



Dédicaces

A ma très chère mère,

Affable, honorable, aimable : Tu représentes pour moi le symbole de la bonté par excellence, la source de tendresse et l'exemple du dévouement qui n'a pas cessé de m'encourager et de prier pour moi.

A mon Père,

Aucune dédicace ne saurait exprimer l'amour, l'estime, le dévouement et le respect que j'ai toujours eu pour vous. Rien au monde ne vaut les efforts fournis jour et nuit pour mon éducation et mon bien être. Ce travail est le fruit de tes sacrifices que tu as consentis pour mon éducation et ma formation.

A mon idole Ahmed BEN MAHMOUD,

Je te suis profondément reconnaissant pour ce que tu as fait pour moi. Une chose est sûre : je n'oublierais jamais.

A tous les membres de ma famille, petits et grands,

Veillez trouver dans ce modeste travail l'expression de mon affection.

À tous mes amis,

En Souvenir des plus beaux instants qu'on a passé ensemble.

Aussi bien à tous ceux qui m'ont aidé,

Merci

Oussama HASNI



Remerciements

Parce que nous avons beaucoup estimé ceux qui nous ont écoutés, conseillés, critiqués, encadrés, je tiens à leur faire-part de toute ma gratitude.

Avant de présenter mon projet, je tiens à exprimer mes profonds sentiments de gratitude et de reconnaissance pour mon encadrent Mr Marouene ARFAOUI, pour sa confiance placée en notre personne et son accompagnement enrichissant tout au long de ce projet.

Aussi, je tiens à exprimer ma gratitude envers les membres de mon équipe : Mme. Imen BEL Aazi, M. Helmi BARGAOUI et M. Seifeddine ADNANI, pour leurs soutiens, leurs écoutes et leurs conseils précieux qui m'ont aidé à accomplir ce projet dans une ambiance chaleureuse. Je remercie mon manager M. Khaled KSONTINI pour m'avoir très bien accueilli parmi eux.

Je tiens à remercier très chaleureusement Mme. Inès EL OUED qui m'a permis de bénéficier de son encadrement. Les conseils qu'elle m'a prodigués, la patience, la confiance qu'elle m'a témoignée ont été déterminants dans la réalisation de ce travail.

Un grand merci pour les membres du jury qui ont bien accepté d'évaluer ce modeste travail, je leur témoigne toute ma gratitude et mon profond respect. Enfin, je remercie tous les enseignants de L'ENSIT qui a contribué à ma formation universitaire et tous les membres de ma famille ainsi que mes amis qui n'ont cessé de m'encourager.



Table des matières

Introduction générale	1
1 Contexte du projet	3
Introduction	3
1.1 Présentation de l'organisme d'accueil	3
1.1.1 Sofrecom groupe	3
1.1.2 Secteur d'activités	4
1.1.3 Sofrecom Tunisie	5
1.1.4 Direction d'accueil	5
1.2 Contexte du projet	6
1.2.1 Justification du choix du sujet et motivations	6
1.2.2 Modèle de travail actuel	6
1.2.3 Problématique et objectifs du projet	7
1.3 Méthodologie de travail	8
1.3.1 Modèle de processus en Spirale	8
1.4 L'approche DevOps comme solution	10
1.4.1 L'approche DevOps	10
1.4.2 Livraison Continue	11
1.4.3 Gestion des versions	11
1.4.4 Intégration Continue	11
1.4.5 Automatisation des tests	12
1.4.6 Orchestration	13
1.4.7 Déploiement automatique	13
Conclusion	14

2	Analyse et spécification des besoins	15
	Introduction	15
2.1	Spécification des besoins	15
2.1.1	Acteurs	16
2.1.2	Besoins fonctionnels	16
2.1.3	Besoins non fonctionnels	18
2.2	Raffinement des cas d'utilisation	18
2.2.1	Cas d'utilisation : Gérer les projets de test	18
2.2.2	Cas d'utilisation : Gérer les comptes d'utilisateurs	20
2.2.3	Cas d'utilisation : Gérer les scénarios de test	22
2.2.4	Cas d'utilisation : Gérer les playbooks Ansible	24
2.2.5	Cas d'utilisation : Consulter les rapports d'audit	27
	Conclusion	29
3	Architecture logique et Conception	30
	Introduction	30
3.1	Architecture logique	30
3.2	Diagramme de classes	32
3.3	Diagrammes de séquence objet	34
3.3.1	Diagramme de séquence objet Authentification	34
3.3.2	Diagramme de séquence objet ajouter un utilisateur	35
3.3.3	Diagramme de séquence modifier un playbook Ansible	36
3.3.4	Diagramme de séquence objet consulter un rapport d'audit système	37
3.3.5	Diagramme d'activité	38
	Conclusion	40
4	Réalisation	41
	Introduction	41
4.1	Environnement de travail	41
4.1.1	Environnement matériel	41
4.1.2	Environnement logiciel	42
4.1.3	Choix techniques de réalisation	42
4.2	Étude comparative	44
4.2.1	Comparaison entre Ansible et Chef	44
4.2.2	Comparaison entre GitLab et SVN	46
4.3	Architecture logicielle globale	48

4.4	Aperçu sur le travail réalisé	49
4.4.1	Gestion des projets	49
4.5	Gestion des playbooks Ansible	51
4.5.1	Gestion des scénarios de test	53
4.5.2	Gestion des utilisateurs	55
	Conclusion	56
	Conclusion générale	57
	Nétographie	59
	Glossaire	61
	Annexe A : ZEBRA	62
A.1	Cadre de l'application	62
A.2	Vue fonctionnelle globale	62
A.3	Architecture globale de ZEBRA	64
	Annexe B : Ansible	67
B.1	Playbooks	67
B.2	Rôles	68
B.3	Variables	68
B.4	Inventaires	69
	Annexe C : Développement d'un module Ansible	70
C.1	Modules	70
C.2	System check	70
	Annexe D : Développement d'un plugin Ansible	74
D.1	Callback Plugin	74
D.2	JSON callback plugin	74
	Annexe E : Exemple d'un test système	77
E.1	Les tests système	77



Table des figures

1	Sofrecom dans le monde.	4
2	Secteurs d'activités de Sofrecom.	5
3	Les phases de déroulement du cycle en spirale.	9
4	Diagramme de cas d'utilisation global.	17
5	Cas d'utilisation « Gérer les projets de test ».	19
6	Diagramme de séquence système « Ajouter un projet ».	20
7	Cas d'utilisation « Gérer les comptes d'utilisateurs ».	21
8	Cas d'utilisation « Gérer les scénarios de test ».	23
9	Diagramme de séquence système « Exécuter un test sécurité ».	24
10	Cas d'utilisation « Gérer les playbooks Ansible ».	25
11	Diagramme de séquence système « Modifier les variables d'un playbook ». . .	27
12	Cas d'utilisation « Consulter les rapports d'audit ».	28
13	Architecture logique.	30
14	Patron de conception MVVM.	31
15	Architecture d'un projet MEAN.	32
16	Diagramme de classes.	33
17	Diagramme de séquence objet « Authentification ».	35
18	Diagramme de séquence objet « Ajouter un utilisateur ».	36
19	Diagramme de séquence objet « Modifier un playbook Ansible ».	37
20	Diagramme de séquence objet « Consulter un rapport d'audit système ».	38
21	Diagramme d'activité.	39
22	Popularité de Chef et Ansible.	46
23	Popularité de GitLab et SVN.	48

24	Architecture logicielle globale.	48
25	Interface de gestion des projets.	50
26	Interface d'ajout d'un projet.	50
27	Playbook de configuration du projet.	51
28	Interface de gestion d'un inventaire Ansible.	52
29	Interface de gestion des variables d'un playbook Ansible.	53
30	Rapport d'audit sécurité.	54
31	Courbe représentatif sur l'avancement des tests.	54
32	Interface de gestion des utilisateurs.	55
33	Interface de modification d'un profil.	56
34	Architecture fonctionnelle globale de ZEBRA	64
35	Marco Design de l'architecture ZEBRA	66
36	Un playbook Ansible.	67
37	Structure d'un rôle Ansible.	68
38	Structure d'un inventaire Ansible	69
39	Code source du module « System check ».	73
40	Code source du plugin « JSON Callback ».	76
41	Exemple d'un test système.	78



Liste des tableaux

1	Raffinement de cas d'utilisation « Ajouter un projet ».	19
2	Raffinement de cas d'utilisation « Modifier le compte d'un utilisateur ».	22
3	Raffinement de cas d'utilisation « Configurer un test ».	23
4	Raffinement de cas d'utilisation « Modifier les variables d'un playbook ».	26
5	Raffinement de cas d'utilisation « Consulter les rapports d'audit ».	28
6	Principales collections du schéma de la base de données.	34
7	Configuration de l'environnement matériel.	42
8	Comparaison entre Ansible et Chef.	45
9	Comparaison entre SVN et GitLab.	47



Introduction générale

De nos jours, toute entreprise doit œuvrer pour allier entre la minimisation du temps et du coût de production et le respect des exigences en termes de qualité et de sécurité.

Dans leur course pour augmenter le gain et la productivité, les entreprises ont tendance à mettre la pression sur leurs équipes pour accélérer leur livraison. Cependant, à ce rythme, la qualité des livrables se trouve généralement impactée. En effet, les équipes de test, sous pression, ne disposent pas toujours du temps nécessaire pour détecter toutes les anomalies d'un produit logiciel, ce qui augmente considérablement le taux d'échec du produit lors de sa mise sur le marché. Ceci nuit à l'image de l'entreprise qui risque de perdre sa crédibilité et ses clients.

Pour cela, les méthodologies ont évolué afin d'alléger le cycle des livraisons et augmenter le rendement des équipes.

C'est ainsi, qu'un nouveau mouvement appelé DevOps a vu le jour. Il prône une nouvelle culture basée sur la collaboration et le partage d'un objectif commun. Son application au sein des entreprises permet un gain de temps de livraison avec une qualité stable des produits livrés.

Beaucoup d'entreprises ont adopté cette nouvelle approche dont Sofrecom. C'est dans ce cadre que s'inscrit ce projet de fin d'études qui, en s'inspirant du mouvement DevOps, permettra de mettre en place un nouveau processus de validation des tests d'acceptation au sein du groupe Orange.

Ce travail vise à améliorer la performance de l'équipe projet en automatisant le processus de test et de validation.

Afin d'illustrer la démarche de notre travail, nous présentons dans ce qui suit l'organisation générale du présent rapport qui s'articule autour de quatre grands chapitres :

Dans le premier chapitre, nous nous focalisons sur la présentation de l'organisme d'accueil, ainsi que la mise en contexte du projet par la présentation de la problématique du sujet et la démarche adoptée tout au long de l'élaboration du projet.

Le deuxième chapitre est consacré à l'analyse et à la spécification des besoins où nous présentons la capture des besoins fonctionnels et non fonctionnels ainsi que les futurs acteurs de notre application.

Le troisième chapitre met le point sur la conception du système ainsi que la description détaillée de l'architecture logique choisit.

Le dernier chapitre est réservé à l'implémentation technique du système conçu. Au cours de ce chapitre nous argumentons nos différents choix techniques et nous présentons, par la suite, les tâches accomplies et la description du travail effectué à travers l'exposition des interfaces homme/machine.

Nous clôturons finalement ce rapport par une conclusion générale dans laquelle nous évaluons les résultats atteints et nous exposons les perspectives éventuelles du présent projet.

Chapitre

1

Contexte du projet

Introduction

Ce chapitre est réservé à la présentation du cadre général dans lequel nous avons réalisé notre travail. Nous commençons par une présentation de l'entreprise d'accueil puis nous introduisons les différents aspects de notre projet à savoir : le contexte, la problématique qu'il traite et l'objectif que nous voulons atteindre. Par la suite, nous étudions le modèle de travail actuel ainsi que ses limites. Enfin nous introduisons l'approche DevOps et ses avantages pour résoudre les problèmes rencontrés.

1.1 *Présentation de l'organisme d'accueil*

Afin de situer notre travail dans son environnement de réalisation, nous présentons dans cette section l'organisme d'accueil en exposant ses secteurs d'activité.

1.1.1 Sofrecom groupe

Sofrecom est une filiale du groupe Orange, développe depuis 50 ans un savoir-faire unique dans les métiers de l'opérateur, ce qui en fait un leader mondial du conseil et de l'ingénierie de télécommunication. Le groupe accompagne ses clients (opérateurs, industriels, institutions,

gouvernement) dans leurs projets de gestion, de création, de reprise ou de transformation en s'appuyant sur ses domaines d'expertise phares : les réseaux et services, les Solutions IT et le Conseil en entreprise, couvrant ainsi tous les métiers des opérateurs. Le groupe Sofrecom compte aujourd'hui plus de 1500 experts répartis sur 10 implantations à travers le monde, ce qui a permis d'accompagner plus de 200 opérateurs, gouvernements et bailleurs de fonds, dans une centaine de pays (voir Figure 1).

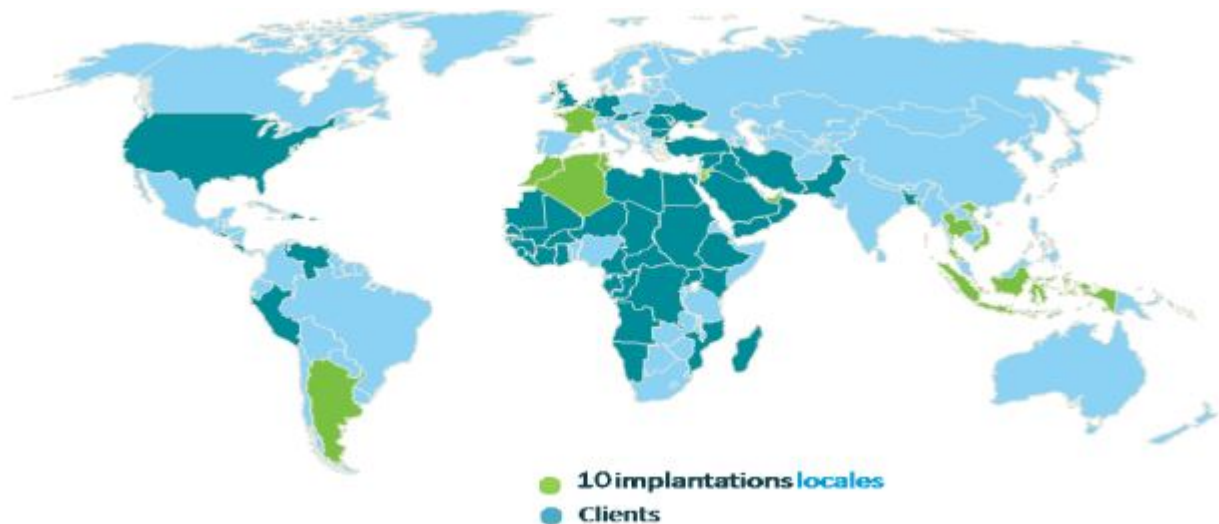


FIGURE 1. *Sofrecom dans le monde [1].*

1.1.2 Secteur d'activités

Sofrecom a pu confirmer son expérience dans plusieurs domaines d'expertise comme représenté dans la figure 2 :



FIGURE 2. *Secteurs d'activités de Sofrecom [1].*

1.1.3 Sofrecom Tunisie

Inaugurée en Octobre 2012, Sofrecom Tunisie est un acteur majeur du conseil et d'ingénierie en télécommunications sur le marché local. C'est la plus jeune et la plus importante filiale du groupe Sofrecom en zone Afrique et Moyen Orient. En 6 ans seulement, Sofrecom Tunisie compte déjà Plus de 500 Experts, elle a acquis sa force et sa crédibilité au travers de centaines de projets diversifiés à l'international, lui permettant d'avoir une longueur d'avance sur ses concurrents. Faisant partie du Groupe Orange, Sofrecom Tunisie bénéficie d'une renommée internationale, elle travaille pour deux clients majeurs du groupe Orange : DSI France et OLS. Elle connaît une très forte croissance depuis sa création avec plus de 500 collaborateurs en 2017 et une croissance continue en 2018 [1].

1.1.4 Direction d'accueil

Notre stage est effectué au sein du département OLT « Orange Labs Tunisie » Créé en novembre 2011, situé à mi-chemin entre l'Europe et l'Afrique, OLT est un acteur clé de l'innovation, du déploiement de produits et de services en Afrique et au Moyen-Orient. OLT

regroupe 160 experts, architectes, chefs de projet, ingénieurs système et intégration. Elle contribue sur des projets majeurs pour le groupe Orange en l'occurrence parmi lesquels nous citons : Cloud PRO, Orange Energy, Zebra, Orange Money, PROMISE, B-use, USSO, Orange Money, OOPRO [1].

1.2 Contexte du projet

Dans cette partie, nous mettons notre projet dans son contexte à travers un tour d'horizon sur le modèle de travail actuel. Ensuite, nous exposons la problématique et les motivations pour l'élaboration de ce projet.

1.2.1 Justification du choix du sujet et motivations

Notre travail a été élaboré au sein de l'équipe Zebra qui est chargée de veiller à la disponibilité de l'application Zebra. Il s'agit d'une plateforme de service qui permet le rechargement des comptes de téléphone mobile. Elle est déployée dans les filiales du groupe orange en Afrique, le moyen Orient, l'Amérique et l'Europe (voir Annexe A). L'équipe gère également les évolutions de la plateforme : upgrade, patch et migration. Elle prend en charge aussi la validation des tests de recettes de la plateforme.

Ce projet vise principalement à améliorer le rendement de l'équipe en automatisant les différents tests de validité. Il s'inscrit dans la politique de Sofrecom à introduire la culture DevOps dans les projets de ses collaborateurs pour faciliter leur travail. Ce projet nous a porté intérêt vu qu'il réunit plusieurs axes de la discipline informatique, à savoir : le système d'exploitation, la sécurité, la base de données, l'architecture et le réseau.

1.2.2 Modèle de travail actuel

La plateforme Zebra est déployée chez dix-neuf filiales dispatchées entre l'Afrique et le Europe de l'Est. Des réunions hebdomadaires sont organisées pour faire le suivi des projets de déploiement et apporter le support technique et managérial aux différentes filiales.

Le cycle de déploiement comporte une étape très importante qui nécessite beaucoup de temps et de rigueur pour pouvoir mettre en place un service fiable, sécurisé et conforme aux normes du groupe Orange, c'est l'acceptation technique. Cette étape est composée d'un nombre volumineux de vérifications exécuté manuellement ce qui impacte la durée de la mise en service.

Le processus d'acceptation commence par déployer toute nouvelle version de scripts de tests sur les serveurs cibles en respectant plusieurs paramètres de configuration tels que les adresses IP, les chemins d'exécution, les ports d'écoute mais aussi les permissions d'accès en lecture et en écriture. Vient ensuite la partie de récupération des rapports de tests. Il s'agit de parcourir tous les rapports afin de réaliser un rapport final qui résume le tout. Les tâches décrites précédemment s'exécutent sur chacun des serveurs cible un par un. Il s'agit de répéter le même travail sur une centaine de serveurs dans un temps bien déterminé.

Ce processus a rapidement atteint ses limites car les tests qu'il faut réaliser sont trop nombreux et répétitifs. De même, le processus est sujet à de nombreuses fautes humaines surtout dans la partie de validation. Nous pouvons facilement nous tromper et oublier de vérifier une configuration critique et primordiale. Par conséquent, cette tâche est devenue fastidieuse pour l'équipe. Si un problème est détecté au niveau fonctionnel, l'équipe doit déboguer les scripts pour détecter les sources d'erreurs. Donc, cette phase est énormément coûteuse en temps ce qui influe la productivité ainsi que d'autres services qui tournent en dépendance.

1.2.3 Problématique et objectifs du projet

Lorsque l'équipe Zebra reçoit un nouveau ticket d'acceptation technique dont elle se charge, un processus de travail est déclenché. Son objectif est d'arriver à exécuter cette tâche sur une centaine de serveurs en respectant les règlements et les configurations. Ce processus est composé par des étapes qui se répètent sur chaque serveur.

La méthodologie appliquée par l'équipe comporte deux problématiques majeures :

- Les itérations qui augmentent drastiquement le taux d'erreur rallongent les délais de traitement.

- Les tâches répétitives deviennent rapidement fastidieuses et engendrent une certaine monotonie dans l'équipe.

Au final, nous nous trouvons face à un modèle de travail très long et non stable. Notre objectif est de concevoir et mettre en place un système permettant de minimiser le cycle de travail tout en diminuant la probabilité d'erreur. Ce système doit automatiser les tâches répétitives en particulier les tests d'acceptation technique.

Les objectifs principaux peuvent ainsi se résumer en :

- La proposition et la mise en place d'une nouvelle méthodologie de travail.
- L'automatisation des tests d'acceptation technique.
- La génération des rapports d'audit avec les statistiques de chaque scénario de test.
- La traçabilité des exigences et des échanges avec les clients finaux.
- Réalisation d'un portail ayant des interfaces graphiques intuitives à la place de l'utilisation de la console.

1.3 Méthodologie de travail

Le succès d'un projet dépend de son adéquation au processus de développement choisi. Ce choix constitue une étape décisive pour l'élaboration d'une application indépendante de toute plateforme d'exécution et de tout langage de programmation.

1.3.1 Modèle de processus en Spirale

Ce type de projet nécessite l'implication de plusieurs acteurs puisque chaque module de test doit être accompagné et validé par un acteur différent.

Ainsi nous avons opté pour une approche caractérisée par un style de conduite de projets itératif, incrémental et adaptatif. Son action est centrée sur l'autonomie de ressources humaines impliquées dans la spécification et la validation des différents modules intégrés et testés en continu. C'est le modèle en spirale.

Dans le cadre de ce type de projet, chaque acteur élabore sa vision du produit à réaliser et spécifie les différentes fonctionnalités et exigences. Pour cette raison le projet doit être réparti en incréments afin de livrer un module après chaque incrément.

Le modèle en spirale met l'accent sur l'activité d'analyse des risques et chaque cycle de la spirale se déroule en quatre phases illustrées dans la figure 3.

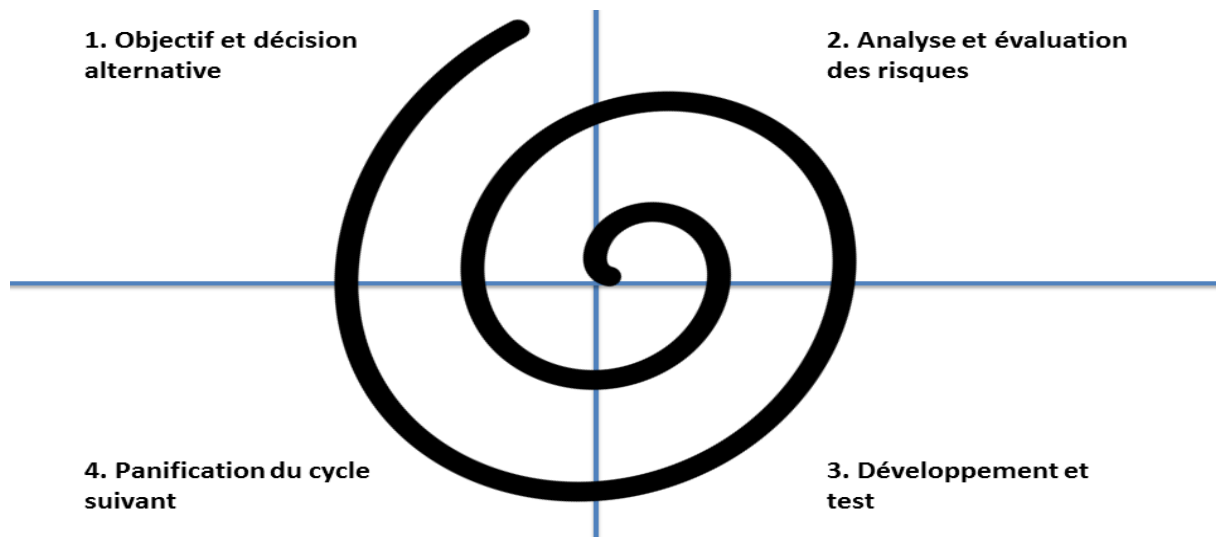


FIGURE 3. *Les phases de déroulement du cycle en spirale [2].*

Le modèle en spirale se caractérise par les cycles suivants :

- **Objectif et décision alternative** : les objectifs sont déterminés conjointement avec le client. Dans le même temps, les alternatives possibles sont discutées et les conditions cadres sont spécifiées (par exemple le système d'exploitation, l'environnement et langage de programmation).
- **Analyse et évaluation des risques** : les risques potentiels sont identifiés et évalués. Les alternatives en question sont également évaluées, tandis que les risques sont enregistrés, estimés puis réduits à l'aide de prototypes, des simulations et des logiciels d'analyse. Dans ce cycle, plusieurs prototypes existent sous forme de modèles de conception ou de composants fonctionnels.
- **Développement et test** : les prototypes sont encore plus étendus et des fonctionnalités sont ajoutées. Le code réel est écrit, testé et migré vers un environnement de test plusieurs fois jusqu'à ce que le logiciel puisse être implémenté dans un environnement productif.

- Panification du cycle suivant : le cycle à venir est planifié à la fin de chaque cycle. Si des erreurs se produisent, les solutions sont recherchées. Si une meilleure alternative est une solution envisageable, elle sera préférée au sein du cycle suivant [2].

1.4 L'approche DevOps comme solution

1.4.1 L'approche DevOps

DevOps est un terme anglais qui a fait son apparition au début des années 2009. Il s'agit de la concaténation des abréviations anglaises de développeurs «Dev» et d'opérateurs IT «Ops». Ce nom n'a pas été adopté par hasard car il signifie l'union de ces deux acteurs. C'est un mode de travail utilisée par les grandes entreprises informatiques comme : Google, Microsoft, Amazone et Facebook [3].

L'approche DevOps vise à réunir toutes les équipes du système d'information que ce soit développeurs, exploitants, intégrateurs et administrateurs autour d'un objectif commun. Elle permet d'éviter les conflits qui surviennent entre les différentes équipes en proposant un processus commun à toutes les équipes. Le DevOps propose d'élargir les pratiques agiles à toute équipe contribuant au processus de livraison. Cette approche permet de mettre en place un processus de livraison continue commun à tous les membres de l'équipe. La chaîne se base sur un ensemble d'outils axés sur l'automatisation.

Les avantages de l'approche DevOps sont :

- Fiabilité : Garantir une qualité de produit stable et fiable tout en gardant un rythme de livraison accéléré,
- Collaboration améliorée : les équipes collaborent de façon efficiente en utilisant un processus unifié qui minimise toutes les sources de conflits.

1.4.2 Livraison Continue

La livraison continue, dans le domaine de l'ingénierie logicielle, est une méthodologie permettant d'avoir des logiciels dans un cycle court. Le but était de construire, tester et déployer le plus rapidement possible. La livraison continue vise à mettre en place un processus itératif qui se base sur les principes du DevOps :

- La gestion de versions : Suivre le changement de code source et l'évolution de l'application,
- Intégration continue : les tests doivent être réalisés en continu en amont et en aval du processus,
- Notification : Notifier les intervenants pour chaque étape du processus,
- Déploiement automatique : Aussitôt l'étape de test clôturée, l'application doit être directement déployée en production. Ceci comprend la préparation de l'environnement technique et ainsi que la gestion de la configuration.

1.4.3 Gestion des versions

La gestion de versions permet de suivre les modifications apportées au code source des applications pour avoir un historique. Étant donné que les projets impliquent plusieurs développeurs, la gestion de versions nous permet de savoir qui a modifié le code. Nous pouvons aussi gérer les versions des applications en les hébergeants dans un dépôt spécifique et en utilisant les gestionnaires de paquets. Ceci permet d'avoir l'accès à plusieurs versions, ce qui facilite la procédure de rétrogradation (procédure permettant de revenir à une version ancienne d'un logiciel). Le code source doit être partagé entre tous les membres de l'équipe sur un serveur dédié pour la gestion des dépôts leur permettant ainsi de se synchroniser sur la même version.

1.4.4 Intégration Continue

L'intégration continue consiste à tester l'application de façon continue. Plus précisément, lorsqu'il y a modification du code source, l'application est testée de façon à vérifier l'absence

de régression. Il faut s'assurer que les modules non modifiés n'ont pas régressé et ne comportent pas de bogues. Il s'agit d'un processus totalement automatique géré par un serveur appelé serveur d'intégration continue. Lorsqu'un développeur enregistre ses changements, le serveur les détecte et déclenche le processus de construction puis teste l'application. De manière générale, pour toute version de l'application, le serveur doit la tester pour avoir un retour d'information rapide.

Les tests représentent l'une des parties les plus importantes de l'intégration continue et de toute la chaîne de livraison. Ils permettent de détecter les problèmes et d'avoir un feedback rapide. C'est pour cela qu'il faut les intégrer dès le début de la chaîne. Il existe plusieurs types de tests :

- Test unitaire : Il s'agit de tester le bon fonctionnement d'une portion de code ou bien d'un module du logiciel. Plusieurs logiciels sont dédiés pour les tests unitaires ; JUnit étant le plus célèbre,
- Test fonctionnel : Il s'agit de tester les différentes fonctionnalités du logiciel pour vérifier sa conformité aux spécifications techniques et fonctionnelles,
- Test de performance : Il comprend plusieurs types, notamment, les tests de charge où nous testons le système pour un nombre prédéfini d'utilisateurs, les tests de stress où nous augmentons la charge maximale sur le système pour voir comment il réagit. Le but est d'avoir une idée sur le comportement du logiciel en conditions extrêmes,
- Test de sécurité : Il s'agit d'auditer le logiciel pour pouvoir détecter d'éventuelles failles critiques et d'évaluer sa robustesse contre les attaques classiques.

Notons que le serveur d'intégration ne permet pas d'exécuter lui-même tous ces tests. Cependant, il offre la possibilité d'intégrer d'autres outils qui permettent de faire ces tests.

1.4.5 Automatisation des tests

Le test logiciel est une activité fastidieuse et coûteuse en ressources lorsqu'elle est entièrement manuelle. Automatiser l'ensemble du processus de tests, malgré un coût initial certain, améliore l'organisation et la rentabilité à terme.

1.4.6 Orchestration

L'orchestration consiste à l'ordonnancement des différentes étapes du pipeline. C'est-à-dire, enchaîner dans le bon ordre les étapes. Dans un pipeline, il y a une multitude d'outils ayant chacun un but précis (construction, test, déploiement). Il est alors fondamental de les synchroniser. L'ordre des étapes est considéré de façon logique. En effet, un binaire ne peut être déployé qu'après sa construction. L'orchestration doit être gérée de façon autonome sans intervention externe. En général, c'est le serveur d'intégration continue qui s'occupe de l'orchestration car il peut intégrer tous les outils à son interface.

1.4.7 Déploiement automatique

Le déploiement automatique consiste à déployer l'application sur les serveurs de production ou bien sur des serveurs de tests de façon autonome sans intervention humaine. Il s'agit d'installer l'application avec toutes ses dépendances et la configurer pour être fonctionnelle. Au début, les opérateurs système utilisaient des scripts pour automatiser le processus d'installation et éviter les erreurs de manipulation. Ces scripts traditionnels ont rapidement atteint leurs limites vues leurs complexités, leurs maintenances difficiles et leurs manques de flexibilité. Par ailleurs, lorsque le déploiement de l'application devait s'opérer sur plusieurs serveurs, il fallait importer le script ce qui rendait la tâche peu pratique.

Aujourd'hui, il existe des outils de gestion de configuration spécialisés pour le déploiement qui décrit l'état voulu du serveur dans un fichier similaire à un script. Nous parlons ainsi du concept de « L'infrastructure en tant que code ». Ainsi, nous pouvons déclarer la procédure de déploiement de l'application dans un fichier de configuration. Ce dernier est hébergé dans un serveur responsable du déploiement qui s'occupe de son exécution sur les machines cibles. Le déploiement peut être exécuté plusieurs fois dans le temps pour s'assurer que la configuration n'a pas changé. Dans ce cas, le serveur central vérifie si la configuration des machines clientes est conforme à la configuration déclarée. Par cette méthode, nous pouvons assurer un déploiement automatique, rapide et stable.

Conclusion

Dans ce chapitre nous avons présenté l'entreprise au sein de laquelle nous avons réalisé notre travail. Nous avons introduit le cadre de notre projet, la problématique résolue et ses objectifs. Nous avons exposé la méthode de travail et l'approche DevOps en tant que solution pour résoudre les problèmes rencontrés. Le deuxième chapitre portera sur l'analyse et la spécification des besoins.

Chapitre

2

Analyse et spécification des besoins

Introduction

Après avoir présenté le cadre général de notre projet, la problématique et les objectifs, nous entamons dans ce chapitre une phase primordiale dans le cycle de développement de notre portail qui va nous permettre d'identifier les acteurs, de formaliser les besoins escomptés des utilisateurs, et de déterminer les fonctionnalités du système. Ainsi, nous présentons dans ce qui suit une étude des besoins fonctionnels et non fonctionnels.

Pour ce faire, nous exploitons les diagrammes UML comme une modélisation des cas d'utilisation du système ainsi que les scénarios d'utilisation et les séquences d'interaction entre acteurs et système.

2.1 *Spécification des besoins*

Dans cette partie, nous entamons la spécification des exigences à laquelle doit répondre notre système. Cette phase a pour but d'identifier les acteurs et de décrire les différents besoins fonctionnels et non fonctionnels.

2.1.1 Acteurs

L'analyse d'une application débute toujours par la détermination des différents acteurs en réalisant une étude d'interaction de notre système avec son environnement. Dans le cadre de notre projet, les utilisateurs sont classés suivant les fonctions dont ils disposent. Il s'agit, en effet, de l'ensemble du personnel de l'équipe Zebra ainsi que le fournisseur Comviva qui accèdent tous aux diverses fonctionnalités du système, selon leurs rôles et permissions.

Nous distinguons deux types d'acteurs : Architecte Zebra et utilisateur externe.

- Architecte Zebra : C'est l'acteur principal du système qui est chargé de l'administration de la plateforme et de la configuration des scénarios des tests.
- Utilisateur externe : C'est l'acteur secondaire du système il peut uniquement gérer un scénario de test et consulter les rapports d'audit.

2.1.2 Besoins fonctionnels

Ayant identifié les acteurs de notre solution, nous passons à dégager les fonctionnalités que l'application devrait satisfaire. Les besoins fonctionnels dégagés se résument dans les points suivants :

- La gestion du système à travers d'un portail.
- L'automatisation des tâches routinières et répétitives.
- La génération des rapports de tests.
- La consultation des rapports et des statistiques de tests.
- La notification des acteurs du résultat de test.

Bien que les services principaux du système aient été listés dans le paragraphe précédent, la figure 4 modélise les dépendances et les relations entre ces derniers

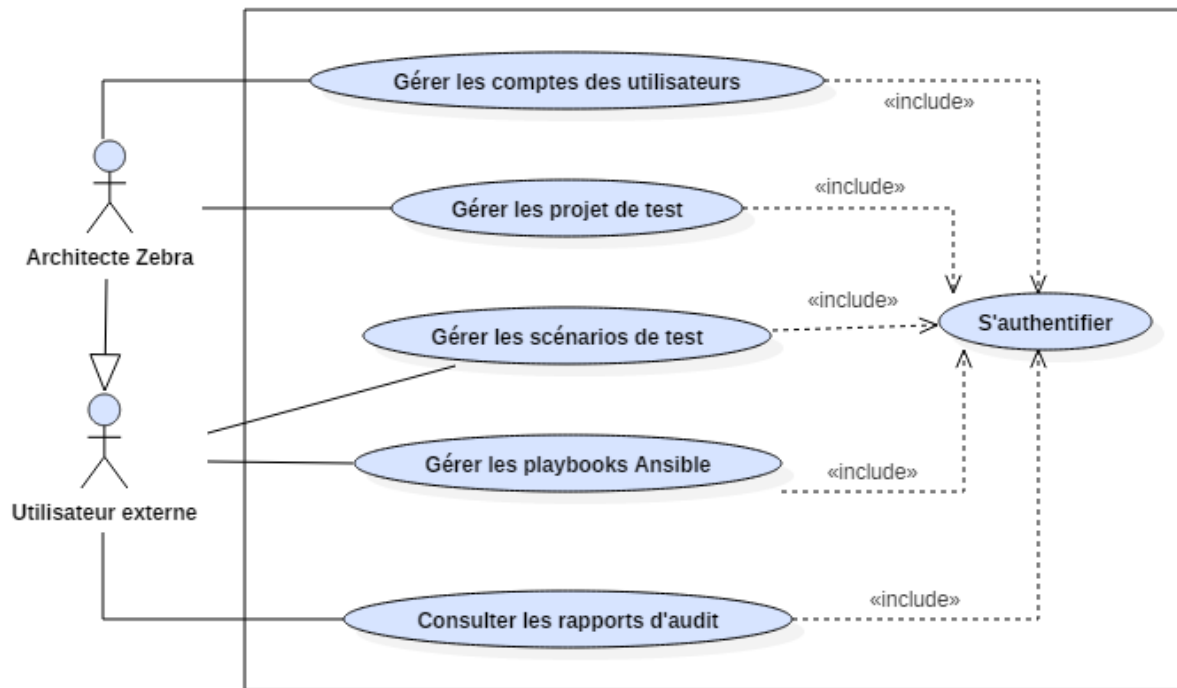


FIGURE 4. *Diagramme de cas d'utilisation global.*

Le système offre à l'acteur cinq possibilités :

- Gérer les projets de tests. Ceci est une phase primordiale lors de l'exécution d'un scénario de test. Elle comprend l'ajout, la modification, la suppression et la consultation d'un projet de test.
- Gérer les comptes des utilisateurs. Ce cas permet d'identifier qui peut avoir accès au système et de définir leurs rôles.
- Gérer les scénarios de tests. Cette tâche inclut la configuration, la consultation et l'exécution d'un test ainsi la sélection du type de test.
- Gérer les playbooks Ansible. L'utilisateur de portail peut gérer un playbook Ansible à travers l'interface graphique sans recours à l'utilisation de la console.
- Consulter les rapports d'audit. L'acteur consulte les rapports et les statistiques détaillant chaque audit lancé.

Ces cas d'utilisation sont détaillés ultérieurement.

2.1.3 Besoins non fonctionnels

Outre les besoins fonctionnels, le système doit assurer les besoins non fonctionnels qui constituent l'ensemble des contraintes qui améliore la qualité de ce dernier. Il doit donc répondre à ces contraintes, il faut que notre système assure :

- Disponibilité : pour garantir le bon déroulement de notre système, ce dernier doit être fonctionnel dans toutes circonstances et fournir un temps de réponse minimal en cas d'erreur.
- Ergonomie : Les interfaces utilisateurs ajoutées doivent s'intégrer avec les interfaces existantes et être facile à utiliser.
- Extensible et évolutif : le système peut intégrer d'autres modules pour étendre ses fonctionnalités.
- Fiabilité : le système doit assurer un niveau de fiabilité élevé vu la critique du domaine.
- Confidentialité : le système doit être capable de garantir un accès sécurisé à l'application toute en respectant les droits et les permissions de chaque utilisateur.

2.2 *Raffinement des cas d'utilisation*

Après avoir cité les différentes fonctionnalités offertes par la solution, nous allons, dans cette section, les modéliser à travers des diagrammes de cas d'utilisation en illustrant les interactions entre les acteurs et le système par des diagrammes de séquences système.

2.2.1 Cas d'utilisation : Gérer les projets de test

La gestion des projets de test est l'une des tâches de l'architecte Zebra. Ce dernier peut ajouter un nouveau projet, modifier ou supprimer un projet existant. La figure 5 décrit un raffinement de cas d'utilisation « Gérer les projets de tests ».

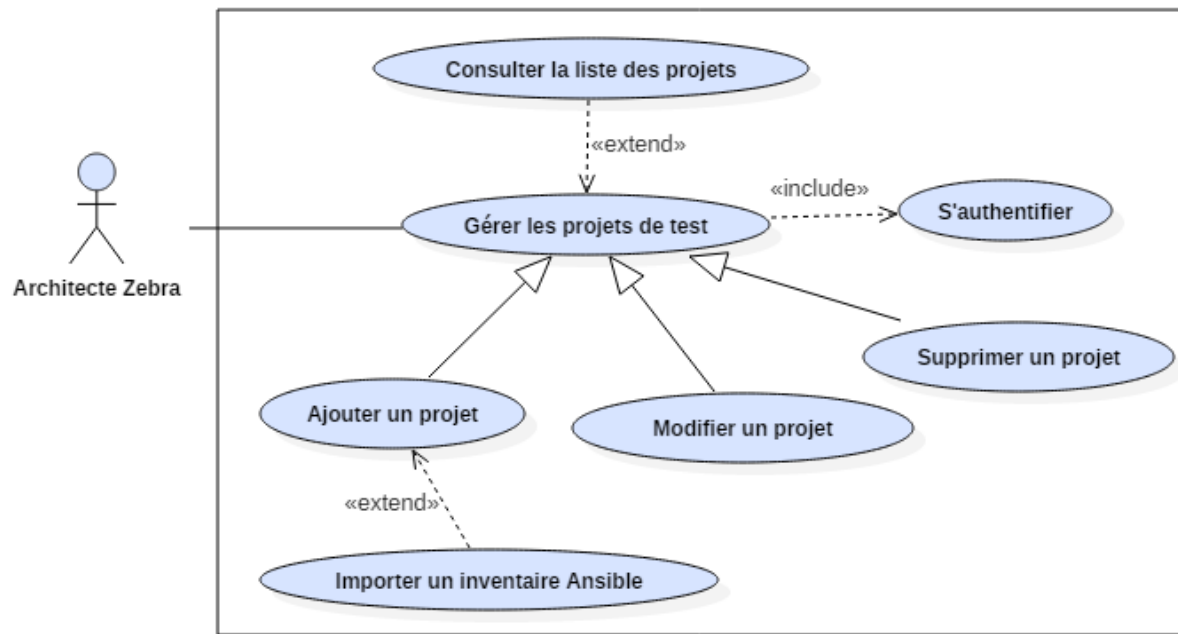


FIGURE 5. Cas d'utilisation « Gérer les projets de test ».

Le tableau suivant représente l'acteur, la description de scénario du cas d'utilisation «Ajouter un projet», ainsi que les exceptions.

Cas d'utilisation	Ajouter un projet
Acteur	Architecte Zebra
Pré condition	Acteur authentifié et autorisé.
Post condition	Projet ajouté
Scénario principale	<ul style="list-style-type: none"> • A travers l'interface 'Projects', l'architecte clique sur le bouton 'Add Project'. Le système affiche le formulaire d'ajout. • L'Architecte remplit les informations sur le projet. • L'Architecte peut choisir de valider ou d'annuler l'ajout.
Exception	<ul style="list-style-type: none"> • Si le nom du projet saisi existe déjà, le système affiche un message d'erreur indiquant que le projet existe déjà. • Si un champ obligatoire manque la saisie, le système affiche un message d'erreur.

TABLE 1. Raffinement de cas d'utilisation « Ajouter un projet ».

Afin de mieux interpréter le processus décrit dans le tableau ci-dessus, nous présentons dans ce qui suit le diagramme de séquence système qui décrit le scénario nominal du cas d'utilisation «Ajouter un projet». Ce diagramme est illustré par la figure 6.

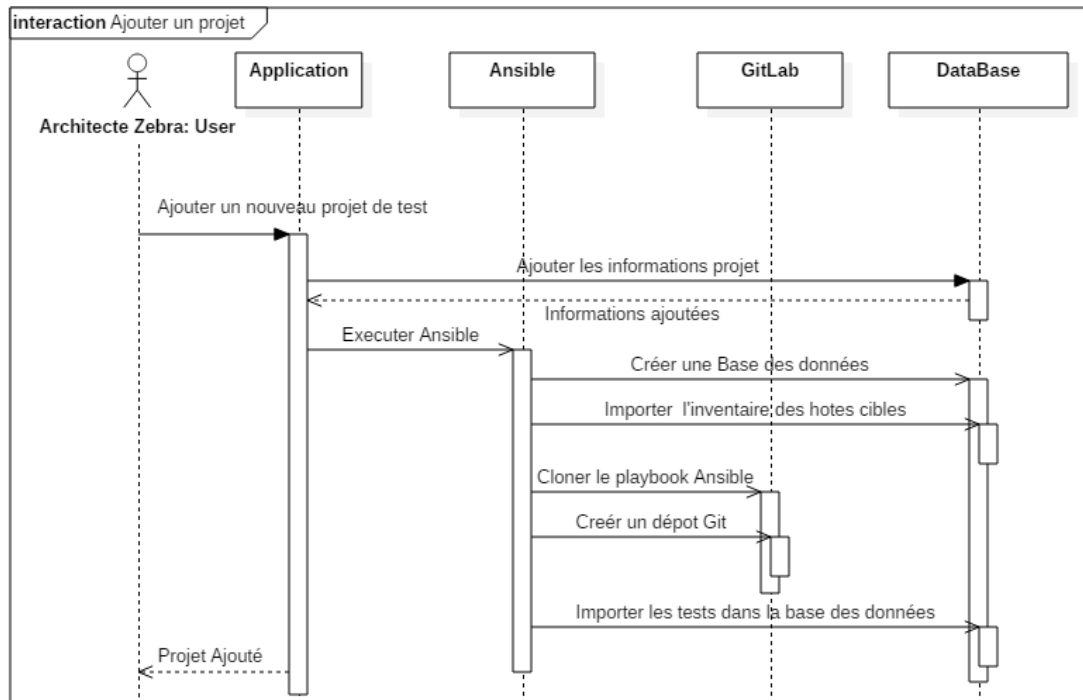


FIGURE 6. diagramme de séquence système « Ajouter un projet ».

2.2.2 Cas d'utilisation : Gérer les comptes d'utilisateurs

La gestion des comptes d'utilisateurs est une partie essentielle de l'administration du portail. L'architecte Zebra crée et modifie les comptes afin de permettre à chacun d'avoir son propre environnement de travail. La figure 7 décrit le cas d'utilisation «Gérer les comptes d'utilisateurs».

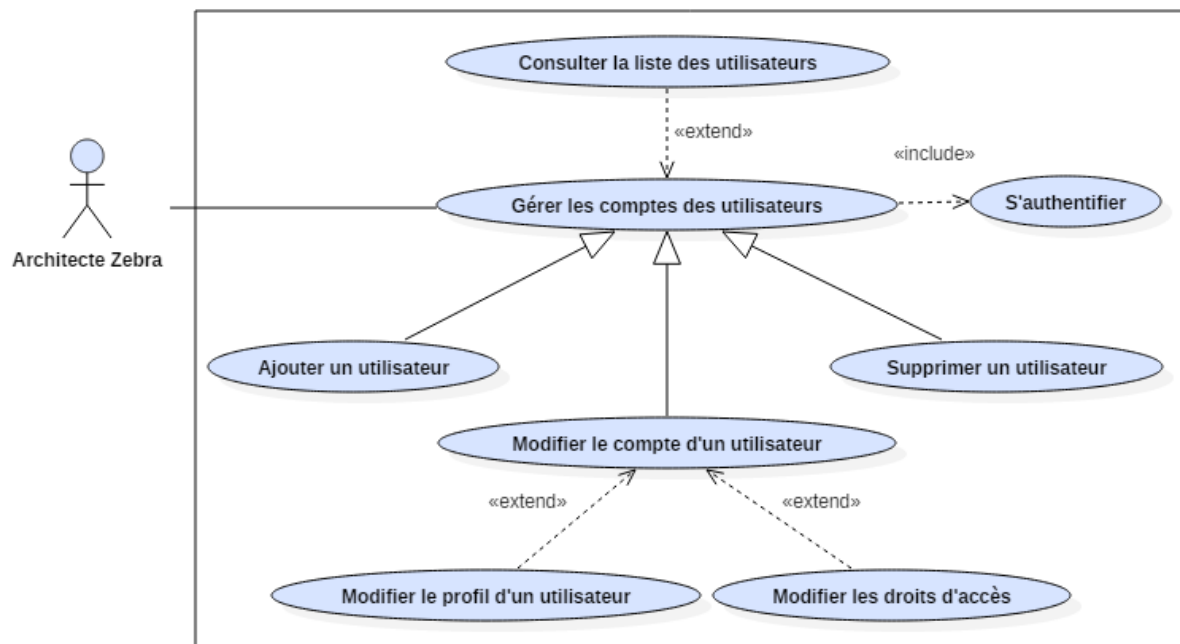


FIGURE 7. Cas d'utilisation « Gérer les comptes d'utilisateurs ».

Pour mieux expliquer ce cas d'utilisation, le tableau suivant représente la description du cas d'utilisation « Modifier le compte d'un utilisateur », ainsi que les exceptions.

Cas d'utilisation	Modifier le compte d'un utilisateur
Acteur	Utilisateur externe, Architecte Zebra
Pré condition	Acteur authentifié et autorisé.
Post condition	Compte modifié.
Scénario principale	<ul style="list-style-type: none"> • A travers l'interface 'Manage Users', l'architecte doit choisir un utilisateur à modifier. • L'Architecte clique sur le bouton 'Edit', un formulaire s'affiche afin de saisir les modifications nécessaires. • L'Architecte peut choisir de valider ou annuler la modification.
Extension	<ul style="list-style-type: none"> • L'acteur peut modifier le profil d'un utilisateur. • L'acteur peut modifier les droits d'accès.
Exception	<ul style="list-style-type: none"> • Si l'identifiant ou le nom d'utilisateur saisi existe déjà, le système affiche un message d'erreur indiquant que le compte existe déjà. • Si un champ obligatoire manque la saisie, le système affiche un message d'erreur.

TABLE 2. *Raffinement de cas d'utilisation « Modifier le compte d'un utilisateur ».*

2.2.3 Cas d'utilisation : Gérer les scénarios de test

Avant l'exécution d'un test, l'utilisateur du portail peut configurer un test en apportant les modifications nécessaires. La figure 8 décrit le cas d'utilisation «Gérer les scénarios de test».

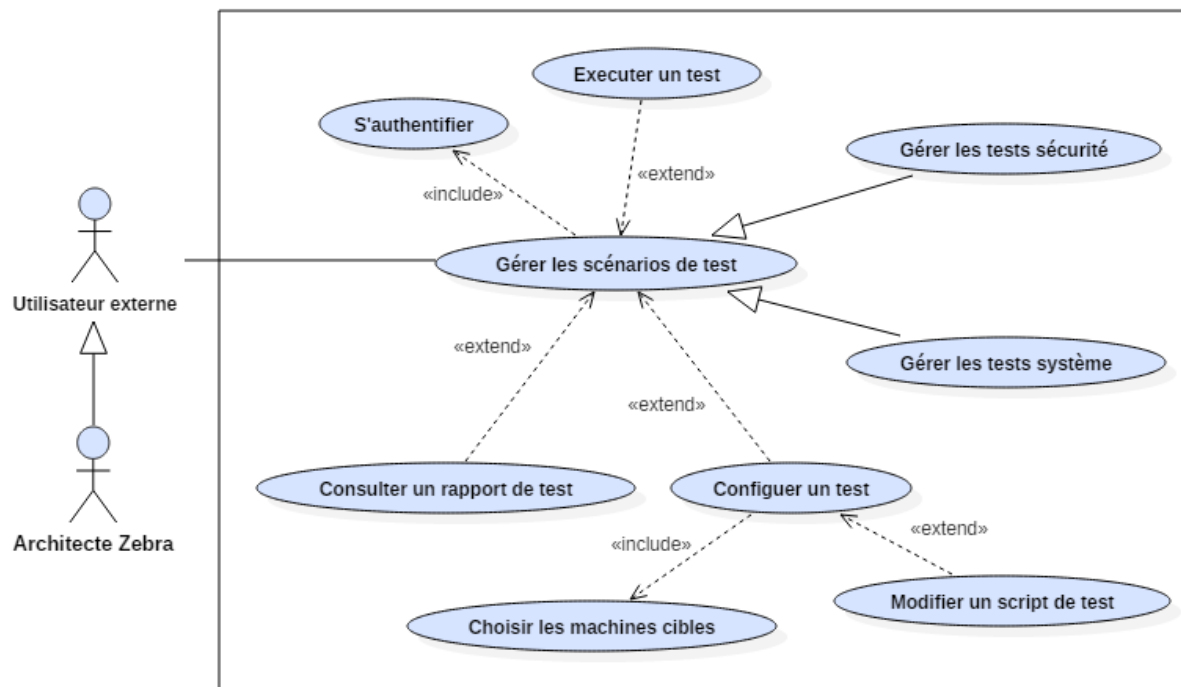


FIGURE 8. Cas d'utilisation « Gérer les scénarios de test ».

Nous détaillons la description du cas d'utilisation «Configurer un test» par le tableau ci-dessous.

Cas d'utilisation	Configurer un test
Acteur	Utilisateur externe, Architecte Zebra
Pré condition	Acteur authentifié et autorisé.
Post condition	Test configuré.
Scénario principale	<ul style="list-style-type: none"> • L'acteur choisit un type de test (système ou sécurité). • L'acteur doit choisir les machines cibles pour lancer le test.
Extension	L'acteur peut modifier un script de test.
Exception	Si un champ obligatoire manque à la saisie, le système affiche un message d'erreur.

TABLE 3. Raffinement de cas d'utilisation « Configurer un test ».

L'automatisation des tests de sécurité est l'une des fonctionnalités principales de notre portail. Le diagramme de séquence suivant (Figure 9) décrit le scénario nominal du cas d'utilisation «Exécuter un test sécurité».

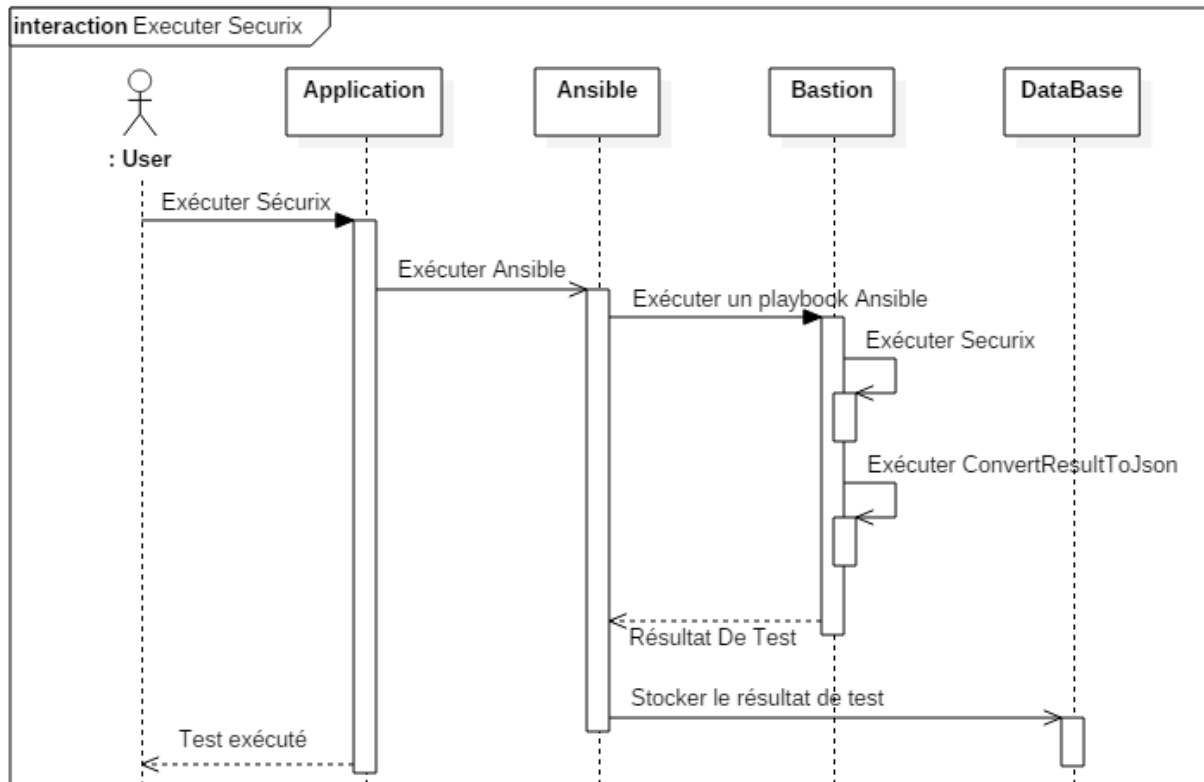


FIGURE 9. Diagramme de séquence système « Exécuter un test sécurité ».

2.2.4 Cas d'utilisation : Gérer les playbooks Ansible

L'utilisateur du portail peut gérer les tests sans recours à la console. Il a la possibilité de modifier les variables d'un playbook, ainsi que les machines cibles. La figure 10 illustre le cas d'utilisation «Gérer un playbooks Ansible».

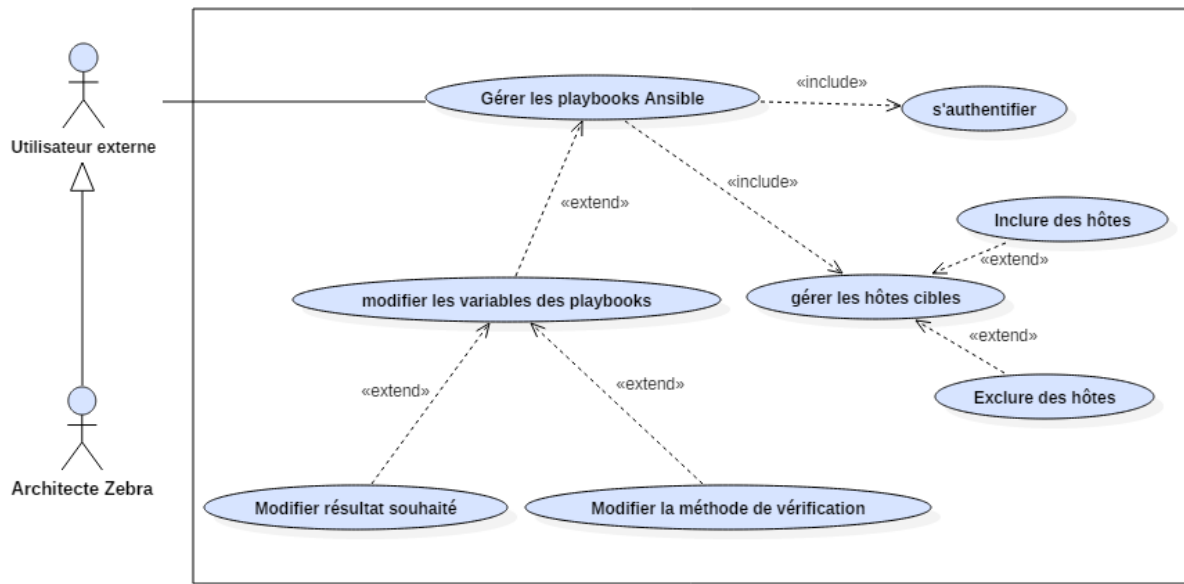


FIGURE 10. Cas d'utilisation « Gérer les playbooks Ansible ».

Outre l'automatisation des tests, notre portail offre la possibilité de gérer un playbook Ansible sans recours à l'utilisation de console. Le tableau suivant illustre l'acteur, la description de scénario du cas d'utilisation «Modifier les variables d'un playbook», les exceptions ainsi que les extensions.

Cas d'utilisation	Modifier les variables d'un playbook
Acteur	Utilisateur externe, Architecte Zebra
Pré condition	Acteur authentifié et autorisé.
Post condition	Playbook modifié
Scénario principale	<ul style="list-style-type: none">• L'acteur choisit la variable à modifier.• Le système affiche un formulaire à remplir.• L'acteur saisit les modifications nécessaires.• Il clique sur le bouton 'Update Variables' pour enregistrer les changements.
Extension	<ul style="list-style-type: none">• L'utilisateur peut modifier la méthode de vérification.• L'utilisateur peut modifier le résultat souhaité.
Exception	Si le système détecte un champ vide, il affiche un message d'erreur

TABLE 4. *Raffinement de cas d'utilisation « Modifier les variables d'un playbook ».*

Le déroulement de l'étape de modification des variables d'un playbook est illustré par la figure 11.

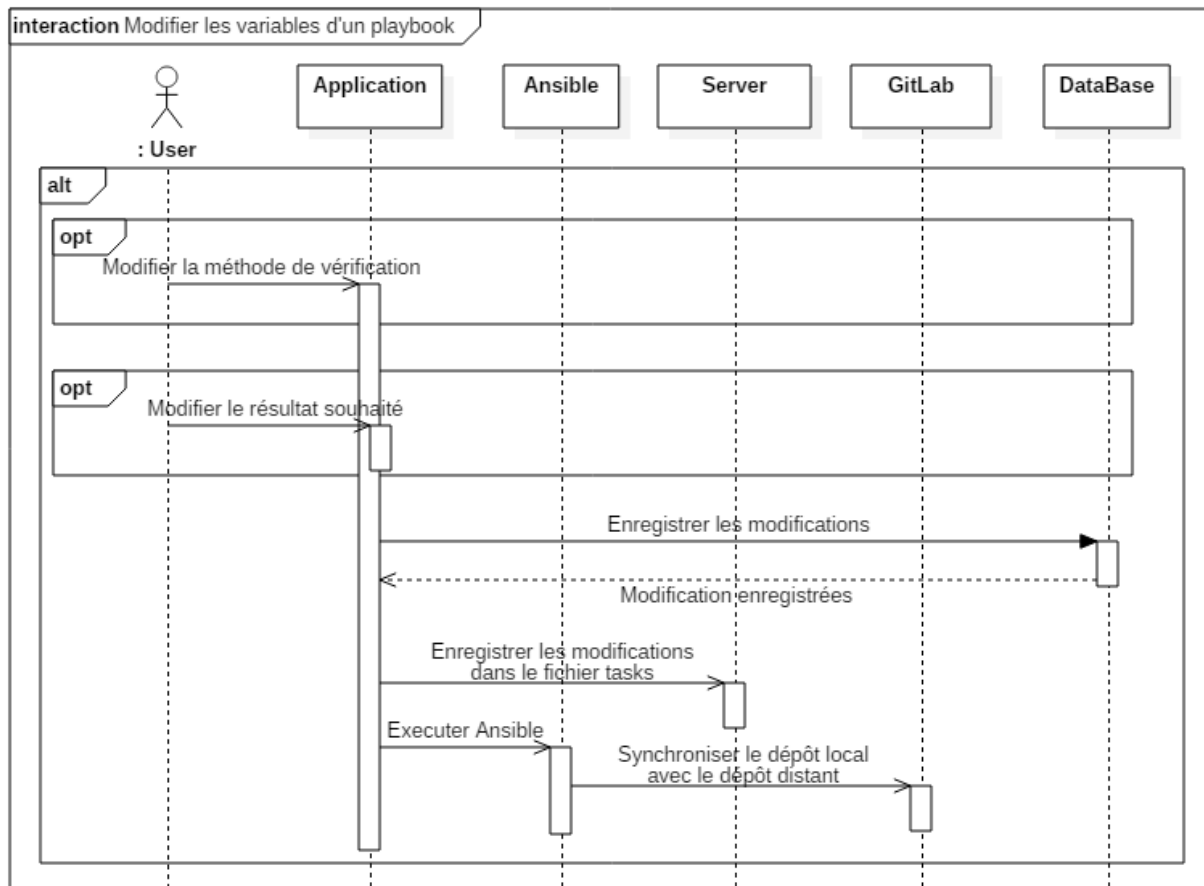


FIGURE 11. Diagramme de séquence système « Modifier les variables d'un playbook ».

2.2.5 Cas d'utilisation : Consulter les rapports d'audit

L'utilisateur du portail peut consulter les différents rapports d'audit en choisissant le type de test : système ou sécurité. La figure 12 décrit le cas d'utilisation « Consulter les rapports d'audit ».

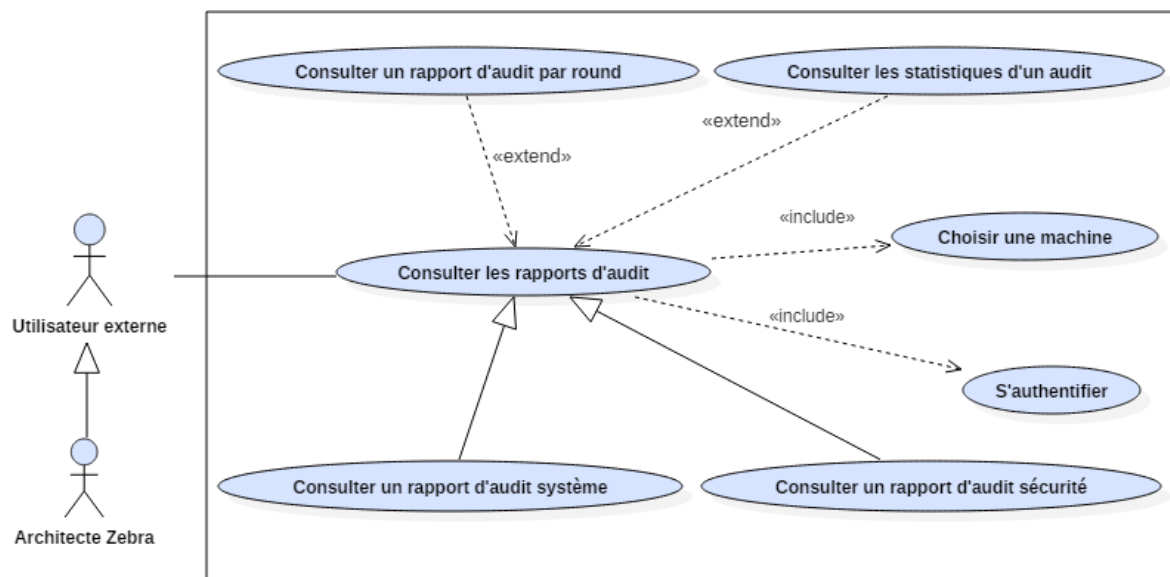


FIGURE 12. Cas d'utilisation « Consulter les rapports d'audit ».

Une fois les tests sont exécutés, notre système génère un rapport d'audit. Le tableau suivant représente l'acteur, la description de scénario du cas d'utilisation « Consulter les rapports d'audit », les exceptions ainsi que les extensions.

Cas d'utilisation	Consulter les rapports d'audit
Acteur	Architecte Zebra, Utilisateur externe
Pré condition	Acteur authentifié et autorisé.
Post condition	Rapport par round spécifié
Scénario principale	<ul style="list-style-type: none"> • L'Acteur choisit le type de test à consulter. • L'Acteur doit choisir la machine à consulter. • Le système affiche le résultat de test par round spécifié.
Extension	L'Architecte peut consulter le rapport d'audit par round.
Exception	S'il n'y a pas des hôtes sélectionnés, le système affiche un message d'erreur.

TABLE 5. Raffinement de cas d'utilisation « Consulter les rapports d'audit ».

Conclusion

Durant ce chapitre, nous avons défini les fonctionnalités de base de notre système à travers les différents diagrammes des cas d'utilisation et séquence système. Dans le prochain chapitre, nous nous abordons la phase de conception ainsi que l'architecture du système réalisé.

Chapitre

3

Architecture logique et Conception

Introduction

Dans ce chapitre, nous détaillons tout d'abord l'architecture logique de notre solution, ensuite nous entamons la conception détaillée de notre système en exposant les différents diagrammes qui constituent une étape primordiale pour procéder au développement.

3.1 *Architecture logique*

L'architecture trois tiers est un modèle logique d'architecture applicative qui oriente la phase de développement de l'application où on doit y introduire les notions et concepts de découpage en couches et mettre en œuvre chacune des couches applicatives. Le but de ce modèle est de présenter le système comme un empilement de multicouches comme l'indique la figure 13.



FIGURE 13. *Architecture logique [4].*

- La couche présentation : associée au client qui est dit « léger ». C'est la couche interactive avec les utilisateurs à travers les interfaces hommes machines. La couche présentation relaie les requêtes de l'utilisateur à sa destination dans la couche métier, et en retour lui présente les informations renvoyées suite aux traitements de cette couche. Dans notre cas, cette couche est développée par Angular.
- La couche métier : c'est la partie fonctionnelle de l'application. Celle qui intègre le logique métier et offre les moyens d'accès aux données en fonction des requêtes des utilisateurs. Elle contient toutes les règles de gestion et de mise en œuvre, et elle recense aussi les objets métiers manipulés par l'application.
- La couche d'accès aux données : cette couche est appelée aussi couche de persistance de données. Elle présente un niveau d'abstraction des données qui prend en charge l'accès aux données.

En effet, cette architecture précédemment expliquée, soutenu par notre architecture logique qui se divise entre coté client, coté serveur et base de données, nous permettra de déduire l'utilisation d'un patron de conception du type MVVM « modèle Vue Vue-Modèle » comme schématisé ci-dessous.

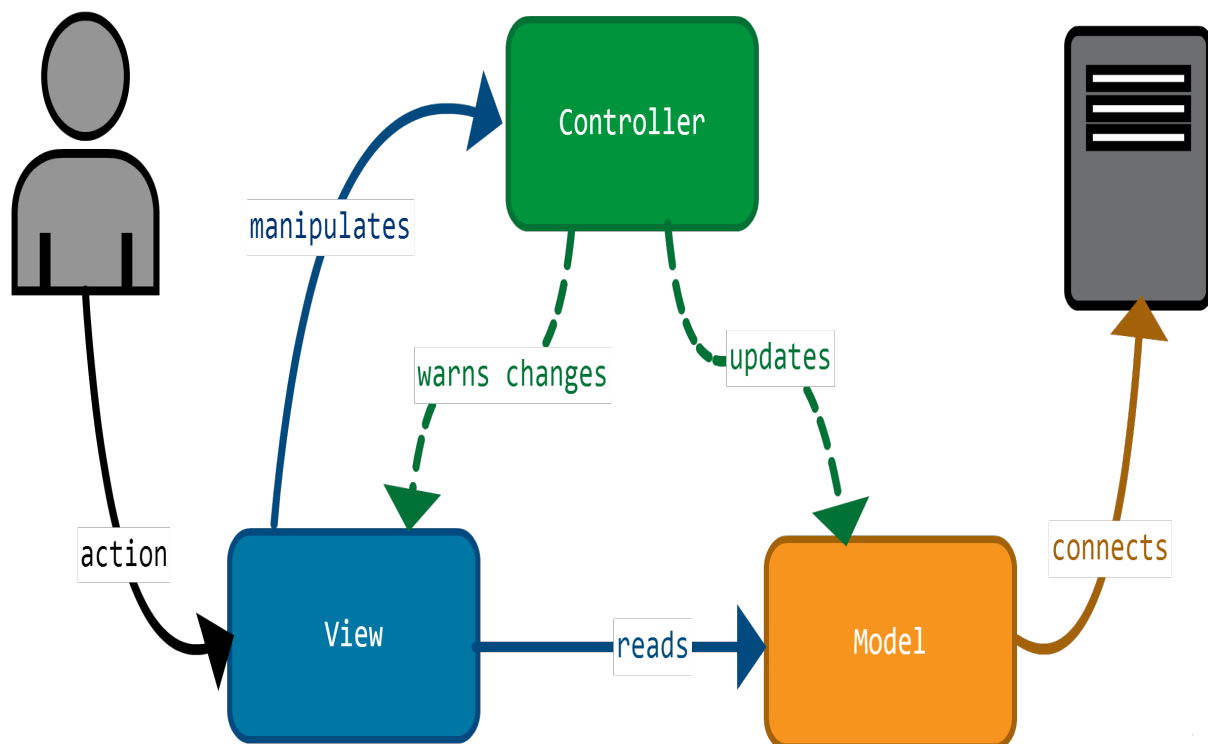


FIGURE 14. Patron de conception MVVM [5].

Ce pattern a spécialement été conçu pour améliorer la séparation entre les données et la vue qui les affichent. Le lien entre la vue et le modèle de données est fait par des mécanismes de binding. Ce mécanisme permet de faire des liaisons entre des données de manière dynamique. Ce qui veut dire que si A et B sont liés, le fait de modifier A va être répercuté sur B et inversement. Le modèle MVVM est composé par trois parties :

- Le modèle : le modèle contient les données. Généralement, ces données proviennent d'une base de données ou d'un service externe comme un API.
- La vue : la vue correspond à ce qui est affiché (la page web dans notre cas). La vue contient les différents composants graphiques (boutons, liens, listes) ainsi que le texte.
- La vue-modèle : ce composant fait le lien entre le modèle et la vue. Il s'occupe de gérer les liaisons de données et les éventuelles conversions. C'est ici qu'intervient le binding [6].

Afin d'avoir une idée claire sur le processus de notre architecture logique nous avons simulé l'application de ce design patron sur un projet MEAN, qui sera schématisé par la figure 15.

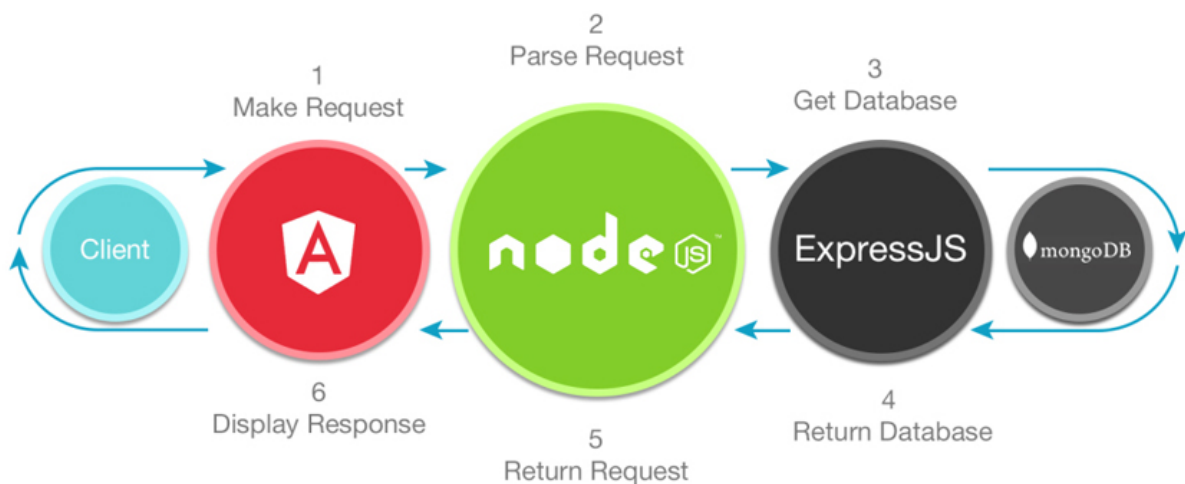


FIGURE 15. Architecture d'un projet MEAN [7].

3.2 Diagramme de classes

Le diagramme de classe représente les classes intervenant dans le système. Il s'agit d'une représentation statique des éléments qui composent un système et de leurs relations. Nous

Afin de mieux comprendre la conception de la base de données le tableau suivant décrit les différentes entités utilisées par notre portail.

ProjectInformation	Regroupe les informations sur l'ensemble des projets de tests de l'application.
User	Regroupe les informations sur les utilisateurs de l'application.
Host	Contient la liste des machines utilisées par notre inventaire Ansible.
SystemChecksVariable	Contient l'ensemble des tests système ainsi que les méthodes de vérification utilisées.
SystemChecksReport	Contient un rapport d'audit généré suite à un test système.
SecurixReport	Contient un rapport d'audit généré suite à un test sécurité.

TABLE 6. *Principales collections du schéma de la base de données.*

3.3 Diagrammes de séquence objet

Les principales informations contenues dans un diagramme de séquence sont les messages échangés entre les lignes de vie présentés dans un ordre chronologique.

3.3.1 Diagramme de séquence objet Authentification

Avant d'entrer au menu du portail et faire l'ensemble des autres scénarios l'utilisateur doit se connecter en utilisant son identifiant et son mot de passe. La figure 17 présente l'enchaînement de la phase d'authentification.

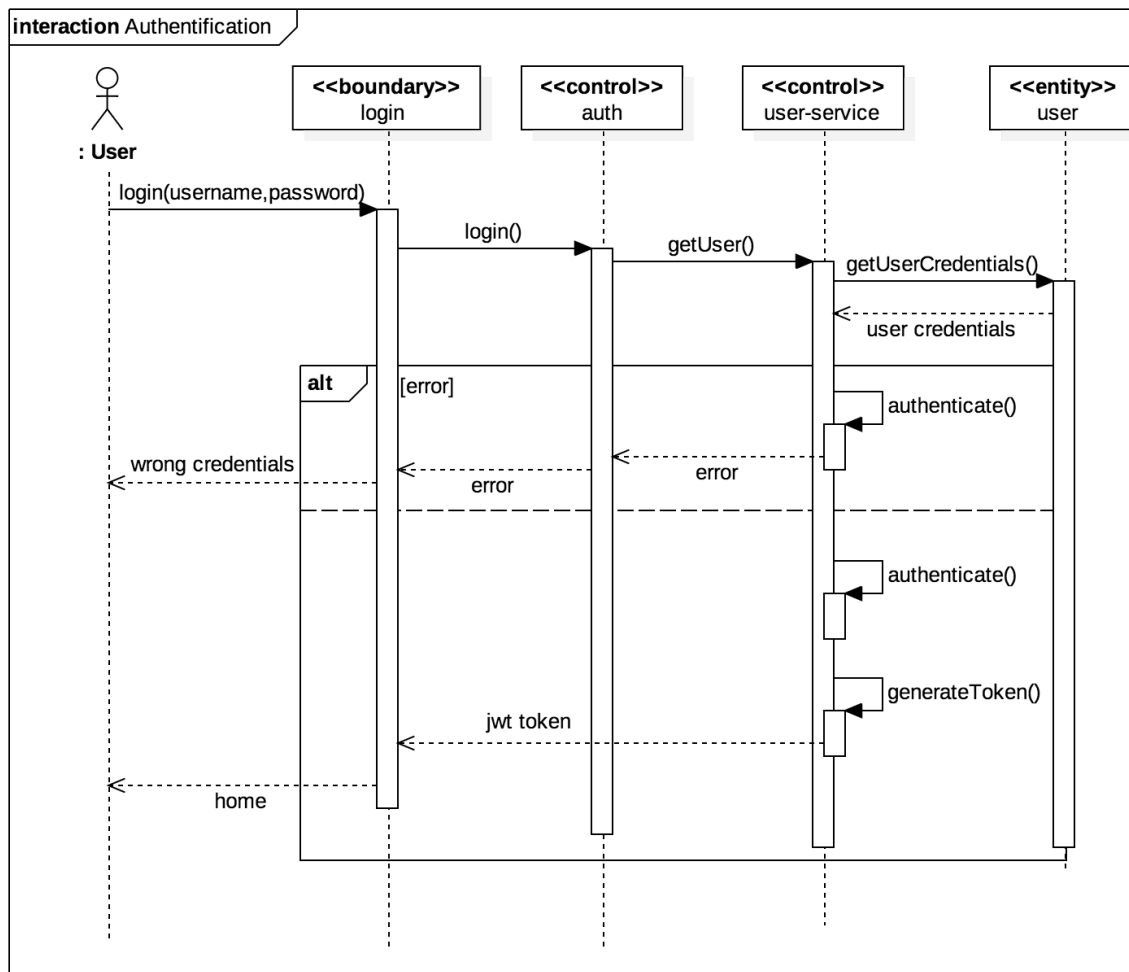


FIGURE 17. Diagramme de séquence objet « Authentification ».

3.3.2 Diagramme de séquence objet ajouter un utilisateur

Parmi les tâches effectuées par l'Architecte Zebra est la gestion des utilisateurs puisqu'il s'agit de l'administrateur de notre portail. Ce dernier peut ajouter un utilisateur en remplissant un formulaire dédié et en attribuant un rôle. La figure 18 nous montre le processus d'ajout d'un nouvel utilisateur.

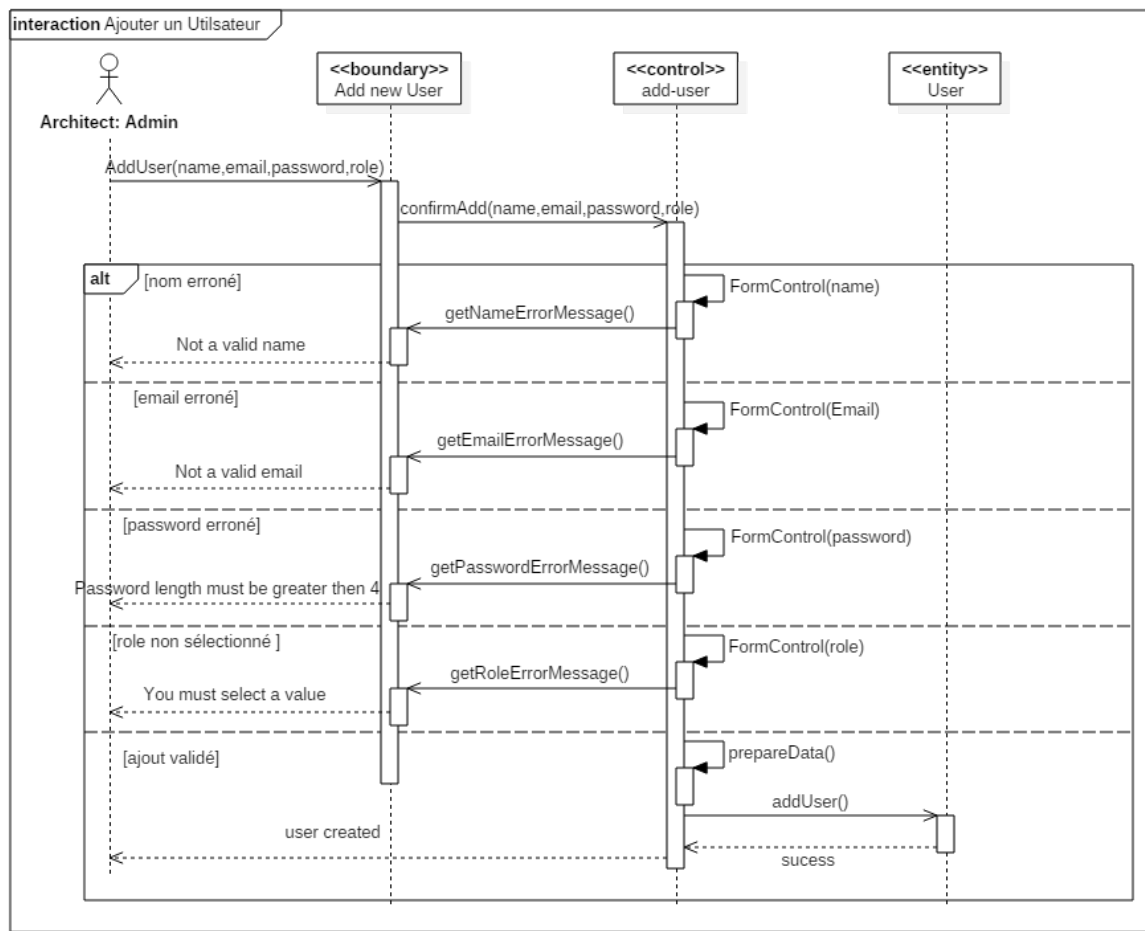


FIGURE 18. Diagramme de séquence objet « Ajouter un utilisateur ».

3.3.3 Diagramme de séquence modifier un playbook Ansible

Afin de modifier un playbook Ansible, l'utilisateur doit consulter la liste des variables et choisir un test à modifier. Ensuite il doit remplir le formulaire de mise à jour convenablement. Une fois les modifications sont enregistrées, elles seront par la suite sauvegardées dans la base des données. Le diagramme de séquence «Modifier un playbook Ansible» est détaillé dans la figure 19.

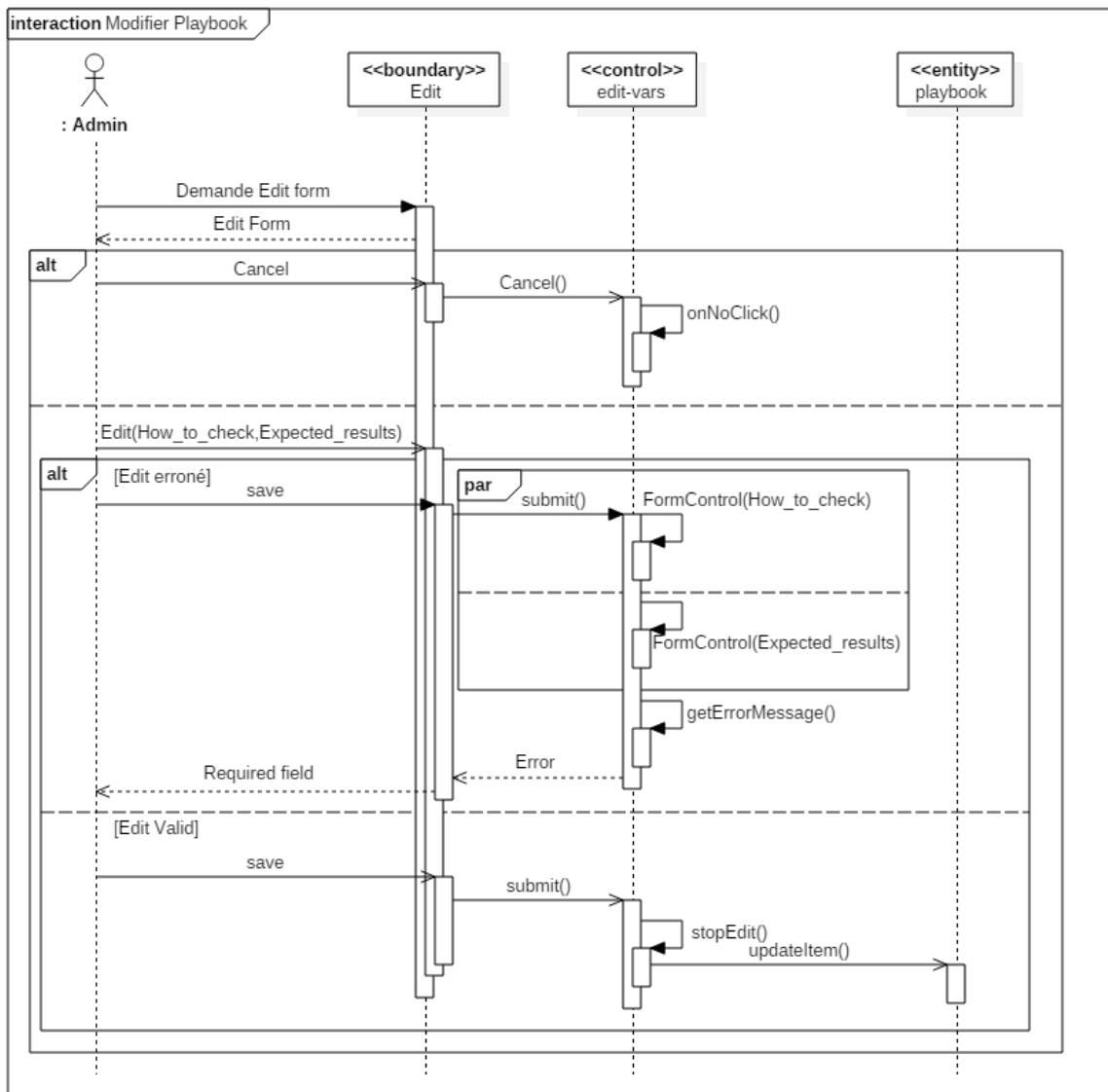


FIGURE 19. Diagramme de séquence objet « Modifier un playbook Ansible ».

3.3.4 Diagramme de séquence objet consulter un rapport d'audit système

Après chaque audit, l'utilisateur de portail consulte les représentations graphiques des statistiques et les finalités des tests. Le système offre la possibilité ainsi d'exporter le rapport d'audit et les courbes. Le diagramme de séquence « Consulter un rapport d'audit système » est détaillé dans la figure 20.

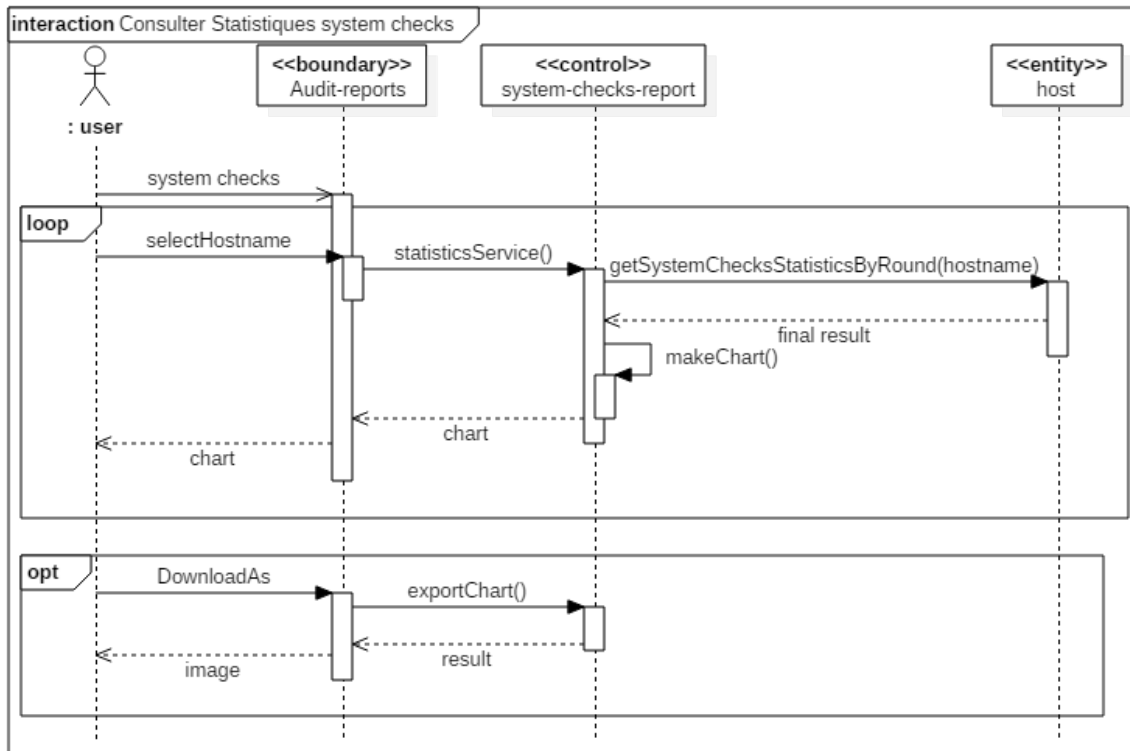


FIGURE 20. Diagramme de séquence objet « Consulter un rapport d’audit système ».

3.3.5 Diagramme d’activité

Un diagramme d’activité fournit une vue du comportement d’un système en décrivant la séquence d’actions d’un processus. La figure 21 illustre le diagramme d’activité du principaux cas d’utilisation fournis par notre système.

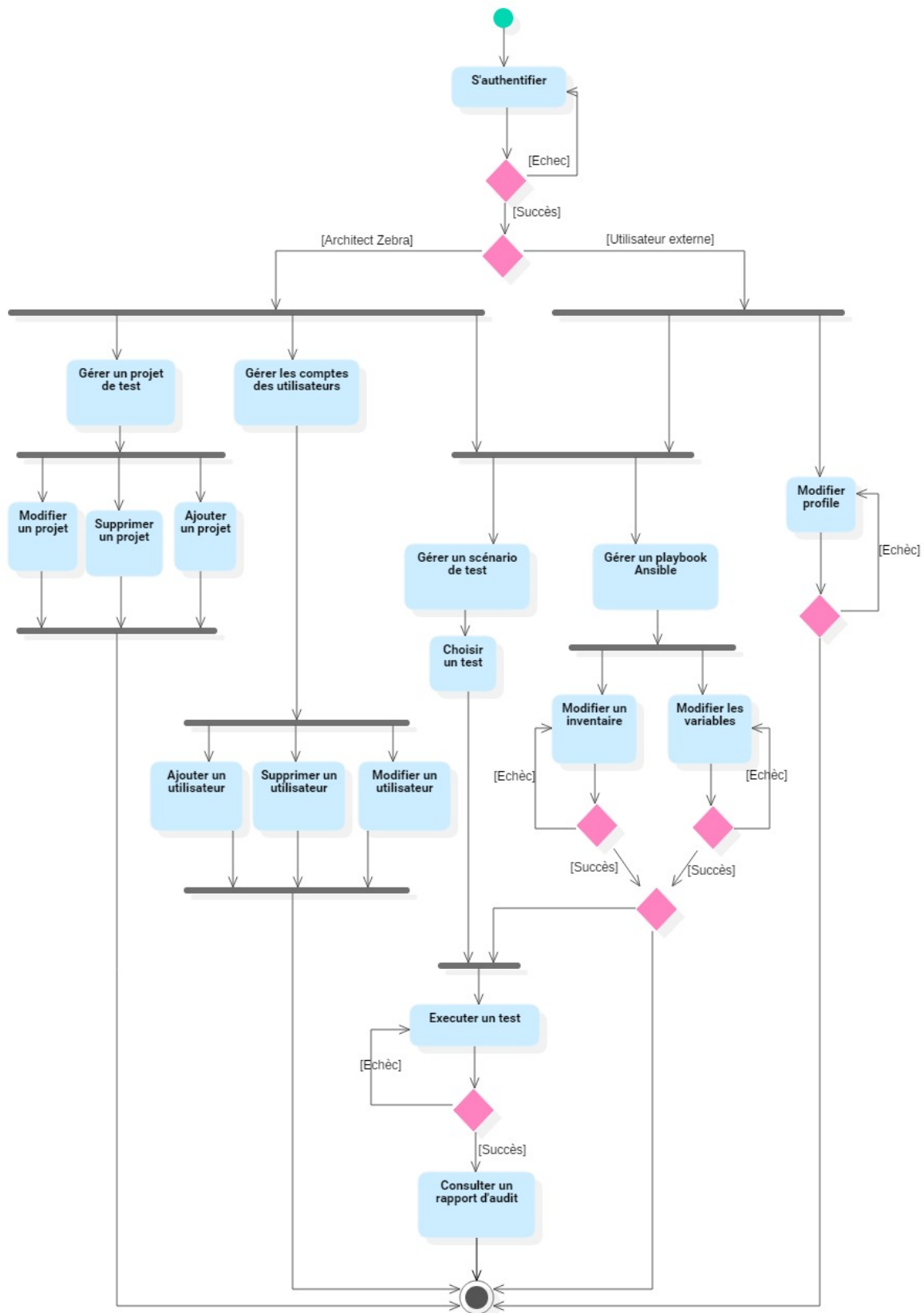


FIGURE 21. Diagramme d'activité.

Conclusion

Durant ce chapitre, nous avons présenté l'architecture logique utilisée lors de l'implémentation de notre solution. Par la suite, nous avons illustré le modèle de données et les traitements possibles du système à travers les diagrammes de séquence objet. Et enfin, nous avons représenté le déroulement des différents cas d'utilisation en utilisant le diagramme d'activité. Dans le prochain chapitre, nous allons détailler les outils et les langages utilisés dans la réalisation de notre travail ainsi que l'architecture physique utilisée.

Chapitre

4

Réalisation

Introduction

Une des étapes de cycle de vie d'un projet, aussi importante que la conception, est l'implémentation. Cette étape constitue la phase d'achèvement et d'aboutissement du projet. Dans ce chapitre, nous commençons par une description de l'environnement matériel, logiciel et de programmation. Par la suite, nous décrivons l'architecture logicielle utilisée lors de l'implémentation de notre solution. Enfin, nous détaillons le travail tout au long de la période du projet en l'illustrant avec des captures écrans des interfaces les plus intéressantes.

4.1 *Environnement de travail*

Dans cette partie, nous présentons les différents outils et ressources matérielles exploitées au cours du présent projet ainsi que l'environnement logiciel qui a permis l'élaboration de notre application.

4.1.1 Environnement matériel

L'environnement matériel auquel nous avons eu recours est présenté dans le tableau ci-dessous.

	Machine de développement
Mémoire	8 GO
Processeur	Core i5
Système d'exploitation	Windows 7 64 bits
Capacité	500 Go

TABLE 7. *Configuration de l'environnement matériel.*

4.1.2 Environnement logiciel

La mise en place de ce projet et l'élaboration du présent rapport ont nécessité un certain nombre de logiciels qui sont :

- IntelliJ IDEA : C'est un environnement de développement intégré spécialisé dans le langage Java. Il fournit une pléthore d'outils vous permettant de développer des applications Java, et ce, sur plusieurs plateformes y compris mobiles [8].
- StartUML : Il s'agit d'un logiciel de modélisation UML, cédé comme open source par son éditeur, à la fin de son exploitation commerciale, sous une licence modifiée de GNU GPL. On a choisi pour la modélisation de notre solution [9].
- Robo 3T : Il s'intègre avec le shell MongoDB (notre base de données NoSQL orientée document) pour vous donner toute la puissance sur les outils de stock. C'est pour ça qu'on a choisi cet outil pour la gestion et l'administration de la base de données [10].

4.1.3 Choix techniques de réalisation

Dans cette partie, nous illustrons les choix techniques des langages de programmation et des Framework utilisés.

- JavaScript : C'est un langage de script léger, orienté objet, principalement connu comme le langage de script des pages web. Mais il est aussi utilisé dans de nombreux environnements extérieurs aux navigateurs web tels que node.js ou Apache CouchDB.

(JavaScript) Ce langage suit, parmi plusieurs, un cadre de programmation fonctionnel, impératif et orienté objet [11].

- Framework ExpressJS :Ce framework fournit une infrastructure d'applications Web Node.js minimaliste, flexible et ultra-rapide grâce au moteur V8 et son fonctionnement non bloquant.Il garantit un ensemble de fonctionnalités robustes pour les applications Web et mobiles. La création d'une API est devenue une tâche simple et rapide grâce aux méthodes utilitaires HTTP et des middlewares offerts par ExpressJs [12].
- Python : C'est un langage de programmation puissant et facile à apprendre. Il dispose de structures de données de haut niveau et permet une approche simple mais efficace de la programmation orientée objet. Parce que sa syntaxe est élégante, que son typage est dynamique et qu'il est interprété, Python est un langage idéal pour l'écriture de scripts et le développement rapide d'applications dans de nombreux domaines et sur la plupart des plateformes [13].
- Script Shell : Un script shell permet d'automatiser une série d'opérations. Il se présente sous la forme d'un fichier contenant une ou plusieurs commandes qui seront exécutées de manière séquentielle [14].
- YAML : Nous avons utilisé le langage de sérialisation YAML qui fournit de puissants paramètres de configuration. YAML est l'acronyme de «Yet Another Markup Language», il se définit comme étant « un standa de sérialisation de données pour tous les langages, facile à utiliser pour les humains » [15].
- MongoDB : C'est un système de gestion de base de données orientée documents, répartitionnable sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données. Il est écrit en C++. Le serveur et les outils sont distribués sous licence AGPL, les pilotes sous licence Apache et la documentation sous licence Creative Commons2. Il fait partie de la mouvance NoSQL [16].
- Ansible : Il s'agit d'une technologie d'automatisation informatique simple, sans agent, capable d'améliorer vos processus existants, d'assurer la migration des applications pour une optimisation plus efficace et de créer un langage commun à toutes les pratiques DevOps de l'entreprise (voir Annexe B)[17].

- GitLab : C'est la première application unique pour toutes les étapes du cycle de vie DevOps. Gitlab offre une visibilité inégalée, des niveaux d'efficacité plus élevés et une gouvernance complète. On a choisi Gitlab pour la gestion des versions ainsi l'intégration continue [18].

4.2 *Étude comparative*

La réussite des approches DevOps repose principalement sur deux éléments majeurs, le premier c'est la mise en place de nouveaux processus de collaboration, le deuxième c'est la mise en œuvre d'outils dédiés. C'est pourquoi on va procéder par une étude comparative de différents outils que nous allons l'adopté pour l'implémentation de notre solution.

4.2.1 **Comparaison entre Ansible et Chef**

Suite à une comparaison entre Ansible et Chef, nous avons opté pour Ansible grâce à sa prise en main facile, sa simplicité d'utilisation, et au sein d'Orange les spécialistes d'Ansible ont créé une communauté très active pour aider les collaborateurs à utiliser Ansible. Les critères de comparaison sont les suivants :

- Facilité d'installation et architecture : la mise en place de l'architecture de déploiement.
- Management des nœuds : la façon avec laquelle l'utilitaire de déploiement gère les machines clients.
- Interopérabilité : Quels sont les systèmes d'exploitation couverts ?
- Activité de la communauté des développeurs : est-ce-que les développeurs maintiennent leur produit et avec quelle fréquence ?
- Popularité de la solution.

Le tableau suivant illustre une comparaison suivant les différents critères cités précédemment.

Critères	Ansible	Chef
Facilité d'installation et architecture	Architecture Master-Node : <ul style="list-style-type: none"> • Un seul Master fonctionnant sur un serveur, aucun agent sur les machines. • Utilise SSH pour se connecter aux machines clientes. • Les machines clientes ne nécessitent pas d'installation d'agents. 	Master-agent : <ul style="list-style-type: none"> • Le serveur Chef fonctionne sur une machine master. • Les clients Chef tournent comme des agents sur chaque machine cliente.
la gestion de configuration	Gestion facile par les playbooks. Ils sont écrits en YAML qui est un langage facile à apprendre et à maîtriser.	Il faut être un programmeur pour gérer les configurations. Les cookbooks sont écrits en Ruby un langage un peu difficile.
Activité de la communauté	<ul style="list-style-type: none"> • Ansible possède plus de 1000 contributeurs au projet qui totalisent plus 13000 commit. • Au sein d'Orange les spécialistes d'Ansible sont nombreux et ils ont créé une communauté active pour aider les collaborateurs à utiliser. 	<ul style="list-style-type: none"> • Chef possède plus de 300 contributeurs qui totalisent plus 12000 commit. • Chez Orange la communauté Chef est réduite et n'est pas active.

TABLE 8. *Comparaison entre Ansible et Chef [19].*

En prenant en compte ses différents critères, Ansible répond plus à notre besoin car nous gérons un grand nombre de machines ce qui engendre une perte de temps énorme pour l'installation et la configuration des agents sur les clients.

En outre, Ansible est maintenue par une grande communauté qui assure sa pérennité. De plus l'ensemble des collaborateurs du groupe Orange ont la possibilité de bénéficier d'une

communauté interne d'expert en la matière. Le projet est maintenu par une grande communauté assurant son avenir et des mises à jour 22 constantes. Pour son utilisation nous pouvons nous reposer sur l'assistance des spécialistes d'Orange.

En guise de conclusion, Google Trends nous permet d'avoir une idée sur la popularité des produits selon la recherche des utilisateurs. Comme indique la figure 22 Ansible est le plus populaire en termes de recherches.



FIGURE 22. Popularité de Chef et Ansible [20].

4.2.2 Comparaison entre GitLab et SVN

GitLab et SVN sont les deux outils de gestion de version les plus répandus sur le marché. Le tableau 9 est un tableau comparatif entre ces deux outils qui nous permettra par la suite d'effectuer notre choix technique. Le tableau ci-dessous compare entre les deux outils de gestion de versions GitLab et SVN en se basant sur les éléments suivants :

- Architecture : Les systèmes informatiques peuvent être déployés selon deux grands types d'architecture, soit centralisée ou distribuée.
- Point unique de défaillance : Une panne entraîne l'arrêt complet du système ?
- Contrôle d'accès : La simplicité d'accès au dépôt.
- Stockage du contenu : Comment sont stockées les données ?

Critères	SVN	GitLab
Architecture	<p>Centralisé :</p> <ul style="list-style-type: none"> • Uniquement le répertoire central a l'historique complet des changements. <p>Les utilisateurs doivent communiquer via le réseau avec le répertoire central pour obtenir l'historique.</p> <ul style="list-style-type: none"> • Les backup sont gérés et maintenu indépendamment du SVN. 	<p>Distribué :</p> <ul style="list-style-type: none"> • Tous les développeurs qui vérifient le code à partir d'un dépôt / serveur central auront leur propre référentiel cloné installé sur leur machine. • Ceci permet aux développeurs de créer une nouvelle branche, de faire un « commit » sur un fichier et de revoir une version même en absence de connexion.
Point unique de défaillance	<ul style="list-style-type: none"> • Si le répertoire central est endommagé, alors uniquement les données enregistrées dans le dernier backup sont récupérable. 	<ul style="list-style-type: none"> • La notion de « single point of failure » n'existe pas car il y a autant de copie du dossier qu'il y'en a d'utilisateur si ce n'est plus.
Contrôle d'accès	<ul style="list-style-type: none"> • Nécessité d'un «commit access » due au fait de la centralisation. 	<ul style="list-style-type: none"> • Non nécessité d'un « Commit access » puisque le système est distribué. Il faut juste décider de fusionner quoi à partir de quel utilisateur.
Stockage de contenu	<ul style="list-style-type: none"> • Stockage des métadonnées des fichiers sous forme de dossier caché. 	<ul style="list-style-type: none"> • Stockage de tout le contenu dans un dossier .git, c'est le dépôt du code cloné sur la machine client.

TABLE 9. Comparaison entre SVN et GitLab [21].

Suite à une comparaison effectuée par Google Trends nous avons constaté que GitLab est plus populaire que SVN voir la figure 23.



FIGURE 23. Popularité de GitLab et SVN [20].

4.3 Architecture logicielle globale

L'architecture logicielle de la plate-forme est dépliée dans cette partie du rapport. De nombreux composants logiciels interagissent entre eux afin d'assurer le bon fonctionnement de notre système. Cette architecture offre un large degré d'évolutivité, d'interopérabilité, d'extensibilité et de généricité. La figure 24 représente l'architecture logicielle globale de notre solution ainsi que la répartition des composants et les connexions entre eux.

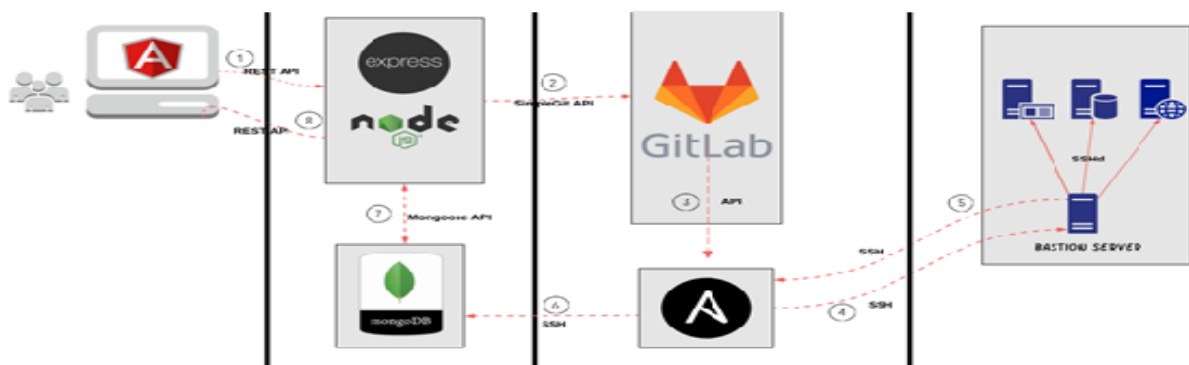


FIGURE 24. Architecture logicielle globale.

Notre architecture peut être décomposée en trois zones. Chaque zone a une fonction dédiée et un ensemble de composants spécifiques :

- L'application web se compose d'Angular comme framework coté client, node.js pour le framework serveur et MongoDB pour la base de données. Node.js sert à créer une API sur le serveur pour manipuler les données et Angular est responsable de l'affichage des pages grâce à son moteur de template.
- La gestion des dépôts est assurée par GitLab. Une version générique des tests est déployée dans ce dernier afin d'appliquer les modifications spécifiques de chaque projet.
- La gestion de configuration et l'exécution des tests sont assurés par Ansible. Il joue le rôle d'orchestrateur dans le système. Ce dernier établit des connexions SSH à des machines distantes pour exécuter les tests et récupérer les rapports.

4.4 Aperçu sur le travail réalisé

Dans cette section, nous présenterons la solution proposée en résumant le travail accompli, et en détaillant les principales fonctionnalités offertes.

4.4.1 Gestion des projets

L'interface de gestion des projets est accessible seulement par l'administrateur du système. Elle permet essentiellement d'ajouter de nouveaux projets de tests, de supprimer un projet et d'en modifier les existants. Suite à l'accès à cette interface, l'utilisateur peut consulter la liste des projets déjà existants. Il aura un accès rapide à un projet donné en faisant un filtrage sur le nom du projet. L'utilisateur pourra visualiser ou modifier les informations d'un projet. Cette interface est illustrée par la figure 25.

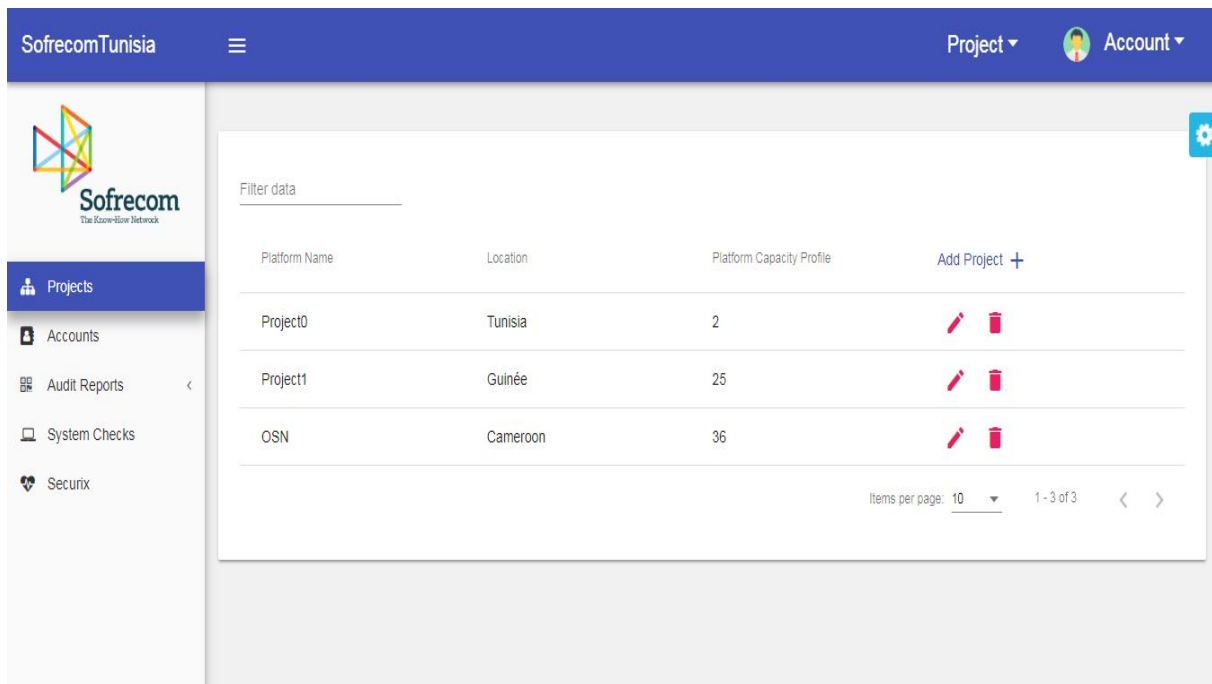


FIGURE 25. *Interface de gestion des projets.*

Pour ajouter un nouveau projet, un bouton intitulé « Add Project » est mis à la disposition de l'utilisateur. L'interface d'ajout est accessible suite au clic sur ce bouton. L'administrateur doit introduire le nom du projet ainsi que sa location, la capacité du projet et un fichier contenant la liste des machines cibles comme illustré par la figure 26.

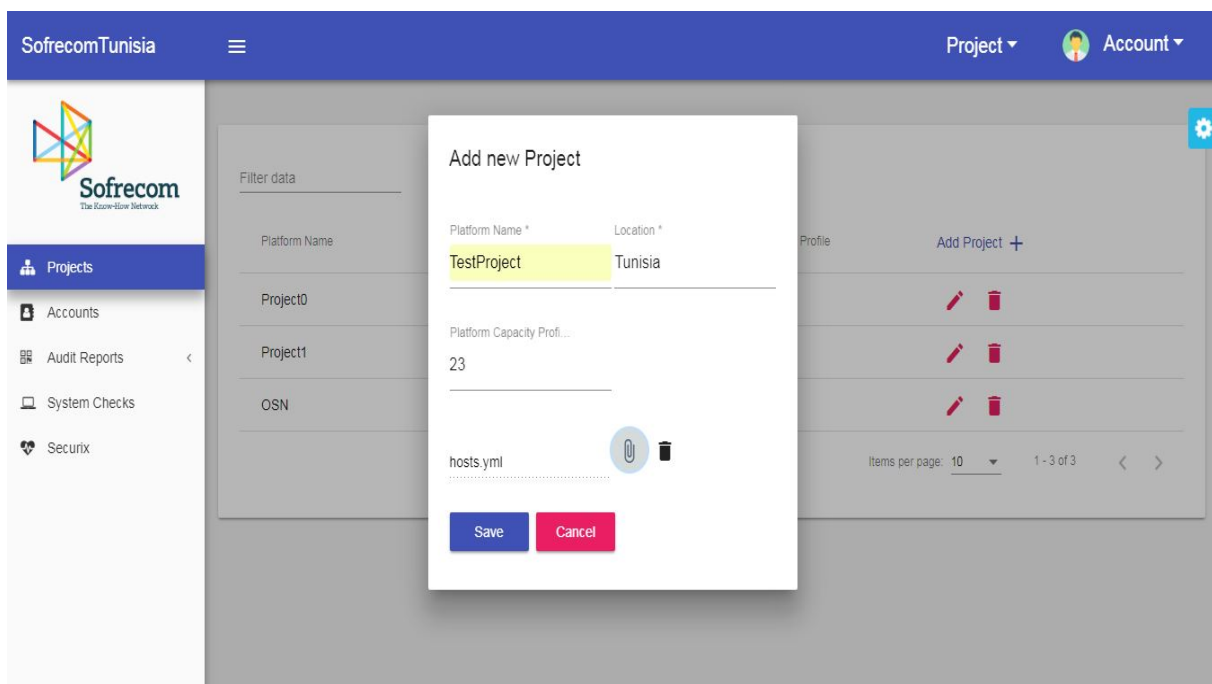
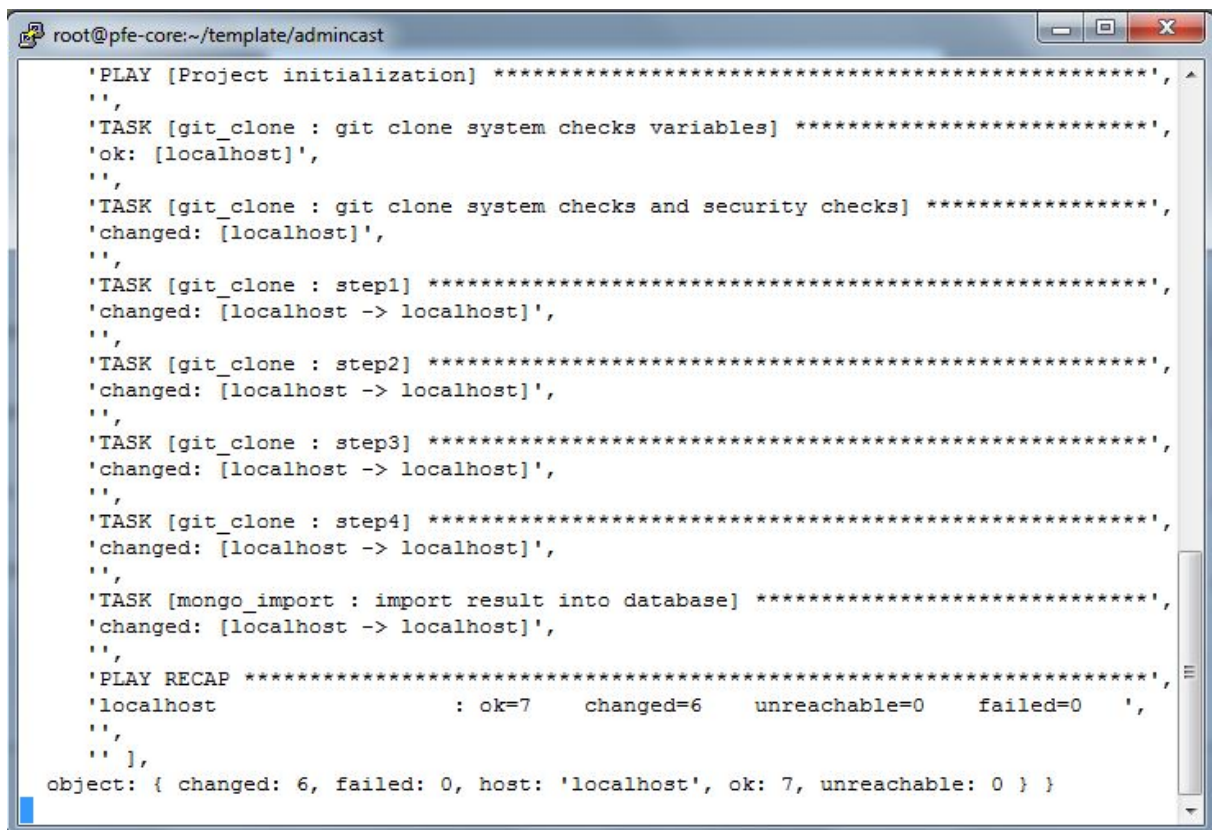


FIGURE 26. *Interface d'ajout d'un projet.*

Chaque projet de test possède sa propre base de données. Une fois l'administrateur a rempli les champs nécessaires, un playbook Ansible sera exécuté pour accomplir la phase de configuration. En premier lieu, Ansible va cloner une version générique des tests déployés sur GitLab et la placer sous le nœud maître. Ensuite, il crée une base de données dans laquelle il inscrit une collection des tests système et la liste des machines cibles. La dernière étape consiste à créer un dépôt Git intitulé avec le même nom du projet afin de garder une trace sur toutes les modifications. La figure 27 résume les étapes décrites précédemment.



```
root@pfe-core:~/template/admincast

'PLAY [Project initialization] *****',
'',
'TASK [git_clone : git clone system checks variables] *****',
'ok: [localhost]',
'',
'TASK [git_clone : git clone system checks and security checks] *****',
'changed: [localhost]',
'',
'TASK [git_clone : step1] *****',
'changed: [localhost -> localhost]',
'',
'TASK [git_clone : step2] *****',
'changed: [localhost -> localhost]',
'',
'TASK [git_clone : step3] *****',
'changed: [localhost -> localhost]',
'',
'TASK [git_clone : step4] *****',
'changed: [localhost -> localhost]',
'',
'TASK [mongo_import : import result into database] *****',
'changed: [localhost -> localhost]',
'',
'PLAY RECAP *****',
'localhost                : ok=7    changed=6    unreachable=0    failed=0 ',
'',
'  ]',
object: { changed: 6, failed: 0, host: 'localhost', ok: 7, unreachable: 0 }
```

FIGURE 27. Playbook de configuration du projet.

4.5 Gestion des playbooks Ansible

Un playbook Ansible est composé de deux parties essentielles :

- Un inventaire : ce fichier contient la liste des machines cibles et les paramètres d'accès SSH tel que le nom d'utilisateur et le mot de passe. Les nœuds peuvent être inclus dans un groupe afin d'exécuter un playbook Ansible sur un groupe particulier. Notre portail

offre la possibilité de mettre à jour cet inventaire sans recours à la console. Il suffit de choisir les groupes et/ou les hôtes cibles et cliquer sur le bouton « Update hosts ». Par la suite Ansible va s'occuper du reste du travail. L'interface de gestion de l'inventaire est montrée dans la figure 28.

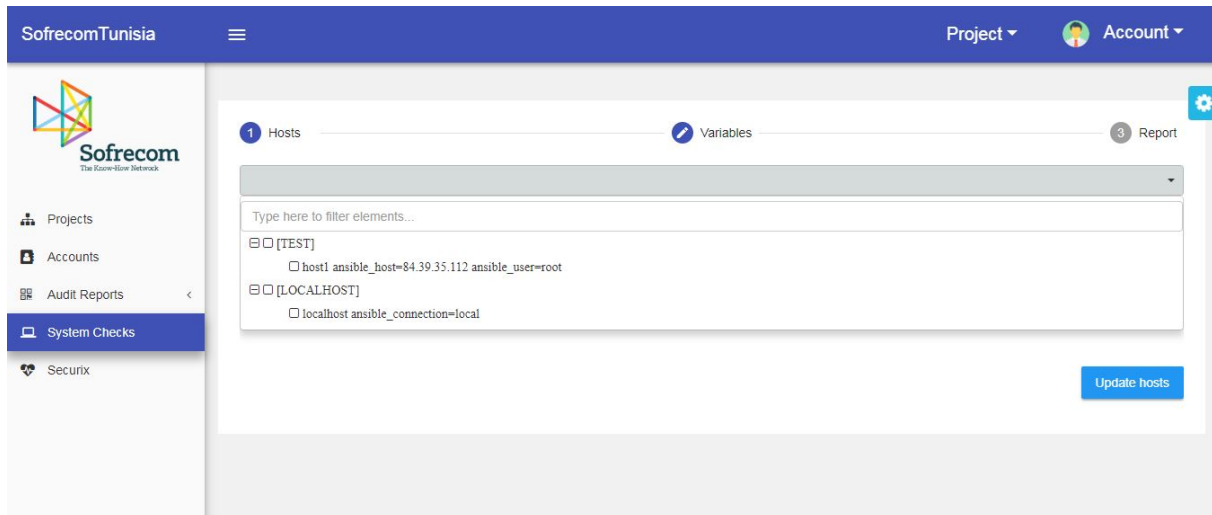


FIGURE 28. *Interface de gestion d'un inventaire Ansible.*

- Une liste des tâches : Chaque tâche englobe un ensemble de modules à exécuter dans une machine distante. Un module est une liste des commandes regroupées afin d'automatiser une configuration, une installation ou une vérification. Pour mieux répondre aux besoins de notre projet, nous avons développé notre propre module Ansible intitulé « system_check » (voir Annexe C).

Ce dernier prend en paramètre l'intitulé de check, le nom de la base de données où il stockera le résultat de test et la liste des tests à exécuter. Chaque test est composé par un identifiant du groupe, une description de test, une commande à exécuter et le résultat attendu. Une fois la commande exécutée, ce module va comparer le résultat obtenu à celle attendue. Par la suite un plugin Ansible va prendre en charge la conversion des résultats sous format JSON et le stockage des informations finales dans la base des données correspondantes (voir Annexe D). Vu la criticité de ce module, nous avons développé une interface graphique qui facilite la gestion des paramètres de ce module. Une fois l'utilisateur accède à cette interface, il peut consulter la liste des tests à exécuter.

Il aura un accès rapide à un test donné en faisant un filtrage sur l'un des paramètres cités précédemment.

L'utilisateur pourra modifier la commande de test ou le résultat souhaité. Cette interface est illustrée par la figure 29.

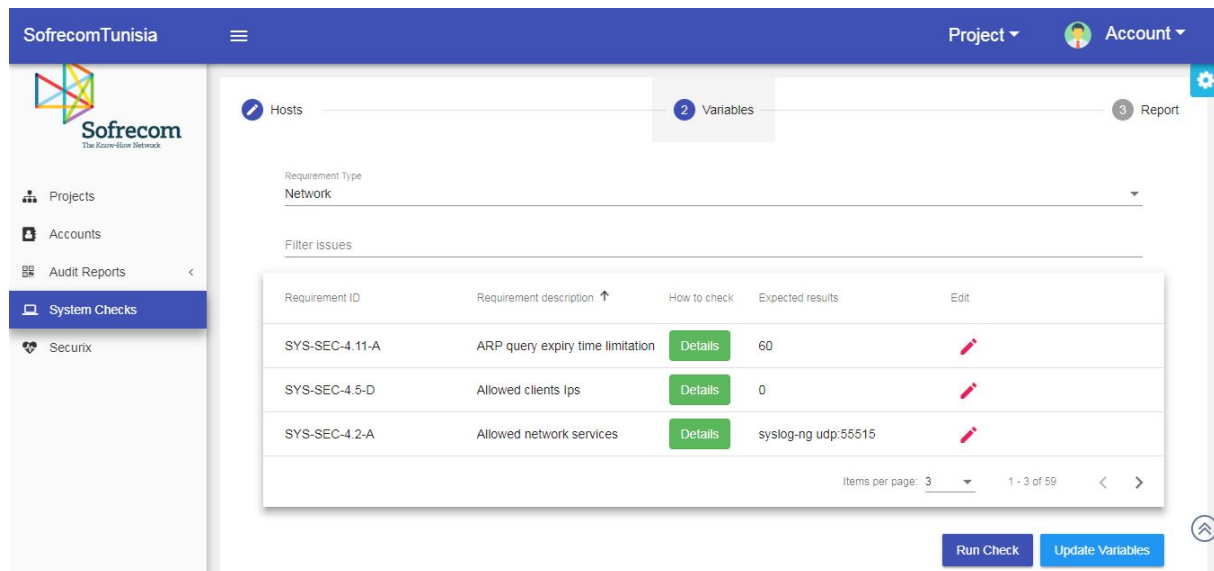


FIGURE 29. Interface de gestion des variables d'un playbook Ansible.

4.5.1 Gestion des scénarios de test

Une fois les machines cibles sont fixées et les méthodes de test sont configurées, notre audit est prêt à être exécuté. Notre application automatise deux types d'audits :

- Un audit système : Ce dernier a pour but de vérifier les configurations des différents services, la version des applicatifs et les droit d'accès des fichiers et des répertoires.
- Un audit sécurité : Il s'agit d'un ensemble des scripts Shell basé sur OPENSAP. C'est une solution de vérification de conformité standardisée pour infrastructures Linux de niveau entreprise. Il s'agit d'une ligne de spécifications maintenue par le NIST (« National Institute of Standards and Technology ») pour la maintenance de la sécurité de systèmes pour des systèmes en entreprise.

A chaque exécution d'un audit, un rapport détaillé est généré afin de mieux comparer la conformité des résultats finaux au cahier de charger. Ces rapports sont regroupés par machines.

Chacun contient des tableaux de bord et des courbes représentent une mesure de chaque round de check. Ces derniers vont nous permettre de rapidement afficher et analyser les résultats. La figure suivante représente un rapport de sécurité généré suite à l'exécution d'audit sécurité.

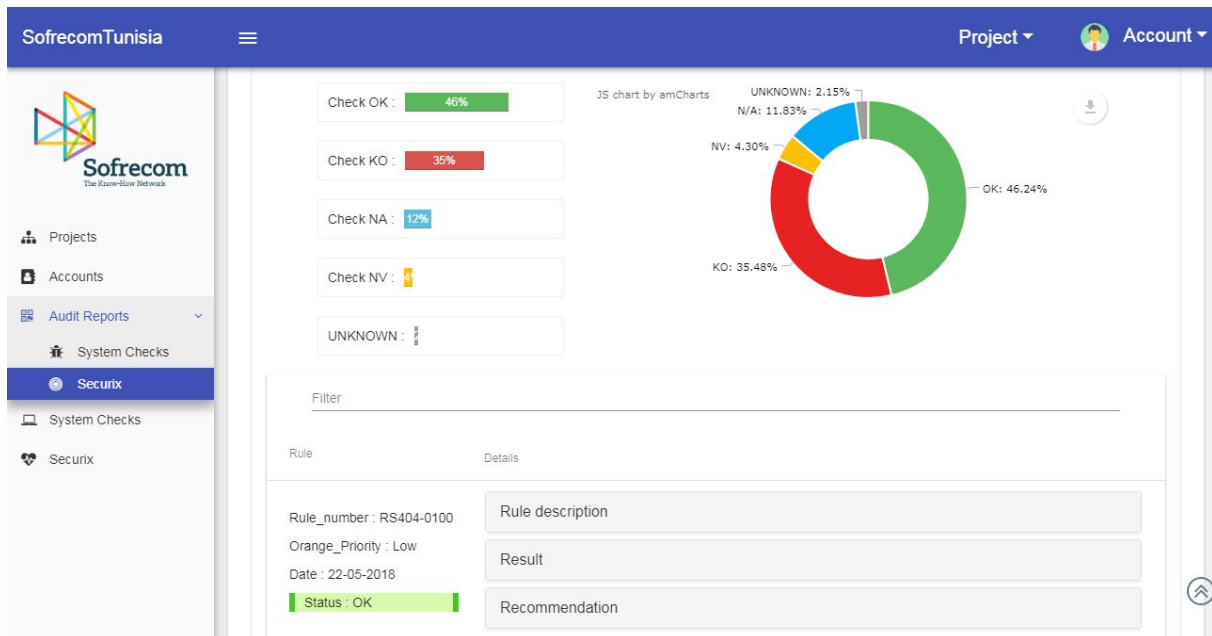


FIGURE 30. Rapport d'audit sécurité.

Une vue globale sur les résultats générés à la fin de chaque tour est résumée par la courbe de la figure suivante.



FIGURE 31. Courbe représentatif sur l'avancement des tests.

4.5.2 Gestion des utilisateurs

La gestion des utilisateurs s'effectue en accédant au menu « Accounts ». Cette interface est accessible seulement par l'administrateur de l'application. La figure ci-dessous, illustre la page qui visualise la liste des utilisateurs.

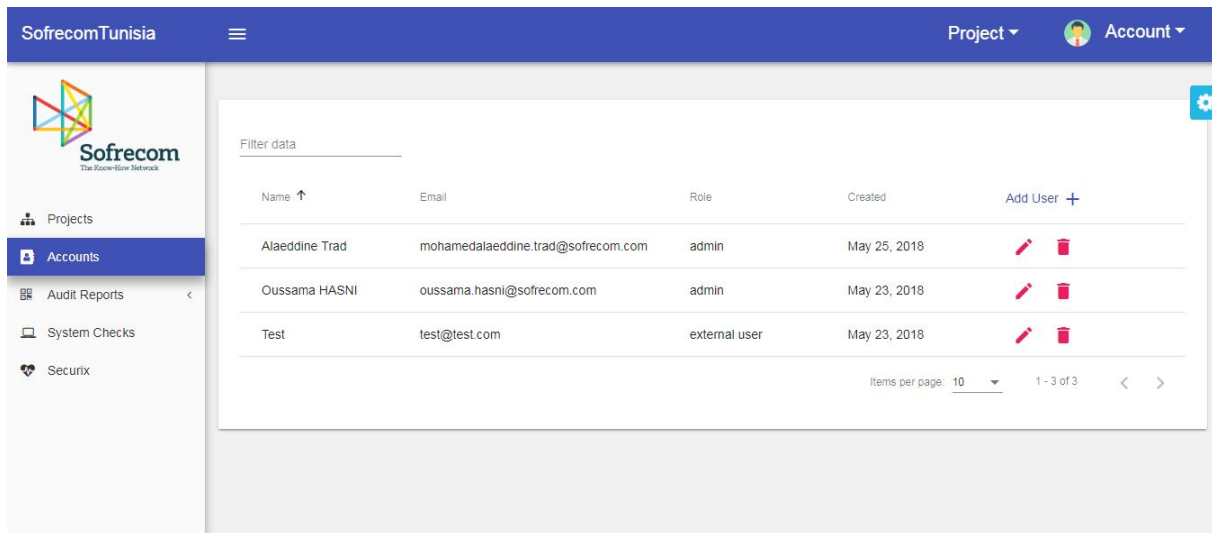


FIGURE 32. *Interface de gestion des utilisateurs.*

Il est possible d'ajouter un nouvel utilisateur, de modifier un utilisateur et de supprimer un utilisateur. Notre portail est accessible par plusieurs utilisateurs. Chacun possède des permissions qui diffèrent de l'autre. Donc il est indispensable d'attribuer un rôle à chaque nouvel utilisateur ajouté. Les utilisateurs externes n'ont pas la permission de gérer les projets de tests ou de modifier les comptes d'autre utilisateur. Cependant, ils ont le droit de modifier leurs propres profils en accédant à l'interface suivante.

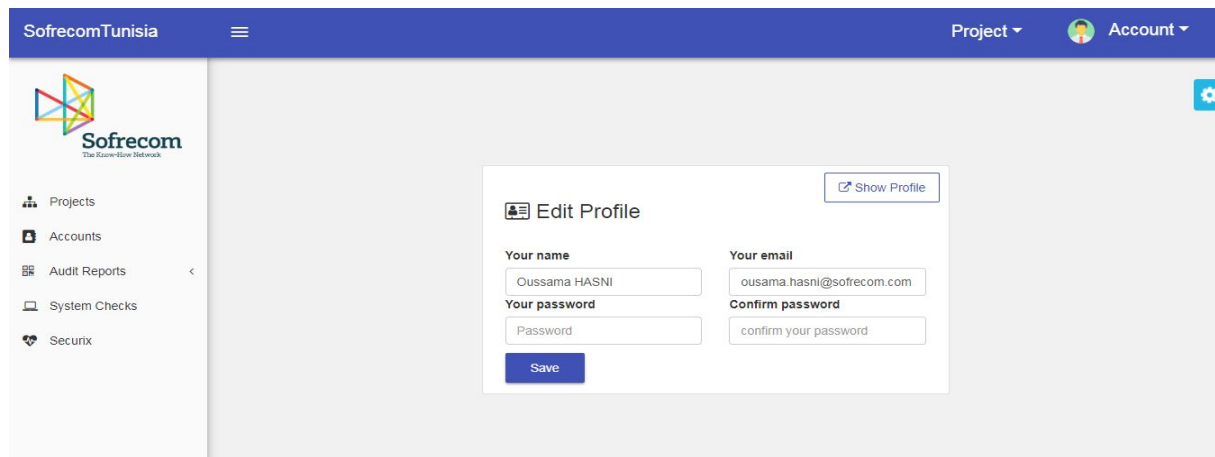


FIGURE 33. *Interface de modification d'un profil.*

Conclusion

Dans ce chapitre de réalisation, nous avons présenté les plates-formes matérielles et logicielles sur lesquelles nous avons développé notre projet, ainsi que les technologies employées. Nous avons, par la suite, présenté les interfaces les plus significatives de notre application.



Conclusion générale

L'automatisation des tests est devenue un besoin récurrent et stratégique pour le groupe Orange. Plusieurs solutions répondant à ce besoin existent sur le marché mais elles restent souvent limitées ou peu matures pour diverses raisons. Ce travail que nous avons effectué avait pour but de mettre en place un portail d'automatisation des tests pour la plateforme de rechargement électronique ZEBRA.

Le présent projet s'inscrit dans la stratégie de Sofrecom Tunisie à instaurer la culture DevOps au sein de ses équipes. Cela permet à l'entreprise de livrer des services de qualité avec des délais plus courts.

Pour atteindre cet objectif, nous avons débuté par situer notre projet dans son contexte méthodologique en adoptant une démarche DevOps. Cette approche a orienté les étapes de l'acheminement de notre travail. Ainsi, dans un premier temps, nous nous sommes intéressés à établir une étude de modèle de travail actuel. Celle-ci nous a permis de comprendre le processus, de dégager ses limites et de proposer une solution satisfaisante. Dans un deuxième temps, nous étions responsables de la mise en place de l'architecture de la solution ainsi que la collecte des besoins, la conception et la réalisation. Enfin, nous avons achevé l'étape de l'implémentation, au cours de laquelle nous avons essayé de mettre en place une solution qui répond pratiquement aux objectifs énoncés au début du stage. En effet, nous avons pu développer les besoins fonctionnels et non fonctionnels initialement dégagés.

A la fin de ce rapport, nous devons noter que ce stage de fin d'études nous a été très instructif de point de vue des connaissances acquises. En effet, c'était pour nous une bonne occasion de nous familiariser avec l'approche DevOps. D'autre part, il nous a procuré une opportunité pour consolider nos connaissances dans les différents langages de programmation comme JavaScript et Python. Au-delà de l'aspect technique, ce projet a été une opportunité pour découvrir le milieu professionnel avec tout ce qu'il implique de responsabilité, de discipline, de sérieux et de travail d'équipe.

Grâce à son caractère évolutif, le travail effectué sur la plateforme ZEBRA est bien applicable pour les autres plateformes de service jugées critiques. Dans le futur, notre système d'automatisation des tests pourrait être généralisé et partagé avec tous les collaborateurs Orange dans le but de résoudre le problème de lenteur du cycle de la validation en déchargeant l'équipe des tâches répétitives.



Nétographie

- [1] *SOFRECOM*. URL : <https://www.sofrecom.com/fr> (visité le 03-2018).
- [2] *RYTE*. URL : https://fr.ryte.com/wiki/Modele_en_spirale (visité le 03-2018).
- [3] *SUPINFO*. URL : <https://www.supinfo.com/articles/single/6811-introduction-devops> (visité le 03-2018).
- [4] *ENGICONSLT*. URL : <http://www.engiconsult.com/> (visité le 05-2018).
- [5] *DOCDOKU*. URL : <https://www.docdoku.com/> (visité le 05-2018).
- [6] *DOTNETDOJO*. URL : <https://www.dotnetdojo.com/mvvm/> (visité le 05-2018).
- [7] *EXCELLENTWEBWORLD*. URL : <https://excellentwebworld.com/> (visité le 05-2018).
- [8] *JETBRAINS*. URL : <https://www.jetbrains.com/idea/> (visité le 06-2018).
- [9] *STARUML*. URL : <http://staruml.io/> (visité le 06-2018).
- [10] *ROBOMONGO*. URL : <https://robomongo.org/> (visité le 06-2018).
- [11] *MDN*. URL : <https://developer.mozilla.org/fr/docs/Web/JavaScript> (visité le 06-2018).
- [12] *EXPRESSJS*. URL : <http://expressjs.com/fr/> (visité le 06-2018).
- [13] *PYTHON*. URL : <https://docs.python.org/fr/3/> (visité le 06-2018).
- [14] *DOCUBUNTU*. URL : https://doc.ubuntu-fr.org/tutoriel/script_shell (visité le 06-2018).
- [15] *YAML*. URL : <http://yaml.org/> (visité le 06-2018).
- [16] *MONGODB*. URL : <https://www.mongodb.com/> (visité le 06-2018).

- [17] *ANSIBLE*. URL : <https://www.ansible.com/> (visité le 06-2018).
- [18] *GITLAB*. URL : <https://about.gitlab.com/> (visité le 06-2018).
- [19] *UPGUARD*. URL : <https://www.upguard.com/articles/ansible-vs-chef> (visité le 06-2018).
- [20] *TRENDSGOOGLE*. URL : <https://trends.google.fr/trends/?geo=FR> (visité le 06-2018).
- [21] *STACKSHARE*. URL : <https://stackshare.io/stackups/gitlab-vs-svn> (visité le 06-2018).
- [22] *ORANGE*. URL : <https://www.orange.tn/> (visité le 03-2018).



Glossaire

API Application Programming Interface

JSON Java Script Object Notation

MEAN MongoDB Express.js Angular Node.js

NIST National Institute of Standards and Technology

OLT Orange Labs Tunisie

SSH Secure Shell

UML Unified Modeling Language

YAML Yet Another Markup Language



ANNEXE A : ZEBRA

A.1 Cadre de l'application

L'application ZEBRA est une plateforme de service qui permet le rechargement des comptes de téléphone mobile. Elle est déployée dans les filiales du groupe orange en Afrique, le moyen Orient, l'Amérique et l'Europe.

Actuellement ZEBRA est présent en Jordanie, Kenya, République centrafricaine, Cameroun, Guinée, Guinée-Bissau, Madagascar, Île Maurice, Mali, Botswana, Côte d'Ivoire, Niger, Sénégal, Uganda, Egypt, Moldavie, Tunisie et Iraq.

Le rechargement est effectué via le web (application web) ou bien à travers le téléphone mobile (USSD, SMS ou bien STK).

Zebra est développé autour de l'architecture 2-tiers. Chaque tiers est basé sur le principe de tolérance aux pannes.

A.2 Vue fonctionnelle globale

Zebra (également connue sous le nom de PreTUPS) est une application de gestion des comptes de téléphonie mobile par le biais d'une application Web ou bien par les terminaux mobiles (SMS / USSD).

Le système Top Up prépayé de PreTUPS gère le cycle de vie complet de la chaîne de distribution Top Up qui commence à partir de l'opérateur et se termine par le rechargement des abonnés prépayés.

L'objectif du système est le suivant :

- Gérer le canal de distribution pour l'opérateur.
- Recharger les abonnés prépayés, Internet et ligne fixe en utilisant le canal de distribution.
- Gérer les virements entre les différents comptes des clients.

Un client (utilisateur final) peut recharger son téléphone mobile prépayé, son compte internet et même sa ligne fixe en utilisant ZEBRA. En outre Il peut payer sa facture post-payé.

Toutes les opérations (transfert / retrait / remboursement / top up) sont effectuées par le biais d'un terminal mobile et / ou une interface web.

Zebra remplace le système de coupons classique par une solution électronique qui fournit une large gamme d'actifs.

Deux grands domaines sont couverts :

- CP2P : client Peer to Peer - c'est-à-dire le transfert de crédits entre les comptes des clients.
- RP2P : Retailer Peer to Peer – c'est-à-dire la gestion de la recharge électronique au sein d'un réseau de distribution d'un détaillant en contrôlant les opérations de Transferts, retraits et remboursements entre les membres de ses canaux de distribution ainsi que la recharge pour les utilisateurs finaux.

Le schéma suivant illustre l'architecture fonctionnelle globale de ZEBRA.

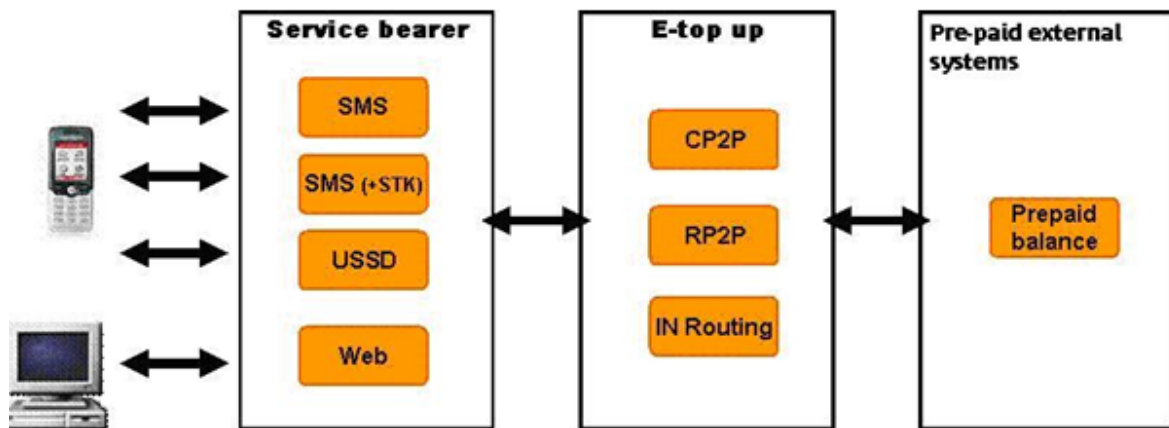


FIGURE 34. *Architecture fonctionnelle globale de ZEBRA*

A.3 Architecture globale de ZEBRA

L'architecture technique de la plate-forme est dépliée dans cette partie du rapport. Une vue générique est d'abord montrée, présentant dans son ensemble, les trois zones principales, ainsi que les services externes ou d'autres plates-formes installées dans l'infrastructure locale de la filiale. La figure 35 montre la plate-forme Zebra au sein des deux sites primaire et secondaire de la filiale. Il présente les trois zones et leurs principaux composants, ainsi que l'interaction avec les services externes. Chaque zone a une fonction dédiée :

La « zone publique » prend le rôle de sécurisation des flux externes :

- Elle constitue une DMZ pour les connexions provenant des réseaux externes (principalement l'Internet) qui sont considérés comme des réseaux non sécurisés et ainsi que l'accès VPN des partenaires.
- Elle vérifie et filtre ces connexions externes avant de les transmettre à la zone privée.
- La zone publique est à la frontière du monde extérieur et le réseau privé. Il est ni aussi sûr que le réseau de confiance interne, ni comme l'insécurité que l'Internet.

La « zone privée » :

- Elle est considérée comme zone de confiance. Elle accepte uniquement les connexions provenant des réseaux de confiance.

- Elle contient les serveurs qui exécutent l'application et les services PreTUPS.

La « zone d'exploitation locale » assure le management et l'administration de la plateforme du service.

- Elle est considérée comme zone de confiance. Elle décline toute connexion des réseaux externes.
- Elle contient les serveurs exécutant les fonctions d'exploitation et les outils de supervision.

La « zone du stockage » permet l'administration et la gestion du stockage.

- Elle n'est pas une zone de réseau.
- Elle contient des unités du stockage pour toutes les données partagées ou accessibles par les serveurs.

La figure suivante montre les différents composants de la zone d'exploitation locale de l'application ZEBRA.

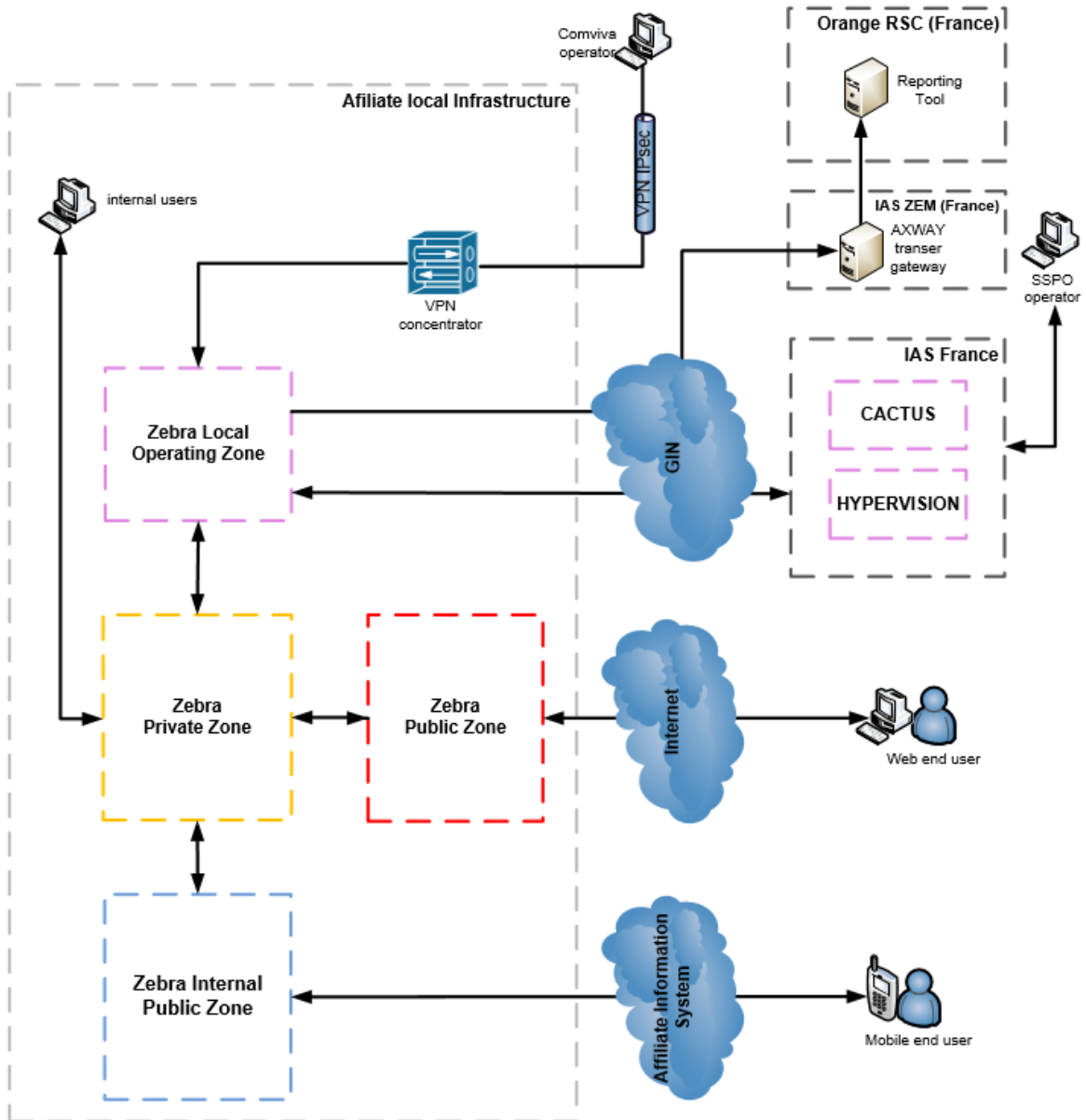


FIGURE 35. Marco Design de l'architecture ZEBRA



ANNEXE B : Ansible

B.1 Playbooks

Un playbook est un fichier contenant du code écrit en Yaml (une forme de syntaxe non balisée). Ce code est structuré en plusieurs parties. Il contient des appels à des rôles et/ou des tâches d'exécution de modules Ansible. Ces modules contiennent des groupements de variables. L'exécution d'un playbook Ansible est faite sur des machines cibles définies dans un fichier inventaire. La figure ci-dessous représente un exemple d'un playbook Ansible.

```
---
- hosts: anstwo.example.com
  user: root
  tasks:
    - name: vgcreate
      lvg:
        vg: vgdata
        pvs: /dev/sdb5,/dev/sdb6

    - name: lvcreate
      lvol:
        vg: vgdata
        lv: dataone
        size: 1500M

    - name: create filesystem
      filesystem:
        fstype: ext4
        dev: /dev/vgdata/dataone

    - name: mount lvm
      mount:
        name: /tmp/lvdata
        src: /dev/vgdata/dataone
        fstype: ext4
        state: mounted
```

15,0-1

FIGURE 36. *Un playbook Ansible*

B.2 Rôles

Un rôle est un ensemble de tâches à exécuter pour aboutir à un but (par exemple configuration d'un serveur Nginx). Il peut prendre plusieurs variables en paramètre qui seront accessibles aux différentes tâches. Il est aussi possible qu'un rôle ne prenne aucune variable en paramètre. La figure suivante illustre la structure d'un rôle Ansible.

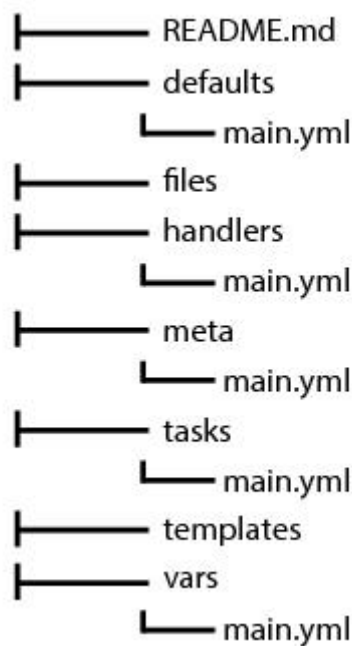


FIGURE 37. *Structure d'un rôle Ansible.*

B.3 Variables

Une variable est connue sous le nom vars. Elle sera passée en input d'une tâche. La valeur d'une variable est introduite dynamiquement. Elle peut être le résultat d'exécution d'une tâche ou un paramètre système qui varie d'un environnement à un autre. Une variable peut être en format YAML, ou JSON.

B.4 Inventaires

Un inventaire n'est tout simplement qu'un fichier qui va contenir la ou les machines cibles sur lesquelles l'utilisateur voudra déployer ses composants ou son application. Cependant il embarque avec lui une notion spécifique qui est celle des groupes.

Un groupe dans l'univers d'Ansible est une entité qui va contenir la liste de différentes machines sur lesquelles nous voulons faire tourner nos playbooks. Comme il est illustré dans la figure ci-dessous, le fichier est donc composé du nom du groupe entre crochets, puis à chaque ligne suivante nous avons le nom ou l'adresse IP d'une machine.

```
[webservers]
192.168.35.140
192.168.35.150

[datacache]
192.168.45.45

[appservers]
192.168.100.1
192.168.100.2

[dbservers]
172.35.0.10
```

FIGURE 38. *Structure d'un inventaire Ansible*



ANNEXE C : Développement d'un module Ansible

C.1 Modules

La communauté Ansible fournit une riche famille des modules permettant de réaliser des tâches très diverses. Ces modules sont développés en langage python, tout en respectant une forme bien déterminé afin d'assurer la comptabilité de ce dernier avec la syntaxe Ansible. Un des grands avantages d'Ansible est de permettre de créer nos propres modules. Il suffit que votre code soit exécutable, l'interface avec Ansible se fait via des paramètres en ligne de commande et la sortie texte du programme.

C.2 System check

Pour répondre aux besoins de projet, nous avons pensé à développer notre propre module appelé (system check). Ce dernier va nous permettre d'automatiser les taches des check systèmes. Il va nous permettre de vérifier la comptabilité de la configuration des différents serveurs de la plateforme ZEBRA avec le cahier des charges fourni par les architectes. Ce module sera le socle de la suite de développement de l'application web grâce aux diverses fonctionnalités offertes par ce dernier. Le code source de ce module est représenté par la figure suivante.

```

1  #!/usr/bin/python
2
3  import socket
4  import sys
5  import json
6  import os
7  import shlex
8  import subprocess
9
10 ANSIBLE_METADATA = {
11     'metadata_version': '1.1',
12     'status': ['preview'],
13     'supported_by': 'Sofrecom Tunisia'
14 }
15
16 DOCUMENTATION = '''
17 ---
18 module: system_check
19 authors:
20     - Oussama HASNI
21 '''
22
23 EXAMPLES = '''
24 ---
25 - system_check:
26     Requirement_Type: "Network"
27     DataBase: "{{ dbname }}"
28     Task_list: [
29         "{{SYS_SEC_1}}",
30         "{{SYS_SEC_2}}"
31     ]
32 '''
33
34 RETURN = '''
35     "checkList" : [
36         {
37             "Final_result" : "KO",
38             "Requirement_ID" : "SYS-SEC-4.1-A",
39             "How_to_check" : "rpm -qa | grep dbx | wc -l ",
40             "Requirement_description" : "Static IPs",
41             "Error" : "",
42             "Actual_results" : "3",
43             "Expected_results" : "20"
44         },
45         {
46             "Final_result" : "KO",
47             "Requirement_ID" : "SYS-SEC-4.2-A",
48             "How_to_check" : "netstat -ltn | sed 1,2d | awk '{n=split($4,a,\":\"); k=split($9,b,\"/\"); print b[k]\" \" \"$1\"\":\"$a[n]}' | grep -i \"xps tcp:5666\"",
49             "Requirement_description" : "Allowed network services",
50             "Error" : "",
51             "Actual_results" : "",
52             "Expected_results" : "xps tcp:5666"
53         }
54     ]
55 '''
56
57
58
59 from ansible.module_utils.basic import AnsibleModule
60

```

```

61 def run_module():
62     # define the available arguments/parameters that a user can pass to
63     # the module
64     module_args = dict(
65         Requirement_Type=dict(type='str', required=True),
66         Task_list=dict(type='list', required=True),
67         DataBase=dict(type='str', required=True)
68     )
69
70     module = AnsibleModule(
71         argument_spec=module_args,
72         supports_check_mode=True
73     )
74
75
76     # seed the result dict in the object
77     # we primarily care about changed and state
78     # change is if this module effectively modified the target
79     # state will include any data that you want your module to pass back
80     # for consumption, for example, in a subsequent task
81     result = dict(
82         changed=False,
83         Tasks=[],
84         Requirement_Type=module.params['Requirement_Type'],
85         hostname=socket.gethostname(),
86         dataBase=module.params['DataBase']
87     )
88
89     # the AnsibleModule object will be our abstraction working with Ansible
90     # this includes instantiation, a couple of common attr would be the
91
92
93     # args/params passed to the execution, as well as if the module
94     # supports check mode
95     module = AnsibleModule(
96         argument_spec=module_args,
97         supports_check_mode=True
98     )
99
100     # if the user is working with this module in only check mode we do not
101     # want to make any changes to the environment, just return the current
102     # state with no modifications
103     if module.check_mode:
104         return result
105
106     # manipulate or modify the state as needed (this is going to be the
107     # part where your module will do what it needs to do)
108     i=0
109     while i < len(module.params['Task_list']):
110         proc = subprocess.Popen(
111             module.params['Task_list'][i][2],
112             stdout=subprocess.PIPE,
113             stderr=subprocess.PIPE,
114             shell=True
115         )
116         (res, err) = proc.communicate()
117         res = res.replace("\n", "")
118         if len(err) != 0:
119             res = "error!"
120         if res == module.params['Task_list'][i][3]:
121             final_res = "OK"
122         else:
123             final_res = "KO";
124         result['Tasks'].append({
125             'Requirement_ID': module.params['Task_list'][i][0],

```



```
126     'How_to_check': module.params['Task_list'][i][2],
127     'Expected_results': module.params['Task_list'][i][3],
128     'Actual_results': res, 'Error': err,
129     'Final_result': final_res
130     })
131     i=i+1
132
133
134     # use whatever logic you need to determine whether or not this module
135     # made any modifications to your target
136     if module.params['Task_list']:
137         result['changed'] = True
138
139
140     # in the event of a successful module execution, you will want to
141     # simple AnsibleModule.exit_json(), passing the key/value results
142     module.exit_json(**result)
143
144 def main():
145     run_module()
146
147 if __name__ == '__main__':
148     main()
```

FIGURE 39. Code source du module « System check »



ANNEXE D : Développement d'un plugin Ansible

D.1 Callback Plugin

Les plugins Callback permettent d'ajouter de nouveaux comportements à Ansible lors de la réponse aux événements. Par défaut, les plugins de callback contrôlent la plupart des sorties affichées lors de l'exécution des programmes de ligne de commande, mais peuvent également être utilisés pour ajouter des sorties, intégrer d'autres outils et rassembler les événements dans un backend de stockage.

D.2 JSON callback plugin

Malgré les nombreuses fonctionnalités offertes par Ansible, ce dernier possède des points de faiblesse. Par défaut la sortie standard d'Ansible est la console. Donc nous ne pouvons pas garder une trace sur les résultats d'exécution. Ceci provoque une perte des données et paralyse la suite de processus. Pour cette raison nous avons pensé à développer un plugin pour couvrir ce besoin. Ce dernier prend en charge la conversion de résultat sous format JSON. Par la suite, il établit une connexion avec la base de données et stocke le résultat. Les données stockées seront traitées par notre application web dans le but d'améliorer l'interaction homme-machine,

et faciliter l'interprétation des rapports générés. La figure suivante illustre le code source de ce plugin.

```

1  from __future__ import (absolute_import, division, print_function)
2  __metaclass__ = type
3
4  DOCUMENTATION = '''
5      callback: json
6      short_description: Ansible screen output as JSON
7      version_added: "1.1"
8      description:
9          - This callback converts all events into JSON output then stock
10             results into dataBase.
11  '''
12
13  import json
14
15  from ansible.plugins.callback import CallbackBase
16
17  from pymongo import MongoClient
18  from datetime import datetime
19
20  class JSONEncoder(json.JSONEncoder):
21      def default(self, o):
22          if isinstance(o, ObjectId):
23              return str(o)
24          return json.JSONEncoder.default(self, o)
25
26  class CallbackModule(CallbackBase):
27      CALLBACK_VERSION = 2.0
28      CALLBACK_TYPE = 'stdout'
29      CALLBACK_NAME = 'json'
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55  def v2_runner_on_ok(self, result, **kwargs):
56      host = result._host
57      res = {}
58      res["hostname"] = result._result['hostname']
59      res["timestamp"] = self.results[-1]['play_start_time']
60      res["checkname"] = self.results[-1]['name']
61      res["requirementType"] = result._result['Requirement_Type']
62      res["checkList"] = result._result['Tasks']
63      self.results[-1]['tasks'][-1][host.name] = res
64      DATABASE_NAME = result._result['dataBase']
65      client = MongoClient('localhost', 27017)
66      db = client[DATABASE_NAME]
67      db['system_checks_report'].insert(res)
68
69  def v2_playbook_on_stats(self, stats):
70      """Display info about playbook statistics"""
71
72      hosts = sorted(stats.processed.keys())
73
74
75      v2_runner_on_failed = v2_runner_on_ok
76      v2_runner_on_unreachable = v2_runner_on_ok
77      v2_runner_on_skipped = v2_runner_on_ok
78

```

```
55 def v2_runner_on_ok(self, result, **kwargs):
56     host = result._host
57     res = {}
58     res["hostname"] = result._result['hostname']
59     res["timestamp"] = self.results[-1]['play_start_time']
60     res["checkname"] = self.results[-1]['name']
61     res["requirementType"] = result._result['Requirement_Type']
62     res["checkList"] = result._result['Tasks']
63     self.results[-1]['tasks'][-1][host.name] = res
64     DATABASE_NAME = result._result['dataBase']
65     client = MongoClient('localhost', 27017)
66     db = client[DATABASE_NAME]
67     db['system_checks_report'].insert(res)
68
69 def v2_playbook_on_stats(self, stats):
70     """Display info about playbook statistics"""
71
72     hosts = sorted(stats.processed.keys())
73
74
75 v2_runner_on_failed = v2_runner_on_ok
76 v2_runner_on_unreachable = v2_runner_on_ok
77 v2_runner_on_skipped = v2_runner_on_ok
78
```

FIGURE 40. Code source du plugin « JSON Callback ».



ANNEXE E : Exemple d'un test système

E.1 Les tests système

Notre portail permet de vérifier la configuration du système d'exploitation de tous les serveurs de la plateforme Zebra. Telle que les paramètres du noyau, les droits d'accès et les services. La figure suivante représente un exemple de tests système exécuté par Ansible.

```

1  [SYS_SEC_9: [
2    'SYS-SEC-4.3-A',
3    'Disable IPv6',
4    "sysctl -n net.ipv6.conf.all.autoconf",
5    'NONE'
6  ]
7  [SYS_SEC_10: [
8    'SYS-SEC-4.5-A',
9    'TCP-Wrappers installation',
10   "rpm -qa | grep \"tcp_wrappers-[0-9]\"",
11   'tcp_wrappers-[0-9]'
12 ]
13 [SYS_SEC_11: [
14   'SYS-SEC-4.5-B',
15   'Default BLOCK policy',
16   "cat /etc/hosts.allow | grep -v \"^#\" | grep -o \"ALL:ALL:DENY\"",
17   'ALL:ALL:DENY'
18 ]
19 [SYS_SEC_12: [
20   'SYS-SEC-4.4-A',
21   'Promiscuous mode',
22   "ifconfig \"interface\" | grep -o PROMISC | uniq",
23   ''
24 ]
25 [SYS_SEC_13: [
26   'SYS-SEC-4.5-B',
27   'Default BLOCK policy',
28   "cat /etc/hosts.deny | grep -v \"^#\" | grep -v \"ALL:ALL\" | wc -l",
29   '0'
30 ]
31 [SYS_SEC_14: [
32   'SYS-SEC-4.5-C',
33   'PARANOID mode',
34   "cat /etc/hosts.allow | grep -v \"^#\" | grep -o \"ALL:PARANOID:deny\"",
35   'ALL:PARANOID:deny'
36 ]

```

FIGURE 41. Exemple d'un test système.

La gestion des tests à travers la console peut être sujet des erreurs humaines. Vu la criticité de cette tâche nous avons pensé à développer une interface graphique pour gérer les variables d'un playbook.