République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université SAAD DAHLAB- BLIDA1

Faculté des sciences

Département informatique

Mémoire pour l'obtention d'un diplôme de Master en informatique.

Option : Ingénierie du Logiciel

Thème

Spring Framework : vers une amélioration du legacy IT de ELIT

Organisme d'accueil :

الجزائر أنفورماسيون تكنولوجي
El Djazaïr Information Technology
Rejoignez une filiale du Groupe SONELGAZ

Réalisé par :

- OUBEL Djamel Eddine
- REZZOUG Zakaria

Présenté le 08 Septembre 2020, devant le jury composé de :

- Mme. CHERFA I — Présidente
- Mme. CHERIGUENE S — Examinatrice
- Mme. LAHIANI N — Promotrice
- Mme. MEGHENEZ L — Encadreur

Promotion 2019-2020

Remerciements

Nous remercions tout d’abord notre dieu ALLAH de nous avoir donnée la santé, le courage, et d’avoir mis des gens extraordinaires sur notre chemin.

Nous tenons à remercier en premier lieu notre promotrice Mme LAHIANI Nesrine qui nous a fait confiance et qui a toujours été présente que ce soit avec ce conseil ou avec ces orientations qui nous ont beaucoup aider.

Nous remercions notre encadreur MEGHENEZ Lynda qui nous a aider et orienter durant notre stage, elle nous a aidé depuis le début nous avons eu le privilège de travailler avec elle et d’apprendre d’elle.

Nous désirons exprimer notre reconnaissance à toute l’équipe de ELIT qui nous ont aidé durant notre stage à chaque fois qu’on avait un problème y’avait toujours quelqu'un pour nous aider.

Enfin nous tenons à remercier l’ensemble du corps enseignant et administrative qui travail pour que les étudiants aient de bonne conditions de travail.

Table des Matières

Résumé

De nos jours la majorité des entreprises spécialisées dans le développement des applications web utilisent Java EE qui est une spécification pour la plate-forme Java d'oracle. Étant donné que JEE est très puissant dans son domaine permettant de gérer toutes les relations entre les modes, les vues et les contrôleurs (model MVC), il coordonne la base de donnée (back-end) avec le front-end mais malgré tout ça il a des défauts qu'on ne peut négliger tell que l'infrastructure et le coût de tout cela.

C'est pour ça que les entreprises commencent à chercher des solutions et l'une de ces solutions est d'utiliser Spring Framework qui permet de corriger la majorité des problèmes de JEE mais pour passer de JEE à Spring on ne peut pas le faire directement cela nécessite des connaissances que ce soit dans JEE ou bien dans Spring et c'est ça l'essence même de notre projet,

Comment ce fait exactement cette migration ?, tous les étapes qu'on doit suivre pour pouvoir faire une migration complète des données la plus optimale qui soit.

Mots clés : Spring, JavaEE, Web, Base de données, Back-end, Front-end, MVC, Java,

Abstract

Nowadays the majority of companies specializing in the development of web applications use JavaEE (is a specification for the Oracle Java platform) because JEE is very powerful in its domain to manage all the relationships between models, views and controllers (MVC model). it coordinates the database (back-end) with the views (front-end) but despite all that it has flaws that we cannot neglect such as the infrastructure and the cost of all this.

This is why companies started to look for solutions and one of these solutions was to use its Spring Framework, which allows to correct the majority of JEE problems but to go from JEE to Spring you cannot do it directly. to do this migration requires knowledge whether in JEE or in Spring and this is the essence of our project,

How exactly does this migration do, all the steps that must be followed to be able to make the most optimal complete data migration possible.

Key words: Spring, JavaEE, Web, Database, Back-end, Front-end, MVC, Java,

ملخص

في الوقت الحاضر ، تستخدم غالبية الشركات المتخصصة في تطوير تطبيقات الويب JavaEE (وهي مواصفات لمنصة Oracle Java الأساسية) لأن JEE قوية جدًا في مجالها لإدارة جميع العلاقات بين النماذج وطرق العرض و المتحكم (نموذج MVC) ينسق قاعدة البيانات (النهاية الخلفية) مع العروض (الواجهة الأمامية) ولكن على الرغم من كل ذلك به عيوب لا يمكننا إهمالها مثل البنية التحتية وتكلفة كل هذا

هذا هو السبب في أن الشركات بدأت في البحث عن حلول وكان أحد هذه الحلول هو استخدام إطار Spring Framework الذي يسمح بتصحيح غالبية مشاكل JEE ولكن الانتقال من JEE إلى Spring لا يمكننا القيام بذلك مباشرة . للقيام بذلك بشكل مباشر، يتطلب الأمر معرفة سواء في JEE أو في Spring وهذا هو جوهر مشروعنا،

كيف يعمل هذا الترحيل بالضبط، كل الخطوات التي يجب إتباعها لتكون قادرًا على تحقيق أقصى قدر ممكن من ترحيل البيانات

كلمات مفتاحية Spring: ، JavaEE ، الويب ، قاعدة البيانات ، الواجهة الخلفية ، الواجهة الأمامية ، MVC، Java

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ABREVIATIONS

JEE Java EntrepriseEdition

EJB Enreprise Java Beans

API Application Programming Interface

JSP Java Server Pages

JSF Java Server Faces

MVC Model View Controller

JTA Java Transaction Api

JPA Java Persistence Api

JSON JavaScript Object Notation

JDNI Java Naming and Directory Interface

POJO plain old Java object

JMS Java Message Service

ORM Mapping Object-Relationnal

IOC Inversion of control

DI Dependecy Injection

AOP Aspect Oriented Programming

Introduction générale

Java EE est en fait un ensemble de spécifications, de technologies et d'APIs pour développer et déployer des applications d'entreprise qui peuvent généralement être classées comme des applications à grande échelle, distribuées, transactionnelles et à haute disponibilité conçues pour prendre en charge les exigences commerciales critiques.

ELIT en sa qualité d'éditeur de solution informatique au profit des filiales du groupe SONELGAZ, a développé plusieurs systèmes d'information basés sur la technologie Java EE full web. Avec l'évolution fulgurante que connait le domaine du développement SI basés sur les technologies Java, on a constaté que plusieurs nouveautés ont vu le jour. Ces nouveautés ont pris en charge beaucoup de faiblesses contournées avec Java EE.

Par ailleurs, le Framework Spring comporte une grande communauté et plusieurs projets simplifiant la gestion du développement et tests de SI en Java.

1. Problématique

Malgré le fait que JEE est un langage ultra puissant dans son domaine il comporte quand même quelques faiblesses c'est pourquoi on doit trouver une autre solution à ces problèmes, et la solution la plus évidente reste celle de faire une migration vers le framework Spring. On peut citer quelques faiblesses :

- Nécessité d'avoir des serveurs web et d'application pour déployer une solution, c'est à dire la nécessité une infrastructure qui est couteuse.
- Lourdeur dans l'intégration de librairies et les interfaces sur les projets Java EE. En effet, en étant sur une solution Java EE, on n'intègre que JSF/JSP en couche présentation, et EJB dans le métier.
- Instabilité du support communautaire par la stagnation de l'évolution de Java EE. En effet, le projet JavaEE a été abandonné par le nouveau propriétaire de Java (Oracle) et n'a été repris par la communauté Eclipse Fondation (Jakarta EE) que récemment.

2. Objectifs de projet

ELIT nous a confié la responsabilité de faire une étude détaillée du framework Spring ainsi que Java EE, et établir une analyse comparative entre les deux technologies afin de tracer une feuille de route pour l'adoption de la technologie Spring. Cette feuille de route permettra à long terme de récupérer le « legacy » des solutions existantes. Pour l'application de Cette feuille de route, le développement d'un module du système gestion de parc roulant, est considéré comme prototype pour le reste du « legacy ».

3. Organisation de mémoire

Afin de présenter notre travail, le présent mémoire est organisé en trois parties et se présente comme suit :

Après une introduction générale du projet, ainsi que la problématique et les objectifs visés. La première partie présente l'état de l'art qu'on a introduit à travers trois chapitres :

- Le chapitre un, présente la définition de JEE ainsi que de tout c'est composant.
- Le chapitre deux présentes la définition de Spring et de tout c'est composant.
- Le chapitre trois, dans ce chapitre nous avons abordé le point de la migration et comment elle ce fait concrètement.

Dans la deuxième partie, nous présentons le travail réalisé au sein du Groupe SONELGAZ à travers les six suivants chapitres :

- Le chapitre quatre a pour vocation de présenter Organisme d'accueil et l'étude de l'existant.
- Le chapitre cinq, contient la conception de notre solution. Il présente entre autre les différents diagrammes d'UML des activités recensées.
- Le chapitre six c'est dans ce chapitre que donnons les définitions de l'environnement de développement et nous avons aussi appliqué la feuille de route déjà tracé dans le chapitre précédant (chapitre 4) tout en montrant les résultats.

Une conclusion générale est proposée afin de synthétiser le travail réalisé et de citer les perspectives du projet.

Chapitre 1 La technologie Java EE

La technologie Java EE

Dans ce chapitre on va étudier la technologie Java EE.

1. Introduction

Aujourd'hui, les développeurs reconnaissent de plus en plus le besoin d'applications distribuées, transactionnelles et portables exploitant la vitesse, la sécurité et la fiabilité de la technologie côté serveur. Les applications d'entreprise fournissent la logique métier d'une entreprise. Ils sont gérés de manière centralisée et interagissent souvent avec d'autres logiciels d'entreprise. Dans le monde de la technologie de l'information, les applications d'entreprise doivent être conçues, construites et produites pour moins d'argent, plus rapidement et avec moins de ressources.

Avec Java Platform, Enterprise Edition (Java EE), le développement d'applications d'entreprise Java n'a jamais été aussi simple et rapide. L'objectif de la plate-forme Java EE est de fournir aux développeurs un ensemble puissant d'API, tout en réduisant les délais de développement, en réduisant la complexité des applications et en améliorant leurs performances.

La plateforme Java EE est développée via le Java Community Process (JCP), responsable de toutes les technologies Java. Des groupes d'experts composés de parties intéressées ont créé des demandes de spécification Java (JSR) pour définir les différentes technologies Java EE. Le travail de la communauté Java dans le cadre du programme JCP contribue à garantir les normes de stabilité et de compatibilité entre plates-formes de la technologie Java.

La plateforme Java EE utilise un modèle de programmation simplifié. Les descripteurs de déploiement XML sont facultatifs. Au lieu de cela, un développeur peut simplement entrer les informations sous forme d'annotation directement dans un fichier source Java et le serveur Java EE configurera le composant lors du déploiement et de l'exécution. Ces annotations sont généralement utilisées pour incorporer dans un programme des données qui seraient autrement fournies dans un descripteur de déploiement. Avec les annotations, vous mettez les informations de spécification dans votre code à côté de l'élément de programme affecté.

Sur la plateforme Java EE, l'injection de dépendance peut être appliquée à toutes les ressources dont un composant a besoin, masquant ainsi la création et la recherche de ressources du code de l'application. L'injection de dépendance peut être utilisée dans les

conteneurs EJB (Enterprise JavaBeans), les conteneurs Web et les clients d'application. L'injection de dépendance permet au conteneur Java EE d'insérer automatiquement des références à d'autres composants ou ressources requis, à l'aide d'annotations.

2. Evolution de JEE

1998 a vu la sortie de la première incarnation d'Enterprise Java, elle a subi plusieurs changements depuis sa création. Nous allons couvrir dans cette partie de ce chapitre la chronologie des différentes versions et les fonctionnalités introduites dans chaque version dans un ordre chronologique.

Le tableau suivant représente la chronologie des différentes versions de JEE aves ses fonctionnalités :

Version	Fonctionnalité
JPE 1.0 (1998)	Annonce du projet JPE (Java Professional Edition) chez Sun.
J2EE 1.2 (1999)	J2EE 1.2 supportait les spécifications suivantes : -JDBC Standard Extension API 2.0 -Java Naming and Directory Interface Specification (JNDI) 1.2 -RMI-IIOP 1.1 -Java Servlet 2.2 -JavaServer Pages (JSP) 1.1 -Enterprise JavaBeans (EJB) 1.1 -Java Message Service API (JMS) 1.0 -Java Transaction API (JTA) 1.0 -JavaMail API 1.1

	-JavaBeans Activation Framework (JAF) 1.0
J2EE 1.3 (2001)	J2EE 1.3 a introduit les fonctionnalités suivantes: -Ajout de la bibliothèque de balises standard JSTL (JavaServer Pages), du service JAAS (Java Authentication and Authorisation Service) et de l'architecture du connecteur J2EE. -Prise en charge abandonnée du modèle d'objet distribué lourd (RMI-IIOP / CORBA) entièrement en faveur d'une solution basée sur XML.
J2EE 1.4 (2003)	En réponse aux nouveaux services Web, J2EE 1.4 a introduit les fonctionnalités suivantes : -le support des services Web basés sur XML. Cette version a également ajouté la technologie JSF (JavaServer Faces) basée sur les composants, destinée à remplacer JSP.
Java EE 5 (2006)	J2EE a été rebaptisé Java EE 5, ce qui correspond à Java SE 1.5. -Beans entité remplacés par une JPA indépendante. Il a également ajouté une API Java pour les services Web basés sur XML (JAX-WS) et SOAP avec API de pièces

	jointes pour Java (SAAJ).
Java EE 6 (2009)	Tirant parti d'autres infrastructures telles que Spring, Java EE 6 a présenté les fonctionnalités suivantes: -Java EE 6 a introduit le concept de profil, qui représente une configuration de la plate-forme adaptée à une classe d'applications donnée. Le profil Web offre une pile complète, avec des technologies d'adressage (JavaServer Faces, JavaServer Pages), de fonctionnalités de conteneur Web (Servlet), de logique métier (Enterprise JavaBeans Lite), de transactions (API Java Transaction), de persistance (Java Persistence API) et plus. -L'API Java pour les services Web RESTful (JAX-RS) inclus dans Java EE6. -Java EE 6 a également standardisé DI, AOP et autres, de Spring en contextes et en injection de dépendances pour Java, la validation de beans, les beans gérés et les intercepteurs.
Java EE 7 (2013)	S'appuyant sur d'autres frameworks tels que Play et sur les tendances du Big Data, Java EE 7 a introduit les fonctionnalités suivantes: -Java EE 7 a ajouté l'API Java pour WebSocket, l'API Java pour le traitement JSON et l'Asynchrone Servlet et le NIO non bloquant, une des principales raisons pour

	lesquelles Play Framework a été démarré. -En réponse à la tendance émergente du Big Data, Java EE 7 a ajouté une solution complète pour les applications batch pour la plate-forme Java.
Java EE 8 (2016)	Java EE 8 a été publié en 2016, JEE 8 a introduit les fonctionnalités suivantes [16]: -Support pour la norme HTTP 2.0 émergente -Support pour les événements envoyés par le serveur -Java API pour la liaison JSON (JSR-367) - Modèle de contrôleur de vue (MVC) basé sur l'action (JSR-371) pour compléter le JSF basé sur les composants -JCache (JSR-107) -Plus support Cloud (configuration, locataires multiples, sécurité, API basées sur REST pour la surveillance et la gestion).

Tableau 1 L'évolution de JEE [w3]

La figure 5 représente l'évolution de JEE selon l'ordre chronologique :

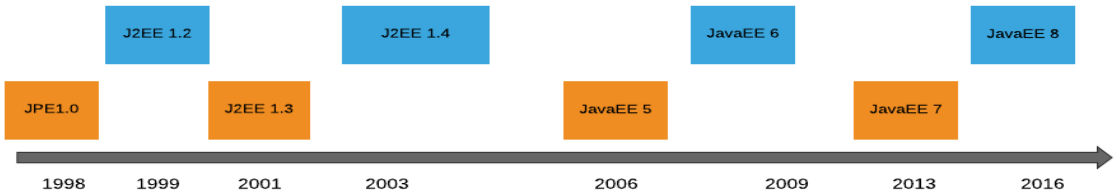


Figure 1 L'évolution de JEE [w4]

3. Le modèle MVC

La plupart des plateformes de développement des applications web y compris Java EE n'imposent aucun rangement particulier du code, c'est-à-dire qu'on peut développer n'importe comment. Le problème c'est que si on développe n'importe comment notre code va être mal organisé et il devient très vite difficile de retrouver un bout de code ou une fonction qu'on veut modifier. Pour éviter ce problème, les développeurs ont aujourd'hui tendance à utiliser les bonnes pratiques de développement qu'on appelle les Patrons de conception(Design Pattern) un modèle de conception est en quelque sorte une ligne de conduite qui permet de décrire les grandes lignes d'une solution. Le modèle MVC découpe littéralement l'application en couches distinctes et de ce fait impacte très forcement l'organisation du code.[4]

Quand on veut développer avec le MVC on segmente son code en trois parties ou couches, chaque couche ayant une fonction bien précise.

3.1 La couche vue

C'est la partie du code qui s'occupera de la présentation des données à l'utilisateur, elle retourne une vue des données venant du modèle, en d'autres termes c'est elle qui est responsable de produire les interfaces de présentation de votre application à partir des informations qu'elle dispose (page HTML par exemple). Cependant, elle n'est pas seulement limitée au HTML ou à la représentation en texte des données, elle peut aussi être utilisée pour offrir une grande variété de formats en fonction des besoins.

3.2 La couche contrôleur

C'est la couche chargée de router les informations, elle va décider qui va récupérer l'information et la traiter. Elle gère les requêtes des utilisateurs et retourne une réponse avec l'aide de la couche Modèle et Vue.

3.3 La couche modèle

C'est la partie du code qui exécute la logique métier de l'application. Ceci signifie qu'elle est responsable de récupérer les données, de les convertir selon les concepts de la logique de l'application tels que le traitement, la validation, l'association et toute autre tâche concernant la manipulation des données. Elle est également responsable de l'interaction avec la base de

données, elle sait en quelque sorte comment se connecter à une base de données et d'exécuter les requêtes (CREATE, READ, UPDATE, DELETE) sur une base de données.

La figure 6 représente l'architecture MVC :

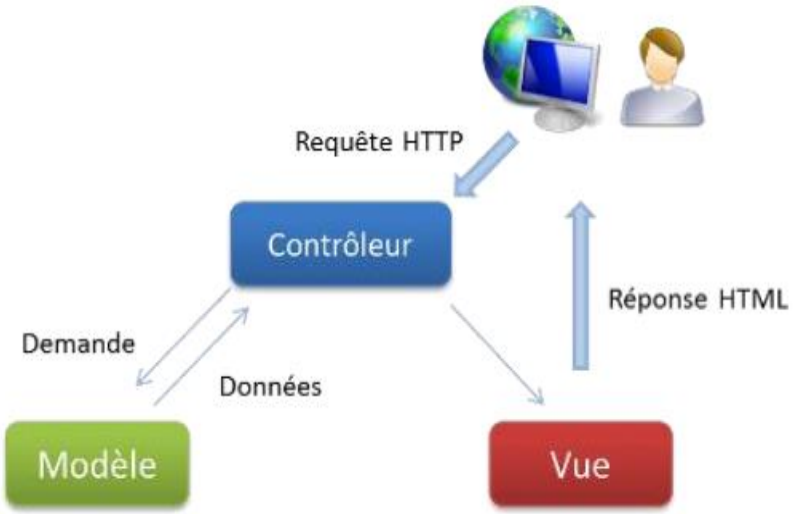


Figure 2 L'Architecture MVC [w5]

4. Comment JAVA EE implémente-t-il le modèle MVC

Avec la plateforme JEE, chaque élément du modèle MVC porte en quelque sorte un nom. Le Contrôleur porte le nom de Servlet. Le modèle est en général géré par des objets Java ou des JavaBeans. Il peut être amené aussi à communiquer avec les bases de données pour stocker les informations, pour les persister et les garder en mémoire le plus longtemps possible. La Vue quant à elle est gérée par les pages JSP (Java Server Pages) qui sont en effet des pages qui vont utilisées du code HTML et du code spécifique en général en JAVA. Cette Vue est donc retournée au visiteur par le Contrôleur.

4.1 Le Servlet

En java EE un Servlet est une classe Java ayant la capacité de permettre le traitement des requêtes et de personnaliser les réponses, c'est-à-dire qu'il est capable de recevoir les requêtes HTTP envoyées depuis le navigateur (Client web) de l'utilisateur et de lui renvoyer une réponse http. Un Servlet http doit toujours hériter de la classe HttpServlet qui est en effet une classe abstraite qui fournit des méthodes qui doivent être redéfini dans la classe héritière.

4.2 Les pages JSP

Une page JSP est destinée à la vue (présentation des données). Elle est exécutée côté serveur et permet l'écriture des pages. C'est un document qui a première vue, ressemble beaucoup à une page HTML.

4.3 Les JavaBeans

Les Javabeans sont un modèle de composants du langage Java. Ce sont des 'pièces logicielles' qui peuvent être réutilisées pour créer des programmes dans un environnement visuel de développement d'applications. Elles représentent généralement les données du monde réel. Les principaux concepts mis en jeu pour un Bean sont les suivants :

Propriétés : Un JavaBean doit pourvoir être paramétrable. Les propriétés c'est à-dire les champs non publics présent dans un bean. Qu'elles soient de type primitif ou objets, les propriétés permettent de paramétrer le bean, en y stockant des données.

La sérialisation : Un bean est construit pour pouvoir être persistant. La sérialisation est un processus qui permet de sauvegarder l'état d'un bean et donne ainsi la possibilité de le restaurer par la suite. Ce mécanisme permet une persistance de données voir de l'application elle-même.

La réutilisation : Un bean est un composant conçu pour être réutilisable. Ne contenant que des données du code métier, un tel composant n'a en effet pas de lien direct avec la couche de présentation, et peut également être distant de la - couche d'accès aux données. C'est cette indépendance qui lui donne ce caractère réutilisable.

L'introspection : un bean est conçu pour être paramétrable de manière dynamique. L'introspection est un processus qui permet de connaître le contenu d'un composant (attributs, méthodes et événements) de manière dynamique, sans disposer de son code source. C'est ce processus, couplé à certaines règles de normalisation, qui rend possible une découverte et un paramétrage dynamique du bean.

Ainsi tout objet conforme à ces quelques règles peut être appelé Bean. Il est également important de noter qu'un JavaBean n'est pas un EJB.

5. Les composants JEE

Les applications Java EE sont constituées des composants. Un composant Java EE est une entité logicielle fonctionnelle autonome qui est assemblée dans une application Java EE avec ses classes et fichiers associés et qui communique avec d'autres composants. Le développement des applications JEE reposent sur un découpage en couche ou tiers (MVC), on parle alors d'applications multi-tiers. Trois grands tiers sont ainsi représentés: Modèle, Vue et Contrôleur dans lesquelles sont repartis les différents composants. La spécification Java EE définie les composants Java EE suivants :

- Les composants Clients ou tiers Client qui sont exécutés côté client
- Les composants Web ou tiers Web exécutés côté serveur (Moteur web)
- Les composants métier ou tiers Métier exécutés côté serveur (Moteur d'application)

5.1 Les composants client

Les composants clients peuvent être des clients web ou des clients riches (Applications clientes).

- **Clients Web**

  Les clients web sont composés de deux parties, des pages web dynamiques qui contenant différents types de langage de balise (HTML, XML, etc.), générées par des composants web s'exécutant dans la couche tiers Web et un navigateur web qui présente ou affiche les pages envoyées par le serveur. Un client web est parfois appelé client léger. Les clients légers ne consultent généralement pas les bases de données, n'exécutent pas de règles métiers complexes ou ne se connectent pas aux applications héritées. Lorsqu'on utilise un client léger, ces opérations lourdes sont déchargées sur les beans entreprise s'exécutant sur le serveur Java EE, où ils peuvent tirer parti de la sécurité, la vitesse, des services et de la fiabilité des technologies Java EE.

- **Applets**

  Une application cliente s'exécute sur un ordinateur client et permet aux utilisateurs de gérer des tâches nécessitant une interface utilisateur plus riche que celle fournie par un langage de balisage. Elle possède généralement une interface utilisateur graphique (GUI) créée à partir de l'API AWT (Swing ou Abstract

Window Toolkit), mais une interface de ligne de commande est certainement possible. Les Applications clientes accèdent directement aux beans d'entreprise s'exécutant dans le niveau métier (voir Figure ci-dessous). Toutefois, si les exigences de l'application le justifient, un client d'application peut ouvrir une connexion HTTP pour établir une communication avec une Servlet s'exécutant dans la couche Web tiers. Les applications clientes écrites dans des langages autres que Java peuvent interagir avec les serveurs Java EE, ce qui permet à la plate-forme Java EE d'interagir avec les systèmes et clients qui ne sont pas développé en Java.

5.2 Les composants Web

Les composants Web Java EE sont des servlets ou des pages créés à l'aide de la technologie JSP (pages JSP) et / ou de la technologie JavaServer Faces (JSF). Les servlets sont des classes de langage de programmation Java qui traitent dynamiquement les requêtes et construisent des réponses. Les pages JSP sont des documents textuels qui s'exécutent en tant que servlets mais permettent une approche plus naturelle de la création de contenu statique. La technologie JavaServer Faces s'appuie sur les servlets et la technologie JSP et fournit une infrastructure de composants d'interface utilisateur pour les applications Web. Les pages HTML statiques et les applets sont regroupés avec des composants Web lors de l'assemblage de l'application, mais ne sont pas considérés comme des composants Web par la spécification Java EE. Les classes d'utilitaires côté serveur peuvent également être regroupées avec des composants Web et, à l'instar des pages HTML, ne sont pas considérées comme des composants Web. Comme le montre la Figure ci-dessous, le Web tiers, tout comme le client tiers, peut inclure un composant JavaBeans pour gérer l'entrée utilisateur et envoyer cette entrée aux beans entreprise s'exécutant dans le tiers métier (Business tiers).

5.3 Les composants métiers

Le code métier, qui est la logique qui résout ou répond aux besoins d'un domaine d'activité particulier tel que la banque, le commerce de détail ou la finance, est géré par des beans entreprise s'exécutant dans côté métier. La Figure ci-dessous montre comment un entreprise bean reçoit des données de programmes client, les traite (si nécessaire) et les envoie au niveau du système d'information d'entreprise pour le stockage. Un

entreprise bean récupère également les données du stockage, les traites (si nécessaire) et les renvoie au programme client.

La figure suivante représente l'architecture de ces composants.

La figure 7 représente l'architecture des composants JEE :

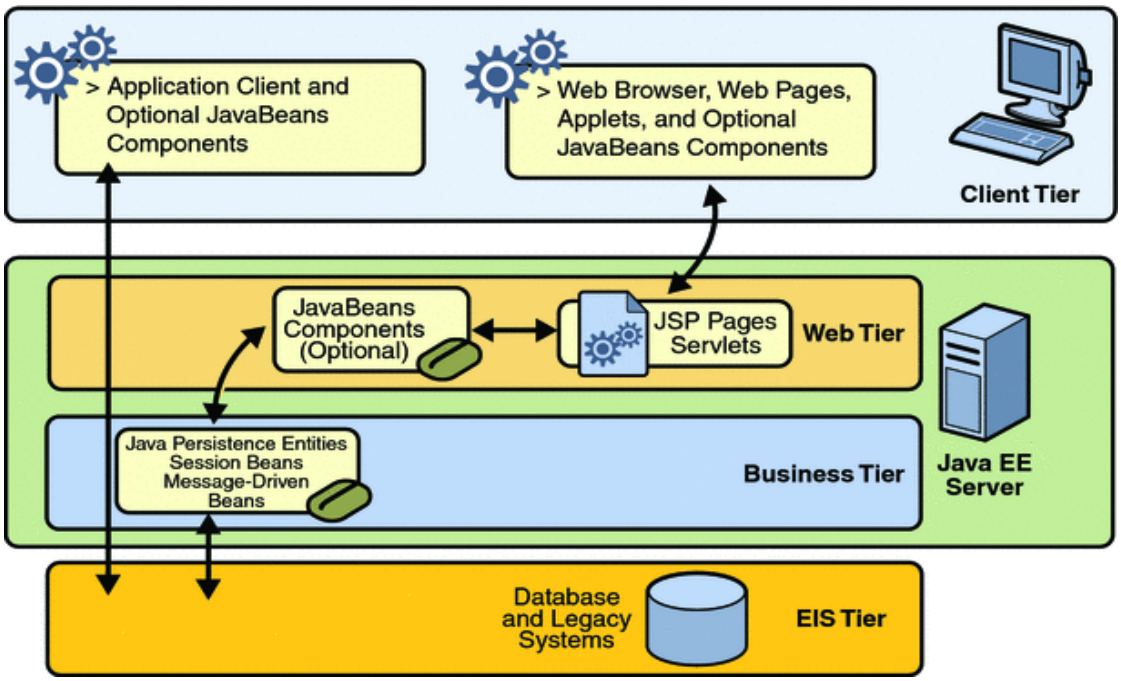


Figure 3 Architecture des composants [w6]

6. Les conteneurs Java EE

Normalement, les applications multi-tiers sont difficiles à réaliser car elles impliquent d'écrire de nombreuses lignes de codes complexes pour gérer les transactions, les accès aux bases de données, le pool de ressources et d'autres détails complexes de bas niveaux. L'architecture Java EE indépendante de ses composants et des plates-formes rend les applications Java EE facile à développer, en effet la logique métier est organisée en composants réutilisables. En outre, le serveur Java fournit des services sous-jacents sous la forme d'un conteneur pour chaque type de composant. Les conteneurs fournissent donc un support ou environnement d'exécution pour les composants des applications Java EE. Ils fournissent une vue fédérée des API Java EE sous-jacentes aux composants de l'application. Les composants d'application Java EE n'interagissent pas directement avec d'autres composants d'applications Java EE. Ils utilisent les protocoles et les méthodes du conteneur pour interagir entre eux avec les services de la plate-forme. Nous avons dit précédemment que les applications JEE utilisaient un serveur d'application et un serveur web, en réalité, il s'agit d'un seul

serveur appelé Serveur JAVA qui propose plusieurs types de conteneurs ou moteur pour chaque type de composant. Le conteneur web qui constitut l'environnement d'exécution des servlets, des pages JSP et JSF et le conteneur EJB quant à lui constitut l'environnement d'exécution des Entreprises Java Bean (EJB) Chaque conteneur à une fonction bien défini et offre aux développeurs un ensemble de services tels que :

- L'annuaire de nommage d'accès aux ressources : Java Naming and Directory Interface (JNDI) qui est une interface unifiée de gestion de nommage pour les services et l'accès à ces derniers par les applications.
- L'injection dynamique des ressources.
- La gestion des accès aux bases de données.
- Le modèle de gestion de la sécurité.
- Le paramétrage des transactions.
- Les connexions distantes.

L'exécution et le développement d'application JEE sont donc directement liés au conteneur utilisé. C'est-à-dire qu'une application JEE Web Tiers utilisera uniquement le conteneur Web pour son développement et de même une application JEE Business tiers utilisera uniquement le conteneur d'application pour son développement et son exécution.

La figure 8 montre les conteneurs JEE :

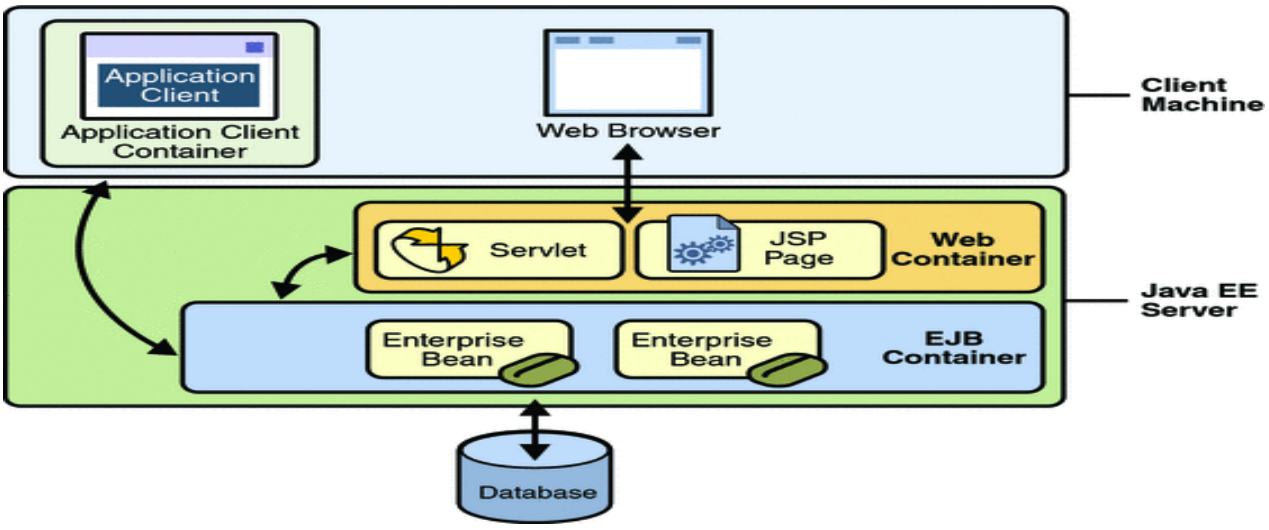


Figure 4 Les conteneurs Java EE [w6]

7. Les APIs de JEE

Nous avons dit précédemment que la plateforme JAVA EE spécifiait les infrastructures de gestion de nos applications au travers composants sous forme d'API En effet cette plateforme

fournit essentiellement un environnement d'exécution et un ensemble de services accessibles via des APIs pour aider à concevoir nos applications. Ainsi donc, tout serveur JEE va fournir à nos applications un ensemble de services comme la connexion aux bases de données, la messagerie, la gestion des transactions et etc. L'architecture JEE unifie l'accès à ces services au sein des API de services d'entreprise, mais plutôt que d'avoir accès à ces services à travers des interfaces propriétaires, les applications JEE peuvent accéder à ces API via le serveur. [8]

La spécification de la plateforme J2EE prévoit un ensemble d'extensions Java standard que chaque plate-forme JEE doit prendre en charge :

- **JNDI**: Java Naming and Directory Interface permet de localiser et d'utiliser les ressources. C'est une extension JAVA standard qui fournit un API uniforme permettant d'accéder à divers services de noms et de répertoires. Derrière JNDI il peut avoir un appel à des services tels que : CORBA, DNS, LDAP, etc. [6]
- **JDBC** : Java Database Connectivity est une API qui permet aux programmes JAVA de se connecter et d'interagir avec les bases de données SQL
- **JMS**: Java Message Service est à la fois une ossature et une API permettant aux développeurs de construire des applications professionnelles qui se servent de messages pour transmettre des données.
- **JTA** : Java Transaction API définit des interfaces standards entre un gestionnaire de transactions et les éléments impliqués dans celle-ci : l'application, le gestionnaire de ressource et le serveur.
- **JSP** : c'est une extension de la notion de Servlet permettant de simplifier la génération de pages web dynamiques. Elle se sert de balises semblables au XML ainsi que de scriplets Java afin d'incorporer la logique de fabrication directement dans le code HTML. [6]
- **Servlet** : Un Servlet est un composant coté serveur, écrit en Java, dont le rôle est de fournir une trame générale pour l'implémentation de paradigmes " requête-réponse ". Ils remplacent les scripts CGI tout en apportant des performances bien supérieures. [6]
- **EJB :** Enterprise JavaBeans (EJB) est une architecture de composants logiciels côté serveur. Elle nous permet de créer des composants distribués, c'est-à-dire déployés sur des serveurs distants écrit en langage de programmation Java hébergés au sein d'un serveur applicatif permettant de représenter des données.[8]

- **JPA :**Java Persistence Api Les entités Beans ont été développées pour le modèle de persistance en Java EE. Ce modèle de composants avait de nombreux détracteurs. Pour apporter des solutions à ce problème, de nouvelles spécifications, des outils de mapping objet / relationnel comme TopLink et Hibernate ont été développés. Java EE apporte donc un nouveau modèle de persistance nommé JPA. JPA s'appuie sur JDBC pour communiquer avec la base de données mais permet d'éviter de manipuler directement les fonctionnalités de JDBC et le langage SQL.[6]
- **Authentification :** JEE fournit des services d'authentification en se basant sur les *concepts d'utilisateur, de domaines et de groupes.

8. Les Frameworks Java EE

Un Framework est un ensemble de composants permettant de réaliser ou de créer l'architecture et les grandes lignes d'une application. Il se présente sous diverse forme qui peut inclure tout une partie des éléments suivants :

- Un ensemble de classes généralement regroupées sous la forme de bibliothèques pour proposer des services plus ou moins sophistiqués.
- Un cadre de conception reposant sur les designs patterns pour proposer tout ou une partie d'un squelette d'application.
- Des recommandations sur la mise en œuvre et des exemples d'utilisation.
- Des normes de développement.
- Des outils facilitant la mise en œuvre.

L'objectif d'un Framework est de faciliter la mise en œuvre des fonctionnalités de son domaine d'activité. Il doit permette au développeur de se concentrer sur les tâches spécifiques de l'application à développer plutôt que sur les tâches techniques récurrentes telles que : l'architecture de base de l'application, l'accès aux données, la sécurité (authentification et gestion des rôles), le paramétrage de l'application, la journalisation des événements (logging), etc.

En ce qui concerne JAVA EE, il existe de nombreux Framework tels que JSF , Struts, Spring, Hibernate etc.

9. Serveur Web et Serveur d'application

9.1 Serveur Web

Un serveur Web gère le protocole HTTP. Lorsque le serveur Web reçoit une requête HTTP, il répond par une réponse HTTP, telle que renvoyer une page HTML. Pour traiter une

requête, un serveur Web peut répondre par une image ou une page HTML statique, envoyer une redirection ou déléguer la génération de réponse dynamique à d'autres programmes tels que des scripts CGI, JSP, servlets, ASP ou une autre technologie côté serveur. Quel que soit leur but, ces programmes côté serveur génèrent une réponse, le plus souvent en HTML, à afficher dans un navigateur Web.

Bien qu'un serveur Web ne prenne pas en charge les transactions ou la gestion de connexion à une base de données, il peut utiliser diverses stratégies de tolérance aux pannes et d'évolutivité, telles que l'équilibrage de charge, la mise en cache et la mise en cluster. [w8]

9.2 Serveur d'application

Un serveur d'applications expose la logique métier aux applications client via divers protocoles, y compris éventuellement HTTP. Alors qu'un serveur Web traite principalement de l'envoi de code HTML pour l'affichage dans un navigateur Web, un serveur d'applications permet d'accéder au logique métier à utiliser par les programmes d'application client.

Dans la plupart des cas, le serveur expose cette logique métier via une API, telle que le composant EJB présent sur les serveurs d'applications. En plus, le serveur d'applications gère ses propres ressources. Ces tâches de sécurité comprennent la sécurité, le traitement des transactions, la mise en commun des ressources et la messagerie. Tout comme un serveur Web, un serveur d'applications peut également utiliser diverses techniques d'évolutivité et de tolérance aux pannes. [w8]

9.3 La différence

Le serveur d'applications contient des conteneurs Web, EJB et un serveur Web en tant que partie intégrée. En revanche, un serveur Web contient uniquement un conteneur Web ou de Servlet et peut utiliser un EJB.

Les serveurs Web sont souhaitables pour le contenu statique alors que les serveurs d'applications sont appropriés pour le contenu dynamique.

Le serveur Web ne prend pas en charge le multithreading, tandis que le serveur d'applications assiste les transactions multithreading et distribuées.

Le serveur Web fournit un environnement pour exécuter une application Web et des fonctionnalités telles que la mise en cache et l'évolutivité. Par contre, le serveur

d'applications fournit un environnement pour exécuter des applications Web avec des applications d'entreprise.

10. Conclusion

En somme, la plate-forme Java EE fournit plusieurs APIs et conteneurs qui fournissent des services de bas niveaux au développeur de telle sorte que ces derniers n'ont plus besoin de développer eux même des fonctionnalités basiques mais portent toute leur attention et les savoir-faire sur le développement des fonctions métier de l'application. Quant aux conteneurs elles constituent des environnements d'exécution pour les composants d'application Java EE et fournissent-elles aussi des services tels que la gestion des transactions, la connexion aux bases de données, etc. Mais l'utilisation des EJBs nécessite des serveurs d'application, ces derniers utilisent les ressources du serveur dans le déploiement d'application mais aussi nécessitent une infrastructure qui est couteuse. Alors qu'il y a le framework Spring qui Peut utiliser à la fois les serveurs d'application et les serveurs web qui utilisent les ressources du client et qui sont aussi peu couteuses. L'infrastructure de Spring est similaire à un serveur d'application JavaEE mais l'avantage de Spring est que les classes n'ont pas besoin d'implémenter une interface pour être prises en charge. C'est en ce sens que Spring est qualifié de conteneur léger.

Spring Framework

Dans ce chapitre on va faire une étude sur Spring Framework.

1. Introduction

Spring Framework est une plateforme Java qui fournit un support d'infrastructure complet pour le développement d'applications Java. Il est effectivement un conteneur dit « léger », c'est-à-dire une infrastructure similaire à un serveur d'application JEE. Spring gère l'infrastructure pour qu'on puisse nous concentrer sur notre application.

Spring nous permet de créer des applications à partir de «plain old Java objects» (POJOs) et d'appliquer des services d'entreprise de manière non invasive aux POJOs. Cette capacité s'applique au modèle de programmation Java SE et à Java EE complet et partiel. Exemples de la façon d'utilisation des avantages de la plateforme Spring:

- Faire exécuter une méthode Java dans une transaction de base de données sans avoir à gérer les API de transaction.
- Faire d'une méthode Java locale une procédure distante sans avoir à traiter avec des API distantes.
- Faire d'une méthode Java locale une opération de gestion sans avoir à traiter avec les API JMX.
- Faire d'une méthode Java locale un gestionnaire de messages sans avoir à traiter avec les API JMS.

2. Evolution de Spring framework

Le tableau suivant représente la chronologie des différentes versions de Spring Framework aves ses fonctionnalités [w8] :

Version	
Spring 1.0 (mars 2004)	implémente les fonctions de base du framework: - Conteneurs qui implémentent des modèles de conception IoC. - Développement orienté POJO. - AOP par déclaration. - Prise en charge des frameworks JDBC, ORM et Web. - Configuration XML basée sur DTD.

Spring 1.2 (mai 2005)	- Prise en charge JMX. - Prise en charge de JDO 2, Hibernate 3 et TopLink. - Prise en charge des séquences JCA CCI et JDBC. Utilisez @Transactionional pour déclarer des transactions
Spring 2.0 (octobre 2006)	Apporte de nombreuses nouvelles fonctions : - Support et utilisation d'AspectJ. - Configuration XML basée sur un schéma XML. - Simplifiez la configuration, en particulier avec des espaces de noms dédiés (Bean, TX, Monkey, Long, Util, Jee, P). - POJO contrôlé par message. - @ référentiel.
Spring 2.5 (novembre 2007)	apporte de nombreuses nouvelles fonctionnalités pour simplifier la configuration [17]: - Ajouter de nouveaux espaces de noms (contexte, JMS) avec de nouveaux noms. - Enrichissement des espaces de noms (jee, aop). - Ajouter des Bean Life Notes (@Service, @Component, @Controller), Autowiring(@Autowired, @Qualifier, @Required), Transaction Management (@Transactional) et la prise en charge des notes standard Java 5 (@PostConstruct, @PreDestroy, @ Ressource). - Teste et commenter l'intégration Junit 4 (@ContextConfiguration, @TestExecutionListeners, @BeforeTransaction, @AfterTransaction).

Spring 3.0 (décembre 2009)	apporte de nombreuses nouvelles fonctionnalités à sa configuration et à ses fonctionnalités: - Possibilité de configurer le contexte à l'aide d'annotations: ces annotations du projet Spring JavaConfig sont ajoutées dans Spring Core (@Configuration, @Bean, @DependsOn, @Primary, @Lazy, @Import, @ImportResource et @Value). Spring Expression Language (SpEL): langage d'expression qui peut être utilisé pour définir des beans dans Spring Core et certaines fonctions dans des projets de portefeuille. - Prise en charge REST. - Mappage objet à XML (OXM): Abstraction utilise une solution de mappage objet / XML initialement proposée par le projet Spring Web Services et intégrée à Spring Core. - Nécessite Java SE 5.0 ou supérieur (révision de l'API pour utiliser Generics, Varargs, java.util.concurrent, etc.). - Nouvelle modularité: la répartition des ressorts dans les verres a été modifiée pour que chaque module ait son propre jar. Le fichier Spring.jar n'est plus fourni. - Prise en charge des moteurs de base de données embarqués (Derby, HSQL, H2). - Prend en charge la validation (JSR 303), la liaison de données et le type de conversion. - Prise en charge de JSR 330. - Planification via configuration, annotations (@Async, @ Scheduled) ou API. - ajouter de nouveaux espaces de noms (task, jdbc, mvc). - Forte compatibilité avec Spring 2.5.

	- Spring MVC prend en charge l'API Porlet 2.0.
Spring 3.1(2011)	- Support des dialogues. - Support des caches. - Le concept de profil a été ajouté, avec lequel une configuration de contexte différente est possible pour chaque environnement - Ajout de nouvelles annotations pour définir certaines fonctions d'espace de noms dans la configuration. - Prend en charge Servlet 3.0.
Spring 4.0 (Décembre 2013)[w10]	Spring 4.0 a été une avancée majeure pour le framework Spring et il a inclus des fonctionnalités telles que la prise en charge complète de Java 8, des dépendances de bibliothèques tierces supérieures (groovy 1.8+, ehcache 2.1+, hibernate 3.6+ etc.), la prise en charge de Java EE 7, groovy DSL pour définitions de bean, prise en charge des websockets et prise en charge des types génériques comme qualificatif pour l'injection de beans.
Spring 5.0 (Juillet 2017)[w10]	- Mise à jour de la ligne de base JDK. - Révision du core du framework. - Mises à jour du conteneur principal. - Programmation fonctionnelle avec Kotlin. - Modèle de programmation réactive. - Support des bibliothèques.

Tableau 2L'évolution de Spring Framework

3. Inversion de contrôle et injection de dépendance

Dans Spring Framework, le module de conteneur principal fournit les fonctionnalités essentielles. Le BeanFactory est un composant principal du conteneur principal et le modèle d'inversion de contrôle (IoC) est appliqué par BeanFactory. IoC est utilisé pour séparer la

configuration d'une application et la spécification de dépendance du code d'application réel et il est réalisé par injection de dépendance.

3.1 Inversion de control IOC

Il est important dans le développement de logiciels pour contrôler le flux des applications tout en assurant une cohésion élevée et un faible couplage.

L'IOC, en revanche, est fondamental pour tout Framework. Avec IOC, un objet d'application est généralement enregistré avec le Framework qui prend la responsabilité d'appeler des méthodes sur l'objet enregistré à un moment ou un événement approprié. Le contrôle est inversé car au lieu que le code d'application invoque l'API du framework, les choses se produisent exactement le contraire.

L'IOC n'est pas un nouveau concept et existe depuis longtemps. Les EJBs, par exemple, prennent en charge IOC. Les différents composants EJBs tels que les beans session, entité et Message-Driven établissent des contrats spécifiques avec le conteneur en implémentant les méthodes définies dans différentes interfaces.

3.2 L'injection de dépendance DI

Il est courant pour les développeurs de croire que IOC et DI sont la même chose. C'est incorrect, ils sont deux concepts différents mais liés. Juste comme IOC s'occupe d'inverser le flux de contrôle dans une application, DI décrit comment un objet résout ou trouve d'autres objets sur lesquels il doit invoquer certaines méthodes. Il existe plusieurs façons d'atteindre DI, et une de ces stratégies est IOC.

L'injection de dépendance existe sous trois formes [19]:

- **Basée-Setter**: les classes sont généralement des JavaBeans, avec un constructeur sans argument et des setters pour le conteneur IoC à utiliser lors du câblage des dépendances. C'est la variante recommandée par Spring. Alors que Spring prend en charge l'injection basée sur les constructeurs, un grand nombre d'arguments constructeurs peuvent être difficiles à gérer.
- **basée-Constructeur:** les classes contiennent des constructeurs avec un certain nombre d'arguments. Le conteneur IoC découvre et appelle le constructeur en fonction

du nombre d'arguments et de leurs types d'objet. Cette approche garantit qu'un bean n'est pas créé dans un état non valide.

• **Basé-Getter** c'est similaire à basée-setter, sauf que vous ajoutez un getter à notre classe. Le conteneur IoC remplace cette méthode lorsqu'il s'exécute à l'intérieur, mais nous pouvons facilement utiliser le getter que nous spécifions lors des tests. Cette approche n'a été discutée que récemment.

4. Architecture modulaire de Spring

Le framework Spring se compose de fonctionnalités organisées en environ 20 modules. Ces modules sont regroupés en Conteneur principal, Accès au données / Integration, Web, AOP (Aspect Oriented Programming), Instrumentation et Test. [w11]

4.1 Le conteneur principal (Core container)

Le conteneur principal comprend les modules Core, Beans, Context et Expression Langage.

- Les modules Core et Bean fournissent les parties fondamentales du framework, y compris les fonctionnalités d'IOC et d'injection de dépendance. Le BeanFactory est une implémentation sophistiquée du modèle d'usine. Il supprime le besoin de singletons programmatiques et vous permet de découpler la configuration et la spécification des dépendances de votre logique de programme réelle.
- Le module Context s'appuie sur la base solide fournie par les modules Core et Beans: c'est un moyen d'accéder aux objets d'une manière de type framework qui est similaire à un registre JNDI. Le module Context hérite de ses fonctionnalités du module Beans et ajoute la prise en charge de l'internationalisation (en utilisant, par exemple, des regroupements de ressources), de la propagation d'événements, du chargement des ressources et de la création transparente de contextes par, par exemple, un conteneur de Servlet. Le module Context prend également en charge les fonctionnalités Java EE telles qu'EJB, JMX et la communication à distance de base. L'interface ApplicationContext est le point focal du module Context. [11]
- Le module Expression LangageSpEL fournit un langage d'expression puissant pour interroger et manipuler un graphique d'objet au moment de l'exécution. Il s'agit d'une extension du langage d'expression unifié tel que spécifié dans la spécification JSP 2.1. Le langage prend en charge la définition et l'obtention de valeurs de propriété, l'attribution de propriété, l'appel de méthode, l'accès au

contexte des tableaux, des collections et des indexeurs, des opérateurs logiques et arithmétiques, des variables nommées et la récupération d'objets par nom à partir du conteneur IOC de Spring. Il prend également en charge la projection et la sélection de listes ainsi que les agrégations de listes courantes. [11]

4.2 Accès au données / Integration

La couche d'accès au données / intégration des données comprend les modules JDBC, ORM, OXM, JMS et Transaction.

- Le module JDBC fournit une couche d'abstraction JDBC qui supprime la nécessité de coder et d'analyser JDBC fastidieux des codes d'erreur spécifiques au fournisseur de base de données.
- Le module ORM fournit des couches d'intégration pour les API de mapping relationnel objet populaires, y compris JPA, JDO, Hibernate et iBatis. En utilisant le package ORM, vous pouvez utiliser tous ces frameworks de mapping O / R en combinaison avec toutes les autres fonctionnalités proposées par Spring, telles que la simple fonctionnalité de gestion déclarative des transactions mentionnée précédemment.
- Le module OXM fournit une couche d'abstraction qui prend en charge les implémentations de mapping objet / XML pour JAXB, Castor, XMLBeans, JiBX et XStream.
- Le module Java Messaging Service JMS contient des fonctionnalités pour produire et consommer des messages.
- Le module Transaction prend en charge la gestion des transactions par programme et déclarative pour les classes qui implémentent des interfaces spéciales et pour tous les POJOs.

4.3 Web

La couche Web comprend les modules Web, Web-Servlet, Web-Struts et Web-Portlet.[11]

- Le module Web de Spring fournit des fonctionnalités d'intégration orientées-Web de base telles que la fonctionnalité de téléchargement des fichiers en plusieurs parties et l'initialisation du conteneur IOC à l'aide d'écouteurs de Servlet et d'un

contexte d'application orienté-Web. Il contient également les parties Web de la prise en charge à distance de Spring.

- Le module Web-Servlet contient l'implémentation Spring MVC pour les applications Web. Le framework MVC de Spring offre une séparation nette entre le code de modèle de domaine et les formulaires Web, et s'intègre à toutes les autres fonctionnalités du framework Spring.

  Le framework web Spring MVC assure le développement de la partie IHM pour les applications web. L'utilisation de ce module peut être remplacée par d'autre framework notamment Struts grâce à la facilité de Spring de s'interfacer avec d'autres framework. [11]
- Le module Web-Struts contient les classes de prise en charge pour l'intégration d'un niveau Web Struts classique dans une application Spring. Notez que cette prise en charge est désormais obsolète à partir de Spring 3.0.
- Le module Web-Portlet fournit l'implémentation MVC à utiliser dans un environnement de portlet et reflète les fonctionnalités du module Web-Servlet.

4.4 AOP et Instrumentation

Le module AOP de Spring fournit une implémentation de programmation orientée aspect conforme à l'AOP Alliance nous permettent de définir, par exemple, des intercepteurs de méthodes et des coupes de points pour découpler proprement du code qui implémente des fonctionnalités qui doivent être séparées. À l'aide de la fonctionnalité de métadonnées au niveau de la source, nous pouvons également incorporer des informations comportementales dans notre code, d'une manière similaire à celle des attributs .NET.

Le module Aspects séparé permet l'intégration avec AspectJ.

Le module Instrumentation fournit un support d'instrumentation de classe et des implémentations de chargeur de classe à utiliser dans certains serveurs d'applications.

4.5 Test

Le module Test prend en charge le test des composants Spring avec JUnit ou TestNG. Il fournit un chargement cohérent de Spring ApplicationContexts et une mise en cache de ces contextes. Il fournit également des objets fictifs que nous pouvons utiliser pour tester notre code de manière isolée.

La figure 9 montre l'architecture modulaire de Spring Framework :

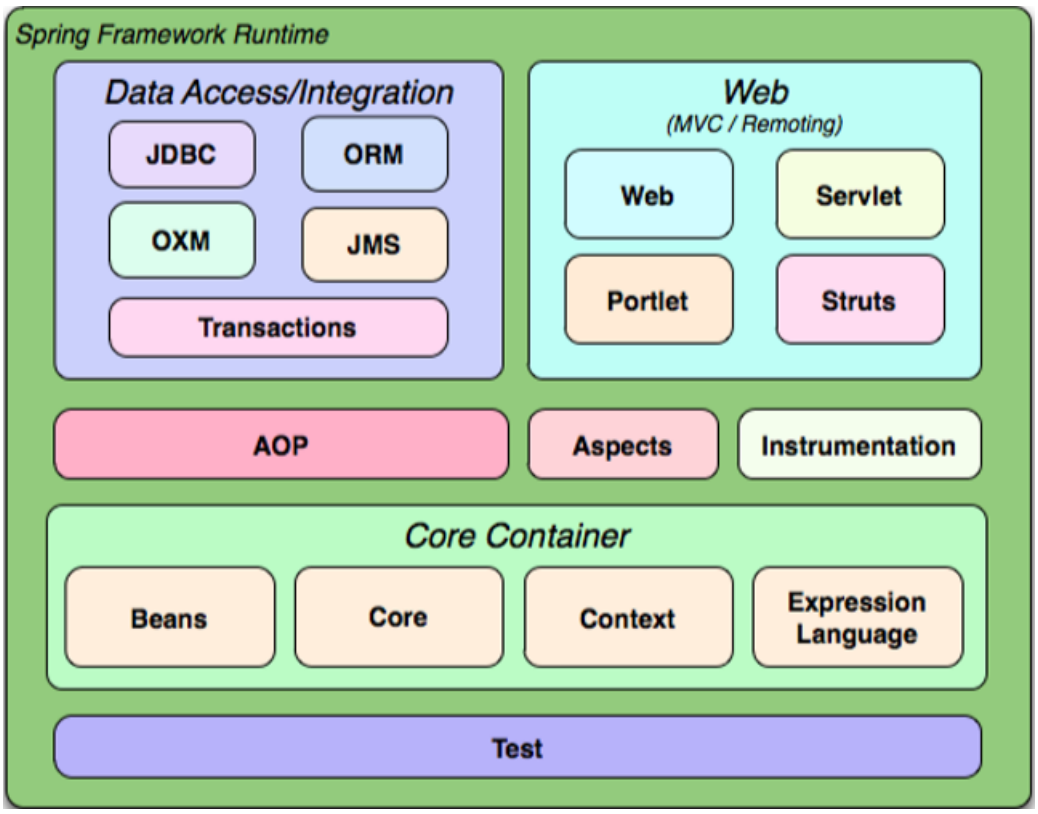


Figure 5 L'architecture modulaire de Spring framework [w11]

5. Les couches d'une application Spring

Une application Spring est découpée en trois couches [6]:

5.1 Couche de présentation

Comme son nom l'indique, Spring MVC fournit des composants de première classe pour créer des applications Web à la suite de patron MVC. Comme tout bon framework Web, Spring MVC aide à construire la couche contrôleur. Il est flexible et prend en charge une variété de composants pour la gestion des vues, y compris les technologies de core JavaEE telles que JSP et d'autres telles que Extensible Stylesheet Language (XSL) et Portable Document Format (PDF). La requête HTTP du navigateur Web du client est d'abord interceptée par le composant contrôleur. Il s'agit d'un servlet de passerelle qui agit comme un seul point d'entrée dans l'application. Le servlet délègue ensuite les demandes des utilisateurs successifs aux gestionnaires respectifs appelés contrôleurs de page. Les contrôleurs de page sont des classes Java simples qui exécutent un cas d'utilisation d'application et appellent les

services métier. Spring MVC contient également une couche de gestion des vues (non représentée par souci de simplicité), qui est chargée de localiser une vue appropriée (qui peut être un autre JSP). Il lie également les objets de données renvoyés par les contrôleurs de page après avoir appelé les objets de modèle dans le niveau métier et envoie enfin la réponse HTTP au client. Tous les objets chargés de la gestion des vues, de la gestion des contrôleurs et des contrôleurs de page eux-mêmes sont enregistrés dans le conteneur Spring IOC sous-jacent. Ainsi, tous les avantages de ce conteneur sont disponibles pour le module Spring MVC. Notez que le servlet de passerelle ne fait pas partie du conteneur Spring IOC, il est plutôt géré par le serveur Web.

Spring MVC est un module polyvalent et prend en charge presque toutes les technologies de vue et de modèle disponibles, il est tout à fait possible d'intégrer des frameworks comme JSF à Spring MVC. Nous pouvons très bien utiliser des moteurs de vue basés sur des modèles tels que Velocity et FreeMarker, des vues basées sur des documents telles que PDF, Microsoft Word et Microsoft Excel; et des interfaces utilisateur riches fournies par Adobe Flex. En ce qui concerne le modèle, Spring MVC non seulement se marie bien avec les composants métier POJO, mais il va aussi bien avec les composants EJB distribués.

5.2 Couche métier

Avec Spring, nous pouvons développer des composants métier en tant que classes Java simples sans aucune dépendance de framework, ce qui contraste complètement avec le modèle de programmation EJB, qui exige la mise en œuvre de plusieurs interfaces différentes ainsi que des descripteurs de déploiement, ce qui complique la vie du développeur. Les objets métier POJOs sont tous configurés dans le conteneur Spring IOC comme tout autre bean. Ils sont responsables de l'exécution des règles métier et, dans le processus, ils manipulent les données d'application à l'aide de l'API d'accès aux données disponible dans le niveau d'intégration. Le module Spring AOP joue un rôle important au niveau de l'entreprise. Il peut être utilisé pour appliquer et contrôler de manière déclarative les transactions et la sécurité sur les composants métier POJOs. Il est possible d'utiliser Spring AOP pour développer des aspects personnalisés qui collectent des informations sur la piste d'audit ou le temps d'exécution de la méthode de l'instrument sans affecter le code d'application existant. Il est possible de développer une solution mixte avec Spring POJOs et EJBs. Spring IOC implémente le modèle de localisateur de service pour rechercher les interfaces d'accueil EJB,

puis injecter ces objets dans les objets métier POJOs. Notons que les services de niveau système fournis par le conteneur EJB sont désormais disponibles pour l'application. Dans ce scénario, Spring Framework a joué le rôle d'un client EJB utilisant un session bean. Spring aide également à la mise en œuvre d'EJB grâce à des superclasses pratiques.

Il est important de noter que Spring MVC n'est pas le seul moyen de se connecter au niveau commercial Spring. Il est possible d'exposer des objets métier Spring en tant que services Web. De même, diverses autres options de communication à distance comme Spring remoting, Burlap-Hessian, etc. peuvent être utilisées pour se connecter aux composants Spring, les rendant disponibles en tant que composants distants.

5.3 Couche intégration

La couche d'intégration de la plupart des applications interagit avec le SGBDR à l'aide de l'API JDBC via des objets d'accès aux données POJOs. Les objets d'accès aux données fournissent une API cohérente pour les objets de niveau métier et encapsulent l'API JDBC. Spring DAO fournit des modèles pour des objets d'accès aux données simples et flexibles. Les objets d'accès aux données mettent à jour les bases de données relationnelles ainsi que les données récupérées. Les données récupérées sont enveloppées dans des objets JavaBeans et renvoyées aux autres couches.

Tout comme les deux autres couches, Spring offre de nombreuses options même dans le niveau d'intégration. Spring ORM permet nous d'utiliser facilement une solution de pont relationnel objet telle que Hibernate ou TopLink. La couche d'intégration dans une application Java EE ne se limite pas à la communication avec des bases de données relationnelles. Il peut être nécessaire de se connecter à un ordinateur central ou à un système ERP ou CRM. Tout comme avec le niveau métier, nous pouvons tirer parti du module Spring JEE pour se connecter à ces systèmes et applications à l'aide de technologies standard telles que JCA ou des services Web.

6. Spring boot

Spring Boot est un micro-framework open source maintenu par une société appelée Pivotal. Il fournit aux développeurs Java une plateforme pour démarrer avec une application Spring auto-configurable, on peut commencer avec des configurations minimales sans avoir besoin d'une configuration complèteSpring. Grâce à lui, les développeurs peuvent démarrer rapidement sans perdre de temps à préparer et à configurer leur application Spring [18].

Spring Boot est construit sur le framework Spring est livré avec de nombreuses dépendances qui peuvent être connectées à l'application Spring. Quelques exemples sont Spring Web Services et Spring Security. Il vise à raccourcir la longueur du code et Augmente la productivité et à nous fournir un moyen facile d'exécuter notre application Spring. On peut exécuter et déployer nos application sur le serveurembarqué de Spring boot (Tomcat par défaut).

Spring Boot ne remplacera pas Spring Framework car Spring Boot est le Spring Framework, nous pouvons regarder Spring Boot comme une nouvelle façon de créer Springapplications avec facilité.[12]

Spring boot a était crée pour :

- éviter la configuration XML compliquéede Spring.
- réduire le temps de développement et exécuter l'application indépendamment grace a son Serveur web embarqué.
- développer plus facilement des applications Spring prêtes pour la production.

7. Conclusion

En conclusion, Spring est un framework puissant pour la création d'applications d'entreprise qui résout de nombreux problèmes courants dans JEE. De nombreuses fonctionnalités Spring sont également utilisables dans une large gamme d'environnements Java, au-delà du classique JEE. Spring offre un moyen cohérent de gérer les objets business et encourage les bonnes pratiques telles que la programmation aux interfaces, plutôt qu'aux classes. Il peut également être facilement intégré avec autre frameworks pour développer des applications d'entreprise efficaces, réduisant ainsi le couplage et la séparation nette des couches. En raison de sa fonction légère, il est facile à utiliser. En utilisant un conteneur léger comme le framework Spring nous permet d'être libres du besoin pour un serveur d'applications. Cela diminue l'administration du serveur d'applications de complexité et permet une plus grande portabilité puisque. Spring ne nécessite qu'un conteneur de servlet. L'utilisation de framework Spring facilite la maintenance des modules et la réutilisation dans d'autres contextes, et les objets sont facilement testables en dehors du conteneur qui fait gagner du temps et diminue.

Chapitre 3

La migration vers Spring

Dans ce chapitre on va traiter le sujet principal de notre recherche, la migration de JEE vers Spring. On va faire une comparaison entre les deux technologies afin de tracer une feuille de route pour faire la migration.

1. Introduction

Il faut faire preuve de prudence lors de la migration des applications existantes et matures hors de l'héritage Java EE full stack vers un autre framework. Comme dans tout plan de projet, les avantages doivent l'emporter sur les coûts. Une compréhension claire de la portée, la complexité et les défis techniques sont des conditions préalables à l'estimation d'un plan de migration. Dans les cas où Java EE full stack est fortement adopté dans une organisation et une application ne continuera pas à être améliorée dans des modes, rester avec Java EE full stack peut avoir plus de sens. Cependant, un certain nombre de considérations peut indiquer qu'il vaut la peine de migrer vers une certaine utilisation de Spring:

- Coût de l'effort de développement avec les technologies traditionnelles s'il est nécessaire d'améliorer ou de modifier les fonctionnalités de l'application de manière significative ou sur une base continue.
- Frais de licence, d'exploitation et de maintenance des technologies lourdes de serveur d'applications Java EE.
- Problèmes de qualité dus à l'incapacité d'implémenter et d'exécuter facilement des tests unitaires et d'intégration dans les applications Java EE traditionnelles.
- Intérêt pour la simplification de l'architecture de déploiement d'une grande plateforme multicouche vers une application Web à une seule couche pour réduire la complexité globale.

Il faut également souligner que l'utilisation de Spring n'est pas une proposition du tout ou rien. Bien qu'il puisse être très bénéfique de refactoriser de manière significative certaine partie du code ou de l'architecture d'une architecture Java EE full stack traditionnelle, c'est assez possible de commencer à utiliser Spring de manière incrémentielle. Dans l'application existante, les nombreuses API de service de Spring peuvent être accédées par programme, ou l'utilisation du conteneur Spring peut être combinée avec l'utilisation continue d'autres modèles de composants, tels qu'EJB.

2. Analyse comparative

JEE est une spécification standard et officielle pour un cycle de développement complet des applications d'entreprise. Elle Comprend des éléments tels que le mappingd'objet-relationnel, la sécurité, les applications Web, la connectivité de base de données, les transactions ...

En plus des spécifications Java EE, il existe des implémentations / serveurs d'applications JavaEE tels que: JBoss, Glassfish, WebSphere, Weblogic.

Spring, d'autre part, est un framework qui fait beaucoup de choses sur les spécifications Java EE, mais sous sa propre forme.Il ne suit pas les spécifications et les API Java EE pour cela. Mais il inclut un Framework Web, la gestion des transactions, la sécurité et plusieurs autres solutions proposées par JEE.

Cette partie représente une analyse comparative selon différentes critères représenté dans le tableau suivant :

Les bases de comparaison	**Java EE**	**Spring framework**
Architecture	- Basé sur un cadre architecturel en trois dimensions, c'est-à-dire des niveaux logiques, des niveaux client et des niveaux de présentation. - Utilisation d'intercepteurs EJB.	- Il est basé sur une architecture simple, plateforme à un seul niveau. - Remplacer par AspectJ de Spring Programmation orientée aspect mécanismes.
Language	-Elle utilise un langage orienté objet de haut niveau qui a un certain style et une certaine syntaxe.	- Il n'a pas un certain modèle de programmation.

Persistance	- Il prend en charge la persistance programmée gérée par les bean et étroitement couplé à JPA.	- Il fournit un framework qui prend en charge l'intégration de diverses technologies de persistance telles que JDBC, Hibernate, JDO et iBATIS.
Les types des beans	- Stateless Session Beans. - Stateful Session Beans. - Beans d'entité gérée par conteneur.	-les stateless session beans n'existent pas dans spring, il existe les spring POJOs en utilisant le scope par défaut singleton. - on peut émuler le comportement des stateful beans en utilisant le scope session dans les spring POJOs ou dans la portée du prototype. - pour les beans d'entités, Simple Data Access Object avec Spring JDBC, ou utiliser Spring ORM et JPA.
Transactions	- Utilisation de UserTransaction JTA API de transaction. - Transactions entièrement distribuées entre plusieurs sources de données.	- Grâce à son interface PlatformTransactionManager, Spring prend en charge plusieurs transactions telles que JTA, Hibernate, JDO et JDBC. - Spring utilise le gestionnaire de transactions JTA pour prendre en charge les transactions distribuées.

Sources de données	-Utilisation des sources de données JNDI.	- Définissez une source de données dans le cadre d'injection de dépendances de Spring et injectez-la dans des objets faisant face aux données. - Conserver la majorité du code tel quel et injecter uniquement la source de données JDBC dans les DAO existants.
Messagerie et services web	- Conteneur JMS intégré, utilise les Message-Driven Beans. - Implémentations de services Web de fournisseurs propriétaires tels que les services web WLI de BEA.	- Implémentez un conteneur JMS externe, tel qu'ActiveMQ ou une alternative commerciale. - Implémentez des services Web basés sur WS-* à l'aide de l'API Spring Web Services.
Sécurité	- Configuration de la sécurité implémentée dans les modules JAAS. - Sécurité attribuée à EJB, composants Web utilisant des descripteurs de déploiement xml.	- Implémentez la sécurité à l'aide de l'API Spring Security. Il peut être connecté à de nombreuses sources de données différentes et peut être configuré pour des méthodes d'autorisation et d'authentification personnalisées. - Utilisez Spring Security, avec des annotations ou dans la configuration Spring xml.

Tableau 3 Analyse comparative entre JEE et Spring

3. Facteurs d'étudier la complexité

Pour faire une telle migration, il faut faire une étude détaillée sur la complexité du projet selon des différents facteurs. Voila les types de complexité :

3.1 Les facteurs d'une faible complexité

- Bonne documentation et compréhension claire du code, de la base de données ou des exigences existants.
- Application en couches bien architecturée.
- Organisation déjà familière avec l'utilisation de technologies légères comme Spring Framework.
- Aucune intégration avec les frameworks de serveurs d'applications propriétaires pour les API.
- Accès simple à la base de données basé sur DAO (à l'aide de JDBC ou ORM).
- Pas de dépendance aux EJB d'entité.
- Pas besoin de support de transaction distribué.

3.2 Les facteurs d'une complexité moyenne

- Dépendance des beans entité.
- Utilisation du fournisseur JMS existant dans le serveur d'applications full stack.
- Sécurité mise en œuvre à l'aide d'API de sécurités tierces telles que SiteMinder, mais souhait d'intégrer ou de passer à Spring Security.
- Forte dépendance organisationnelle d'EJB et de Java EE, Spring n'est pas encore adopté.
- Dépendance à la disponibilité du support des transactions distribuées.

3.3 Les facteurs d'une forte complexité

- Utilisation intensive des technologies spécifiques des EJBs.
- Dépendance à l'égard des API de serveur d'applications personnalisées (ré-architecte pour supprimer d'abord les dépendances).
- Mauvaise documentation et / ou manque de compréhension claire du code, de la base de données ou des exigences existants.

4. Les types de migration

Pour les types de migration on à deux choix, soit on fait une migration complète c'est-à-dire on prend en charge le projet complet et faire la migration des deux cotés back-end et front-end. Sinon on fait une migration partiel c'est-à-dire faire la migration pour les choses qui

doivent être changé dans le coté back-end seulement en gardant le coté front-end. Le choix doit être fait selon la complexité de la migration, et les besoins de l'entreprise.

5. Les étapes de la migration

Dans ce qui suit on va donner les différentes étapes à suivre pour faire la migration d'un projet JEE existant vers Spring Framework :

- La première chose à faire, c'est d'étudier la complexité du projet et décider qu'est ce qu'on va faire, une migration complète ou partiel.
- Pour l'implémentation, de préférence utiliser la dernière version de l'IDE Spring Tool Suit.

Le coté back-end

Dans cette première partie on va commencer avec la migration du back-end c'est à dire on va parler sur les entités et la partie accès au donnée et la couche métier :

- Le point de départ c'est de créer un nouveau projet Spring boot, la classe main sera créée automatiquement par l'IDE STS et annotée avec l'annotation @SpringBootApplication. À partir de cette classe, Spring analysera le chemin de classe et chargera les paramètres trouvés.
- La deuxième chose à faire est Les configurations d'application, l'un des points fort de Spring c'est qu'on a le choix, la configuration peut être basée sur un fichier xml ou on peut créer un classe java annoté avec l'annotation @Configuration pour décrire les différentes configurations et ajouter l'annotation @EnableConfigurations dans la classe main.
- L'étape suivante c'est d'ajouter dans le fichier POM.xml les dépendances des différents API qu'on va utiliser dans notre projet.
- Vu que les APIs JPA et Hibernate peuvent fonctionner correctement avec Spring, du coup on peut utiliser les entités directement sans aucune modification, ce qu'il nous reste à faire c'est d'importer nos entités depuis notre projet JEE.
- Maintenant pour la couche DAO on doit créer des interfaces pour chaque objet au but de gérer les interactions avec la base de données, pour cela il existe deux types de façon de faire :

1- Le premier type c'est de laisser les classes de la couche DAO du projet JEE comme elles sont en ajoutant l'annotation @Repository qui annote les classes au niveau de la

couche de persistance qui agira comme un référentiel de base de donnée. Elle active la traduction des exceptions de persistance pour tous les beans annotés avec @Repository, pour permettre aux exceptions levées par les fournisseurs de persistance JPA d'être converties dans la hiérarchie DataAccessException de Spring. Cela signifie que toutes les exceptions non vérifiées levées dans cette classe seront converties en DataAccessException. On ajoute aussi l'annotation @Transactionnal cette annotation est utiliser pour gérer les transactions (le niveau d'isolement de la transaction, le type de propagation de la transaction, etc.),et aussi elle doit être ajouté dans chaque classe qui apporte une modification a la base de donnée.

Chacune de ces classes doit implémenter l'une des interfaces qu'on a déjà créées car Spring boot a besoin que chaque classe implémente une interface pour pouvoir utiliser l'annotation @Autowired lors de l'injection des dépendances.

Ensuite on doit enlever toutes les annotations des EJBs et les remplacer avec les annotations de Spring.

Le tableau 4illustre quelque annotation qu'on a pu rencontrer dans notre projet :

L'annotation EJB	**L'équivalent Spring**	**Description**
@Singleton @Stateless	Le type Singleton c'est le scope par défaut des objets Spring, et Stateless n'existe pas.	Les beans des types Singleton et Stateless sont traité de la même façon dans Spring, il faut juste définir le type de Spring bean (@Component ...) et laisser le scope par défaut.
@Stateful	@Scope("Prototype")	Il faut définit le type de beans par exemple @Component et ajouter l'annotation @Scope("Prototype"), Prototype signifie que ce bean et de type stateful.
@MessageDriven	@JmsListener	Pour utiliser l'annotation

		@JmsListener il faut ajouter @EnableJms dans la classe de configuration et définir les beans nécessaire.
@Schedule("…")	@Scheduled(cron = "…")	Spring nous permet de créer un timer dont les caractéristiques sont fournies sous la forme d'attributs de l'annotation en utilisant @Scheduled, on ajoute "cron" pour utiliser des paramètres similaire à ceux qu'on trouve à @Schedule.
@LocalBean	N'existe pas	Le bean a une annotation @LocalBean c'est-à-dire le bean n'a pas d'interfacede vue donc c'est un bean qui n'a pas d'annotations de vue et n'implémente aucune interface alors on peut le créer de cette manière sans ajouter l'annotation @LocalBean.
@EJB	@Autowired	Utiliser dans l'injection des bean, spring connaitre le type de bean injecté en utilisant cette annotation.

Tableau 4 Le changement des annotations

Enfin on doit configurer l'EntityManager, et on peut faire cette configuration en utilisant deux méthodes, soit avec du code java en créent une classe de configuration contenant les

beans des différentes propriétés et cela avec l'annotation @Bean comme les figures 10, 11, 12 montrent.

- **EntityManagerFactory**

  Le premier bean qu'on a besoin de déclarer c'est l'EntityManagerFactory montré dans la figure 10. Le rôle principal d'un EntityManagerFactory est de prendre en charge l'instanciation d'EntityManager et fournit un moyen efficace de construire plusieurs instances del'EntityManager pour une base de données spécifique.

```java
@Configuration
@EnableTransactionManagement
public class PersistenceJPAConfig{

   @Bean
   public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
      LocalContainerEntityManagerFactoryBean em
        = new LocalContainerEntityManagerFactoryBean();
      em.setDataSource(dataSource());
      em.setPackagesToScan(new String[] { "com.baeldung.persistence.model" });

      JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
      em.setJpaVendorAdapter(vendorAdapter);
      em.setJpaProperties(additionalProperties());

      return em;
   }

   // ...

}
```

Figure 6 Configuration EntityManagerFactory [w12]

- **Source de données**

  Le deuxième bean à configurer c'est la source de données montré dans la figure 11. C'est la ou on ajoute les propriétés de la base de données

```java
@Bean
public DataSource dataSource(){
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
    dataSource.setUrl("jdbc:mysql://localhost:3306/spring_jpa");
    dataSource.setUsername( "tutorialuser" );
    dataSource.setPassword( "tutorialmy5ql" );
    return dataSource;
}
```

Figure 7 Configuration de la source des données [w12]

- **TransactionManager et exceptionTranslation beans**

  Enfin on va déclarer les deux dernier beans, le gestionnaire de transaction et la translation d'exception, montré dans la figure 12.

```java
@Bean
public PlatformTransactionManager transactionManager() {
    JpaTransactionManager transactionManager = new JpaTransactionManager();
    transactionManager.setEntityManagerFactory(entityManagerFactory().getObject());

    return transactionManager;
}

@Bean
public PersistenceExceptionTranslationPostProcessor exceptionTranslation(){
    return new PersistenceExceptionTranslationPostProcessor();
}

Properties additionalProperties() {
    Properties properties = new Properties();
    properties.setProperty("hibernate.hbm2ddl.auto", "create-drop");
    properties.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

    return properties;
}
```

Figure 8 Configuration de transaction et exception [w12]

Sinon on peut faire la configuration dans un fichier xml tel qu'il est montré dans la figure ci-dessous. On utilisant les balises nécéssaires, <bean> pour déclarer le bean, et la balise <property> pour ajouter les différentes propriétes de notre bean.

```xml
<bean id="myEmf"
  class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="packagesToScan" value="com.baeldung.persistence.model" />
    <property name="jpaVendorAdapter">
        <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter" />
    </property>
    <property name="jpaProperties">
        <props>
            <prop key="hibernate.hbm2ddl.auto">create-drop</prop>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</prop>
        </props>
    </property>
</bean>

<bean id="dataSource"
  class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/spring_jpa" />
    <property name="username" value="tutorialuser" />
    <property name="password" value="tutorialmy5ql" />
</bean>

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="myEmf" />
</bean>
<tx:annotation-driven />

<bean id="persistenceExceptionTranslationPostProcessor" class=
  "org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor" />
```

Figure 9 fichier de configuration xml [w12]

2- Le deuxième type c'est de prendre les interfaces déjà créer et on les notes avec l'annotation @Repository mais dans celle-ci on a plusieurs façon de faire, les méthodes qu'on a utilisé dans notre projet et qui sont les plus optimale.

- Notre interface doit hériter d'une interface générique tell que JpaRepository<T,ID> ou bien CrudRepository<T,ID>(T : type de l'entité, ID le id de l'entité)ou bien Repository<T, ID> tel qu'il est montré dans la figure 14.C'est interfaces contiennent des méthodes basic pour interagir avec la base de donnée.

```java

@Repository
public interface ProductRepository extends JpaRepository<Product, Integer> {

```

Figure 10 Interface Repository

Le tableau 5 présente quelque méthode basic de l'interface JPARepository.

Les méthodes	**Description**
count()	Renvoie le nombre d'entités existantes, le type de retour est long.
delete(T entité)	Supprime une entité donnée.

deleteAll()	Supprime toutes les entités gérées par la répertoire.
deleteAll(Iterable<? extends T> entités)	Supprime les entités données.
deleteById(ID id)	Supprime l'entité avec l'ID donné.
existsById(ID id)	Renvoie un booléen pour vérifier si une entité avec l'ID donné existe.
findAll()	Renvoie toutes les instances du type.
findAllById(Iterable<ID> ids)	Renvoie toutes les instances du type T avec les ID donnés.
findById(ID id)	Récupère une entité par son identifiant.
save(S entité)	Enregistre une entité donnée.
saveAll(Iterable<S> entités)	Enregistre toutes les entités données.

Tableau 5 Méthodes de JPARepository

- on peut en plus ajouter de nouvelle fonction plus spécifique grâce à un mécanisme de Spring, et cella ce fait en écrivant seulement les signatures des fonctions dans notre interface en respectent une syntaxe bien définie en utilisant des mots clés, Spring Data JPA va par la suite effectue une vérification des propriétés et parcourt les propriétés imbriquées après elle sera traduite en requête sql automatiquement. La figure 15 montre cette méthode.

```
//équivalant à ("select p from product p where p.price=?")
 public List<Product> findAllByPrice(Double price);
```

Figure 11 Méthode spécifique

Le tableau suivant illustre les mots clés nécessaires pour utiliser cette méthode :

Mot clés	**Description**	**Exemples**	**Requête équivalente**
find	Equivalant à select	findAll	"Select * from entité e"
By	Utilisé pour préciser les champs de la recherche	findAllByCode	"Select * from entité e where e.Code= ? "
exists	Utilisé pour vérifier l'existence, elle retourne un booléen	existsByCode	"SELECT CASE WHEN count(pl)> 0 THEN true ELSE false END FROM entité e where e.code=?"
And	Représente l'opérateur "et"	findByCodeAndLibelle	"Select e from entité e where e.Code= ? and e.libelle= ?"
Or	Représente l'opérateur "ou"	findByCodeOrLibelle	"Select e from entité e where e.Code=Code or e.libelle= ?"
Is,Equals	Représente l'operareur égale	findByCode, findByCodeIs, findByCodeEquals	"Select e from entité e where e.Code= ? "
Between	Pour spécifier un intervalle donné	findByDateBetween	"Select e from entité e where e.Date between ?and ?"
LessThan	L'opérateur inferieur à"<"	findByAgeLessThan	"Select e from entité e where e.age <?"
GreaterThan	L'opérateur superieur à">"	findByAgeGreaterThan	"Select e from entité e where e.age >?"

After	Opérateur pour rechercher des dates superieur a une date donnée	findByDateAfter	"Select e from entité e where e.Date >?"
Before	Opérateur pour rechercher des dates inferieur a une date donnée	findByDateBefore	"Select e from entité e where e.Date >?"
In, NotIn	Utilisé pour vérifier l'existence d'une valeur dans un ensemble	findByCodeIn(Collection <Code> codes) findByCodeNotIn(Collection<Code> codes)	"Select e from entité e where e.Code in ? " "Select e from entité e where e.Code not in ? "
OrderBy	Utilisé pour ordonner et trier la list des résultats	findByCodeOrderByLibelleDesc	"Select e from entité e where e.Code= ? order by e.libelle desc "

Tableau 6 Mots clés pour les requetes

- Pour utiliser notre propre requêtes SQL on prend les interface et on écrit les signatures des fonctions, et on annote ces fonctions avec @Query("requête Sql") et a l'intérieur de l'annotation on écrit notre requête et le nom des variables doit être le même dans la requête et la fonction comme la figure 16 montre.

```
@Query("select p from product p where p.name=name and p.price=price ")
public List<Product> RechProductNamePrice(String name,Double price);
```

Figure 12 L'utilisation de @Query

- Il existe aussi une autre façon, pour celle-ci c'est un peu différant on créer une autre interface avec les fonctions qu'on veut et cette interface sera implémenté dans une classe pour pouvoir redéfinir ces fonctions après on revient à notre interface Repository et au lieu d'hériter seulement une interface générique, on va cette fois-ci hériter aussi de l'interface qu'on aura créé du coup elle aura un double héritage.

- Ensuite pour la couche Service on commence par créer des interfaces qui vont contenir les signatures des fonctions qui seront par la suite implémenté dans une classe qui va implémenter cette interface, après cella ont créé nos classes qui seront annotées

avec l'annotation @Service qui vont implémenter les méthodes des interfaces, et les interfaces de la partie DAO seront injectées dans ces classes et c'est grâce à ces injections qu'on pourra effectuer des modifications sur la base de donnée.

La figure suivante représente un exemple d'une classe service :

```
import java.util.Date;
@Service
@Transactional
public class BanqueServiceImp implements IBanqueSerivce {
    @Autowired
    private CompteRepository compteRepository;
    @Autowired
    private OperationRepository operationRepository;
    @Override
    public Compte consutlerCompte(String codeCpte) {
    Compte cp= compteRepository.findById(codeCpte).orElse(null);
    if (cp==null) throw new RuntimeException("Compte introuvable");
        return cp;
    }

    @Override

    public void verser(String codeCpte, double montant) {
        Compte cp=consutlerCompte(codeCpte);
        Versement v= new Versement(new Date(),montant, cp);
        operationRepository.save(v);
        cp.setSold(cp.getSold()+montant);
        compteRepository.save(cp);


    }
```

Figure 13 Exemple d'une classe service

- Ensuite on configure notre fichier Application. properties qui est stocké dans ''src/main/resources'' qui est utilisé pour conserver plusieurs propriétés dans un seul fichier afin d'exécuter l'application dans des environnements différents par exemple c'est dans ce fichier qu'on choisit le port à utiliser ou bien configurer la base de donnée on peut aussi choisir ou les vues sont stocké et leur préfix ou bien suffixe (*.jsf,*.html,*.xhtml,…etc.).

  La figure 18 présente un exemple de configuration d'un fichier Application.properties.

```
# ================================
# = DATA SOURCE
# ================================
# Set here configurations for the database connection
spring.datasource.url=jdbc:postgresql://localhost:5432/zaki
spring.datasource.username=postgres
spring.datasource.password=postgres
spring.datasource.driver-class-name=org.postgresql.Driver
# Keep the connection alive if idle for a long time (needed in production)
spring.datasource.testWhileIdle=true
spring.datasource.validationQuery=SELECT 1
# ================================
# = JPA / HIBERNATE
# ================================
# Show or not log for each sql query
spring.jpa.show-sql=true
# Hibernate ddl auto (create, create-drop, update): with "create-drop" the database
# schema will be automatically created afresh for every start of application
spring.jpa.hibernate.ddl-auto=update
# Naming strategy
spring.jpa.hibernate.naming.implicit-strategy=org.hibernate.boot.model.naming.ImplicitNamingStrategyLegacyHbmImpl
spring.jpa.hibernate.naming.physical-strategy=org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy

# Allows Hibernate to generate SQL optimized for a particular DBMS
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Figure 14 Le fichier Application.properties

Le coté front-end

Dans cette partie on va parler sur le front-end, c'est-à-dire la couche vue. On a essayé plusieurs méthodes pour le frontend, on a le choix soit de faire la migration aussi coté front ou bien la laisser comme elle est.

- La première méthode C'est de le laisser comme il est car dans l'entreprise Elit pour gérer le front il utilise JSF et Primerfaces (JSF est une spécification Java qui favorise le développement d'interfaces utilisateur basées sur des composants pour les applications Web), (Primefaces est un framework d'interface utilisateur open source pour JSF qui comprend plus d'une centaine de composants).

  Mais pour utiliser JSF avec Spring on doit ajouter quelques dépendances au fichier pom.xml pour être utilisable :

```
<dependency>
        <groupId>org.apache.myfaces.core</groupId>
        <artifactId>myfaces-impl</artifactId>
        <version>2.2.12</version>
</dependency>

<dependency>
        <groupId>org.apache.myfaces.core</groupId>
        <artifactId>myfaces-api</artifactId>
        <version>2.2.12</version>
</dependency>
```

```
<dependency>
        <groupId>org.apache.tomcat.embed</groupId>
        <artifactId>tomcat-embed-jasper</artifactId>
</dependency>

<dependency>
        <groupId>org.primefaces</groupId>
        <artifactId>primefaces</artifactId>
        <version>6.1</version>
</dependency>
```

Les deux premières dépendances, myfaces-api et myfaces-impl, sont la spécification de l'interface JSF (-api) et l'implémentation (-impl). La troisième dépendance, tomcat-embed-jasper, est nécessaire pour que la JVM puisse analyser et exécuter la vue JSF à l'exécution. Et la dernière c'est pour utiliser PrimeFaces.

Pour utiliser JSF on doit créer un fichier Web.xml Habituellement, sur une application Spring Boot standard, nous n'avons pas besoin de ce fichier. Mais, puisque nous allons utiliser JSF, nous devons configurer le servlet FacesServlet et quelques écouteurs. La configuration est montrée dans la figure ci-dessous.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="3.1">
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
    </servlet-mapping>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
    </listener>
    <listener>
        <listener-class>org.springframework.web.context.request.RequestContextListener</
    </listener>
</web-app>
```

Figure 15 Configuration de fichier web.xml

Les deux premiers éléments de ce fichier sont responsables de la mise en place de FacesServlet et de sa configuration.

Servlet-mapping indique à ce servlet de traiter les requêtes adressées aux URL *.jsf et de les traiter dans le contexte de JSF.

Les deux derniers éléments, les éléments listener, sont responsables de l'intégration de JSF dans le contexte Spring.

Après on doit créer un autre fichier xml du nom faces-config.xml Tout ce que ce fichier fait est d'enregistrer un ELResolver (Expression Language resolver) qui délègue au contexte WebApplicationContext de Spring la responsabilité de résoudre les références de nom. Avec lui, nous pouvons utiliser des beans gérés par Spring dans le contexte JSF. La figure 20 montre la configuration du fichier faces-config.xml.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
              version="2.2">
    <application>
        <el-resolver>org.springframework.web.jsf.el.SpringBeanFacesELResolver</el-resolver>
    </application>
</faces-config>
```

Figure 16 Le fichier faces-config.xml

Après nous devons modifier notre MainApplication en ajoutant 2 beans et en la faisant hérité de la class SpringBootServletInitializer comme la figure suivante montre Après ça nous serons capables d'utiliser JSF avec PrimeFaces tel qu'il est montré dans la figure 21.

```java
public class Application extends SpringBootServletInitializer {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Bean
    public ServletRegistrationBean servletRegistrationBean() {
        FacesServlet servlet = new FacesServlet();
        return new ServletRegistrationBean(servlet, "*.jsf");
    }

    @Bean
    public FilterRegistrationBean rewriteFilter() {
        FilterRegistrationBean rwFilter = new FilterRegistrationBean(new RewriteFilter());
        rwFilter.setDispatcherTypes(EnumSet.of(DispatcherType.FORWARD, DispatcherType.REQUEST,
                DispatcherType.ASYNC, DispatcherType.ERROR));
        rwFilter.addUrlPatterns("/*");
        return rwFilter;
    }
}
```

Figure 17 Configuration JSF de la classe main

Enfin on utilise les contrôleurs déjà crée dans notre projet JEE pour ce faire @ManagedBean le managedBean de JSF (c'est-à-dire annotés avec javax.faces.bean.ManagedBean) ne fonctionnaient pas correctement. Son injection et son

instanciation de dépendances ne se sont pas produites, elles restent donc avec une valeur nulle. Cependant, si vous utilisez javax.annotation.ManagedBean à la place, le conteneur Spring peut effectuer correctement l'injection des dépendances. De cette façon, il est possible de continuer à utiliser une annotation JEE, @ManagedBean, au lieu de mettre une annotation Spring (@Controller ou @Component). L'avantage d'utiliser une annotation de spécification est de la maintenir indépendante et moins couplée à tout cadre spécifique utilisé pour le moment, et les annotations tell que @EJB qui sont propres a JEE doivent être enlevé et remplacer avec @Autowired, notre contrôleur va contenir des méthodes qui vont analyser les requetés et exécuter les actions à faire en utilisant la couche métier.

- La deuxième méthode consiste à utiliser Thymeleaf. Il s'agit d'un moteur de template XML / XHTML / HTML5 il est très simple à utiliser avec Spring.

- Tout d'abord il faut ajouter ces dépendances dans le fichier pom.xml

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

- Et pour utiliser les layouts on doit ajouter une autre dépendance

```
<dependency>
<groupId>nz.net.ultraq.thymeleaf</groupId>
<artifactId>thymeleaf-layout-dialect</artifactId>
</dependency>
```

- Dans Thymeleaf tous les fichiers front-end doivent être stocké dans le fichier resources les template dans un dossier template et les fichier static dans un fichier static tell que Css et JavaScript.

  La figure 22 montre le chemin des dossiers et des ressources utilisé dans la vue.

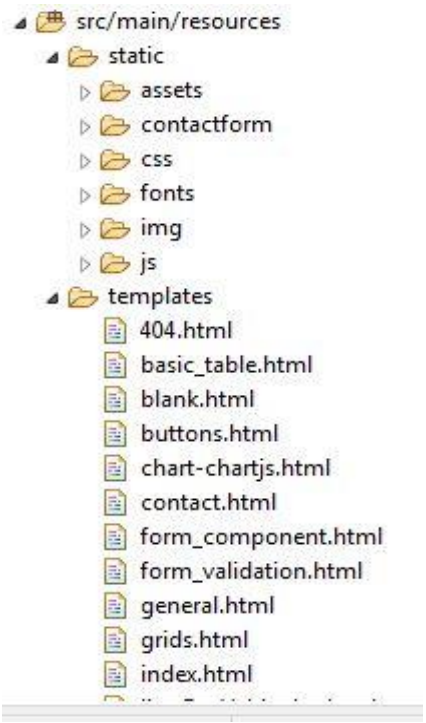


Figure 18 Dossier des ressources

- Et si on veut modifier leur emplacement on a juste besoin de la cité sur le fichier Application.properties on utilisant seulement les deux paramètre montré dans la figure 23.

```
spring.mvc.view.prefix: /WEB-INF/jsp/
spring.mvc.view.suffix: .jsp
```

Figure 19 propriété du chemin des ressources

- Après on doit apporter les modifications nécessaires à nos fichiers html ou xhtml pour pouvoir les utiliser on doit commencer par remplacer les entêtes des fichiers.
  La figure 24 montre l'entête dans le cas d'utilisation de thymeleaf.

```
<html
    xmlns="http://www.thymeleaf.org"
    xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
    layout:decorate="~{template1}">
```

Figure 20 Entête Thymeleaf

La figure 24 montre l'entête dans le cas d'utilisation de JSF et PrimeFaces.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:c="http://java.sun.com/jsp/jstl/core"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:a="http://elit.dz/ui"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">
```

Figure 21 Entête JSF/PrimeFaces

- Les annotations des variables et des liens de Primefaces sont différentes de celle de Thymeleaf,

  Sur PrimeFaces les variables sont noté avec un "#", exemple : ="#{bundleCommun.lbl_date_debut}"

  Alors que sur Thymeleaf sont noté avec un "$", exemple ="${bundleCommun.lbl_date_debut}"
- Enfin on modifie tout ce que doit être modifié (la majorité des balises doivent être changé)

Mais dans notre cas comme L'entreprise Elite utilise JSF+PrimeFaces ainsi le projet GPARC est un projet immense pour faire la migration des vues, donc la deuxième méthode est déconseillée.

6. Evaluation

Dans cette partie on va évaluer cette expérience de migration avec des points positifs et négatifs qu'on a remarqués dans notre recherche.

3.1 Les points positifs

Le développement d'une application Spring Boot depuis le début vers une version en cours d'exécution est très rapide, principalement en raison des configurations automatiques et des archétypes déjà existants.

Son utilisation est simple, de plus Spring a déjà été un framework assez connu et aussi très utilisé dans la communauté Java, donc il y a plutôt du contenu pour référence.

3.2 Les points négatifs

Dans certains cas, utiliser Spring API au lieu de Java EE équivaut a presque la même chose.

Il existe également des cas où la configuration de Spring Boot est complètement différente de celle de Java EE. C'est un point négatif dans les endroits où Java EE est déjà mature et bien connu.

Il y a aussi des problèmes d'intégration, il n'était pas possible d'utiliser CDI(Context Dependency injection) combiné avec Spring duo pour les conflits entre eux. En fait, cela est prévisible, car CDI et Spring Core travaillent tous deux sur le traitement des annotations (injection).

Un autre point qui mérite d'être souligné est l'utilisation de JSF dans cette architecture. Il est possible d'utiliser complètement ces technologies, mais il y a cette limitation dont Spring ne peut pas gérer et injecter javax.faces.bean.ManagedBean.

4. Conclusion

Spring a commencé à devenir très puissant lorsque la plateforme Java EE était en version 5, dans laquelle de nombreuses configurations étaient très complexes et de nombreuses fonctionnalités n'étaient pas disponibles ou matures. Dans ce contexte, Spring était un complément à Java EE avec des configurations simplifiées. Dans ce scénario, c'était mieux que Java EE. Cependant Java EE dans les versions 6 et 7, a beaucoup évolué. Ainsi, une grande partie de sa configuration est déjà simple. De la même manière, Spring a continué d'évoluer et de s'améliorer. Dans ce contexte, le cœur de Spring est une alternative à Java EE (plus un complément).

Actuellement, dans la plupart des cas, il n'y a aucune raison d'utiliser Spring Core sur Java EE (sauf par préférence).

D'un autre côté, il y a encore de nombreuses parties de Spring que Java EE n'a pas encore couvertes. Même avec ces alternatives, Spring Boot a certainement son espace. Principalement pour les nouvelles applications (où vous n'avez pas besoin de l'adapter à partir de Java EE), ou celles axées sur le développement rapide. Ou même pour les équipes n'ayant pas beaucoup de connaissances sur la pile Java EE, car elle nécessite une connaissance plus approfondie des configurations pour la faire fonctionner. À l'avance, Spring propose des mises à jour plus rapides que Java EE.

Chapitre 5

Organisme d'accueil et étude de l'existant

Dans ce chapitre on va présenter l'organisme d'accueil SONELGAZ et l'entreprise ELIT, ensuite on va faire une étude sur la solution informatique existante.

1. Introduction

Notre projet est proposé par ELIT, filiale du groupe SONELGAZ. Dans ce chapitre nous allons faire une présentation globale de l'organisme d'accueil SONELGAZ, la structure d'accueil ELIT ou nous avons effectué notre stage de projet fin d'étude, tel que le contexte général ainsi que les principaux objectifs de notre travail.

2. Organisme d'accueil SONELGAZ

2.1 Présentation

SONELGAZ, acronyme de Société Nationale de l'Electricité et du Gaz. C'est l'opérateur historique dans le domaine de la fourniture des énergies électrique et gazière en Algérie.Ses missions principales sont la production, le transport et la distribution de l'électricité ainsi que le transport et la distribution du gaz par canalisations. Son nouveau statut de « Groupe » acquit depuis quelques années a permis la création de nombreuses filiales et structures fonctionnelles, qui se distinguent par leurs différents métiers et activités professionnelles

Aujourd'hui, le Groupe SONELGAZ est composé de 16 sociétés directement pilotées par la Holding, de 18 sociétés en participation avec des entités du Groupe et de 10 sociétés en participation avec des tiers. Elle a le monopole de l'énergie en Algérie Depuis 1969 En Remplaçant L'entité précédente Electricité et Gaz en Algérie (EGA) [w1].

2.2 Diagramme circulaire du groupe SONELGAZ

Les filiales métiers de base de SONELGAZ assurent la production, le transport et la distribution de l'électricité, ainsi que le transport et la distribution du gaz par canalisations. Ses filiales travaux sont en charge de la réalisation des infrastructures électriques et gazières du pays. Ses filiales de prestations de service activent principalement dans les domaines de la fabrication et de la maintenance d'équipements énergétiques, la distribution de matériel électrique et gazier, le transport et la manutention exceptionnels [w1].

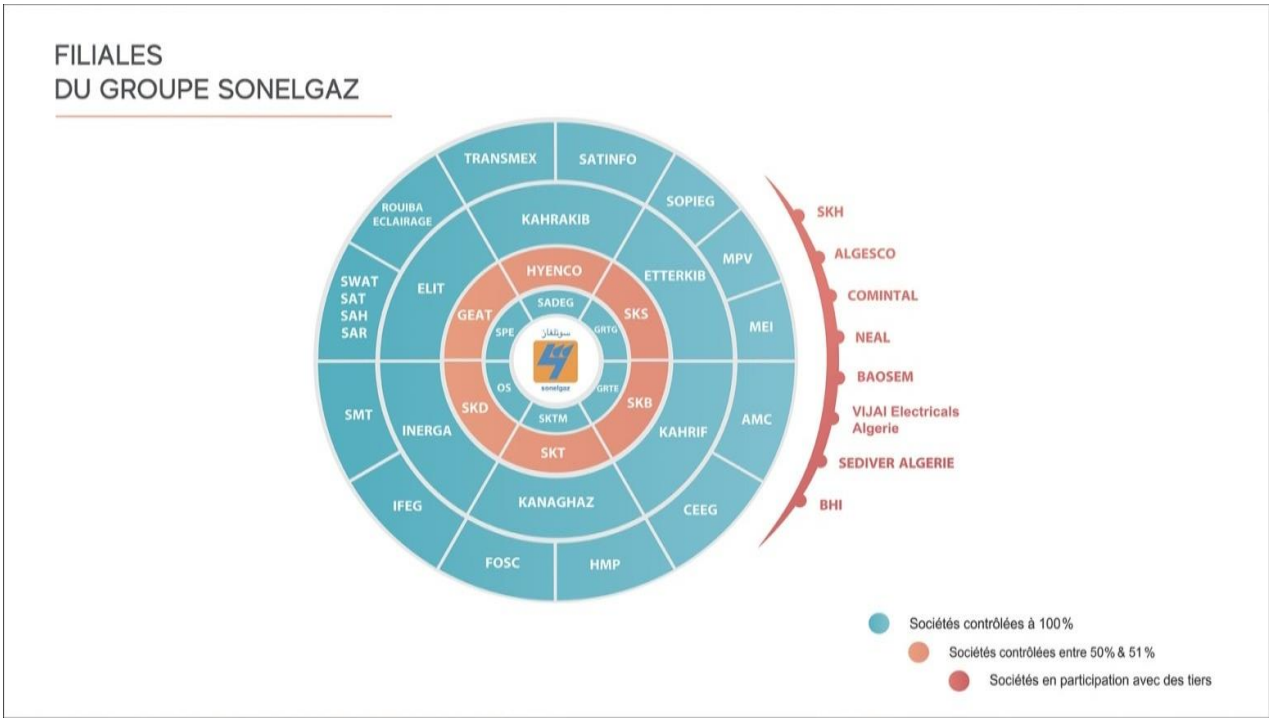


Figure 22 Digramme circulaire du groupe SONELGAZ [w1]

Cette figure représente le diagramme circulaire des différentes filiales du groupe SONELGAZ.

3. Structure d'accueil « ELIT»

4.1 Présentation

Le 1er janvier 2009, l'activité système d'information, confiée à une direction générale au niveau de la maison mère du groupe SONELGAZ, a été érigée en société par actions, dénommée « EL-Djazair Information Technology », par abréviation "ELIT Spa" au sein de laquelle notre projet sera réalisé. ELIT est une société algérienne spécialisée dans les technologies de l'information et de la communication, comptant plus de 300 ingénieurs informaticiens, plus de 40 clients et des infrastructures hautement disponibles et sécurisées.

ELIT, filiale du Groupe SONELGAZ est chargée de mettre en place un système d'information global pour l'ensemble des sociétés du Groupe, en premier lieu, et pour le marché national, en second lieu.

Avec sa filière IT en pleine expansion, des projets d'envergure et une stratégie ambitieuse, ELIT, dans ses politiques Ressources Humaines, doit, de façon permanente, tout

mettre en œuvre pour anticiper ses besoins en menant une véritable gestion prévisionnelle des compétences et anticiper les recrutements à venir.

L'un des gisements privilégié, permettant à ELIT de combler ses besoins prévisionnels, en nouvelles compétences, est constitué des grandes écoles et autres institutions universitaires. Le potentiel de compétences est donc composé de futurs diplômés, dont le parcours universitaire est couronné par la réalisation d'un projet de fin d'études.

Afin de bâtir de vraies passerelles entre l'université et le monde de l'entreprise, ELIT s'est résolument tournée vers des choix de thèmes de stage de fin d'études, inspirés de problématiques métiers avérées, tirés du portefeuille de projets du schéma directeur informatique du Groupe SONELGAZ et déclinés dans le plan de charges de ses différentes structures métiers. Cette approche fondée sur le savoir faire d'ELIT dans le domaine IT, permettra à coup sûr aux stagiaires de vivre de véritables expériences liées à ses métiers.

Intégrés au sein d'équipes projets pluridisciplinaires, les stagiaires prennent part à la vie d'un vrai projet. En y jouant leur rôle, ils peuvent combler leur parcours académique par une phase de formation-action, laissant plus de place à la pratique et à un travail mené de façon plus collaborative dans les domaines de la conduite du projet, des études, de la conception et le développement. Ceci tout en s'appuyant sur les bonnes pratiques et l'utilisation des technologies les plus avancées, et ce, dans le seul but de préparer ces jeunes talents à intégrer facilement le monde du travail.

ELIT a été créée pour répondre à [w2] :

- La stratégie du groupe SONELGAZ de développer des moyens propres de maitrise d'œuvre dans le domaine des systèmes d'information, et de disposer d'un pôle de compétences technologiques au service de ses sociétés.
- La volonté du groupe SONELGAZ de confier la propriété des systèmes d'information à une entité spécialisée et de focaliser les capacités de ses sociétés sur leurs métiers de base respective.

4.2 Missions et taches

Les missions assignées à ELIT se déclinent en deux grandes parties, stratégique et opérationnelle. Pour la partie stratégique, ELIT contribue à la stratégie du groupe SONELGAZ par :

- L'élaboration de la politique des systèmes d'information et des technologies de l'information et de la communication du groupe SONELGAZ.
- La prise en charge des besoins des sociétés du groupe SONELGAZ en matière d'informatique et de télécommunication.

Pour la partie opérationnelle, ELIT s'emploie à :

- Elaborer et mettre en œuvre les systèmes d'information destinés au pilotage et à la gestion des différentes activités des sociétés du groupe SONELGAZ.
- Mettre à la disposition des sociétés du groupe SONELGAZ les moyens informatiques et des télécommunications (logiciels, matériels, infrastructures . . . etc.) nécessaires pour assurer le niveau de service attendu.
- Assurer La Maintenance Et l'administration des systèmes d'information, des plateformes et des équipements mis à la disposition des utilisateurs.
- Assurer l'accès à l'information et à l'application et en garantir la sécurité, l'intégrité et la fiabilité.
- Mettre à la disposition des utilisateurs l'expertise technique indispensable à la satisfaction de leurs besoins.
- Proposer, à terme, tous les services construits pour les sociétés du groupe aux clients externes.

Les services fournis par ELIT portent sur :

- Le développement d'applicatifs métiers.
- Les outils de travail collaboratif.
- L'infogérance.
- Les réseaux et télécoms.
- La formation.
- L'assurance et le conseil.

4.3 L'organigramme d'ELIT

La figure 2 représente l'organigramme de l'entreprise ELIT filale du groupe SONELGAZ.

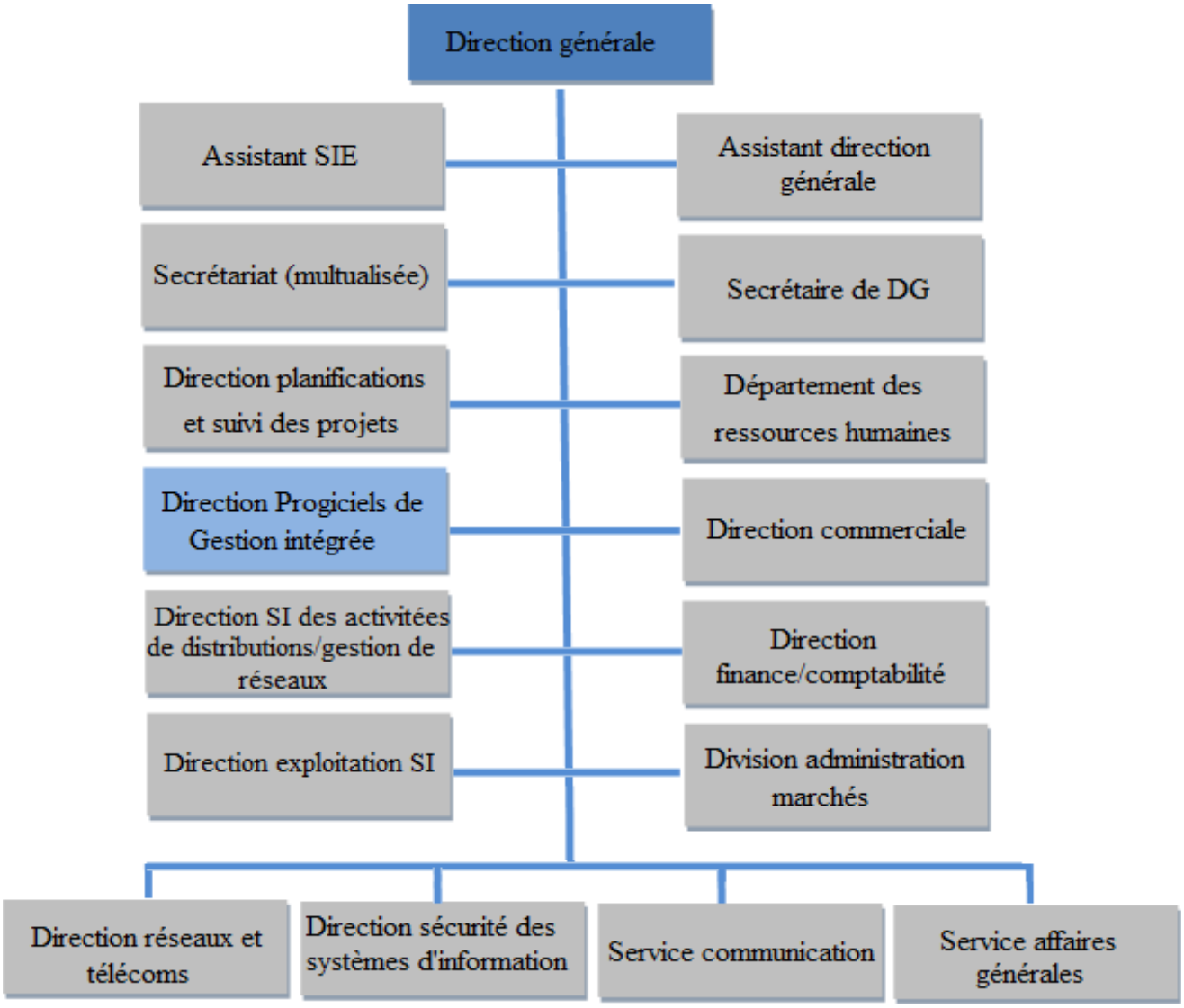


Figure 23 Organigramme de la filiale du groupe SONELGAZ ELIT. [s2]

3.3.1 Direction Progiciels de Gestion Intégrée

1- Mission

La direction Progiciels de Gestion Intégrée, à laquelle appartient la structure d'accueil ELIT, est chargée de la production, du déploiement et de la maintenance des systèmes d'information, capable de gérer l'ensemble des processus d'une organisation, en incluant de façon intégrée. La gestion comptable et financière, la gestion des ressources humaines, de la production, de la maintenance, de la logistique, des ventes, des achats, etc.

2- Attributions

- Etudier les besoins en systèmes d'information de gestion et mettre en œuvre les solutions adaptées pour l'ensemble des sociétés du Groupe SONELGAZ.

- Organiser et planifier la réalisation des projets, depuis leur conceptions jusqu'à leur achèvement, (Conception, développement, tests, intégration, migration de données, etc.), en en s'appuyant sur des compétences internes ou externes.
- Assurer la maintenance corrective et évolutive des SI développés.
- Préparer la société à placer, à termes, ses produits sur le marché.
- Assurer la veille technologique.

5. Etude de l'existant

Le parc de SONELGAZ est estimé à environ 8000 véhicules dans la catégorie légère uniquement, ce qui rend difficile la gestion des parcs, et les dépenses liées sont de plus en plus importantes. En vue de ce besoin, ELIT s'est engagée à développer un système de gestion des parcs automobiles " GPARC" pour le profit du groupe. GPARC est une application web développée en utilisant la technologie Java EE full web, avec le Patron de conception (Design Pattern) MVC. GPARC est basé sur l'API " EJB" pour le coté back-end et le framework JSF pour le coté front-end.

Le système GPARC offre plusieurs fonctionnalités :

• Gestion des véhicules (acquisition, affectation et disponibilité).

• Suivi des sinistres, engagements et maintenance.

• Gestion des missions.

• Gestion du carburant.

• Gestion des opérateurs (chauffeurs et autres intervenants).

• Reporting.

Le système GPARC comporte plusieurs modules, parmi ceux on cite cinq principaux modules :

Module Véhicules et Engins : Du quel se charge l'Administrateur et le chef de projet, sert à gérer les véhicules et engins de la flotte, leurs acquisitions, leurs caractéristiques.

Module Opérateur : Duquel se charge le chef de projet, sert à gérer les différentes informations sur les intervenants et opérateurs de la flotte.

Module Carburant : Duquel se charge le gestionnaire de carburant, sert à gérer la consommation de carburant par intervenant, ainsi que les mouvements d'entrées et sorties des bons et cartes carburants.

Module Mouvement du Parc : Du quel se charge le superviseur et le chef de projet, sert à gérer les missions, leurs demandes et leurs planifications.

Module Reporting: sert à fournir les rapports nécessaires à la gestion d'un parc automobile.

Ainsi que d'autres modules complémentaires :

Module Administration: Du quel se charge l'administrateur, sert à gérer les utilisateurs, leurs profiles et privilèges, ainsi que la structure organisationnelle de l'entreprise.

Module Données de base: sert à gérer les différentes données statiques non techniques utilisées pour accomplir les tâches effectuées par le système comme(affaire, client, fournisseur, . . .).

Module Maintenance: Duquel se charge le gestionnaire de maintenance, sert à gérer la maintenance préventive ainsi que la gestion des sinistres.

La figure 3 représente ces modules :

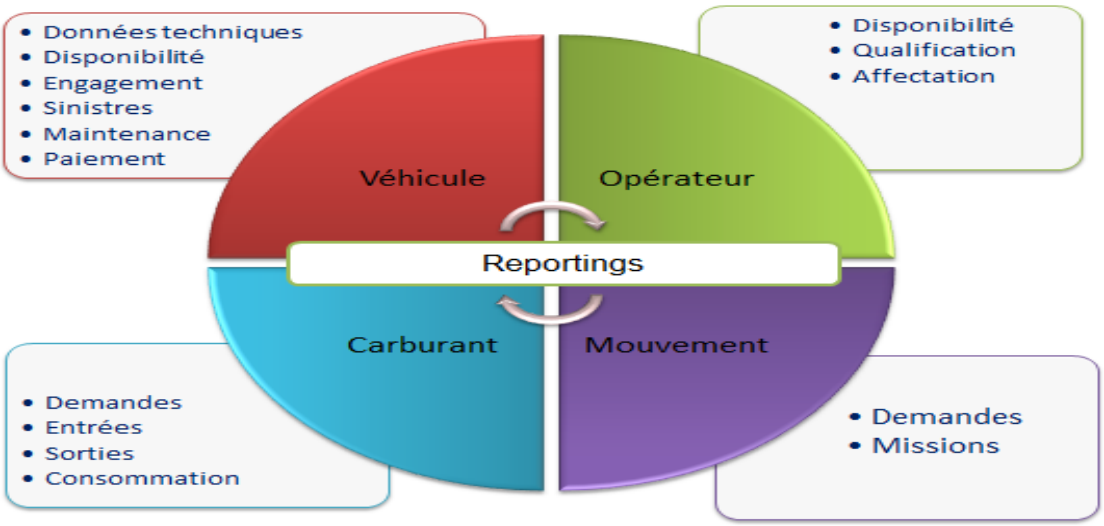


Figure 24 Les modules de GPARC

Pour mieux expliquer le système GPARC actuel, on présente le schéma général dans la figure suivante :

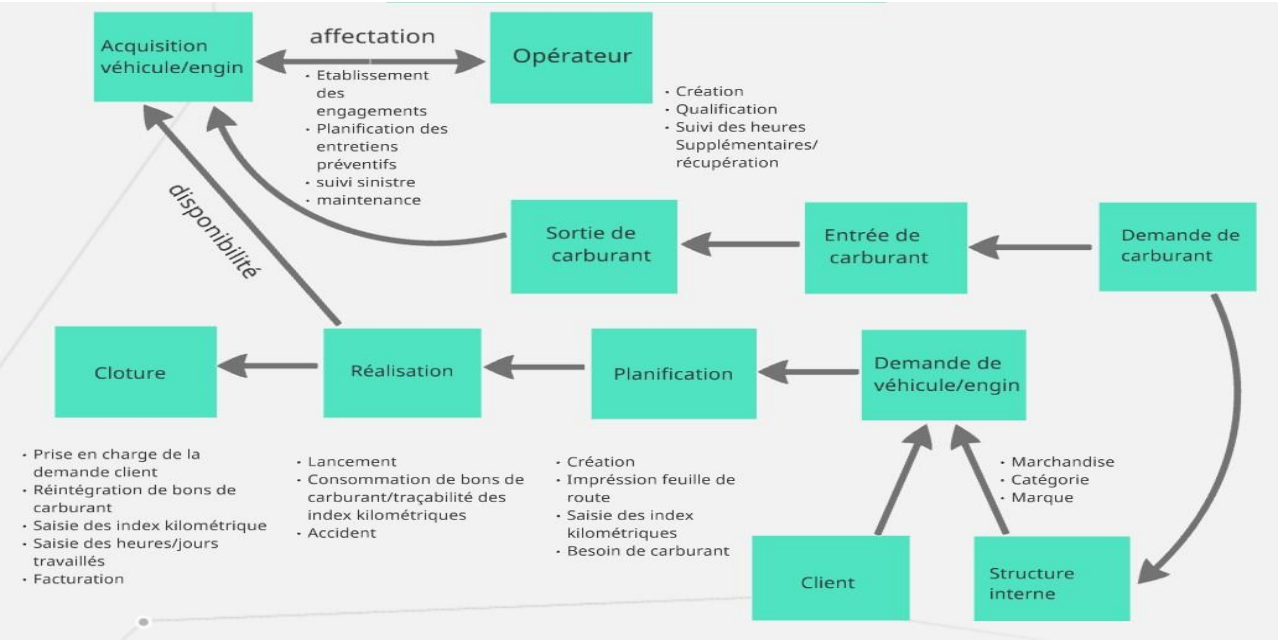


Figure 25 Le système existant GPARC

Le processus commence une fois le véhicule est acquis, pour lequel on établit des engagements : un contrat d'achat, un contrôle technique, et une assurance etc.., migration des contrats d'achat dans le cas d'un ancien véhicule. On planifie aussi les entretiens préventifs (tous ce qui est vidange, rénovation moteur etc..). Ensuite, le véhicule est affecté à un opérateur (qui sera créé dans le cas d'un nouveau conducteur) selon les qualifications de ce dernier. Le client qui est soit un client interne ou externe lance une demande de véhicule, exige la catégorie, la marque de ce dernier, spécifie la marchandise (dimensions, largeur, poids...), les itinéraires (début, fin et arrêts). Aussi, les dates début et fin de la mission. Si le client est interne les charges de la mission seront imputées sur les structures concernées, sinon les données seront envoyées vers le système facturation. Une fois la demande de véhicule est reçue, la mission est planifiée, les documents (feuilles de route, ordre de mission, autorisation de sortie de véhicule, autorisation de matériel et plan de contrôle etc..) sont crées et imprimées. Ainsi, la saisie des index kilométriques et le besoin en carburant. La sortie du carburant dépend du besoin en carburant qui dépend à son tour des itinéraires de la mission ainsi que les dimensions des colis. L'étape réalisation ne peut être lancée qu'à la disponibilité du véhicule. Une fois planifiée, la mission peut être lancée et les index kilométriques sont tracés d'où la consommation de bons de carburant est commencée. En cas d'accident, un suivi sinistre est déclaré et le véhicule est entré en maintenance.Une fois la mission est terminée,

elle est clôturée, et du coup: la prise en charge de la demande client, la réintégration de bons de carburant, la saisie des index kilométriques, la saisie des heures/jours travaillés et finalement la facturation. Une entrée de carburant est effectuée suite à une demande de carburant exprimée par le gestionnaire si le stock de sécurité est atteint.

6. Critique de l'existant

Le système GPARC est développé avec la technologie JEE en utilisant les technologies spécifiques des EJBs intensivement. Le serveur GlassFish est utilisé comme un serveur d'application.

Les conteneurs lourds comme les EJBs Nécessitent d'avoir des serveurs web et d'application pour déployer une solution, donc un grand projet tel que GPARC qui utilise les EJB nécessite une grande infrastructure, pour cela il faut faire une migration vers un conteneur léger tel que Spring Framework.L'entreprise a une forte dépendance d'EJBs et de Java EE, Spring n'est pas encore adopté.

Donc notre travail c'est d'étudier la faisabilité de cette migration afin de tracer une feuille de route l'adoption de la technologie Spring.

Pour le prototype, on s'intéresse au module véhicule et engins pour l'application de la feuille de route.

7. Conclusion

Dans ce chapitre introductif, nous avons présenté l'organisme d'accueil, et on a étudié la solution existante. Cette phase d'étude nous a permis de nous familiariser avec le sujet et de bien cadrer et cibler nos objectifs et adopter une méthode de travail. Dans le chapitre suivant on va présenter la technologie Java EE.

Modélisation et Conception

Dans ce chapitre on va faire la modélisation et la conception de notre projet on utilisant les différents diagrammes UML.

1. Introduction

Avant de commencer à coder la partie applicative, nous nous intéressons à la phase de spécification pour bien définir, clarifier les grandes fonctionnalités de notre application. Ce chapitre consiste à donner une définition précise des besoins fonctionnels et non fonctionnels ainsi que les objectifs visés.

L'approche utilisée dans la conception de notre travail est l'approche objet. Elle consiste à obtenir un modèle informatique d'une collection d'éléments d'une partie du monde réel en un ensemble d'entités appelées objets. Cette approche présente plusieurs avantages dont les principaux sont : la modularité, l'extensibilité et la réutilisation. Nous allons choisir pour notre application, la modélisation basée sur le langage UML et le processus de développement UP (Unified Process).

2. Définition d'UML(UnifiedModelingLanguage)

UML (Unified Modeling Language ou langage unifié de modélisation) est un langage graphique destiné à la modélisation des systèmes et de processus. C'est un langage basé sur l'approche par objets, celle qui a d'abord conduit à la création des langages de programmation comme Java, C++ ou Smalltalk. UML est unifié et permet de décrire un système à l'aide des diagrammes [14]. Ces derniers permettant d'exprimer une représentation simplifiée du problème. Chaque type de diagramme offre une vue d'un système. La combinaison des différents types du diagramme offre une vue complète du système [15].

2.1 Les différents diagrammes d'UML

L'UML offre beaucoup de diagrammes qui servent à la modélisation des systèmes, nousallons présenter la définition de quelques diagrammes :

- Diagrammes de cas d'utilisation (statique) : servent à représenter les fonctionnalités
- offertes au système ainsi que les acteurs et les relations existantes entre eux.
- Diagrammes de séquence (dynamique) : servent à montrer les interactions entre objets
- du point de vue temporel.
- Diagrammes de classes (statique) : servent à exprimer, d'une manière générale, la

- structure statique du système.
- Diagrammes de package : montrent l'organisation logique du modèle et les relations
- entre les packages.
- Diagrammes d'activité (dynamique) : servent à montrer l'écoulement des actions
- pendant un processus dans le système.
- Diagrammes de déploiement (statique) : servent à représenter l'environnement d'implémentation du système [13].

2.2 Pourquoi la méthode UML ?

L'UML est un langage formel et normalisé qui permet durant la phase de conception [1] :

- ✓ Un gain de précision
- ✓ Un gain de stabilité
- ✓ Encourage l'utilisation d'outils

Le langage UML est un support de communication performant [1] :

- ✓ Il encadre l'analyse.
- ✓ Il facilite la compréhension de représentation abstraite complexe.
- ✓ Son caractère polyvalent et sa souplesse en font un langage universel

2.3 Processus Unifié

L'UML propose des diagrammes pour décrire les différents aspects d'applicationmais ne précise pas la séquence d'étape à suivre ou la démarche à suivre pour laréalisation de ces diagrammes.

Un processus unifié est un processus de développement logiciel construit sur la notation UML. Il est itératif et incrémental, centré sur l'architecture, conduit par lescas d'utilisation et piloté par les risques.La gestion d'un tel processus est organisée par quatre phases : pré étude, élaboration,construction et transition. Ses activités de développement sont la capture des besoins,l'analyse et la conception, l'implémentation, le test et le déploiement

3. Spécification des besoins du système

L'analyse de la thématique et des différentes problématiques posées par les outils existants ainsi que la compréhension des besoins utilisateurs a permis de dégager les fonctionnalités qu'offre notre application finale. Les contraintes auxquelles est soumis le système pour sa réalisation et son bon fonctionnement seront décrites par la suite comme étant besoins non fonctionnels

3.1 Identification des acteurs

Un acteur représente l'abstraction d'un rôle joué par des entités externes (utilisateur, dispositifs matériels ou autres systèmes) qui interagissent directement sur le système étudié.

Administrateur : c'est la personne qui gère le système GPARC.

Superviseur : la consultation du tableau de bord et la gestion de la disponibilité des véhicules.

Gestionnaire de la maintenance : c'est la personne qui gère la maintenance.

Chef de projet : c'est la personne chargée de la consultation du tableau de bord et le suivi des heures travaillées.

Gestionnaire de carburant : c'est la personne qui gère le module carburant.

3.2Expression des besoins

L'objectif fondamental de cette étape est de déterminer les acteurs et les fonctionnalités de l'application à développer.

Nous nous intéressons donc, dans cette partie, à la réalisation de diagramme de cas d'utilisation qui nous montre les fonctionnalités de notre projet. Les principaux cas d'utilisation seront accompagnés d'une description textuelle ainsi qu'un diagramme de séquence système.

3.2.1Les besoins fonctionnels

Cette phase représente un point de vue «fonctionnel» de l'architecture système par le biais des cas d'utilisation. Un cas d'utilisation est une entité courante représentant une

fonctionnalité visible de l'extérieur. Il permet de mettre en évidence les relations fonctionnelles entre les acteurs et le système étudié.

Notre Système offre les fonctionnalités suivantes :

- La gestion des véhicules et des engins. l'ajout, la modification et la suppression.
- La gestion des marques, des modèles et des catégories des véhicules.
- La gestion des relevés des compteurs.
- Gérer la disponibilité des véhicules.
- Suivi des sinistres.
- Suivi de la maintenance.
- La gestion de carburant.
- Gérer les utilisateurs.

Suivi des heures travaillées.

3.2.2. Les besoins non fonctionnels

Les besoins non fonctionnels décrivent toutes les contraintes auxquelles est soumis le système pour sa réalisation et son bon fonctionnement.

1. Existence d'un réseau Wifi (Réseau sans fil).

2. Existence d'un serveur de Données avec le SGBD PostgreSQL.

3. Ergonomie et souplesse : l'application doit offrir une interface conviviale et ergonomique exploitable par l'utilisateur en envisageant toutes les interactions possibles à l'écran du support tenu.

4. Efficacité : l'application doit être fonctionnelle indépendamment de toute circonstances pouvant entourer l'utilisateur.

5. L'application doit optimiser les traitements et l'utilisation de ses ressources (autonomie batterie, mémoire disponible…) afin de ne pas épuisé ses dernières dans des requêtes inutiles.

6. L'accès à la base de données doit être souple et rapide.

7. Rapidité du traitement : l'application doit assurer une rapidité de traitement pour qu'elle s'approche au maximum d'une exécution en temps réel.

4. Analyse des besoins

4.1 Diagramme de cas d'utilisation :

Le diagramme de cas d'utilisation représente la structure des grandes fonctionnalités nécessaires à l'utilisateur du système. C'est le premier diagramme du modèle UML, celui où s'assure la relation entre l'utilisateur et les objets que le système met en œuvre. [1]

Pour améliorer notre diagramme, nous allons organiser les cas d'utilisation et les regrouper en ensembles cohérents. Pour cela, nous utilisons le mécanisme général de regroupement d'éléments en UML, qui s'appelle-le paquetage (package). [26]

La figure ci-dessous présente le diagramme de cas d'utilisation général de notre application

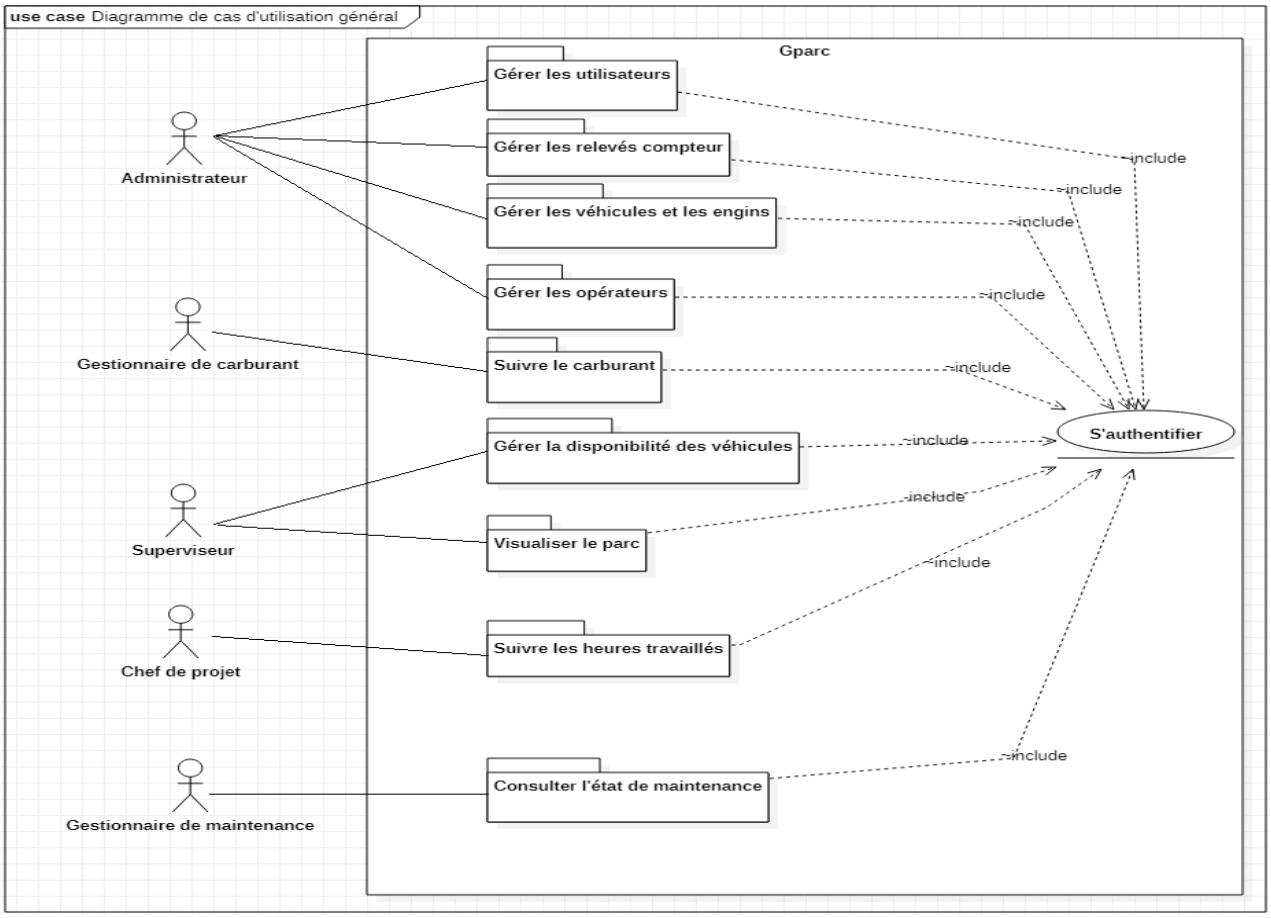


Figure 26 Diagramme de cas d'utilisation général

- Diagramme de cas d'utilisation du paquetage ‘‘Gérer les utilisateurs’’

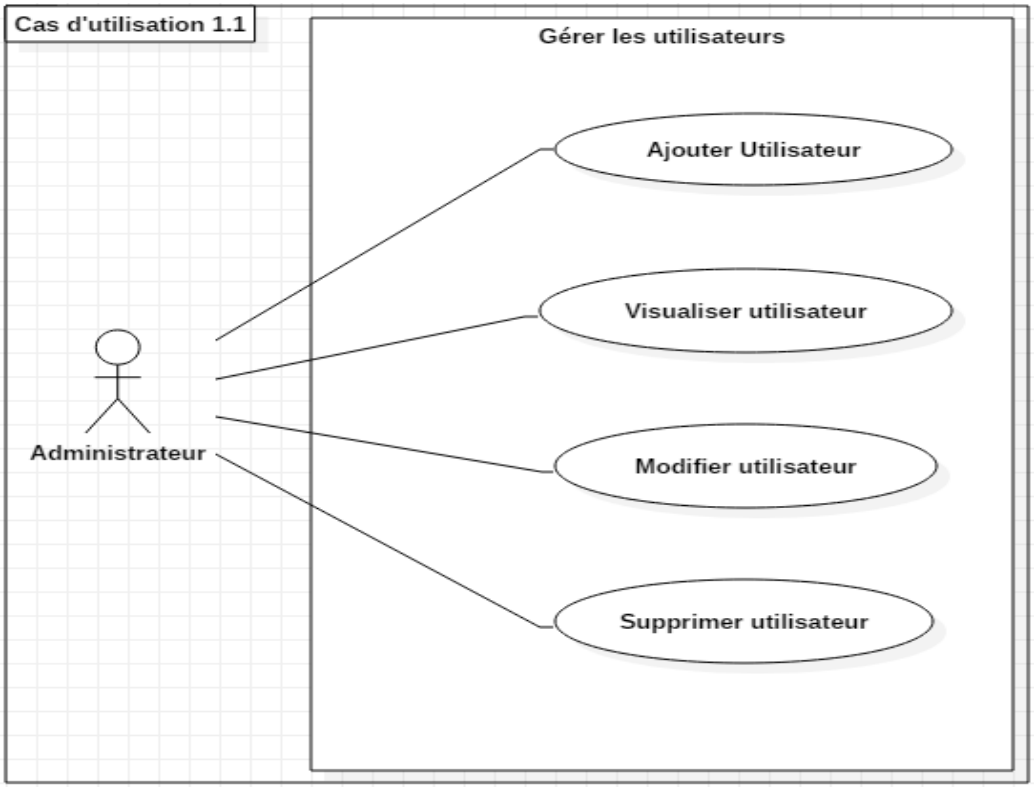


Figure 27 Diagramme de cas d'utilisation ‘‘Gérer les utilisateurs’’

- Diagramme de cas d'utilisation du paquetage ‘‘Gérer les relevés compteurs’’

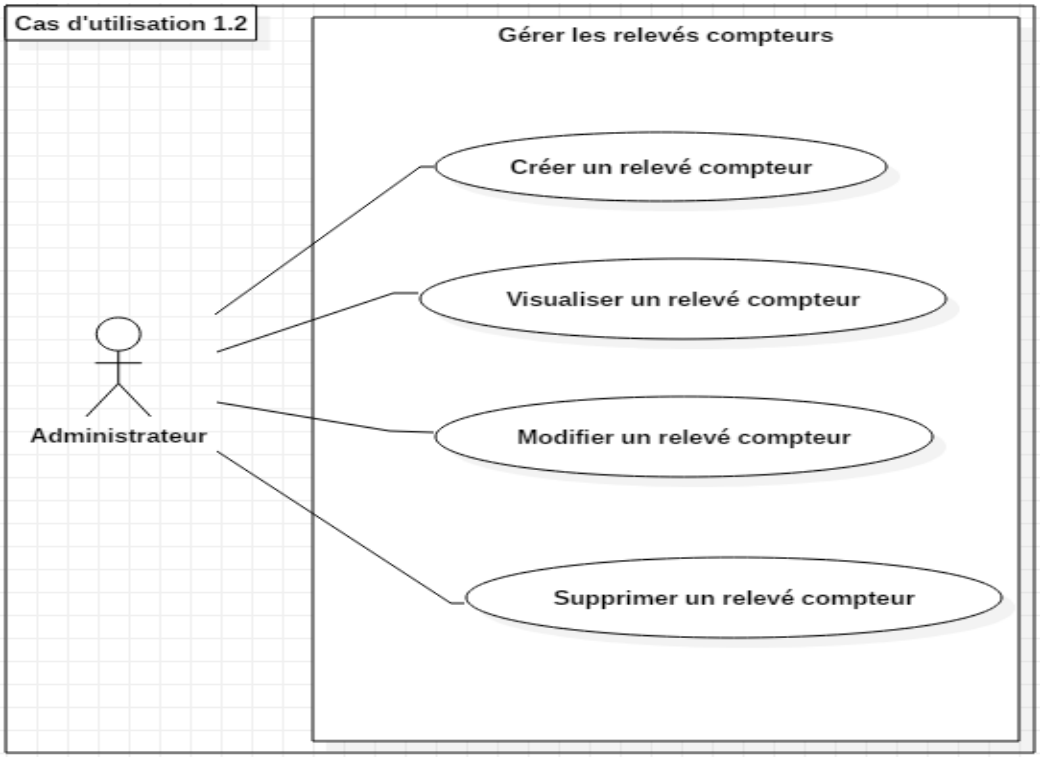


Figure 28 Diagramme de cas d'utilisation ‘‘Gérer les relevés compteurs’’

- Diagramme de cas d'utilisation du paquetage '' Gérer les véhicules''

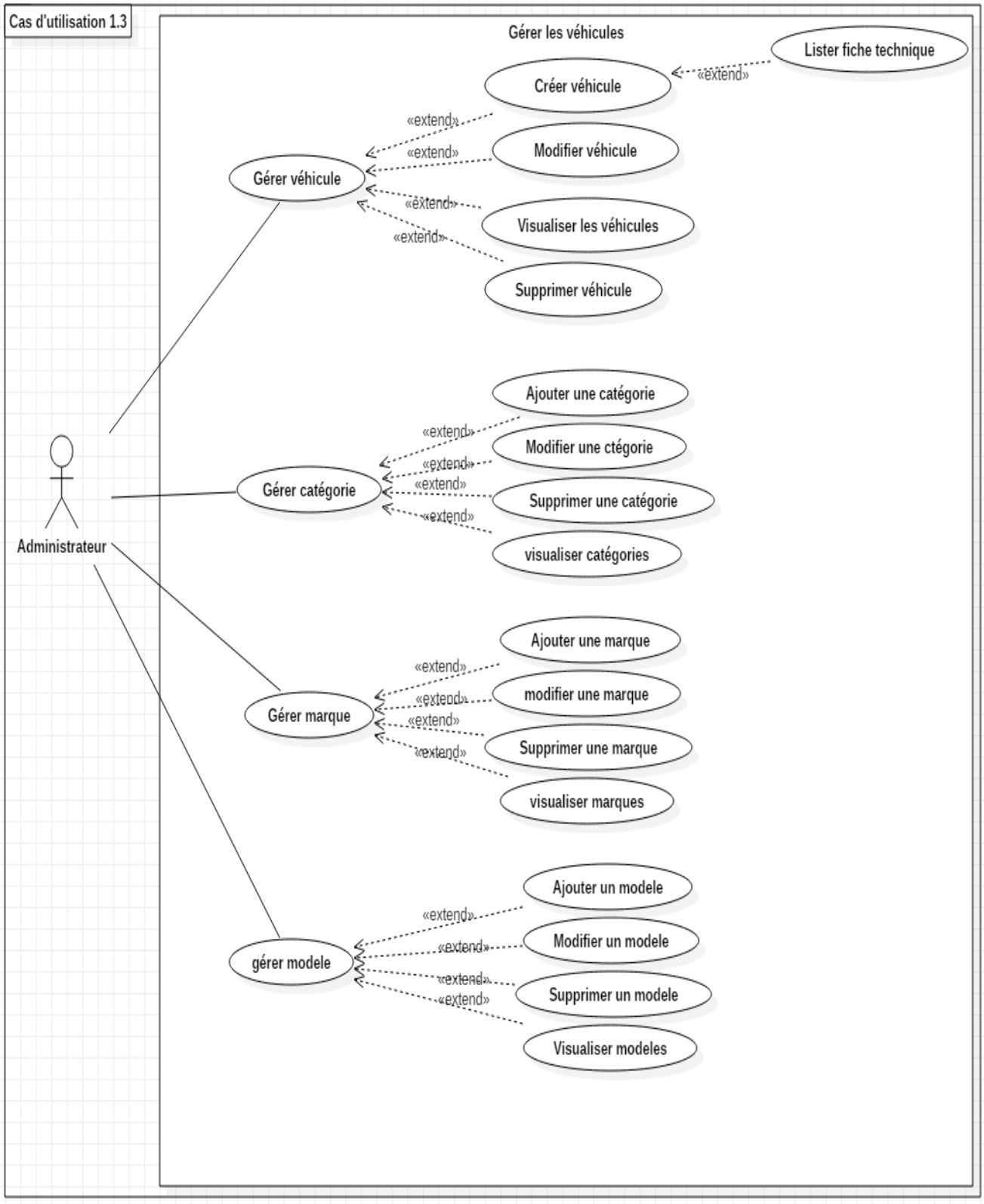


Figure 29 Diagramme de cas d'utilisation '' Gérer les véhicules''

- Diagramme de cas d'utilisation du paquetage ‘‘Suivre le carburant’’ :

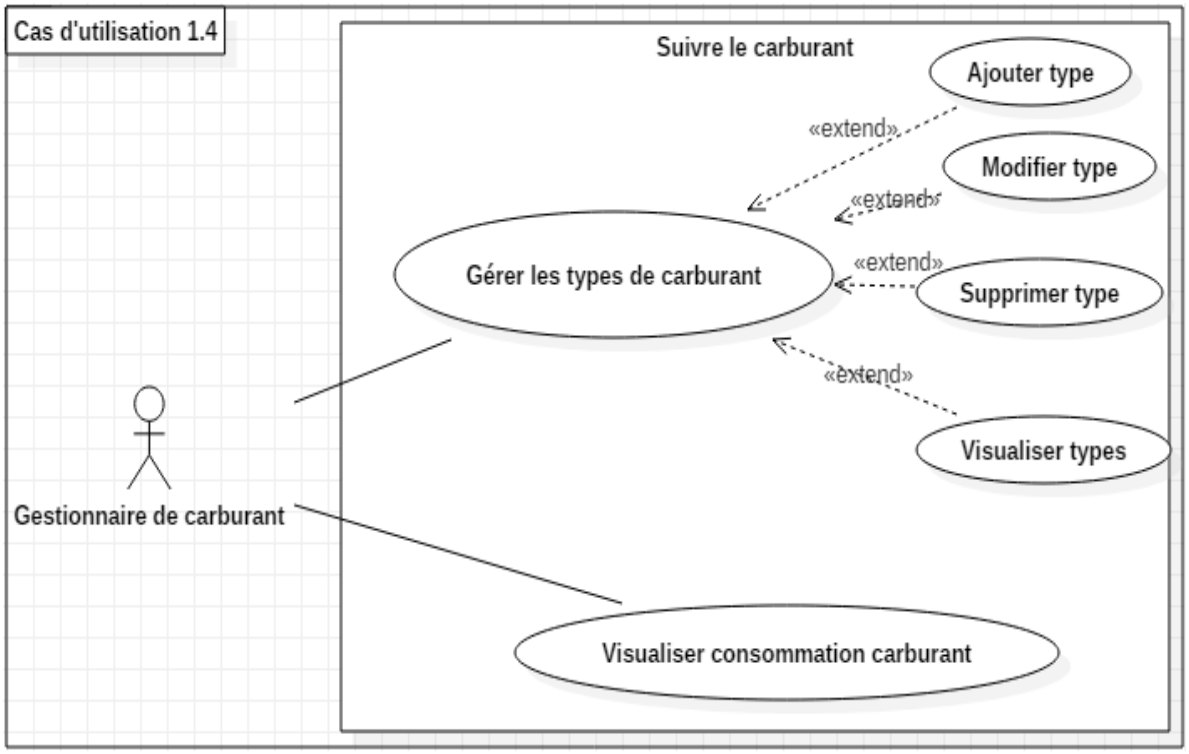


Figure 30 Diagramme de cas d'utilisation ‘‘Suivre le carburant’’

- Diagramme de cas d'utilisation ‘‘Gérer disponibilité véhicule’’

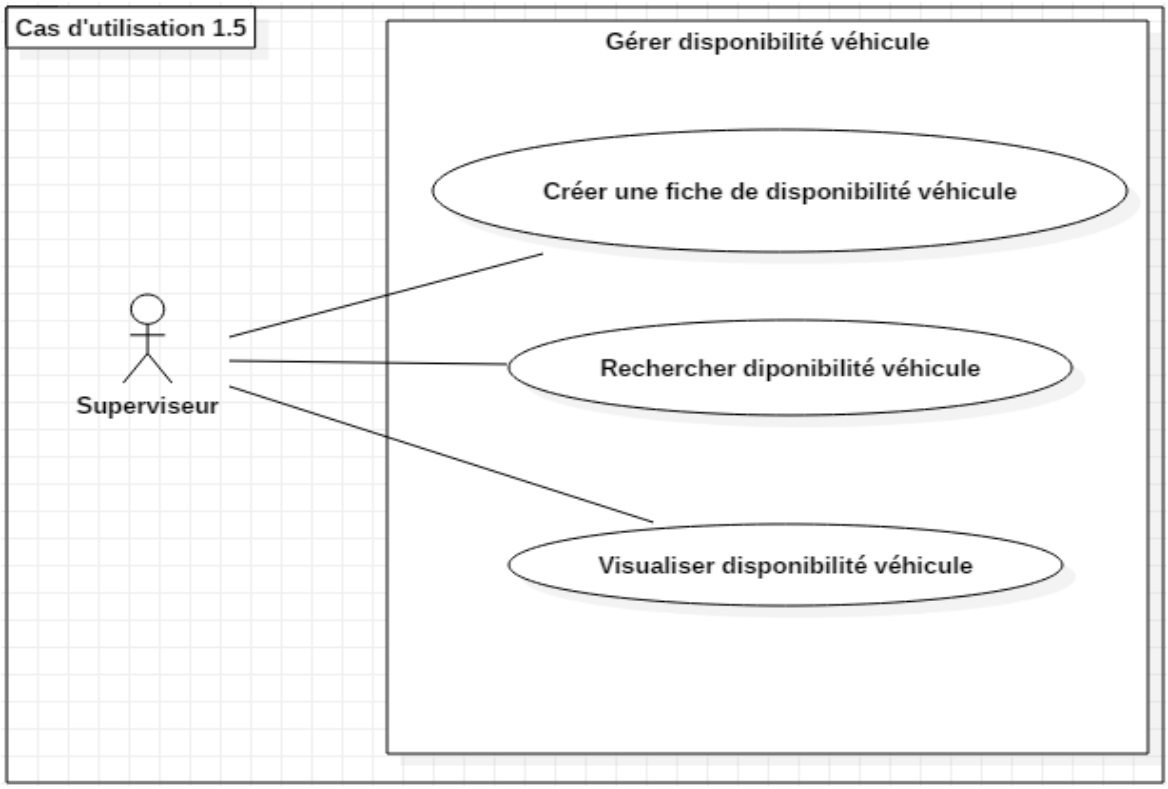


Figure 31 Diagramme de cas d'utilisation ‘‘Gérer disponibilité

- Diagramme de cas d'utilisation ''Visualiser l'activité du parc'' :

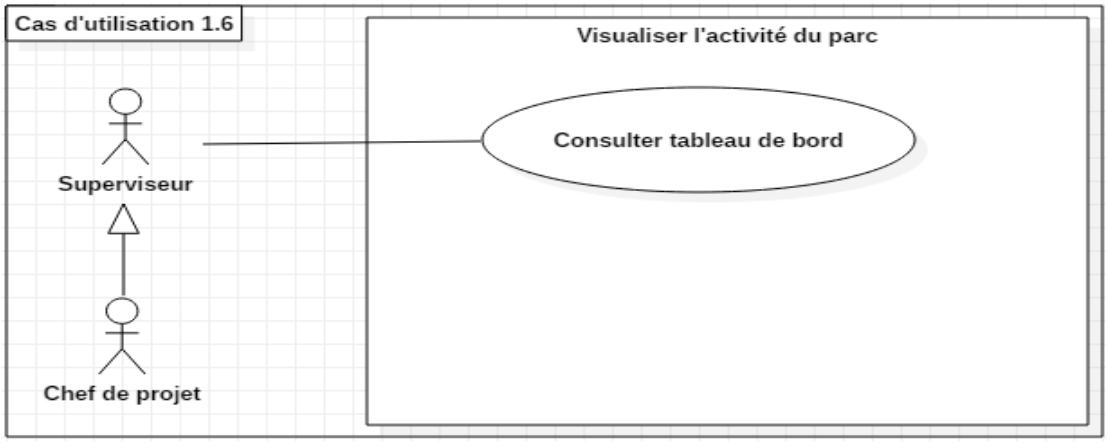


Figure 32 Diagramme de cas d'utilisation ''Visualiser l'activité du parc''

- Diagramme de cas d'utilisation ''Suivre les heures travaillées'' :

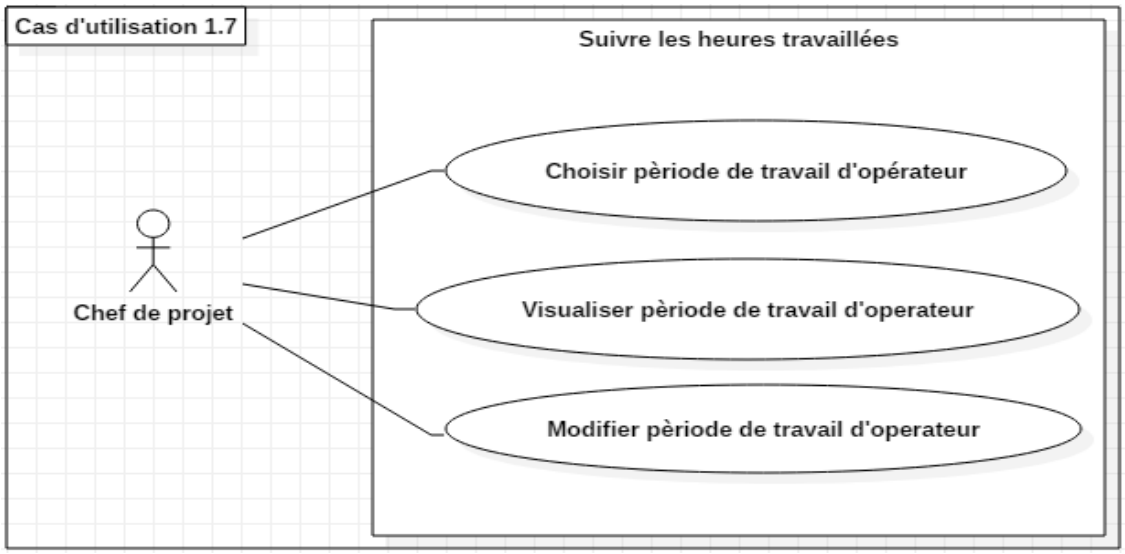


Figure 33 Diagramme de cas d'utilisation ''Suivre les heures travaillées''

- Diagramme de cas d'utilisation ''Consulter l'état de maintenance'' :

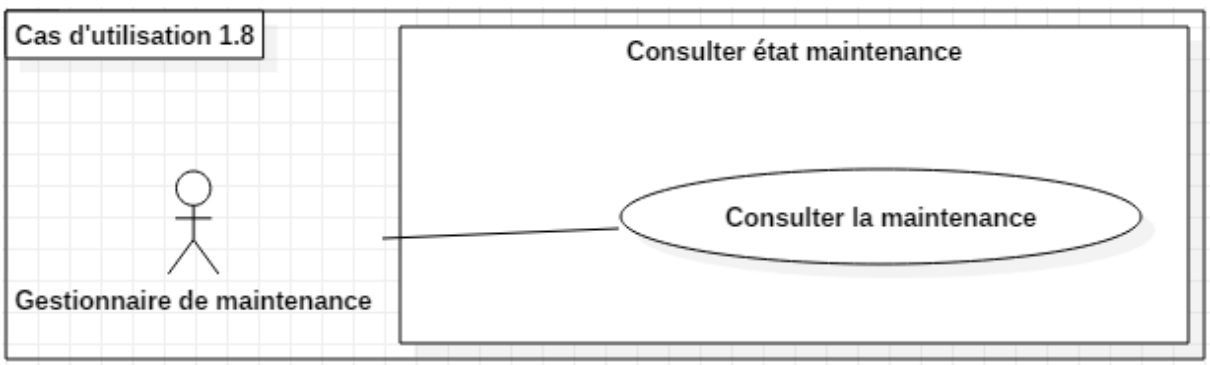


Figure 34 Diagramme de cas d'utilisation ''Consulter l'état de maintenance''

4.2 Diagramme de séquence :

Le diagramme de séquence permet d'établir un lien entre les diagrammes de cas d'utilisation et les diagrammes de classes : il montre comment des objets (i.e. des instances de classes) communiquent pour réaliser une certaine fonctionnalité. Il apporte ainsi un aspect dynamique à la modélisation du système.

Pour produire un diagramme de séquence, il faut focaliser son attention sur un sous-ensemble d'éléments du système et étudier leur façon d'interagir pour décrire un comportement particulier.

Les principales informations contenues dans un diagramme de séquence sont les messages échangés entre les lignes de vie, présentés dans un ordre chronologique. Ainsi, contrairement au diagramme de communication, le temps y est représenté explicitement par une dimension (la dimension verticale) et s'écoule de haut en bas [1].

- **Description des cas d'utilisation:**

La description textuelle des cas d'utilisation permet de décrire la chronologie des actions qui devront être réalisées par les acteurs et par le système lui-même, clarifier le déroulement de la fonctionnalité qui sera représenté par la suite par des diagrammes de séquence système.

Les diagrammes de séquence système sont la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique dans la formulation UML.

- Cas d'utilisation ''Ajouter utilisateur''
  Description textuelle :

Nom du cas : Ajouter utilisateur.

Acteur principal : Administrateur.

Post condition : L'utilisateur est ajouté.

Enchainement nominal:

1. L'administrateur choisit d'ajouter un utilisateur.
2. Le système lui affiche un formulaire à remplir.
3. L'administrateur remplit et valide le formulaire.
4. Le système vérifie les informations remplies.
5. Le système lui affiche un message de confirmation.

Enchaînements alternatifs :

3.1. L'administrateur n'a pas saisi tous les informations nécessaires ou l'utilisateur existe déjà.
3.2. Le système renvoie un message d'erreur et signale à l'administrateur de recommencer.

Diagramme de séquence :

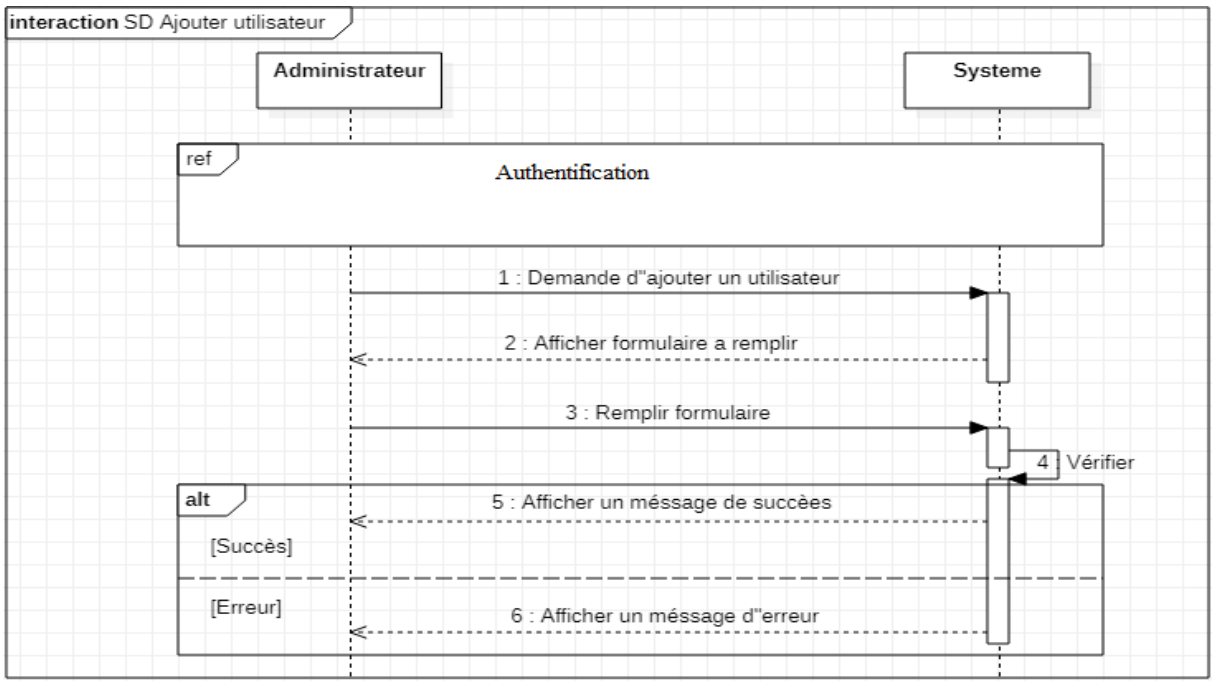


Figure 35 Diagramme de séquence Ajouter utilisateur

- Cas d'utilisation ''Modifier un utilisateur''

  Description textuelle :

Nom du cas : Modifier utilisateur.

Acteur principal : Administrateur.

Post condition : L'utilisateur estmodifié.

Enchainement nominal:

1. L'administrateur demanderla liste des utilisateurs.
2. Le système lui affiche la liste.
3. L'administrateur choisit un utilisateur.
4. Le système lui affiche l'utilisateur.
5. L'administrateur modifie l'utilisateur.
6. Le système vérifie.
7. Le système lui affiche un message deconfirmation.

Enchaînements alternatifs :

5.1. L'administrateur n'a pas saisi tous les informations correctes.
5.2.Le système renvoie un message d'erreur et signale à l'administrateur de recommencer.

Diagramme de séquence :

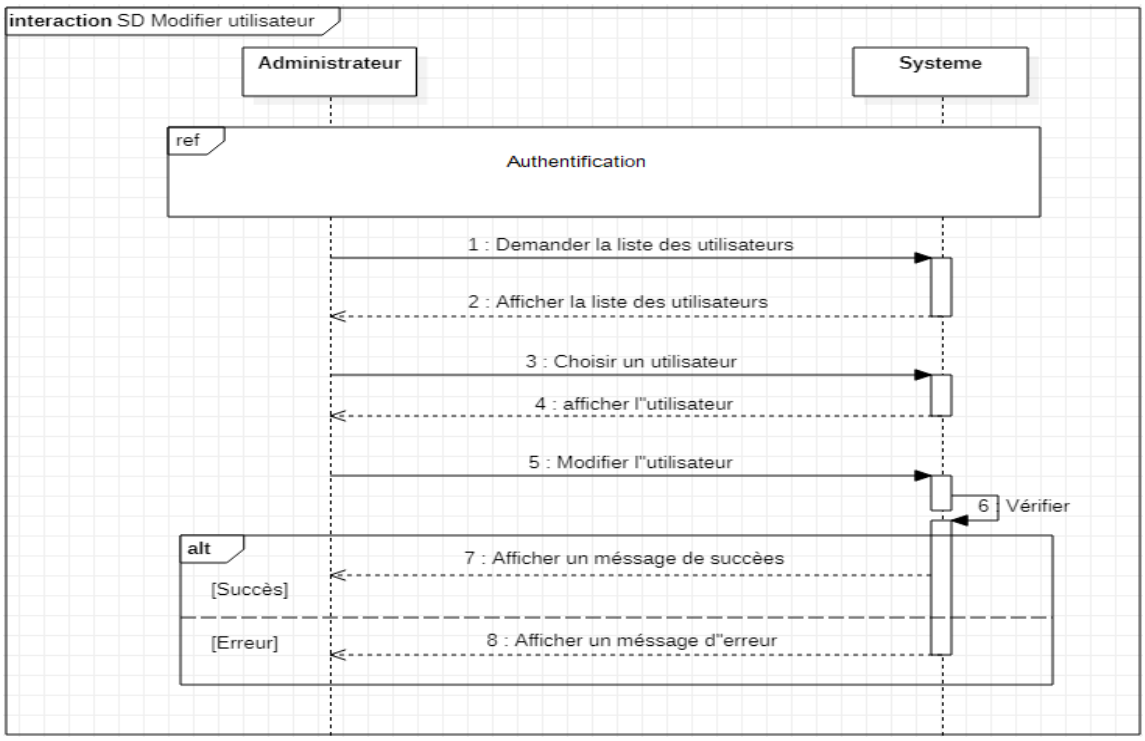


Figure 36 Diagramme de séquence Modifier utilisateur

- Cas d'utilisation ‘‘Supprimer utilisateur’’
  Description textuelle :

Nom du cas : Supprimer un utilisateur.

Acteur principal : L'administrateur.

Post condition : l'utilisateur est supprimé.

Enchaînement nominal :

1. L'administrateur demande la liste des utilisateurs.
2. Le système affiche la liste.
3. L'administrateur choisit l'utilisateur à supprimer.
4. Le système demande la confirmation de la suppression.
5. L'administrateur confirme la suppression.
6. Le système supprime le véhicule et affiche un message de confirmation.

Enchaînements alternatifs :

4.1. L'administrateur annule la suppression d'utilisateur.
4.2. Le système réaffiche la liste des utilisateurs.

Diagramme de séquence :

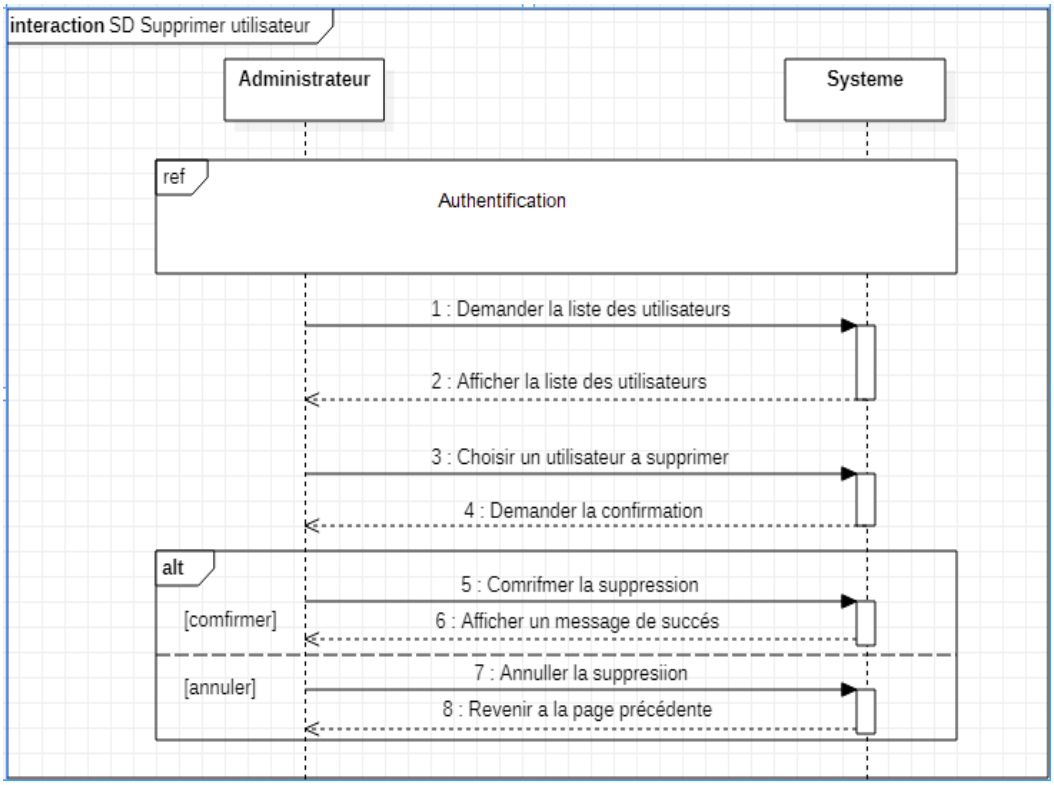


Figure 37 diagramme de séquence Supprimer utilisateur

- Cas d'utilisation ''Créer un relevé compteur''
  Description textuelle :

Nom du cas : Créer relevé compteur.

Acteur principal : L'administrateur.

Post condition : Le relevé compteur est créé.

Enchaînement nominal :

1. L'administrateur demande de créer un relevécompteur.
2. Le système lui affiche un formulaire à remplir.
3. L'administrateur remplit et valide le formulaire.
4. Le système vérifie les informations remplies.
5. Le système lui affiche un message de confirmation.

Enchaînements alternatifs :

3.1. L'administrateur n'a pas saisi les informations correctes (nouvel index kilométrique inférieura l'ancienindex).
3.2. Le système renvoie un message d'erreur et demande à l'administrateur de ressaisir.

- Diagramme de séquence :

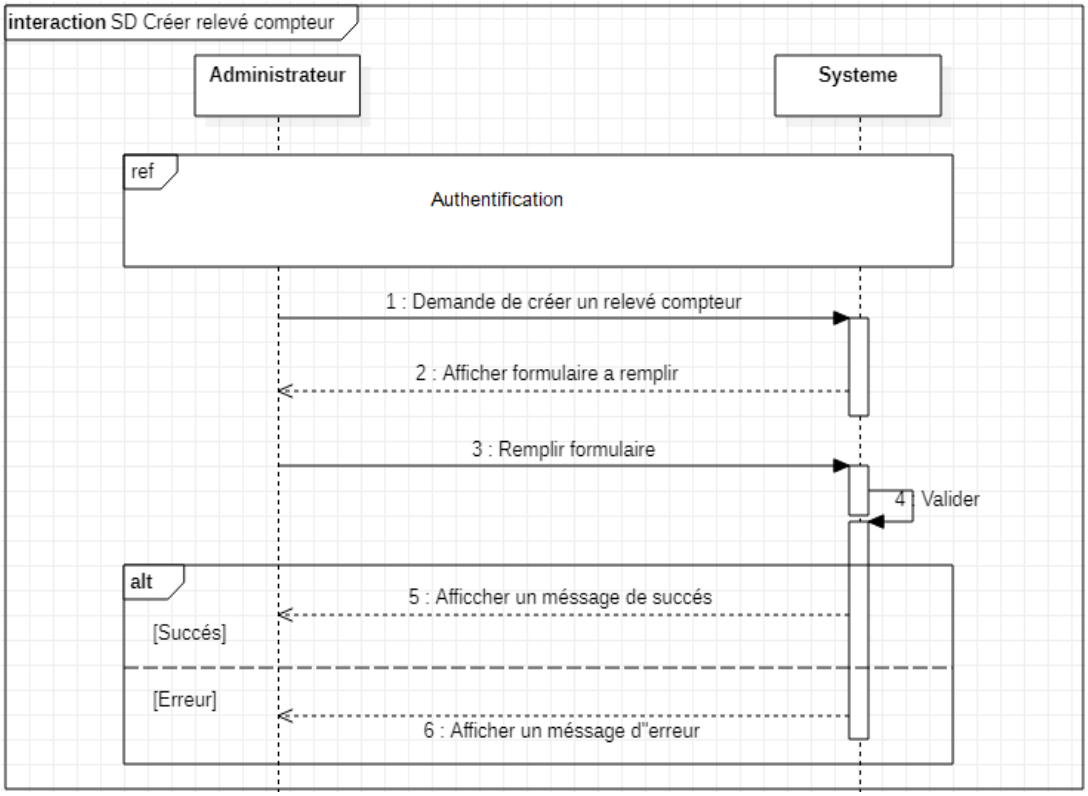


Figure 38 Diagramme de séquence Créer relevé compteur

- Cas d'utilisation ‘‘Créer véhicule’’
  Description textuelle :

Nom du cas : Créerun véhicule.

Acteur principal : L'administrateur.

Post condition : Le véhicule est créé.

Enchaînement nominal :

1. L'administrateur demande de créer un véhicule.
2. Le système lui affiche une fiche véhicule.
3. L'administrateur remplit et valide la fiche.
4. Le système vérifie les informations remplies.
5. Le système lui affiche un message de confirmation.

Enchaînements alternatifs :

3.3.L'administrateur n'a pas saisi les informations correctes
3.4.Le système renvoie un message d'erreur et demande à l'administrateur de ressaisir.

Diagramme de séquence :

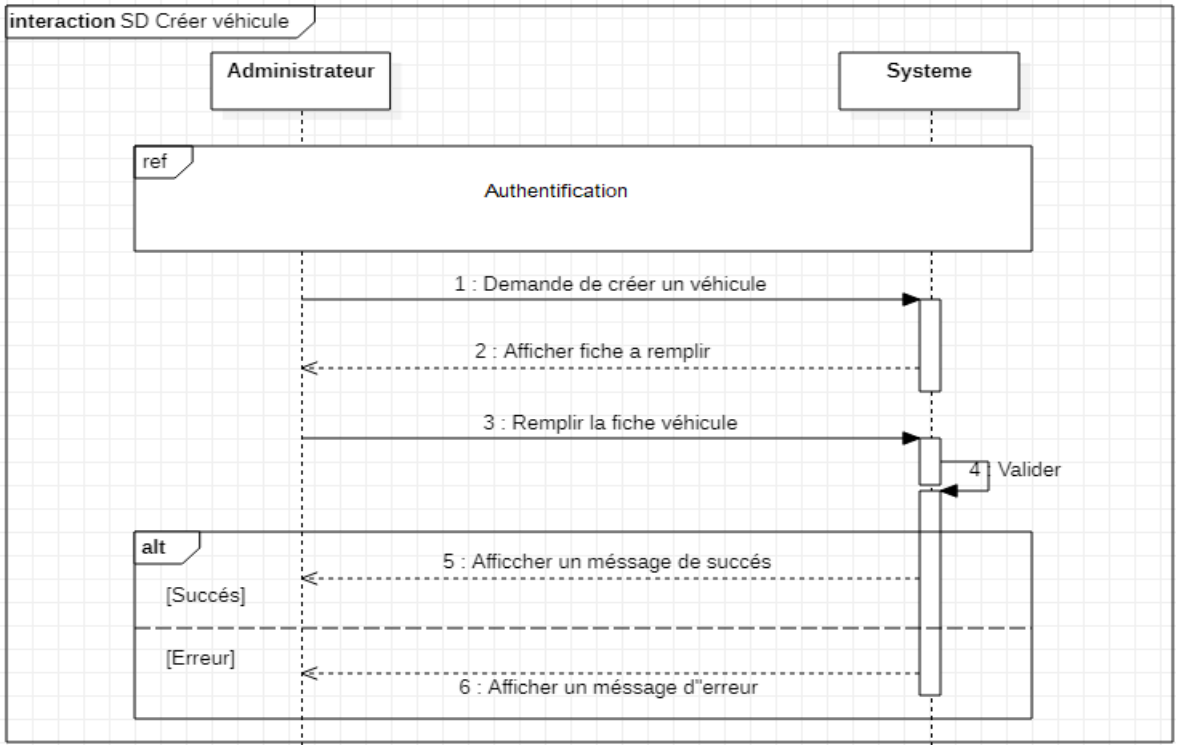


Figure 39 Diagramme de séquence créer un véhicule

- Cas d'utilisation ''Modifier véhicule''
  Description textuelle :

Nom du cas : Modifier véhicule.

Acteur principal : Administrateur.

Post condition : Le véhicule estmodifié.

Enchainement nominal :

1. L'administrateur demanderla liste des véhicules.
2. Le système lui affiche la liste.
3. L'administrateur choisit un véhicule.
4. Le système lui affiche la fiche de véhicule choisi.
5. L'administrateur modifie le véhicule.
6. Le système vérifie.
7. Le système lui affiche un message deconfirmation.

Enchaînements alternatifs :

5.1. L'administrateur n'a pas saisi tous les informations correctes.
5.3.Le système renvoie un message d'erreur et signale à l'administrateur de recommencer.

Diagramme de séquence :

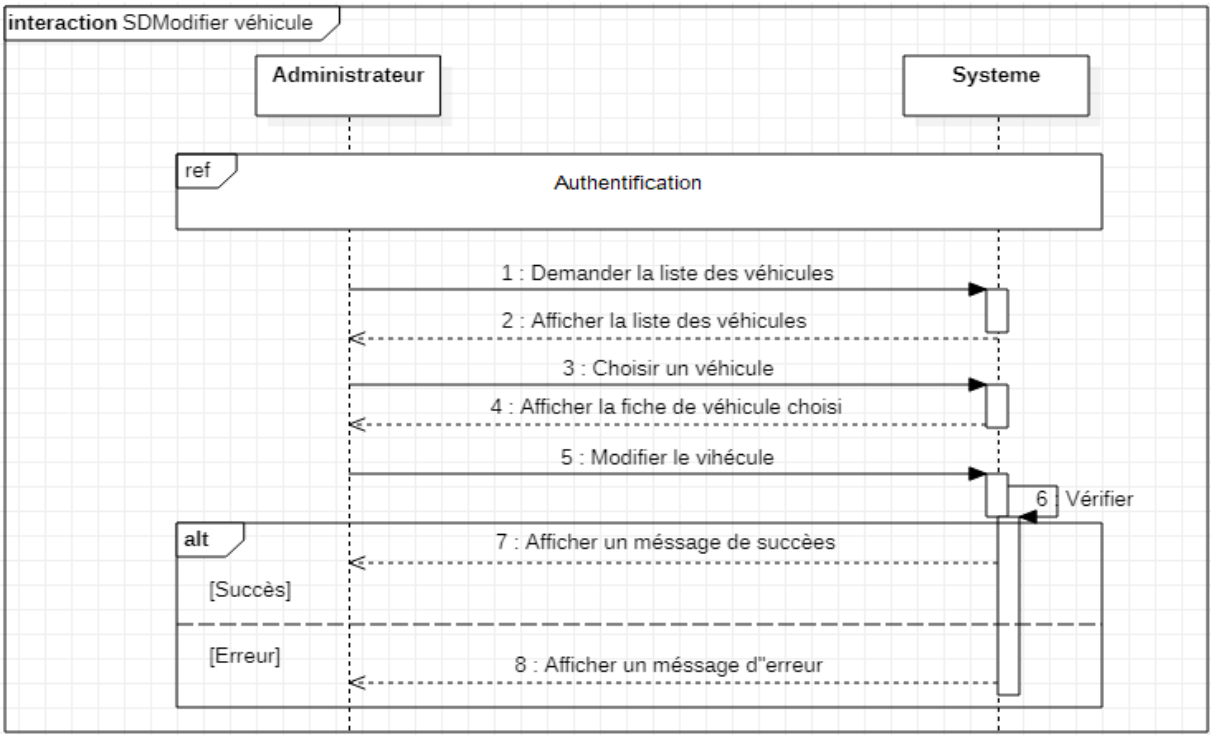


Figure 40 Modifier un véhicule

- Cas d'utilisation ''Supprimer véhicule''
  Description textuelle :

Nom du cas : Supprimer un véhicule.

Acteur principal : L'administrateur.

Post condition : Le véhicule est supprimé.

Enchaînement nominal :

7. L'administrateur demande la liste des véhicules.
8. Le système affiche la liste.
9. L'administrateur choisit le véhicule à supprimer.
10. Le système demande la confirmation de la suppression.
11. L'administrateur confirme la suppression.
12. Le système supprime le véhicule et affiche un message de confirmation.

Enchaînements alternatifs :

4.3. L'administrateur annule la suppression d'utilisateur.
4.4. Le système réaffiche la liste des utilisateurs.

Diagramme de séquence :

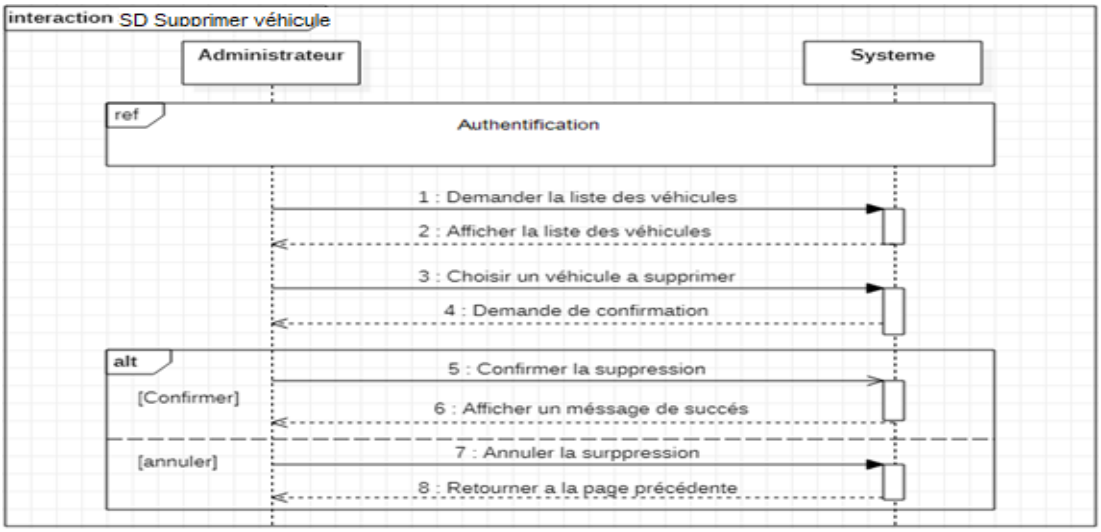


Figure 41 Diagramme de séquence Supprimer véhicule

Remarque :

Les autres cas d'utilisation d'ajout, modification et suppression des marques, modèles, catégorie, les opérateurs, les sinistres et les autres entités se déroulent de la même façon que les précédents.

- Cas d'utilisation ''Modifier période de travail d'opérateur''

  Description textuelle :

Nom du cas : Modifier période de travail d'opérateur.

Acteur principal : Chef de projet.

Post condition : La période est modifiée.

Enchaînement nominal :

1. Le chef de projet demande la liste des operateurs.
2. Le système affiche la liste.
3. Le chef de projet choisit un operateur.
4. Le système affiche les informations de l'operateur choisi.
5. Le chef de projet modifie la période de travail de l'opérateur.
6. Le système vérifie.
7. Le système affiche un message de confirmation.

Enchaînements alternatifs :

5.1.La période saisi par le chef de projet est incorrecte.
5.2.Le système affiche un message d'erreur.

Diagramme de séquence :

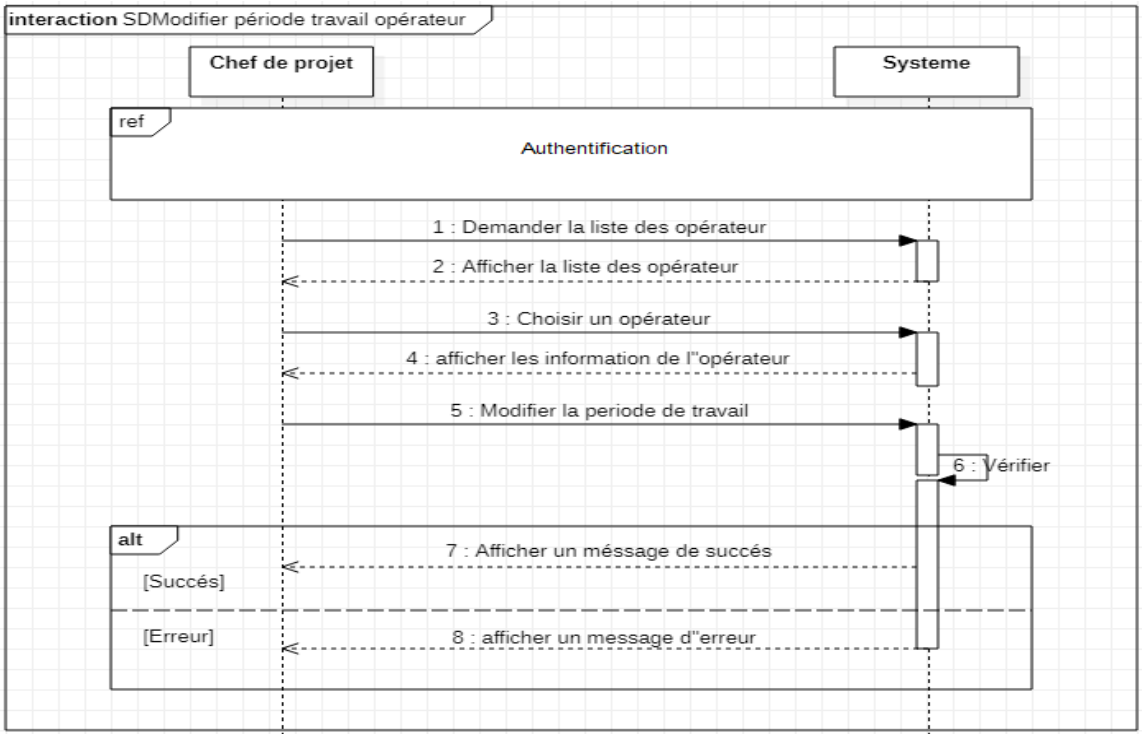


Figure 42 Diagramme de séquence Modifier période de travail operateur

- Cas d'utilisation ''Consulter l'état de maintenance''

  Description textuelle :

Nom du cas : Consulter l'état de maintenance.

Acteur principal : Gestionnaire de maintenance.

Enchaînement nominal :

1. Le gestionnaire de maintenance demande la liste des véhicules en maintenance.
2. Le système affiche la liste.
3. Le gestionnaire de maintenance choisit un véhicule.
4. Le système affiche les informations du véhicule et les détails de la maintenance.

Diagramme de séquence :

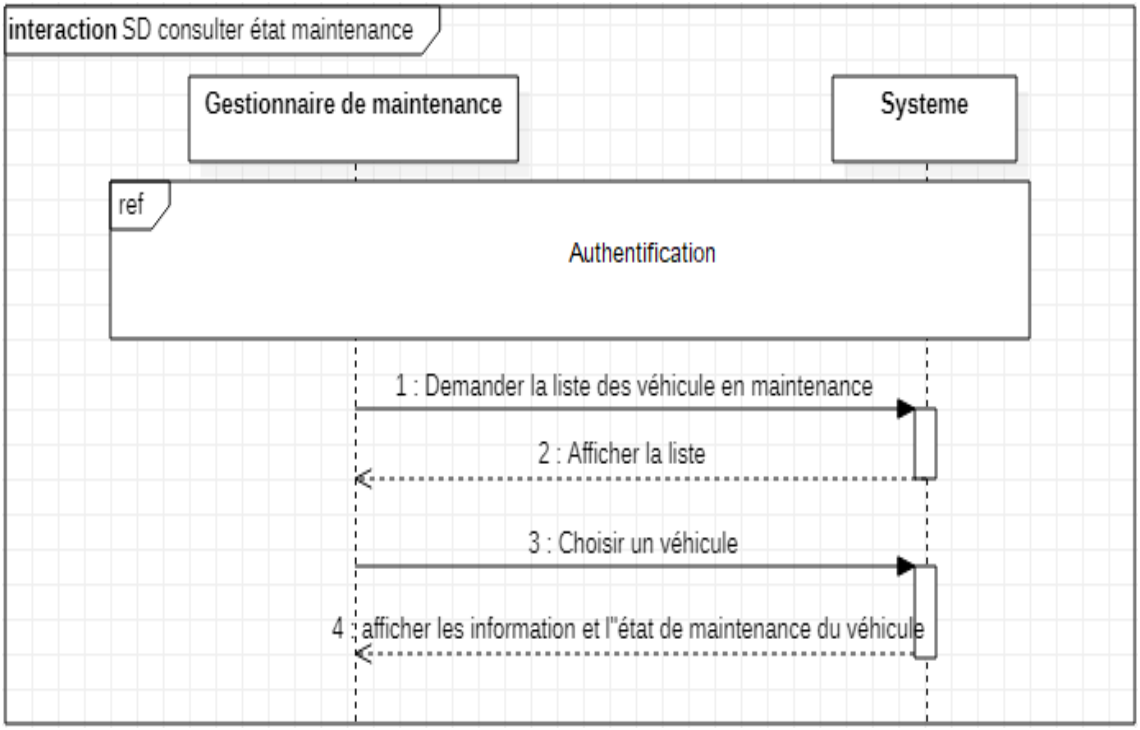


Figure 43 Diagramme de séquence Consulter état maintenance

- Cas d'utilisation ''Vérifier disponibilité véhicule''

Nom du cas : Vérifier la disponibilité d'un véhicule.

Acteur principal : Superviseur.

Enchaînement nominal :

1. Le superviseur choisit de vérifier la disponibilité d'un véhicule.
2. Le système affiche la liste des véhicules.
3. Le superviseur choisit la période de disponibilité et le véhicule à vérifier.
4. Le superviseur lance la recherche de ce véhicule.
5. Le système affiche un message de succès.

Enchaînement alternatif :

4.1. Le véhicule n'est pas disponible.
4.2. Le système affiche un message d'erreur et lui redirige vers la liste des véhicules.

Description textuelle :

Diagramme de séquence :

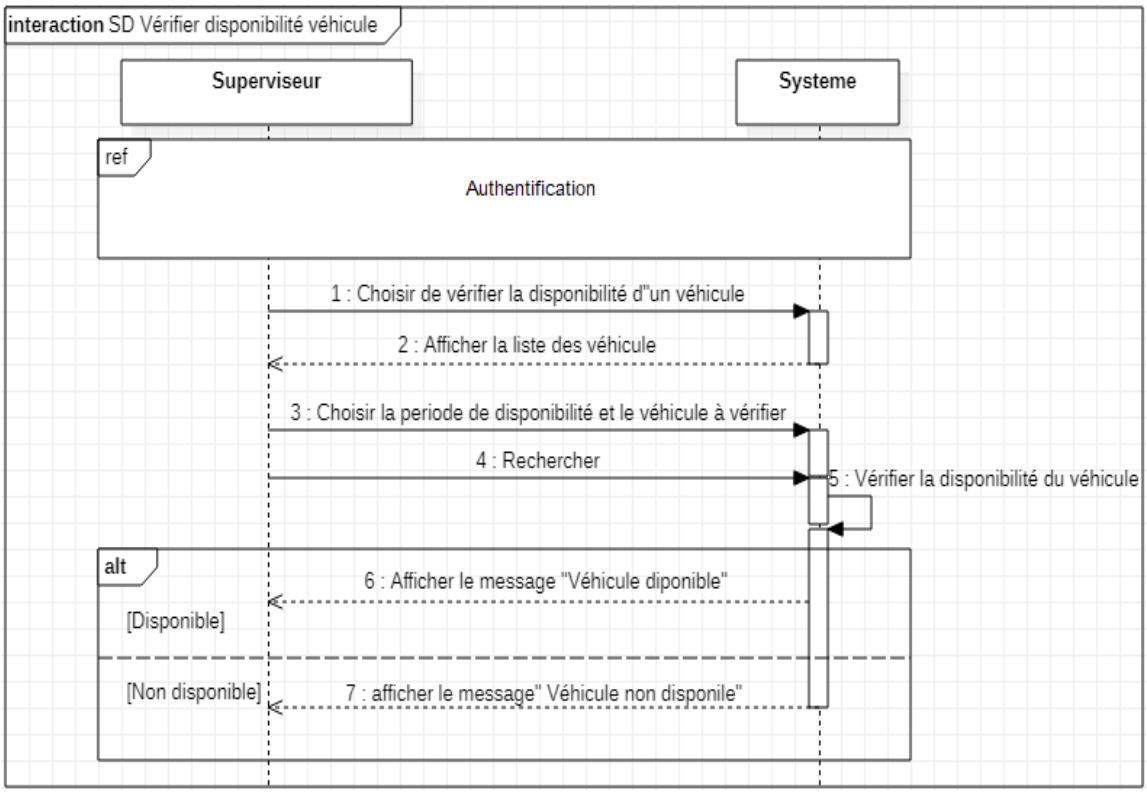


Figure 44 Diagramme de séquence vérifier la disponibilité véhicule

- **Les diagrammes des séquences détaillés :**

  Les diagrammes des séquences détaillés sont obtenus en se basant sur les diagrammes des séquences système, et cela en remplaçant le système vu comme une boite noire par un ensemble d'objets en interaction. Ces objets sont répartis en trois catégories [10]:

  1. Objets d'entités : représentent les concepts du système qu'on a définit comme classe.

  2. Objets d'interfaces : représentent l'interaction avec les utilisateurs.

  3. Objets de contrôles: qui sont utilisés pour contrôler les objets Interface et Entité, et qui représentent un transfert d'informations.

Les figures suivantes représentent les diagrammes de séquence détaillés

de notre projet :

- Diagramme de séquence détaillé ‘‘Ajouter utilisateur’’ :

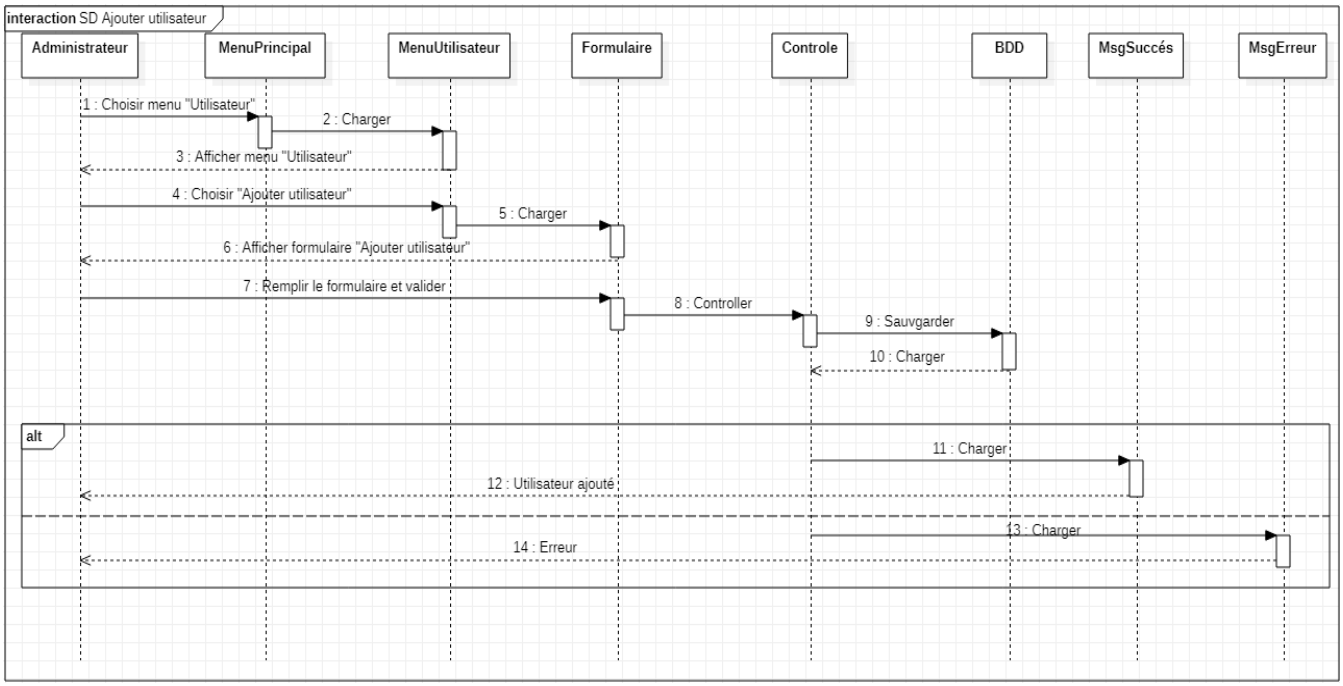


Figure 45 Diagramme de séquence détaillé Ajouter utilisateur

- Diagramme de séquence détaillé ‘‘Modifier véhicule’’ :

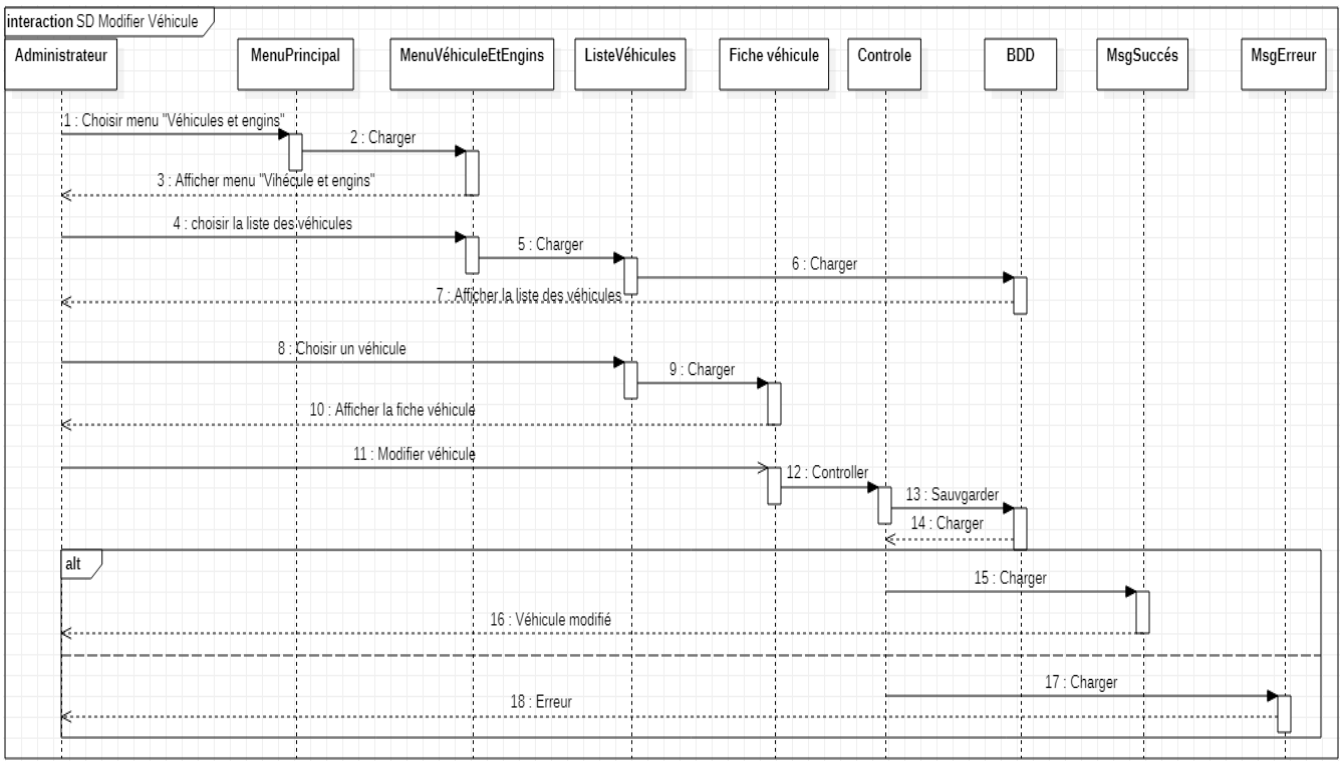


Figure 46 Diagramme de séquence détaillé Modifier véhicule

- Diagramme de séquence détaillé ‘‘Supprimer véhicule’’ :

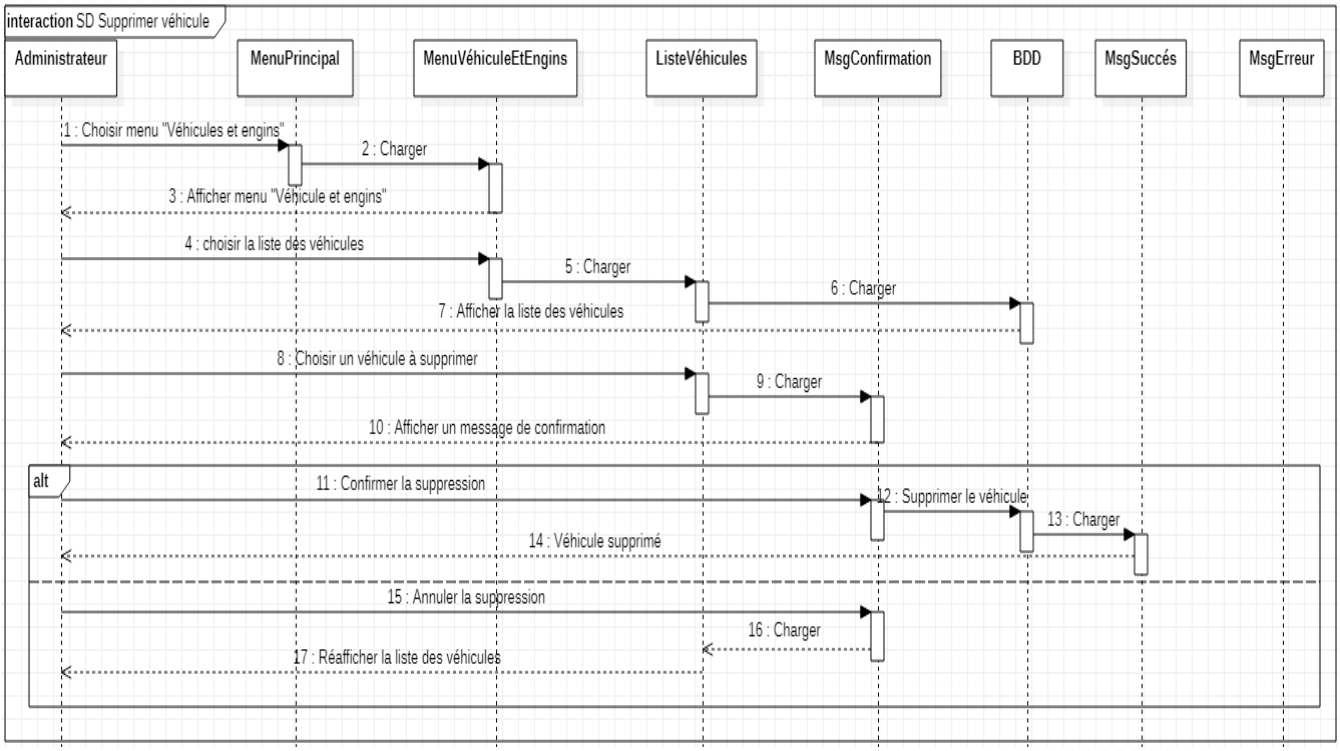


Figure 47 Diagramme de séquence détaillé Supprimer véhicule

- Diagramme de séquence détaillé ‘‘Créer relevé compteur’’ :

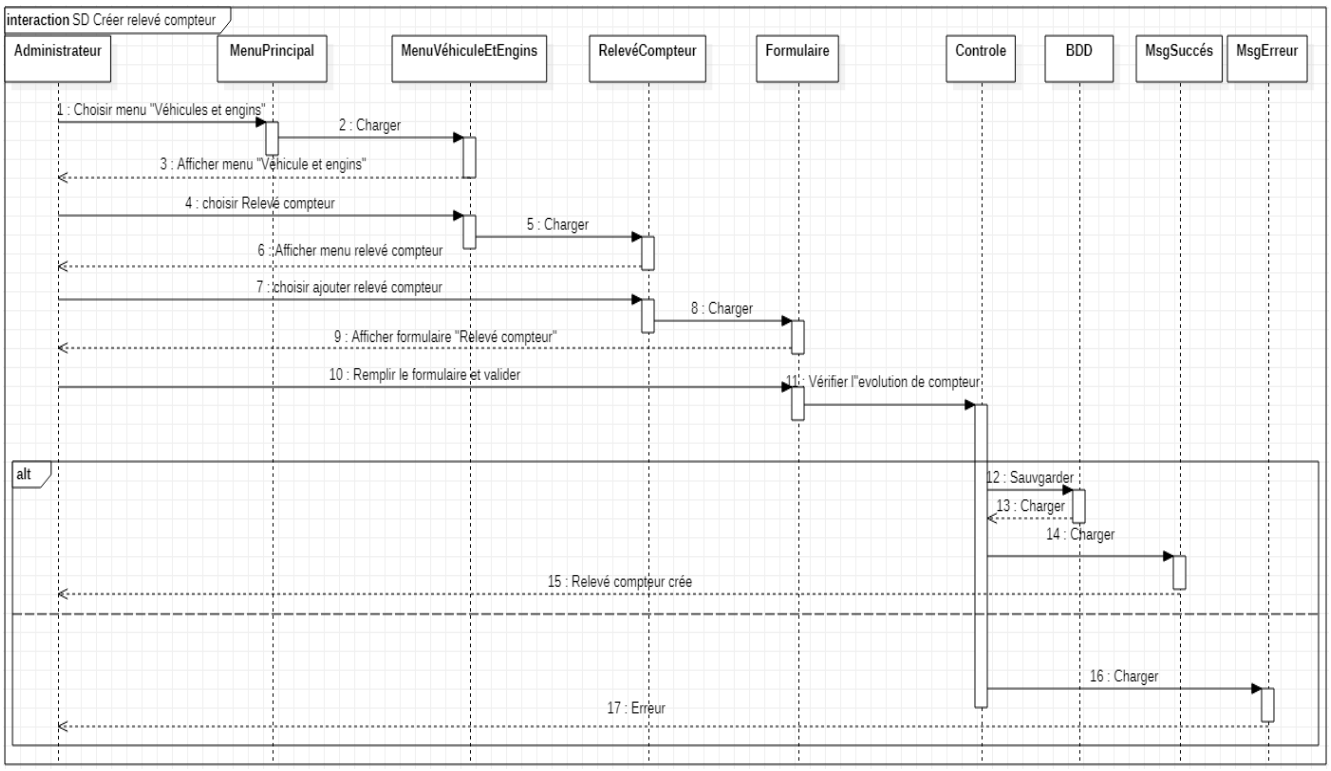


Figure 48 Diagramme de séquence détaillé Créer relevé compteur

- Diagramme de séquence détaillé ‘‘Modifier relevé compteur’’ :

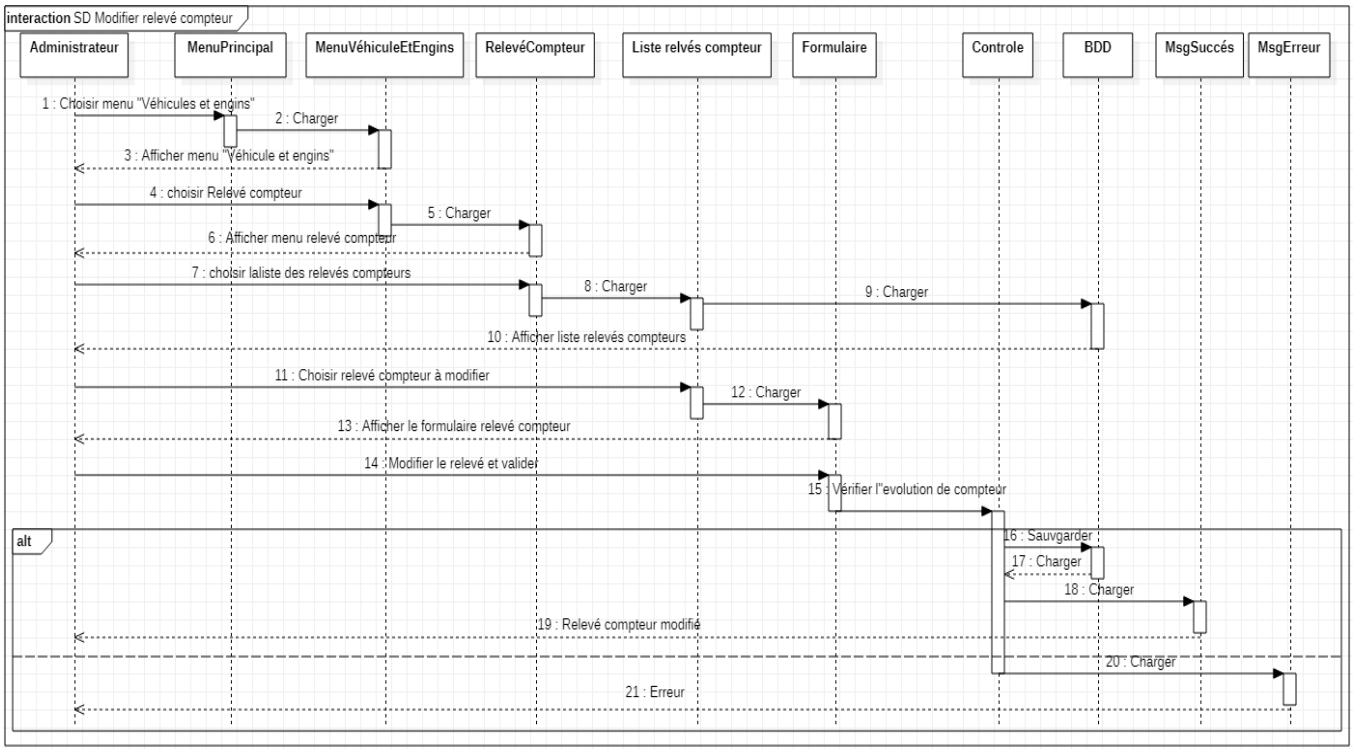


Figure 49 Diagramme de séquence détaillé Modifier relevé compteur

- Diagramme de séquence détaillé ‘‘Vérifier disponibilité véhicule’’:

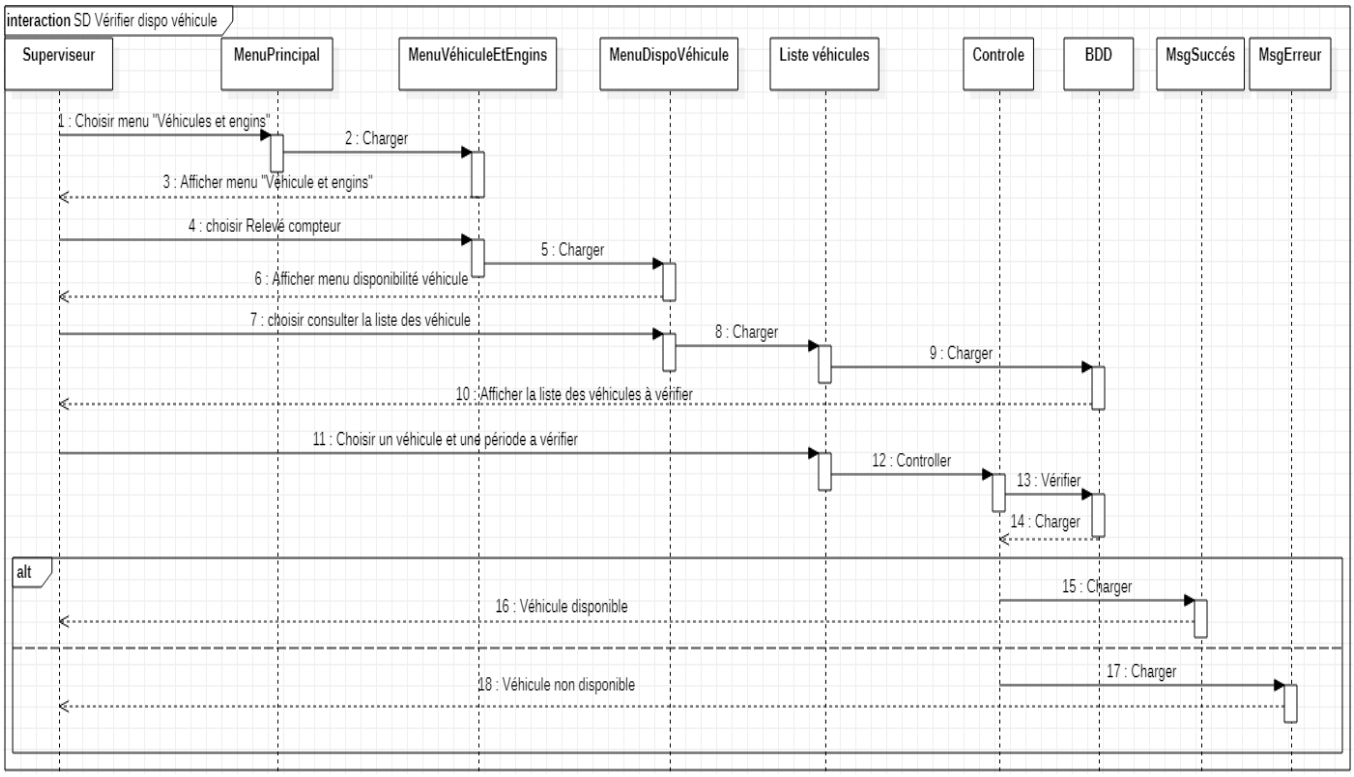


Figure 50 Diagramme de séquence détaillé Vérifier disponibilité véhicule

- Diagramme de séquence détaillé ‘‘Consulter état de maintenance‘‘:

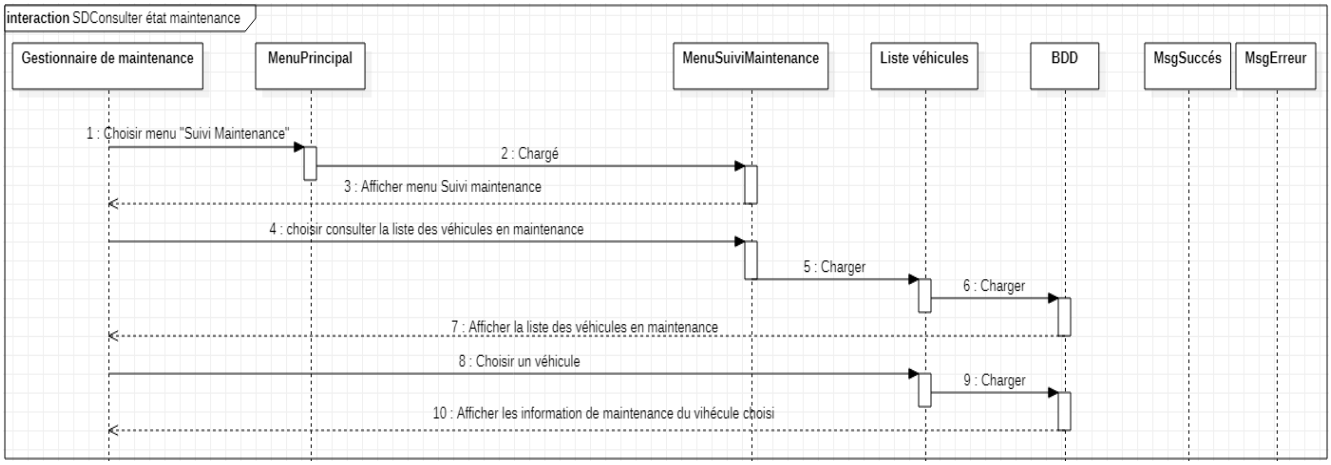


Figure 51 Diagramme de séquence détaillé Consulter état maintenance

4.3 Diagramme d’activité

Un diagramme d’activité permet de spécifier des traitements a priori séquentiels. Il offre un pouvoir d’expression très proche des langages de programmation objet : spécification des actions de base (déclaration de variables, affectation. . .), structures de contrôle (conditionnelles, boucles), ainsi que les instructions particulières à la programmation orientée objet (appels d’opérations, exceptions. . .). Il est donc bien adapté à la spécification détaillée des traitements en phase de réalisation. [10]

- La figure ci-dessous représente le diagramme d’activité de notre projet :

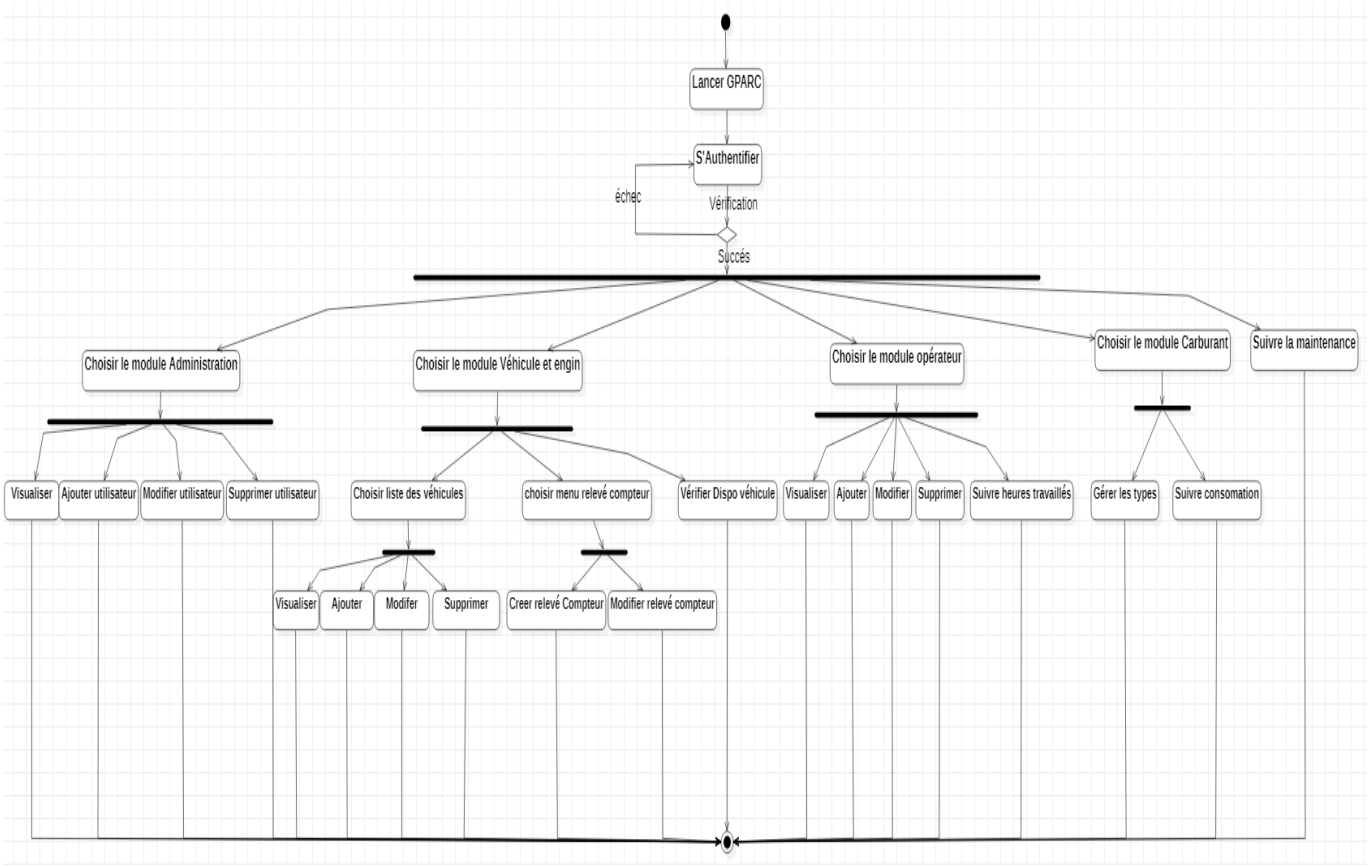


Figure 52 Diagramme d'activité

5. Conception du système

5.1 Diagramme de classe :

Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet. Le diagramme de classes montre la structure interne. Il permet de fournir une représentation abstraite des objets du système qui vont interagir ensemble pour réaliser les cas d'utilisation.Il est important de noter qu'un même objet peut très bien intervenir dans la réalisation de plusieurs cas d'utilisation.

Le diagramme de classes modélise les concepts du domaine d'application ainsi que les concepts internes créés de toutes pièces dans le cadre de l'implémentation d'une application.

Le diagramme de classes permet de modéliser les classes du système et leurs relations indépendamment d'un langage de programmation particulier.

Les principaux éléments de diagramme de classes sont : les classes et leurs relations : association, généralisation et plusieurs types de dépendances [1].

Le Diagramme de classe général de notre application est le suivant :

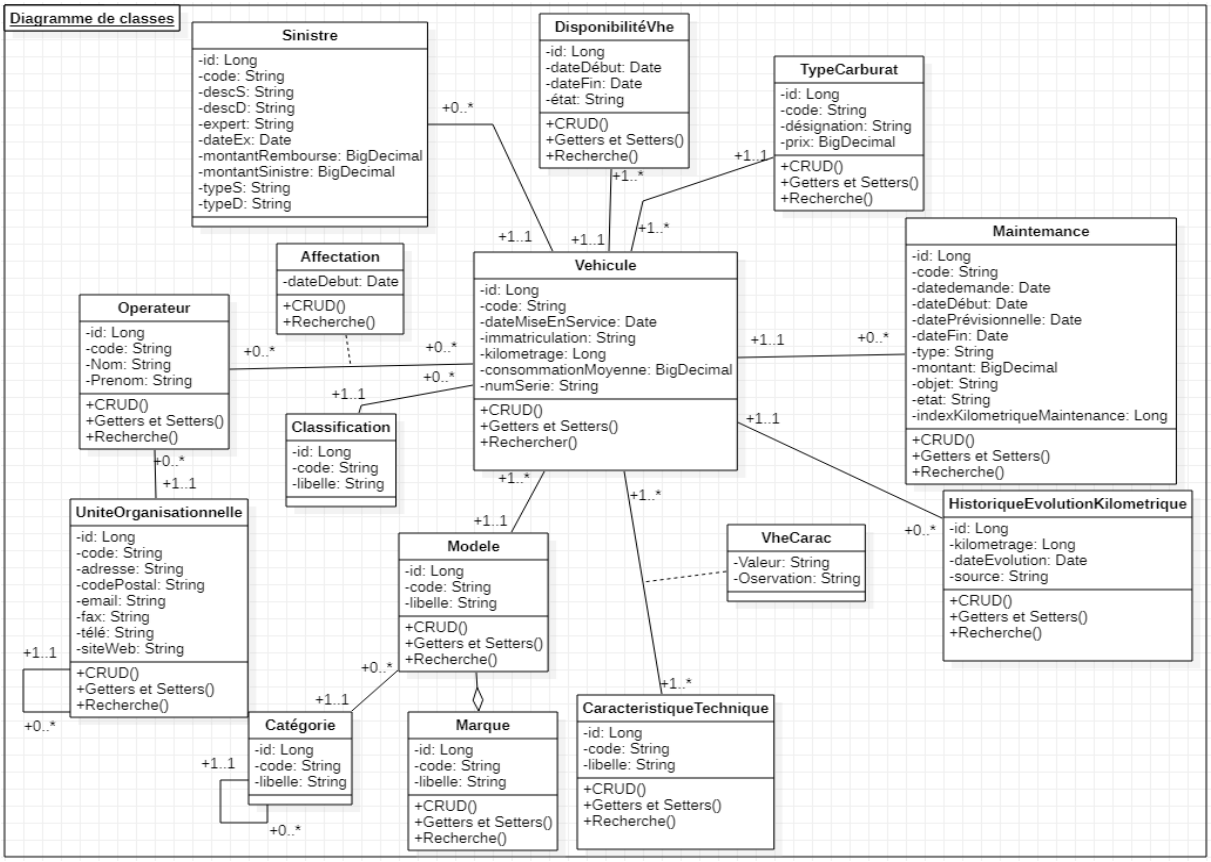


Figure 53 Diagramme de classes

- **Dictionnaire des données**

Classe	Nom	Description	Type	Taille
Operateur	Id	Identifiant d'un opérateur	Long	
	Code	Code de l'opérateur	Chaine de caractère	50
	Nom	Nom de l'opérateur	Chaine de caractère	50
	Prénom	Prénom de l'opérateur	Chaine de caractère	50
Unité Organisationnelle	Id	Identifiant d'une unité	Long	
	Code	Code de l'unité	Chaine de caractère	50
	Adresse	Adresse de l'unité	Chaine de caractère	255
	codePostal	code Postal de l'unité	Chaine de caractère	255
	Email	Email de l'unité	Chaine de caractère	50
	Fax	Numéro de fax	Chaine de caractère	50
	Téléphone	Numéro téléphone de l'unité	Chaine de caractère	50
	siteWeb	Site official de l'unité	Chaine de caractère	50
Dispo	Id	Identifiant de la disponibilité du véhicule	Long	
	date_debut	Date de début de disponibilité du véhicule	Date	

Véhicule	date_fin	Date fin de disponibilité du véhicule	Date	
	Etat	Etat de disponibilité du véhicule	Chaine de caractère	50
Catégorie	Id	Identifiant de catégorie	Long	
	Code	Code de catégorie	Chaine de caractère	50
	Libelle	Libelle de catégorie	Chaine de caractère	50
Véhicule	Id	Identifiant du véhicule	Long	
	Code	Code de véhicule	Chaine de caractère	
	dateMiseEnService	Date d'acquisition de véhicule	Date	
	Immatriculation	Immatriculation de véhicule	Chaine de caractère	50
	kilometrage	Index kilométrique	Long	
	consomationMoy	Consommation moyenne du carburant	BigDécimal	
	numSérie	Numéro de série véhicule	Chaine de caractère	50
	KliomMiseEnServ	Index kilométrique mise en service	Long	
Marque	Id	Identifiant de la marque	Long	
	Code	Code de la marque	Chaine de caractère	50
	Libelle	Libelle de la marque	Chaine de caractère	50
Modèle	Id	Identifiant du modèle	Long	
	Code	Code du modèle	Chaine de caractère	50
	Libelle	Libelle du modèle	Chaine de caractère	50
Type carburant	Id	Identifiant de type carburant	Long	
	Code	Code de type	Chaine de caractère	50
	Désignation	Désignation du type	Chaine de caractère	50
	Prix	Le prix de carburant	BigDécimal	
Caractéristique technique	Id	identifiant	Long	
	Code	Code caractéristique	Chaine de caractère	50
	Libelle	Libelle du caractéristique	Chaine de caractère	50
	Valeur	La valeur du caractéristique	Chaine de caractère	50
Classification	Id	Identifiant de la classification	Long	
	Code	Code de la classification	Chaine de caractère	50
	Libelle	Libelle de la classification	Chaine de caractère	50
Historique Evolution Kilometrique	Id	Identifiant de l'évolution	Long	
	kilométrage	Le kilométrage de l'évolution	Long	
	Date	Date d'évolution	Date	
	Source	Source d'évolution	Chaine de caractère	50
	Id_source	Identifiant de la source	Long	
Maintenance	Id	identifiant	Long	
	Code	Code de la maintenance	Chaine de caractère	50
	dateDemande	Date de demande maintenance	Date	
	dateDébut	Date début de maintenance	Date	
	datePrévisionnelle	Date Prévisionnelle de fin	Date	

		de maintenance		
	dateFin	Date de fin de maintenance	Date	
	Type	Type de maintenance	Chaine de caractère	50
	Montant	Montant de maintenance		
	Objet	Objet de la maintenance	Chaine de caractère	50
	Etat	Etat de maintenance	Chaine de caractère	50
	Index	Index kilométrique après la maintenance	Long	
Sinistre	Id	identifiant	Long	
	Code	Code du sinistre	Chaine de caractère	50
	descS	Description du sinistre	Chaine de caractère	150
	descD	Description des dégat	Chaine de caractère	150
	Expert	Expert du sinistre	Chaine de caractère	50
	dateEx	Date d'expertise	Date	
	montantRembourse	Montant remboursé par l'assurance	BigDécimal	
	montantSinistre	Montant causer par le sinistre	BigDécimal	
	typeS	Type du sinistre	Chaine de caractère	50
	typeD	Type des dégats	Chaine de caractère	50

Tableau 7 Dictionnaire des données

- **Règles de gestion**

1. On peut affecter plusieurs véhicules à un opérateur mais dans des dates différentes.
2. Un véhicule a plusieurs caractéristiques techniques.
3. Un véhicule appartient un seul modèle, et le modèle peut contenir plusieurs véhicules.
4. Un modèle appartient une seule marque et une seule catégorie, et chaque marque et chaque catégorie peut contenir plusieurs modèles.
5. Un véhicule a une seule classification.
6. Chaque maintenance concerne un seul véhicule à la fois et ce dernier peut nécessiter plusieurs maintenances.
7. Une évolution kilométrique concerne un seul véhicule à la foi, et le véhicule peut avoir plusieurs.
8. Plusieurs sinistres peuvent arriver à un seul véhicule.
9. Une unité organisationnelle a plusieurs opérateurs, et chaque opérateur appartient à une seule unité.

10. Chaque unité organisationnelle a une seule unité parente.
11. Chaque catégorie a une seule catégorie parente.
12. La disponibilité d'un véhicule peut être effectuée sur un seul véhicule à la foi.

- **Le modèle relationnel**

Véhicule(id,code,dateMiseEnService,immatriculation,kilométrage, consommationMoyenne, numSérie, #id_modèle, #id_classification,#id_typeCarburant)
Opérateur(id, code,nom, prénom, #id_unitéOrganisationnelle**)**
Affectation(#id_véhicule,#id_opérateur, dateDébut**)**
UnitéOrganisationnelle(id,code,adresse,codePostal,email,fax,télé,siteWeb,#id_unitéParente**)**
Modèle(id, code, libelle, #id_marque, #id_catégorie**)**
Marque(id,code,libelle**)**
Catégorie(id,code,libelle,#id_catégorieParente**)**
CaractéristiqueTéchnique(id,code,libelle**)**
VheCarac (#id_véhicule,#id_caractéristiqueTéchnique,valeur,observation**)**
Classification(id,code,libelle**)**
Maintenance(id, code, dateDemande, dateDébut, datePrévisionnelle, dateFin,type, montant, objet, état, indexKilometriqueMaintenance, #id_véhicule**)**
DisponibilitéVhe(id, dateDébut, dateFin, état, #id_véhicule**)**
HistoriqueEvolutionKilometrique(id, kilométrage, dateevolution, source,#id_véhicule**)**
Sinistre(id, code, descS, descD, expert, dateEx, montantRembourse, montantSinitre, typeS, typeD, #id_véhicule**)**
TypeCarburant(id, code, désignation, prix**)**

6. Conclusion

Dans ce chapitre,nous avons mis en avant les phases nécessaires à la réalisation de notre application. Une conception détaillée des différents modules est réalisée, par le langage UML et ses différents diagrammes (cas d'utilisations, classes, séquence,..).

Dans ce qui va suivre, nous entamerons l'implémentation de notre projet.

Chapitre 6 Implémentation

Implémentation

1. Introduction

Cette partie constitue le dernier volet de ce rapport. Après avoir terminé la phase despécification et conception, la solution étant déjà choisie et étudiée, il ne reste que de se décider dans quel environnement nous allons travailler, exposer les choix techniques utilisés et les langages adoptés, et enfin présenter quelques interfaces graphiques, qui servent à expliquer le fonctionnement de notre application.

2. Choix technique

2.1 Java

Java est un langage de programmation orienté objet, développé par Sun Microsystems et destiné à fonctionner dans une machine virtuelle, il permet de créer des logiciels compatibles avec des nombreux systèmes d'exploitation.

Java et non seulement un langage de programmation puissant et international, mais aussi un environnement de développement qui est continuellement étendu pour fournir des nouvelles caractéristiques et des bibliothèques permettant de gérer de manière élégante des problèmes traditionnellement complexes dans les langages de programmation classiques, tels que le multithreading, les accès aux bases des données, la programmation réseau, l'informatique répartie. En plus java est considéré comme un langage adaptable à plusieurs domaines puisqu'une application web implémentée par celle-ci peut avoir des extensions ou des modifications dans le futur. De plus, java permet de réduire le temps de développement d'une application grâce à la réutilisation du code développé.[11]

On a utilisé ce dernier car il est très connue et donc la disponibilité de documentation, et il permet de mieux structurer une application JEE en assurant l'indépendance entre présentation (jsp ou jsf) et code (service,contrôleur, bean...).

2.2 XML

L'eXtensible Markup Language (langage de balisage extensible) est un langage permettant le balisagededocuments,sertàenregistrerdesdonnéestextuellesetpermetdefaciliterl'éch

ange d'information sur l'internet.[w13]

XML est utilisé pour faire les différentes configurations dans les applications d'entreprise.

3. Outils matériels

La réalisation de notre projet a été effectuée sur deux (02) ordinateurs portables :

- HP ProBook 650 G2 doté de la configuration suivante :
- Intel(R) Core(TM) i5-6200UCPU @ 2.30 GHZ.
- 8.00 Go de RAM.
- Système d'exploitation Windows 10 64bits.
- Dell Inspiron 15 doté de la configuration suivante :
- Intel(R) Core(TM) i5-3337UCPU @ 1.80 GHZ.
- 6.00 Go de RAM.
- Système d'exploitation Windows 7 64bits.

4. Environnement de développement

4.1 Technologies de développement

- **Spring Framework**

  Spring Framework est une plate-forme Java qui fournit une prise en charge complète de l'infrastructure pour le développement d'applications Java. Spring gère l'infrastructure afin que vous puissiez vous concentrer sur votre application.

- **Hibernate**

  Hibernate est un framework open source gérant la persistance des objets en base de données relationnelle.

Hibernate est adaptable en termes d'architecture, il peut donc être utilisé aussi bien dans un développement client lourd, que dans un environnement web léger ou dans un environnement Java EE complet.

Hibernate apporte une solution aux problèmes d'adaptation entre le paradigme objet et les SGBD en remplaçant les accès à la base de données par des appels à des méthodes objet de haut niveau.

- **Thymeleaf**

Thymeleaf est un moteur de template Java moderne côté serveur pour les environnements Web et autonomes.

Avec des modules pour Spring Framework, une multitude d'intégrations avec nos outils préférés et la possibilité de connecter vos propres fonctionnalités, Thymeleaf est idéal pour le développement Web HTML5 JVM moderne - bien qu'il puisse faire beaucoup plus.

5. Outils Logiciels

5.1 SpringToolSuite

STS est l'outil IDE proposé par SpringSource pour les développeurs d'applications Spring. Il comprend un certain nombre de Plug-ins, qui prennent en charge le développement d'applications basées sur Spring, ainsi que s'intègrent à d'autres plug-ins Eclipse (tels que m2e, AspectJ Développement Tool, etc.).

Figure 54 SpringToolSuite

5.2 NetBeans

NetBeans IDE vous permet de développer rapidement et facilement des applications Java de bureau, mobiles et Web, ainsi que des applications HTML5 avec HTML, JavaScript et CSS. L'EDI fournit également un excellent ensemble d'outils pour les développeurs PHP et C / C ++. Il est gratuit et open source et possède une large communauté d'utilisateurs et de développeurs à travers le monde.

5.3 PostgreSQL

PostgreSQL est un système de gestion de base de données Objet-Relationnel(SGBDOR) open source, il prend en charge presque toutes les fonctionnalités de base de données relationnelles et offre d'autres fonctionnalités absentes dans d'autres moteurs SGBDR. Ce système est concurrent à d'autres systèmes de gestion de base de données, qu'ils soient libres (comme MySQL et Firebrid), ou propriétaires (comme Oracle, Sybase, DB2 et Microsoft SQL Server). [w49] La version choisie dans notre projet est 9.5.

Figure 55 PostgreSQL

6. L'application de la feuille de route

Dans cette partie du chapitre on va appliquer la feuille de route tracée dans le chapitre 4 sur le module véhicule et engins du système GPARC.

6.1 Etude de la complexité

On va étudier la complexité selon les facteurs mentionnés dans le chapitre 4. Alors voilà ce qu'on peut dire sur notre projet et notre stage :

- Manque de documentation, code compliqué difficile à comprendre.

- Dépendance des beans entité.
- Forte dépendance organisationnelle d'EJB et de Java EE, Spring n'est pas encore adopté.
- Utilisation intensive des technologies spécifiques des EJBs.
- Un grand projet qui contient beaucoup de classes avec grande dépendance entre eux.
- L'utilisation de la technologie JSF dans le coté front, JSF est déconseillée parce qu'elle n'est pas compatible avec Spring.

Donc on est face d'un grand projet d'une forte complexité.

6.2 Choix de type de la migration

Vu qu'on a un grand projet d'une forte complexité, on est obligé de faire une migration partielle en gardant les pages JSF du coté font et les intégrer dans notre projet Spring et faire la migration pour le coté back-end.

Mais le problème qu'on a trouvé c'est qu'on ne peut exécuter les pages JSF en dehors de l'entreprise ELIT, parce que les ressources utilisée dans ces pages sont stockés dans le nexus du serveur de l'entreprise, donc ces pages dépend du serveur dont on n'a pas l'accès à cause de la pandémie et des raisons de sécurité. Donc on a décidé de faire la migration pour une partie du front-end en développant quelque page nécessaire en utilisant Thymeleaf pour prouver la fiabilité de notre travail.

6.3 Processus de la migration

On a essayé d'appliquer toute les méthodes qu'on a mentionné dans le chapitre 4. Donc après la création d'un nouveau projet SpringBootStarter, on ajouter les dépendances des différents APIs qu'on a besoins dans le fichier POM.xml montré dans la figure

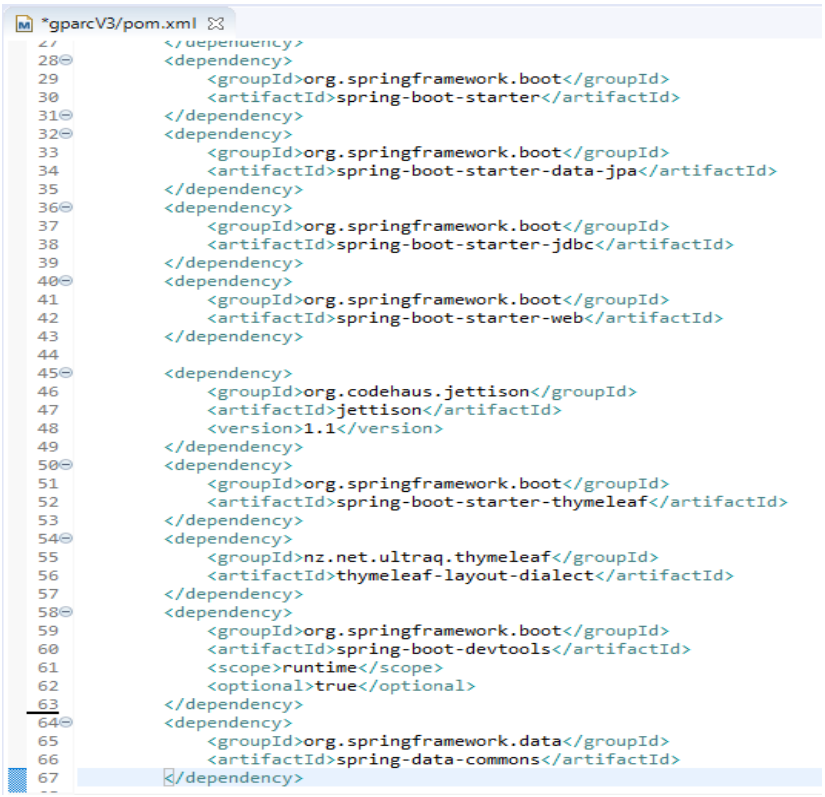


Figure 56 fichier POM.xml

Ensuite, On a ajouté les entités dans qui ne posent aucun problème lors de la migration. Après cela on a traité la couche d'accès aux données, et dans cette dernière on a plusieurs façon pour la migrer vers Springcomme on a vu dans le chapitre 4, donc on appliquer toutes ces méthodes dans notre prototype.

Les deux figures suivantes montrent un exemple de la première méthode de garder les classes DAO du projet JEE existant en appliquant les modifications nécessaires qu'on a mentionné dans la feuille de route dans le chapitre 4tel que l'héritage d'un interface et l'ajout des différentes annotation.

La figure 57 montre la classe d'accès aux données de l'objet VheVehiculedans projet existant JEE avant la migration.

```
1  ...5 lines
6  package dz.elit.gparc.vehicule.service;
7
8  import ...45 lines
53
54 /**...4 lines */
58 @Stateless
59 public class VheVehiculeFacade extends AbstractFacade<VheVehicule> {
60
61     @PersistenceContext(unitName = "Gparc-ejbPU")
62     private EntityManager em;
63     @EJB
64     EngEngagementFacade engEngagementFacade;
65     @EJB
66     private VheHistoriqueAffectationFacade vheAffectationVheOperFacade;
67     @EJB
68     private CrbTypeCarburantFacade typeCarburantFacade;
69     @EJB
70     private MaintCompteurGammeMaintenanceVehiculeFacade maintCompteurGammeMaintenanceVehiculeFacade;
71     private List<VehiculeMaintenancePreventive> listVehiculeMaintenancePreventives;
72     @EJB
73     private CrbMouvementCarburantFacade mouvementCarburantFacade;
74     @EJB
75     private VheArchiveKilometriqueFacade archiveKilometriqueFacade;
76     @EJB
77     private VheHistoriqueEvolutionKilometriqueFacade historiqueEvolutionKilometriqueFacade;
78     @EJB
79     private VheDocumentFacade documentFacade;
80     @EJB
81     private AdministrateurSingleton administrateurSingleton;
82     @Override
83     protected EntityManager getEntityManager() {
84         return em;
85     }
86
87     public VheVehiculeFacade() {
88         super(VheVehicule.class);
89     }
90
91     public List<VheVehicule> findListVheControleTechniqueExpirer(TypeEngagement type) {
92         List<VheVehicule> result = new ArrayList<>();
93         Query query = em.createQuery("Select  lgEng.vehicule ,MAX(lgEng.engagement.dateFin) from VheV
94         query.setParameter("type", type);
95         query.setParameter("date", new Date());
```

Figure 57 Classe DAO JEE

La figure 58 montre cette classe après la migration vers Spring :

```
61 @Repository
62 @Transactional
63 public class VheVehiculeFacade extends AbstractFacade<VheVehicule>implements VheVehiculeFacaderepo {
64
65     @PersistenceContext(unitName = "Gparc-ejbPU")
66     private EntityManager em;
67     @Autowired
68     EngEngagementFacade engEngagementFacade;
69     @Autowired
70     private VheHistoriqueAffectationFacade vheAffectationVheOperFacade;
71     @Autowired
72     private CrbTypeCarburantFacade typeCarburantFacade;
73     @Autowired
74     private MaintCompteurGammeMaintenanceVehiculeFacade maintCompteurGammeMaintenanceVehiculeFacade;
75     private List<VehiculeMaintenancePreventive> listVehiculeMaintenancePreventives;
76     @Autowired
77     private CrbMouvementCarburantFacade mouvementCarburantFacade;
78     @Autowired
79     private VheArchiveKilometriqueFacade archiveKilometriqueFacade;
80     @Autowired
81     private VheHistoriqueEvolutionKilometriqueFacade historiqueEvolutionKilometriqueFacade;
82     @Autowired
83     private VheDocumentFacade documentFacade;
84     @Autowired
85     private AdministrateurSingleton administrateurSingleton;
86     @Override
87     protected EntityManager getEntityManager() {
88         return em;
89     }
90
91     public VheVehiculeFacade() {
92         super(VheVehicule.class);
93     }
94
95     public List<VheVehicule> findListVheControleTechniqueExpirer(TypeEngagement type) {
96         List<VheVehicule> result = new ArrayList<>();
```

Figure 58 Classe DAO Spring

On peut facilementremarquer la différence entre les deux classes, pour l'interface implémentée c'est une juste une interface vide ms on doit faire ça sinon on va rencontrer une erreur lors de l'utilisation d'une instance de cette classe. C'est une règle de Spring d'implémenter une interface pour qu'il peut reconnaisse le bean lors de l'injection.

On appliquer aussi la deuxième méthode d'accès au donnée, celle qui utilise les interfaces générique sur quelque classes. La figure 59 représente l'application de cette méthode pour l'entité VheModele comme un exemplaire.

```java
@Repository
public interface VheModeleRepository extends JpaRepository<VheModele, Long> {

    public List<VheModele> findByCodeAndLibelle(String code,String libelle);

    public VheModele findByCode(String code);

    @Query("SELECT d FROM VheModele d WHERE d.marque.id = :idmarque")
    public List<VheModele> findAllByMarque(Long idmarque);

    @Query("SELECT d FROM VheModele d WHERE d.libelle = :libelle")
    public List<VheModele> findAllByLibelle(String libelle);

    public boolean existsByCode(String code);

    public boolean existsByLibelle(String libelle);

}
```

Figure 59 VheModeleRepository

Comme il est mentionné dans la feuille de route, les différentes méthode de cette interface vont etre utilisées dans une classe service, la figure 60 montre cette classe.

```java
@Service
public class VheModeleService {
@Autowired
VheModeleRepository vmr;

public void sauvgarder(VheModele modele) {
    vmr.save(modele);}

public void supprimer(VheModele modele) {
    vmr.delete(modele);}

public void suppById(Long id) {
    vmr.deleteById(id);}

public List<VheModele> getAllModele(){
    return vmr.findAll();}

public List<VheModele> getByParam(String code,String libelle){
    return vmr.findByCodeAndLibelle(code, libelle);}

public List<VheModele> getAllByMarque(Long idmarque){
    return vmr.findAllByMarque(idmarque);}

public List<VheModele> getAllByLibelle(String libelle){
    return vmr.findAllByLibelle(libelle);}

public VheModele getBycode(String code) {
    return vmr.findByCode(code);}

public Optional<VheModele> getById(Long id) {
    return vmr.findById(id);}

public boolean exitebylibelle(String libelle) {
    return vmr.existsByLibelle(libelle);}

public boolean exitebycode(String code) {
    return vmr.existsByCode(code);}
}
```

Figure 60 VheModeleService classe

Il est préférable de créer les classes DAO de cette façon parce qu'elle très efficace, on peut remarqué facilement que cette façon nous offre plus de méthode(JPARepository contient plusieurs méthodes plus les méthodes personnalisées qu'on a ajoutées) et mois de

code(presque 50 ligne seulement) par rapports la façon clasique(5 méthode seulement et presque 90 ligne de code) montré dans la figure 61.

```
@Stateless
public class VheMarqueFacade extends AbstractFacade<VheMarque> {

    @PersistenceContext(unitName = "Gparc-ejbPU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public VheMarqueFacade() {
        super(VheMarque.class);
    }

    public List<VheMarque> findByCodesDesignationDesactive(String codeMarque, String libelleMarque, Boolean desactive) {
        StringBuilder queryStringBuilder = new StringBuilder("SELECT clt FROM VheMarque AS clt WHERE 1=1 ");
        if (codeMarque != null && !codeMarque.equals("")) {
            queryStringBuilder.append(" AND  UPPER(clt.codeMarque) like :codeMarque ");
        }

        if (libelleMarque != null && !libelleMarque.equals("")) {
            queryStringBuilder.append(" AND  UPPER(clt.libelleMarque) like :libelleMarque ");
        }
        if (desactive != null) {
            queryStringBuilder.append(" AND  clt.desactive = :desactive ");
        }

        queryStringBuilder.append(" ORDER BY clt.codeMarque ");

        Query q = em.createQuery(queryStringBuilder.toString());

        if (codeMarque != null && !codeMarque.equals("")) {
            q.setParameter("codeMarque", codeMarque.toUpperCase() + "%");
        }

        if (libelleMarque != null && !libelleMarque.equals("")) {
            q.setParameter("libelleMarque", libelleMarque.toUpperCase() + "%");
        }
        if (desactive != null) {
            q.setParameter("desactive", desactive);
        }
        return q.getResultList();
    }
```

Figure 61 VheModele DAO classe

Pour la configuration de la persistance de l'existant projet était dans un fichier persistence.xml montré dans la figure 62.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">


    <persistence-unit name="Gparc-ejbPU" transaction-type="JTA">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <jta-data-source>jdbc/GparcJNDI</jta-data-source>
        <exclude-unlisted-classes>false</exclude-unlisted-classes>
        <properties>
            <property name="eclipselink.logging.level.sql" value="FINE"/>
            <property name="eclipselink.logging.parameters" value="true"/>
        </properties>
    </persistence-unit>
```

Figure 62 Fichier Persistence.xml

Dans notre projet on a préféré de la faire on utilisant une classes java qui est montré dans la figure 63.

```java
@Configuration
@EnableTransactionManagement
public class GparcConfig {
        @Bean
        public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
           LocalContainerEntityManagerFactoryBean em
             = new LocalContainerEntityManagerFactoryBean();
           em.setDataSource(dataSource());
           em.setPackagesToScan(new String[] { "dz.elit.gparc.administration.entity", "dz.elit.gparc.carburant.entity",
                     "dz.elit.gparc.dba.entity", "dz.elit.gparc.engagement.entity", "dz.elit.gparc.maintenance.entity",
                     "dz.elit.gparc.mvt.entity", "dz.elit.gparc.operateur.entity", "dz.elit.gparc.vehicule.entity" });
           JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
           em.setJpaVendorAdapter(vendorAdapter);
           em.setJpaProperties(additionalProperties());
           em.setPersistenceUnitName("Gparc-ejbPU");
           return em;
        }
        @Bean
        public DataSource dataSource(){
            DriverManagerDataSource dataSource = new DriverManagerDataSource();
            dataSource.setDriverClassName("org.postgresql.Driver");
            dataSource.setUrl("jdbc:postgresql://localhost:5432/bd_gparc");
            dataSource.setUsername( "postgres" );
            dataSource.setPassword( "postgres" );
            return dataSource;
        }
        @Bean
        public PlatformTransactionManager transactionManager() {
            JpaTransactionManager transactionManager = new JpaTransactionManager();
            transactionManager.setEntityManagerFactory(entityManagerFactory().getObject());
            return transactionManager;
        }
        @Bean
        public PersistenceExceptionTranslationPostProcessor exceptionTranslation(){
            return new PersistenceExceptionTranslationPostProcessor();
        }
        Properties additionalProperties() {
            Properties properties = new Properties();
            properties.setProperty("hibernate.hbm2ddl.auto", "update");
            properties.setProperty("hibernate.dialect", "org.hibernate.dialect.PostgreSQLDialect");
            return properties;
        } }
```

Figure 63 Persistence Config classe

Enfin on fini la configuration nécessaires avec le fichier Application.properties montré dans la figure 64.

```
# ===============================
# = DATA SOURCE
# ===============================
# Set here configurations for the database connection
spring.datasource.url=jdbc:postgresql://localhost:5432/bd_gparc
spring.datasource.username=postgres
spring.datasource.password=postgres
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.testWhileIdle=true
spring.datasource.validationQuery=SELECT 1
spring.jpa.show-sql=true
spring.jpa.hibernate.naming.implicit-strategy=org.hibernate.boot.model.naming.ImplicitNamingStrategyLegacyHbmImpl
spring.jpa.hibernate.naming.physical-strategy=org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Figure 64Application.properties

Concernant la couche vue et les contrôleurs. A cause des problèmes mentionnés dans le chapitre 4 qu'on a rencontrés nous avons développé des nouvelles pages de vues en utilisant thymeleaf et on a créé des nouveaux contrôleurs.

Dans ce qui suit on va présenter le système développé avec des captures des fonctionnalités principales.

7. Présentation du système développé :

Le lancement de notre application commence avec une interface d'authentification montré dans la figure 65.

Figure 65 Login

Après l'authentification, l'utilisateur va être orienté vers l'interface du module véhicules et engins, cette interface contient la liste des véhicules avec des différentes actions d'ajout ou de modification ou suppression, et on voit à la gauche le menu des différents modules.

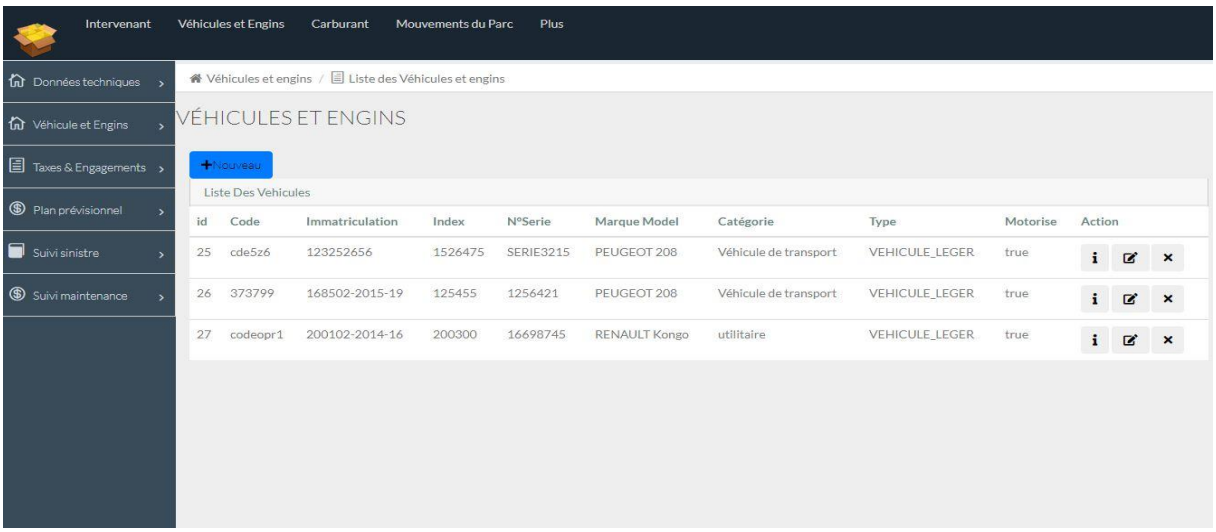


Figure 66 Véhicules Et Engins

L'ajout d'un nouveau véhicule se fait en cliquant sur le bouton nouveau qui nous nous envoie le formulaire d'ajout montré dans la figure.

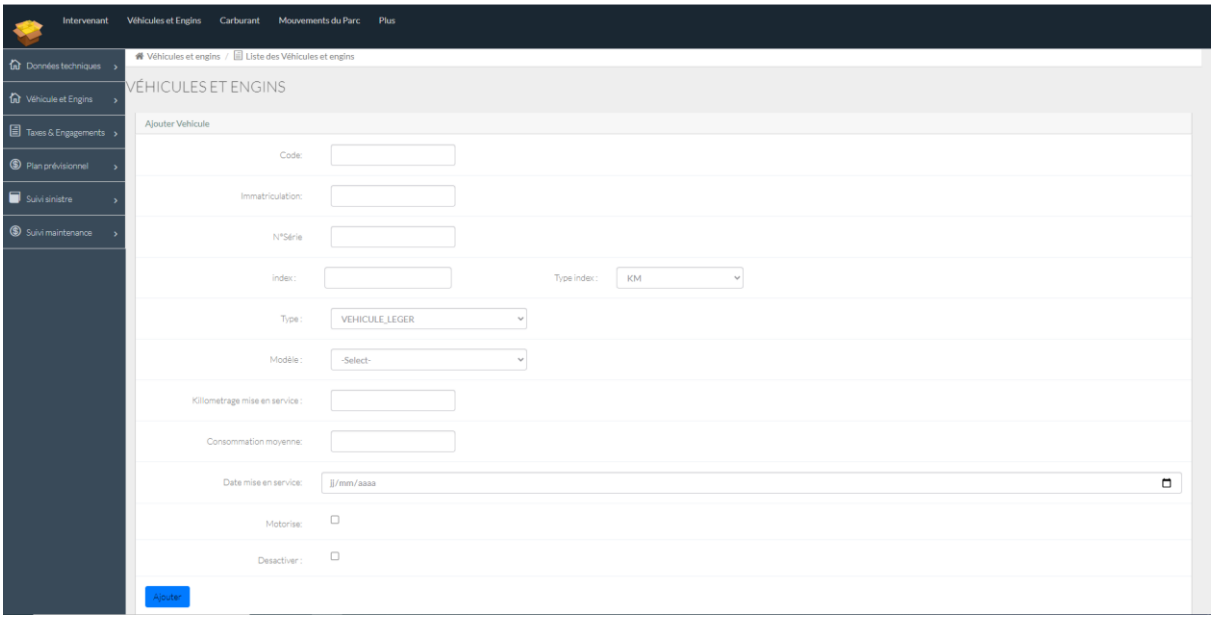
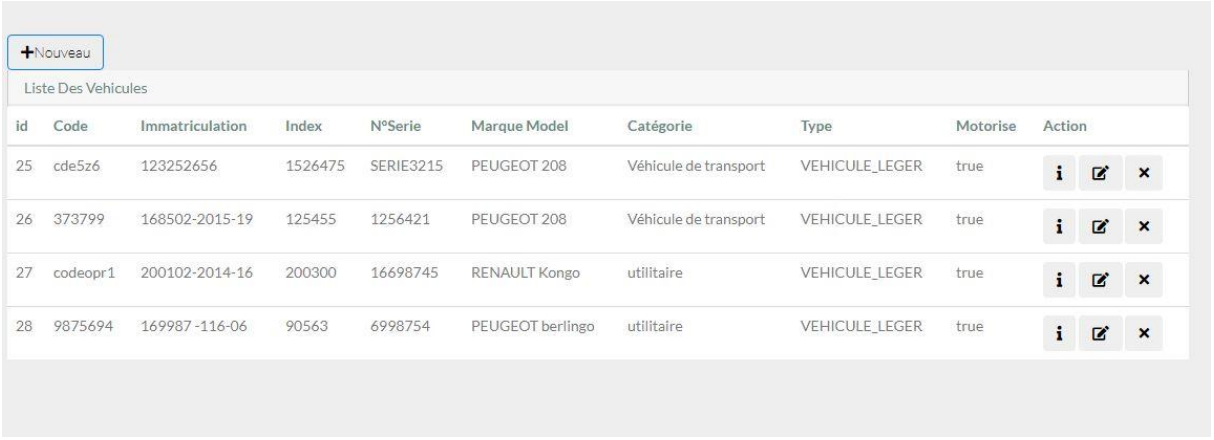


Figure 67 Ajouter Véhicule

Apres l'ajout nous retournons vers notre page qui contient la liste des véhicules et on verra que notre véhicule a été correctement ajouter



id	Code	Immatriculation	Index	N°Serie	Marque Model	Catégorie	Type	Motorise	Action
25	cde5z6	123252656	1526475	SERIE3215	PEUGEOT 208	Véhicule de transport	VEHICULE_LEGER	true	
26	373799	168502-2015-19	125455	1256421	PEUGEOT 208	Véhicule de transport	VEHICULE_LEGER	true	
27	codeopr1	200102-2014-16	200300	16698745	RENAULT Kongo	utilitaire	VEHICULE_LEGER	true	
28	9875694	169987 -116-06	90563	6998754	PEUGEOT berlingo	utilitaire	VEHICULE_LEGER	true	

Figure 68 Apres Avoir ajouter un véhicule

On a aussi la possibilité de Consulter les Marques qu'on a

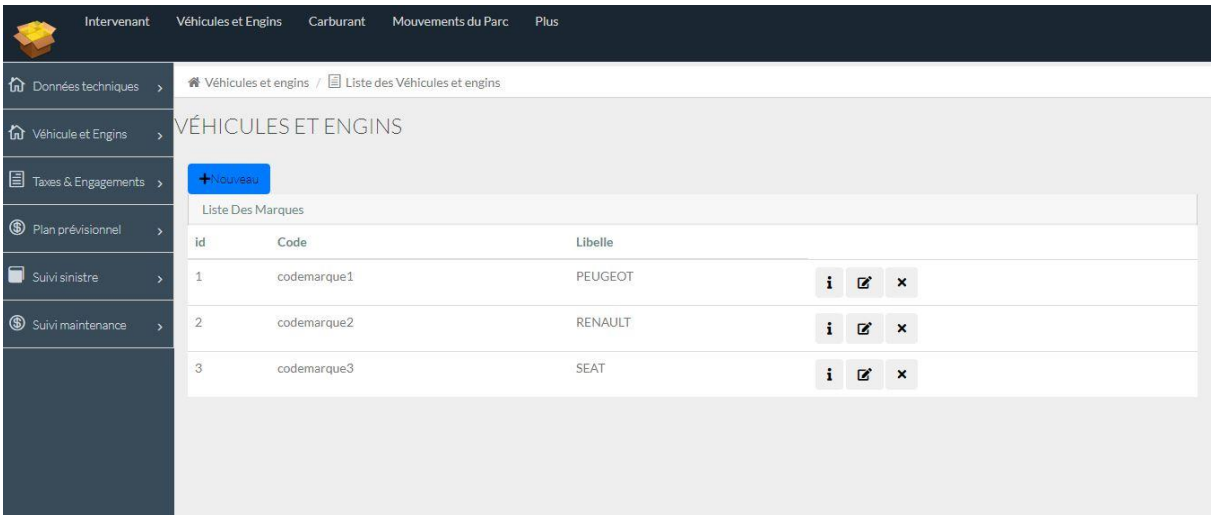


Figure 69 Consulter Marques

Ou bien ajouter une nouvelle d'ajouter des marques si elles n'existentpas.

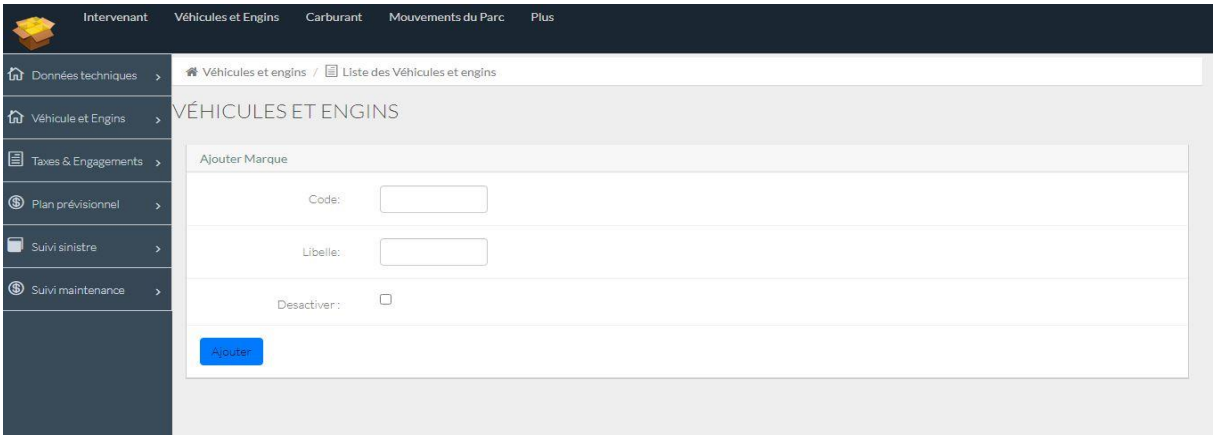


Figure 70 Ajouter Marque

Tout comme les marques on peut consulter les catégories en allons dans la page listecatégorie

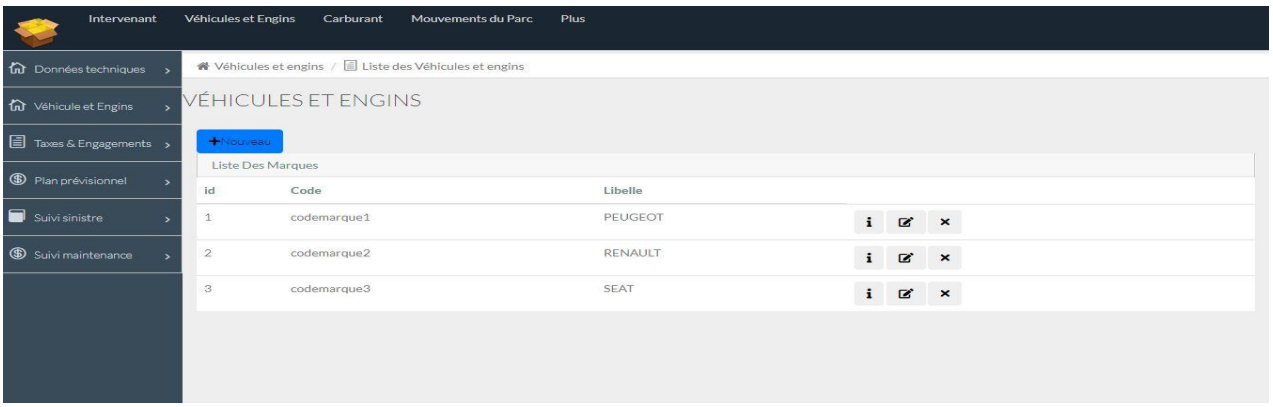


Figure 71 Liste Catégories

On peut aussi consulter les affectations.

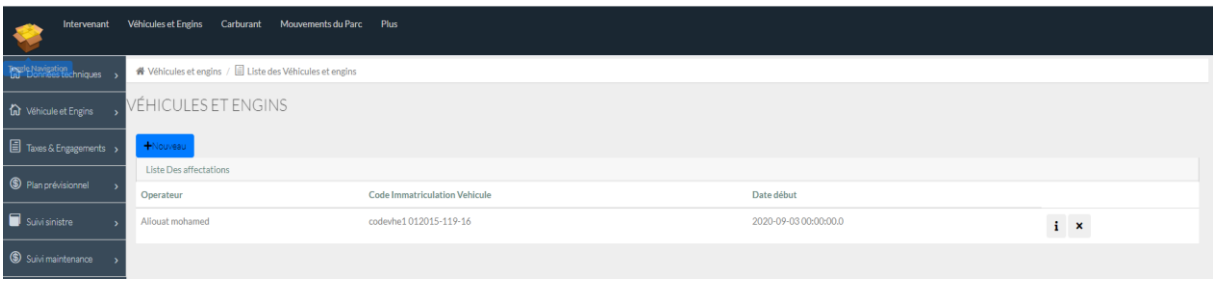


Figure 72 consulter Affectations

Et pour faire des affectations on sélection un véhicule et un opérateur est on fait notre affectation et la date du jour de l'affectation sera enregistrée.

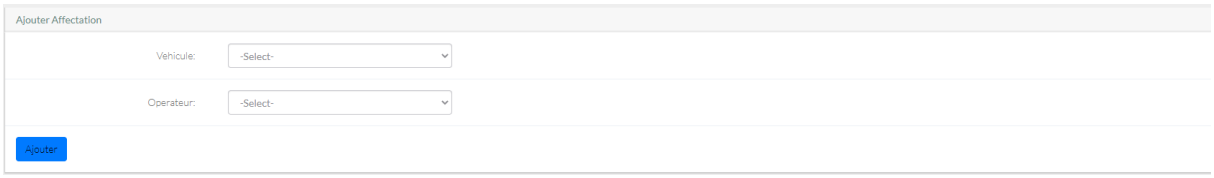
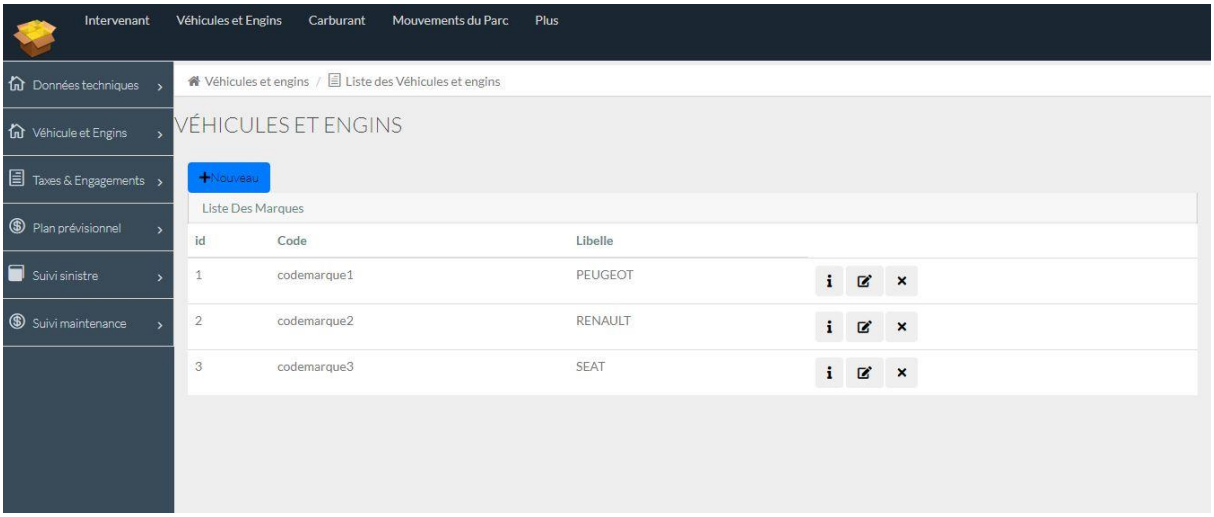


Figure 73 Ajouter Affectations

Enfin on a la fiche relevé compteur qui nous permet de contrôler le mouvement du véhicule après que l'opérateur est pris possession



7. Conclusion

Dans ce chapitre on a montré exactement toutes les technologies et outils utiliser pour exécuter correctement l'application ainsi que les principales fonctionnalités de notre application.

Avec ce chapitre on a pu conclure que la migration est faisable et a été correctement effectuer.

Conclusion Générale

Durant notre travail de recherche pour faire la migration des données d'un langage très utiliser qui est JEE a un autre qui était à la base son framework mais qui c'est tellement développer qu'il est devenu presque un langage à lui tout seul.

Notre méthodologie consiste à apprendre les deux langages pour pouvoir comprendre notre code, qui est à la base était en JEE, et ça en lui-même pose un problème car quand il s'agit d'un très grand projet il est très difficile de distingué les classes et les méthodes de notre projet, du coup il faut une parfaite compréhension de notre projet en JEE de préférence être toujours en contact avec un des ingénieurs qui à développer cette application pour pouvoir avoir des informations.

Mais malgré tout ça y aura toujours des points qu'il voudrait mieux les développer depuis le début car elles seront très difficiles à reproduire si on le voulait.

Et du côté du front-end ce n'est pas un très grand problème on peut le garder comme il est, qui est généralement avec Jsf et Primefaces, il y a juste quelques paramètres à régler pour que Spring puisse les exécuter directement, et bien sur le synchronisé avec les contrôleurs, ou bien on peut utiliser Thymeleaf qui est très pratique avec spring du coup si on veut changer de front vaut mieux opter pour Thymeleaf.

Enfin durant toute cette recherche on a pu conclure que cette migration est faisable et très pratique et que Spring en lui-même facilite le travail du développeur.

Bibliographie

[1] Daniel Rubio, Josh Long Marten Deinum, *Spring 5 Recipes A Problem-Solution Approach*, 4th ed. California, états-unis: Apress, 2017.

[2] doudoux jean-michel, *Développons en Java*, 1st ed. Luxembourg: Oxiane, 2009.

[3] Joseph Gabay, *Merise et UML pour la modélisation des systèmes d'information*, 1st ed. Paris, France: Dunod, 2004.

[4] Lafosse Jérôme, *Java EE :guide de développement d'application web en java*, 1st ed. Saint Herblain, France: Editions ENI, 2009.

[5] Lonchamp Jaques, *Conception d'applications en Java/JEE : Principes, patterns et architecture*, 1st ed. Paris, France: Dunod, 2014.

[6] Colin Sampaleanu. (2008, octobre) Spring source: a guide to migrating entreprise application to spring. Document.

[7] Kayal Dhrubojyoti, *Pro Java EE Spring Patterns: Best Practices and Design Strategies Implementing Java EE Patterns with the Spring Framework*, 1st ed. New York, états-unis: Apress, 2008.

[8] Josh Juneau, *Java EE 8 Recipes A Problem-Solution Approach*, 2nd ed. Illinois, états-unis: Apress, 2018.

[9] Gosling James et Holmes David Ken Arnold, *Le langage Java*, 3rd ed. Paris, France: Vuibert Informatique, 2001.

[10] Sanaa CHABANE, Lydia IZZA Amel BELMAKSENE, "Conception et développement d'une application Android "Guide de l'UMBB"," Université M'HAMED BOUGARA, Boumerdes, Rapport de master 2017.

[11] Hervé Le Morvan, *Java Spring : le socle technique des applications JEE*, 2nd ed. Saint Herblain, France: Editions ENI, 2018.

[12] Felipe Gutierrez, *Pro Spring Boot 2: An Authoritative Guide to Building Microservices, Web and Enterprise Applications, and Best Practices*, 2nd ed. New York, états-unis: Apress, 2018.

[13] Hacene abdel hafid Bouki, "Conception et Developpement d'une application Java sousAndroid(Donnerdusang)," Université Abou Bakr Belkaid, Telemcen, Mémoire de master 2015.

[14] Debrauwer Laurent Van Der Heyden Fien, *UML2:Initiation*, 2nd ed. Saint Herblain, France: Edition ENI, 2008.

[15] Pascal Roques, *UML 2 par pratique: Études de cas et exercices corrigés*, 5th ed. Paris, France: éditions Eyrolles, 2006.

[16] Nardone Massimo, Rathod Chirag et Kodali Raghu Wetherbee Jonathan, *Beginning EJB in Java EE 8*, 1st ed., Apress, Ed. Berkeley, états-unis, 2018.

[17] Bruce Snyder,Christian Dupuis, Sing Li, Anne Horton et Naveen Balani Thomas Van de Velde, *Beginning Spring Framework 2*, 1st ed. New Jersey, états-unis: Wiley, 2008.

[18] Acetozi Jorge, *Pro Java Clustering and Scalability*, 1st ed. New York, états-unis: Apress, 2017.

[19] Raible Matt, *Spring live*, 1st ed. Colorado, états-unis: SOURCEBEAT, 2004.

[w1] http : //www.sonelgaz.dz: description groupe sonelgaz : Janvier 2020.

[w2] http : //www.elit.dz: description elit: janvier 2020.

[w3]https://www.oracle.com:La technologie Java EE: mars 2020.

[w4]https://www.thePolyglotdevloper.com: évolution JEE: mars 2020.

[w5]https://www.supinfo.com:Modele MVC : mars 2020.

[w6]https://www.supinfo.com: Architecture des applications Jee: mars 2020.

[w7]https://fr.wikipedia.org: définition SpringToolSuite, PostgreSQL, EJB: aout 2020.

[w8]https://waytolearnx.com:Serveur web et serveur d'application: mars 2020.

[w9]https://jmdoudoux.developpez.com: Le framework Spring : avril 2020.

[w10]https://www.quickprogrammingtips.com: évolution de spring : avril 2020.

[w11]https://docs.spring.io: définition spring, évolution, architecture modulaire: avril 2020.

[w12]https://www.baeldung.com/:La persistance avec JPA: juillet 2020.

[w13]https://openclassrooms.com: définition xml: aout 2020.