

Université de La Manouba
École Nationale des Sciences de l'Informatique



Rapport de Mémoire de Fin d'Études
Présenté en vue de l'obtention du titre
D'INGÉNIEUR EN INFORMATIQUE

Réalisé par

Donia HAMMAMI

Sujet

REFONTE DE L'APPLICATION MONOLITHIQUE BOTOOL
VERS UNE ARCHITECTURE MICROSERVICES



Organisme d'accueil : Talan Tunisie

Nom du responsable : M. Behjet BOUSSOFARA

Encadré par : Mme. Sonda FRIKHA et M. Adem BEN AMOR

Supervisé par : Mme. Chiraz ZRIBI

Adresse : 10 rue de l'énergie solaire, impasse n° 1, Cedex 2035 Charguia 1 Tunis

Année Universitaire : 2017-2018

Signature et cachet de l'entreprise

Mme. Sonda Frikha
(Talan Tunisie Consulting)

Dédicaces

A mes très chers parents,

Aucun hommage ne pourrait être à la hauteur de l'amour et de l'affection dont ils ne cessent de me combler. Qu'ils trouvent dans ce travail un témoignage de mon profond amour et éternelle reconnaissance.

Que dieu leur procure bonne santé et longue vie. A mes chères soeurs Abir et Salma pour leur soutien durant toutes ces années,

Une spéciale dédicace à cette personne qui compte déjà énormément pour moi, et pour qui je porte beaucoup de tendresse et d'amour.

A toi Mohamed, sans ton aide, tes conseils et tes encouragements ce travail n'aurait vu le jour.

A tous mes amis, Je dédie ce travail ...

Remerciements

Nous tenons, avant de présenter notre travail, à exprimer notre grande reconnaissance envers les personnes qui nous ont, de près ou de loin, apporter leurs soutiens. Qu'ils trouvent ici collectivement et individuellement l'expression de toute notre gratitude.

Nous tenons à remercier tout particulièrement et à témoigner toute notre reconnaissance à Mr Adem Ben Amor pour l'expérience enrichissante et pleine d'intérêt qu'il nous a fait vivre durant la période du stage et pour tous les conseils et les informations qu'il nous a prodigué.

à Mme Sonda Frikha, qui a proposé et dirigé ce travail. Je lui exprime mes profondes reconnaissances pour la confiance qu'elle m'a accordée et son aide précieux, son encouragement et ses conseils continus durant ce stage.

Aussi, nous exprimons notre parfaite reconnaissance et nos remerciements à notre encadrant Dr Chiraz Zribi pour le temps qu'elle a bien voulu consacrer à l'encadrement et le suivi de ce travail ; les conseils qu'il nous a prodigué après son minutieuse lectures et pour les réunions qui ont rythmées les différentes étapes de la rédaction de ce rapport. Les discussions que nous avons tenus ont permis d'orienter ce travail d'une manière sûre et pertinente. Nous le remercions vivement pour son effort, sa disponibilité et surtout ses conseils qui ont largement contribué à rehausser la valeur de ce travail.

Que les membres de jury trouvent, ici, l'expression de nos remerciements pour l'honneur qu'ils nous font en acceptant de juger ce travail.

Table des matières

Introduction générale	1
1 Présentation générale	3
I Présentation de l'organisme d'accueil	3
I.1 Talan Tunisie Consulting	3
I.2 Secteurs d'activités	4
II Contexte du projet	5
II.1 Présentation de l'outil BOTOOL	5
II.2 Limites et critiques de l'existant	6
II.3 Impact de l'architecture monolithique sur l'outil BOTOOL	7
III Solution proposée et travail demandé	10
IV Méthodologie de gestion de projet	10
IV.1 Présentation de la méthode SCRUM	11
IV.2 Les rôles dans SCRUM	11
IV.3 Le sprint agile	12
Conclusion	12
2 Concepts théoriques	13

TABLE DES MATIÈRES

I	L'architecture microservices	13
II	Caractéristiques de l'architecture microservices	14
II.1	La division en composants via les services	14
II.2	L'organisation autour des capacités métiers	14
II.3	Un produit, pas un projet	15
II.4	Une gouvernance décentralisée	15
II.5	Gestion de données décentralisée	15
II.6	Les extrémités intelligentes et les canaux stupides	15
II.7	Automatisation de l'infrastructure	16
II.8	Conception pour l'échec	16
II.9	Une conception évolutive	16
III	Les concepts liés à l'architecture microservices	16
III.1	La conception pilotée par le domaine	17
III.2	Développement à base de composants	17
III.3	Persistance polyglotte	18
IV	Bilan sur l'architecture microservices	18
IV.1	Motivations de l'utilisation des microservices	19
IV.2	Avantages de l'architecture microservices	19
IV.3	Défis soulevés par les microservices	19
IV.4	Inconvénients de l'architecture microservices	20
	Conclusion	20

3	Analyse et spécification des besoins et architecture en microservices proposée	21
----------	---	-----------

I	Capture des besoins globaux	21
I.1	Définition des acteurs	21
I.2	Analyse des besoins globaux	22
I.3	Backlog de produit	23
II	Spécification des besoins fonctionnels globaux	25
III	Division de BOTOOL en microservices	26
IV	L'architecture de l'application en microservices	27
	Conclusion	28
4	Sprint 1 : Microservice Administration	29
I	Sprint Backlog	29
II	Spécification fonctionnelle du microservice Administration	30
II.1	Diagrammes de cas d'utilisation du microservice Administration . .	30
II.2	Description de quelques scénarii	33
III	Conception du microservice Administration	35
III.1	Architecture logicielle du microservice Administration	35
III.2	Diagramme de paquets du microservice Administration	36
III.3	Diagramme de classes du microservice Administration	37
III.4	Description du scénario de mise à jour d'un compte utilisateur . . .	39
IV	Réalisation du microservice Administration	40
V	Phase de test du microservice Administration	44
	Conclusion	45
5	Sprint 2 : Microservice Client	46

TABLE DES MATIÈRES

I	Sprint Backlog	46
II	Spécification fonctionnelle du microservice Client	47
II.1	Diagramme de cas d'utilisation du microservice Client	47
II.2	Description de quelques scénarii	48
III	Conception du microservice Client	49
III.1	Architecture physique du microservice Client	49
III.2	Diagramme de classes du microservice Client	50
III.3	Diagramme d'activités consultation informations client	51
IV	Réalisation du microservice Client	52
V	Phase de test du microservice Client	54
	Conclusion	54
6	Sprint 3 : Microservice Journalisation	55
I	Sprint Backlog	55
II	Spécification fonctionnelle du microservice Journalisation	56
II.1	Diagramme de cas d'utilisation du microservice Journalisation . . .	56
II.2	Scénario du cas d'utilisation consulter le journal d'opérations . . .	56
III	Conception du microservice Journalisation	57
III.1	Architecture logicielle du microservice Journalisation	57
III.2	Diagramme de paquets du microservice Journalisation	58
III.3	Diagramme de classes du microservice Journalisation	59
IV	Réalisation du microservice Journalisation	60
V	Phase de test du microservice Client	61
	Conclusion	61

7	Intégration des microservices de l'outil BOTOOL	62
I	Environnements de travail	62
I.1	Environnements de développement matériel	62
I.2	Environnements de développement logiciel	63
II	Intégration des microservices	65
II.1	Service de découverte Eureka	66
II.2	L'API Gateway Zuul	67
II.3	Communication entre microservices	69
III	Chronogramme	69
	Conclusion	70
	Conclusion générale et perspectives	71
	Bibliographie	73
	Netographie	74

Table des figures

1.1	Talan Consulting à travers le monde	4
1.2	Architecture monolithique de BOTOOL	9
2.1	Carte de contexte	17
3.1	Diagramme de cas d'utilisation global	25
3.2	Carte de contexte de BOTOOL	26
3.3	Les composants d'une architecture microservices	28
4.1	Diagramme de cas d'utilisation gestion des utilisateurs	31
4.2	Diagramme de cas d'utilisation gestion des rôles utilisateurs	31
4.3	Diagramme de cas d'utilisation configuration des écrans	32
4.4	Diagramme de séquence système d'authentification	33
4.5	Diagramme de séquence système de modification de rôle utilisateur	34
4.6	Architecture logicielle du microservice Administration	35
4.7	Diagramme de paquets du microservice Administration	36
4.8	Diagramme de classes du microservice Administration	37
4.9	Diagramme de séquence du scénario de mise à jour d'un utilisateur	39
4.10	Interface d'authentification de l'outil BOTOOL	40

4.11	Liste des utilisateurs de BOTOOL	41
4.12	Ajout d'un nouveau utilisateur de BOTOOL	41
4.13	Liste des rôles	42
4.14	Mettre à jour un rôle existant	42
4.15	Recherche d'un rôle inexistant	43
4.16	Configuration des écrans de BOTOOL	43
4.17	Recherche d'un écrans de BOTOOL	44
4.18	Mise à jour de la configuration d'un écrans de BOTOOL	44
4.19	Tests unitaires du microservice Administration	45
5.1	Diagramme de cas d'utilisation consulter information client	47
5.2	Diagramme de séquence consulter information client	48
5.3	Diagramme de déploiement du microservice Client	49
5.4	Diagramme de classes du microservice Client	50
5.5	Diagramme d'activités du microservice Client	52
5.6	Consulter informations client	53
5.7	Résultat de consultation informations client	53
5.8	Résultat de consultation informations client en cas d'échec	54
5.9	Test unitaire de microservice Client	54
6.1	Diagramme de cas d'utilisation du microservice Journalisation	56
6.2	Diagramme de séquences du microservice Journalisation	57
6.3	Architecture logicielle du microservice Journalisation	58
6.4	Diagramme de paquets du microservice Journalisation	58

TABLE DES FIGURES

6.5	Diagramme de classes du microservice Journalisation	59
6.6	Liste des opérations effectuées	60
6.7	Test unitaire de microservice Journalisation	61
7.1	Architecture technique en microservices	66
7.2	Enregistrement des microservices de BOTOOL dans l'annuaire	67
7.3	Dashboard Eureka	67
7.4	Fonctionnement de l'API Gateway Zuul	68
7.5	Tolérance aux pannes au niveau de l'API Gateway	68
7.6	Diagramme de gantt réel	70

Liste des tableaux

1.1 Inconvénients de l’architecture monolithique 8

3.1 Backlog du produit 23

3.2 Partitionnement des user stories par sprint 27

4.1 Backlog du sprint 1 29

4.2 Les principales classes conçues pour l’élaboration du microservice 38

5.1 Backlog du sprint 2 46

5.2 Les principales classes conçues pour l’élaboration du microservice 51

6.1 Backlog du sprint 3 55

6.2 Les principales classes conçues pour l’élaboration du microservice 59

Introduction générale

*L*e pouvoir d'une entreprise sur le marché peut être évalué en fonction du nombre de ses clients et partenaires et leurs niveaux de satisfaction par les produits et les solutions offertes par l'entreprise. Une entreprise opérant dans le secteur de développement logiciel devra concentrer ses recherches sur les astuces qui rendent ses solutions plus fiables et plus robustes tout en répondant aux besoins évolutifs de ses clients. Elle devra aussi bien fonder ses choix technologiques et logiciels en étudiant les avantages et les limites de chacun en particulier le choix architectural de ses applications.

En effet, l'architecture d'une application présente le squelette de la solution finale. L'architecture d'une application est l'infrastructure composée de modules actifs qui sont en interaction et d'un ensemble de règles qui gèrent cette interaction. À cet effet, l'équipe intégration doit suivre les évolutions technologiques qui ont apporté des changements dans les architectures logicielles et la gestion des cycles de vie des projets informatique. Ces changements ont pour but de développer des applications performantes, maintenables et extensibles.

À cet égard, devant un développement exponentiel et continu de ses processus métier, Talan Tunisie fait face à certains problèmes qui peuvent ralentir le fonctionnement de ses applications existantes tels que : l'augmentation du temps de réponse, la redondance des modules, la difficulté de maintenance et les besoins des clients qui ne cessent pas d'augmenter.

C'est dans ce cadre, que s'inscrit notre projet de fin d'études du cycle des ingénieurs à l'École Nationale des Sciences de l'Informatique, réalisé à Talan Consulting, société de services en ingénierie informatique. Notre tâche consiste à assurer une refonte architec-

turale de quelques modules de l'outil « Back Office Tool » (BOTOOL) d'un opérateur télécom, en passant de l'architecture monolithique vers une architecture à base de microservices. Le résultat souhaité est d'avoir une structure robuste, et fiable qui allège les complexités de l'architecture existante.

Le présent rapport décrit les différentes étapes de notre travail, et il s'articule autour de sept chapitres :

Le premier chapitre comporte une brève présentation de l'organisme d'accueil et du cadre général de ce projet. Il expose aussi l'étude de l'existant et met l'accent sur la solution proposée. Il aborde à la fin la méthodologie de gestion de projet appliquée pour assurer le bon déroulement de notre travail.

Le deuxième chapitre présente les concepts théoriques clés, liés au style architectural microservices ainsi que les caractéristiques de base de notre architecture.

Le troisième chapitre présente notre sprint de démarrage. Il expose, en premier lieu, une analyse détaillée des besoins fonctionnels et non fonctionnels globaux de l'outil BOTOOL. En second lieu, il décrit le cas d'utilisation général ainsi que l'architecture globale de notre solution.

Le quatrième, cinquième et sixième chapitre détaillent le cycle de vie des sprints ayant pour objectif la réalisation des microservice Administration, Client et Journalisation.

Le septième chapitre illustre l'intégration des microservices de l'outil BOTOOL tout en exposant les choix technologiques utilisés pour la réalisation de notre solution, ainsi que les résultats obtenus selon ces technologies.

Nous clôturons par une conclusion générale qui présente une récapitulation du travail réalisé et ouvre quelques perspectives.

Chapitre 1

Présentation générale

Dans ce premier chapitre, nous nous intéresserons tout d'abord au cadre général de notre projet. Il s'agit en effet d'une présentation de l'organisme d'accueil.

Après l'exposition de la problématique, nous abordons l'étude de l'existant et nous exposerons ensuite la solution proposée. Enfin, nous terminerons ce chapitre par énoncer la méthodologie de gestion de projet adoptée.

I | Présentation de l'organisme d'accueil

Nous présentons dans ce qui suit l'organisme d'accueil dans laquelle s'est déroulé notre stage de fin d'études, ses services et ses produits.

I.1 Talan Tunisie Consulting

Talan est une société de conseil spécialisée dans l'intégration des NTIC (Nouvelles Technologies de l'Information et de la Communication) dans le domaine de la relation client. Elle concentre son expertise sectorielle sur les opérateurs de services à savoir les services Finance et Assurance, Télécoms et Média, Énergie et Services Publics et le service

Transport et Logistique.[\[1\]](#)

Talan est fondée en 2002 par Mehdi Houas, Eric Benamou et Philippe Cassoulat. Elle délivre ses savoir-faire métiers, fonctionnels et technologiques à l'échelle internationale avec plus de 1000 collaborateurs, en France, à New York, à Hong Kong, au Royaume-Uni, à Montréal. Pour accélérer son développement, Talan ouvre un centre de développement « nearshore » en Tunisie intitulé « Talan Tunisie Consulting », mobilisant à ce jour environ 180 ingénieurs de développement nouvelles technologies et travaillant avec de grands clients européens.



FIGURE 1.1 – Talan Consulting à travers le monde

I.2 Secteurs d'activités

Talan couvre essentiellement :

- Le secteur de la finance à travers une collaboration avec des banques d'investissement et des assurances.
- Le secteur de la télécommunication à travers des projets destinés aux opérateurs

télécom et un ensemble de fournisseurs d'accès à internet.

- Le secteur du transport et logistique.
- Le secteur de l'énergie à travers des projets de développement visant des opérateurs de service d'électricité, gaz, eau, etc.[\[1\]](#)

Ces différents secteurs se caractérisent par la mise en disposition d'un ensemble de prestations :

- Conseil et assistance à la maîtrise d'ouvrage ;
- Refonte et optimisation des processus métiers ;
- Support aux grands projets de transformation ;
- Alignement des systèmes d'information aux changements d'organisation et à l'accompagnement au changement.[\[1\]](#)

II Contexte du projet

Dans cette section nous allons commencer, dans un premier temps, par présenter l'outil BOTOOL. Nous allons exposer le travail demandé ainsi que la méthodologie de gestion de projet adoptée, dans un deuxième temps.

II.1 Présentation de l'outil BOTOOL

BOTOOL (Back Office Tool) est un outil assurant la gestion des ressources télécom (numéro téléphone, numéro SIM). Il fait appel à plusieurs services web de gestion de ressources et des suivis d'opérations planifiées touchant à des services d'un opérateur de télécommunication international.

BOTOOL qui est une application monolithique, développer en un seul bloc et par la même technologie, permet d'invoquer des services web en utilisant des classes qui sont fortement couplées ce qui rend difficile sa maintenabilité et son évolution.

Le développement de BOTOOL est subdivisé par ordre de priorité : des fonctionnalités qui sont en production, d'autres en développement et d'autres en attente pour des itérations ultérieures. D'un autre côté, les cycles de changement sont répétitifs. Dans ces situations l'architecture monolithique a prouvé des limites en termes de flexibilité et dynamisme. Ce style de structure reste un frein face à la construction et la modification de BOTOOL. Pour ceci Talan Tunisie prévoit de migrer cet outil afin de pallier ces problèmes.

Nous traitons dans le prochain paragraphe des inconvénients de l'architecture monolithique et ses impacts sur l'évolution de l'outil BOTOOL. Cette prochaine étape nous donnera l'élan suffisant pour définir nos axes d'amélioration et nous offrira une vue plus claire sur l'application existante.

II.2 Limites et critiques de l'existant

Après chaque itération l'outil BOTOOL, subit des améliorations. Des fonctionnalités s'ajoutent pour s'aligner plus au besoin du client. Le projet a commencé depuis des années, et selon son plan d'évolution, il continuera à évoluer encore pour quelques années. Ceci a généré plusieurs défis.

En effet, la taille du projet n'a cessé d'augmenter pour devenir une application monolithique gigantesque, difficile à gérer et à comprendre. Même le respect des bonnes pratiques et les efforts fournis pour maintenir un code modulaire et évolutif n'a pas pu éliminer la complexité de ce projet. Avec une application qui comporte des milliers des lignes de code et un grand nombre de classes plusieurs problèmes se présentent.

En premier lieu, faire évoluer l'équipe est devenu de plus en plus coûteux. L'ajout d'un nouveau développeur au projet implique le sacrifice de plusieurs jours et semaines pour comprendre le code existant et afin qu'il soit capable de le modifier ou d'ajouter d'autres fonctionnalités.

En second lieu, la modification de quelques lignes au niveau de l'application entraîne le redéploiement, le test (les tests unitaires, les tests de régression, les tests IHM) et la révision de la qualité de code de toute l'application. La répétition manuelle de toute la

chaîne de déploiement après chaque modification rend le travail de l'équipe de test plus coûteux en termes d'homme/jours.

En troisième lieu, la haute disponibilité, la rapidité de traitement et la fiabilité sont des priorités pour les fonctionnalités de base de l'outil BOTOOL. Or, la réplication de toute l'application afin de garantir ces exigences pour quelques fonctionnalités est très coûteuse en termes de ressources matérielles.

Par ailleurs, un autre problème qui s'impose, depuis toujours, concerne l'agilité au niveau d'équipe qui est constituée traditionnellement de deux sous équipes : une équipe de développement et une équipe opérationnelle. L'équipe de développement collecte les exigences métier, rédige le code, exécute et teste le code dans un environnement isolé. Cette équipe n'est pas souvent préoccupée par l'impact de leur code sur la production. L'équipe opérationnelle exploite le code fourni par l'équipe de développement et elle est plutôt concentrée sur la stabilisation des services et de l'architecture ainsi que sur la performance des instances actuelles.

Cette organisation traditionnelle entre les équipes alourdit énormément la mise en production. En effet, il y a de nouvelles fonctionnalités qui augmente le temps de la mise en production. Potentiellement, cela implique l'augmentation du temps d'indisponibilité pour l'application, du stress pour les équipes et bien entendu de l'argent perdu.

II.3 Impact de l'architecture monolithique sur l'outil BOTOOL

L'architecture monolithique est l'une des plus répandues dans les projets informatiques. Elle consiste à développer des applications entières en une seule pièce. Ces applications produisent de bons résultats pour les petits projets voire les moyens. Cependant après des mois de développement elle limite l'intégration des ouvertures, l'agrégation de nouveaux besoins et l'innovation technologique. Le tableau 1.1 récapitule les principales limites de l'architecture monolithique et leurs impacts sur l'outil BOTOOL.

TABLE 1.1 – Inconvénients de l’architecture monolithique

Limites	Impact sur BOTOOL
Limites technologiques	<ul style="list-style-type: none">- Le choix des technologies est décidé avant que le développement de l’application commence.- Une seule technologie est utilisée pour le développement de l’application.
Mise à l’échelle coûteuse	<ul style="list-style-type: none">- La mise à l’échelle d’une partie qui a besoin de plus de ressources requiert toute l’application.
Modification du code coûteuse	<ul style="list-style-type: none">- Une modification dans une petite partie de l’application requiert un rebuild et redéploiement de toute l’application.
Coût des tests élevé	<ul style="list-style-type: none">- Durée des tests automatiques et des builds est longue.
Tolérance aux pannes limitée	<ul style="list-style-type: none">- Un dysfonctionnement sur une partie du système a un impacte sur toute l’application.
Agilité limitée	<ul style="list-style-type: none">- Agilité réduite de l’équipe et fréquence de livraison limitée à cause du couplage fort entre les composants.- Effort important de montée en compétences pour un nouvel arrivant sur des composants fortement couplés.- Nécessité de l’intervention de plusieurs personnes pour chaque modification.

La figure 1.2 représente l’outil BOTOOL qui est une application web composée principalement de quatre couches techniques :

- Couche présentation : Se base sur le framework PrimeFaces qui permet de développer des applications basées sur le paradigme MVC coté client.
- Couche métier : Représente l’ensemble métier de l’application, elle est développée avec le framework java/JEE.
- Couche persistance : La couche DAO (Data Access Objects) a pour but de transformer les objets métiers en données sérialisées et inversement. Elle est réalisée en se basant sur le framework Hibernate/JPA.

- Couche données : Toutes les données issues de différentes sources sont sauvegardées dans une base de données relationnelle Oracle.

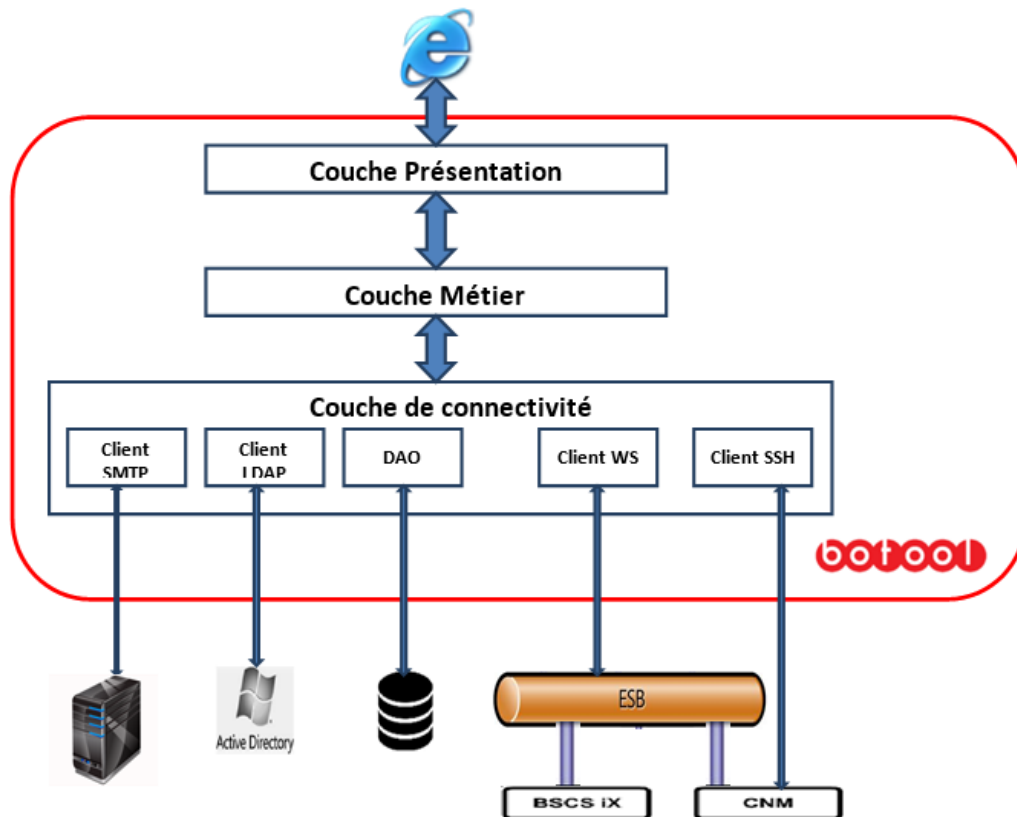


FIGURE 1.2 – Architecture monolithique de BOTOOL

En dépit de cette division en couches et la bonne modularité de BOTOOL, une modification ou ajout d'une fonctionnalité métier touchent toutes ses parties techniques. L'équipe grandit, les développeurs expérimentés quittent et de nouveaux rejoignent l'équipe. Néanmoins, l'effort de montée en compétences sur des composants fortement couplés devient important pour un nouvel arrivant. De même, après chaque itération un redéploiement de la totalité de l'application est nécessaire. Vu ces modifications répétitives et ce cycle de développement bien mouvementé, l'architecture actuelle souffre de manque de flexibilité.

III | Solution proposée et travail demandé

Talan Tunisie consulting intervient dans le secteur des télécommunications pour répondre aux besoins des opérateurs en Tunisie et à l'étranger. Elle s'intéresse principalement à satisfaire ses clients qui sont toujours en quête d'un système d'information performant et facile à utiliser. Après une étude approfondie de la solution existante, et dans le but de surmonter les limites de l'application monolithique ainsi que dans un souci de performance, l'équipe intégration a décidé de faire une refonte de l'outil BOTOOL existant dédié pour un opérateur télécom, afin de surmonter les problèmes de l'architecture monolithique et améliorer la qualité des IHMs.

C'est dans ce cadre que s'inscrit notre projet de fin d'études, qui consiste à migrer quelques modules de la solution existante vers une architecture microservices. Les modules à refondre sont :

- Gestion des utilisateurs et des profils.
- Configuration des écrans de BOTOOL.
- Consultation des informations des clients.
- Consultation de l'historique des opérations effectuées sur BOTOOL.

IV | Méthodologie de gestion de projet

Le choix entre un modèle de processus de développement et un autre, dépend de la nature du projet et de sa taille. Lorsqu'il s'agit d'un projet où les données ne sont

pas réunies dès le départ, où les besoins sont incomplets voire floues, il recommander de s'orienter vers une méthode itérative ou orientée prototypes.

Parmi les méthodes itératives, nous pouvons citer les méthodes AGILE largement utilisées de nos jours à travers le monde. Une méthode AGILE est menée dans un esprit collaboratif et s'adapte aux approches incrémentales. Elle engendre des produits de haute qualité tout en tenant compte de l'évolution des besoins du client. Elle permet aussi de gérer la qualité en continu et de détecter des problèmes le plus tôt au fur et à mesure, permettant ainsi d'entreprendre des actions correctrices sans trop de pénalités dans les coûts et les délais. Il y a plusieurs méthodes AGILE et il ne s'agit pas de choisir la meilleure méthode parmi celles existantes. Pour notre projet, nous nous sommes orientés vers une méthode de type AGILE et plus particulièrement SCRUM. [2]

IV.1 Présentation de la méthode SCRUM

Le principe de la méthodologie SCRUM est de développer un logiciel de manière incrémentale en maintenant une liste totalement transparente des demandes d'évolutions ou de corrections à implémenter. Avec des livraisons très fréquentes, le client reçoit un logiciel fonctionnel à chaque itération. Plus le projet avance, plus le logiciel est complet et possède toujours de plus en plus de fonctionnalités.[2]

IV.2 Les rôles dans SCRUM

La méthodologie SCRUM fait intervenir trois rôles principaux qui sont :

- Product owner : dans la majorité des projets, le responsable produit (Product owner) est le responsable de l'équipe projet client. C'est lui qui va définir et prioriser la liste des fonctionnalités du produit et choisir la date et le contenu de chaque sprint sur la base des valeurs (charges) qui lui sont communiquées par l'équipe,
- Scrum Master : véritable facilitateur sur le projet, il veille à ce que chacun puisse travailler au maximum de ses capacités en éliminant les obstacles et en protégeant l'équipe des perturbations extérieures,

- Equipe : l'équipe s'organise elle-même et elle reste inchangée pendant toute la durée d'un sprint. Elle doit tout faire pour délivrer le produit.[2]

IV.3 Le sprint agile

Le sprint agile représente le cœur de la méthode scrum. Cette qualification lui correspond plutôt bien, puisque tous les développements incrémentaux menant petit à petit au produit final du projet sont réalisés au sein des sprints. [3]

Un périmètre de développement est défini au début d'un sprint et doit être entièrement réalisé lorsqu'il se termine. Chaque sprint doit apporter des fonctionnalités supplémentaires à l'application en cours de développement qui doivent être livrées lorsqu'il se termine. Le product owner est le responsable de définir les sprints et d'organiser le «backlog product» afin de faciliter la construction du produit. [3]

Conclusion

Ce premier chapitre constitue une étape primordiale pour fixer les repères de notre projet. Après avoir présenté l'organisme d'accueil et avoir reconnu ses attentes du projet, nous avons déterminé le cadre du travail ainsi que la méthodologie à emprunter lors de ce stage. Dans le prochain chapitre nous décrirons les caractéristiques de l'architecture microservices ainsi les concepts liés.

Chapitre 2

Concepts théoriques

Afin d'atteindre les objectifs de notre projet, l'étude des concepts théoriques qui lui sont relatifs et des différents moyens mis à notre disposition est une étape primordiale.

Dans ce chapitre, nous nous intéresserons aux concepts de base liés à notre travail. Nous décrirons, tout d'abord, l'architecture microservices. Puis, nous projetterons, les concepts liés à ce style architectural. A savoir, l'approche de la conception pilotée par le domaine, ou encore en anglais Domain Driven Design (DDD), le développement à base de composants et la polyglotte de persistances.

I | L'architecture microservices

Le terme "Microservice" a connu une émergence au cours des dernières années pour décrire un style d'architecture bien particulier. Cette approche consiste à développer une seule application en un ensemble de petits services, isolés, autonomes et indépendamment déployés.

Ces services peuvent communiquer ensemble afin de fournir les fonctionnalités nécessaires. Les microservices sont, dans la plus part du temps, implémentés et pilotés par des équipes de petite taille avec suffisamment d'autonomie. Chacune peut changer l'implé-

mentation de chaque microservice, ajouter ou supprimer des fonctionnalités de ce service avec un impact minimal sur les autres microservices.[B1] Ce style d'architecture présente plusieurs avantages comme l'hétérogénéité technologique, la résistance contre l'échec, la scalabilité sur mesure, la facilité de déploiement, l'alignement organisationnel, la réutilisabilité.[B2]

II | Caractéristiques de l'architecture microservices

D'après Martin FOWLER [4], l'architecture microservices possède neuf principales caractéristiques qu'il est essentiel d'appliquer durant la conception et le développement d'une application en microservices.

II.1 La division en composants via les services

Cette caractéristique est héritée de l'architecture à base de composants. Les microservices sont indépendamment développés, testés et déployés. Un changement dans un service ne nécessite que son déploiement et n'affecte pas l'intégrité du système. Les services permettent d'éviter le couplage fort entre les différents composants en utilisant des mécanismes d'appel distants explicites.

II.2 L'organisation autour des capacités métiers

La division d'une application en microservices est très différente de la décomposition classique qui est souvent basée sur les couches techniques. Chaque microservice est autonome vis à vis de la fonctionnalité qu'il réalise puisqu'il possède son propre code, sa propre interface et gère ses propres données.

II.3 Un produit, pas un projet

Le but de l'utilisation des microservices est de livrer rapidement un morceau de logiciel qui est alors considéré comme terminé. Dans la vision microservices, une équipe est responsable d'un produit durant tout son cycle de vie. Elle est entièrement responsable du logiciel en production.

II.4 Une gouvernance décentralisée

En effet, il est difficile de trouver une seule technologie permettant de résoudre tous les problèmes d'une façon efficace. D'où, il est préférable d'utiliser le bon outil au bon moment. Avec l'architectures microservices, nous pouvons utiliser pour chaque service le langage d'implémentation et la plateforme technologique les plus adéquats pour accomplir le besoin.

II.5 Gestion de données décentralisée

L'architecture en microservices admet l'utilisation de plusieurs bases de données. Dans le cadre d'une application monolithique, nous n'avons qu'une seule base données logique pour les entités persistantes alors que le cadre d'une application en microservices, chaque service a son propre modèle conceptuel et gère sa propre base de données.

II.6 Les extrémités intelligentes et les canaux stupides

Plusieurs entreprises s'investissent dans les canaux de communication intelligents entre les services, alors qu'avec les microservices, l'utilisation de communications stupides est favorisée. Ces communication non intelligentes ne font que transmettre les messages, alors que le microservice s'en charge du reste. L'intercommunication entre les microservices via des protocoles ouverts est privilégiée et beaucoup d'autres interagissent les uns avec les autres via des appels REST ou à travers des systèmes de file d'attente.

II.7 Automatisation de l'infrastructure

Les techniques d'automatisation de l'infrastructure ont connu une évolution considérable ces dernières années. L'évolution du cloud a réduit la complexité opérationnelle de la construction, du déploiement et de l'exploitation de microservices. Les entreprises, qui ont migré vers l'architecture microservices, ont gagné de l'expérience dans la livraison continue et l'intégration continue et elles utilisent maintenant des outils d'automatisation de l'infrastructure.

II.8 Conception pour l'échec

L'un des atouts majeur des microservices est qu'il sont conçus pour être tolérants aux pannes. Dans une application en microservices, si un service échoue, les autres services ne sont pas affectés et adaptent leurs fonctionnements selon l'état du système dans lequel ils évoluent.

II.9 Une conception évolutive

L'un des éléments déterminants dans l'architecture en microservices, est la notion d'indépendance et d'évolutivité. En général, l'évolution d'une application consiste à l'ajout de nouvelles fonctionnalités qui se traduit par la création de nouveaux microservices et ou par la mise à jour des services existants qui implique seulement la mise à jour et le redéploiement du microservice concerné.

III | Les concepts liés à l'architecture microservices

Dans cette section, nous exposerons les concepts liés à l'architecture microservices à savoir la conception pilotée par le domaine, le développement à base de composants

ainsi que le polygotte de persistance.

III.1 La conception pilotée par le domaine

La conception pilotée par le domaine, dite en anglais *domain driven design* (DDD), est une approche de développement qui favorise la création des systèmes informatiques autour des compétences métiers pour combler l'écart entre la réalité de l'entreprise et le code. En pratique, DDD encapsule la logique métier complexe en des modèles métiers.

Le maintien d'un modèle dans un état pur est difficile quand il s'étend sur l'intégralité de l'entreprise, donc c'est préférable de tracer des limites à travers le pattern « Bounded Context » pour les définir. D'après Martin Fowler [5] le contexte borné est un pattern central dans la DDD. Il est au centre de la section de conception stratégique. Ces relations entre les contextes bornés sont généralement décrites par une carte de contexte. La carte de contexte est un document partagé entre ceux qui travaillent sur le projet. Elle met en évidence les différents contextes bornés et leurs liaisons. Cette carte est illustrée par la figure 2.1 :

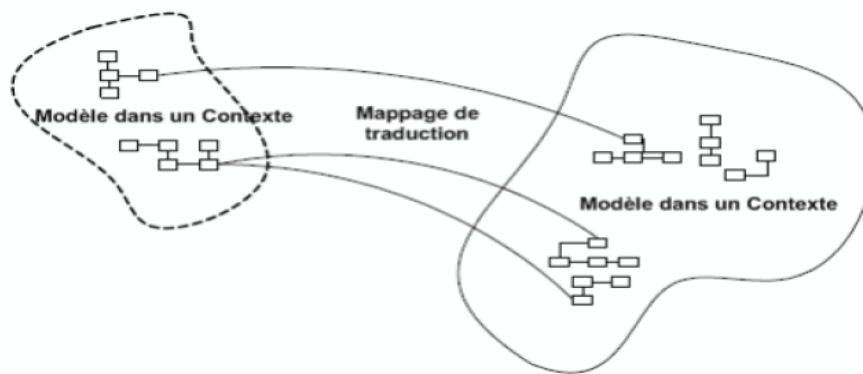


FIGURE 2.1 – Carte de contexte

III.2 Développement à base de composants

Le développement à base de composant est une branche de l'ingénierie logicielle qui met l'accent sur la séparation des préoccupations à l'égard des vastes fonctionnalités disponibles à travers un système de logiciel. C'est une approche basée sur la réutilisation et la redéfinition.

Cependant, un système est un ensemble de préoccupations fonctionnelles et extra-fonctionnelles. Les préoccupations fonctionnelles sont les fonctionnalités métiers que le système doit assurer, alors que les préoccupations extra-fonctionnelles sont des services dont le système a besoin pour effectuer ses fonctionnalités métiers.[B3]

Cette situation augmente la complexité, empêche la réutilisation et gêne l'évolution des systèmes. La séparation avancée des préoccupations permet de séparer les parties extra-fonctionnelles des parties fonctionnelles d'un système. L'objectif escompté étant d'offrir une meilleure réutilisation, faciliter la maintenance, réduire la complexité des systèmes et augmenter leur évolutivité.

III.3 Persistance polyglotte

Le terme persistance polyglotte (Polyglot Persistence en anglais) [6] a été introduit par Scott Leberknight. Il tire son nom de la programmation polyglotte, qui favorise la coexistence de différents langages dans une même application, en tirant profit des forces de chacun d'entre eux.

L'un des atouts majeurs de l'architecture microservices, est que chaque microservice peut être écrit dans un langage de programmation qui lui est propre donnant plus de rapidité et de performance. En raison de l'isolement et de l'indépendance des microservices, les services individuels peuvent être polyglottes en termes de langage de programmation.

IV | Bilan sur l'architecture microservices

Après une étude approfondie des caractéristiques et les concepts liés à l'architecture microservices, nous exposerons, dans cette partie, un bilan sur l'architecture microservices.

IV.1 Motivations de l'utilisation des microservices

Plusieurs motivations poussent à utiliser ce style architectural :

- La frustration de ne pas obtenir le résultat souhaité avec les architectures monolithiques dans les grands projets nous encourage à affronter les défis des microservices.
- L'émergence de nouveaux outils d'automatisation de test, de monitoring et de déploiement nous permettent de voir les microservices sous un nouveau jour et suppriment une partie des problèmes que leur développement et leur déploiement avaient créés.
- L'utilisation des microservices par des grandes entreprises comme Amazon, Netflix, eBay et d'autres, donne assez de confiance pour que ce style d'architecture soit prêt à être évalué et utilisé par les développeurs d'applications professionnelles.

IV.2 Avantages de l'architecture microservices

- La réduction du délai de mise en marché ou « time to market ».
- L'agilité technologique.
- L'extensibilité.
- La factorisation et la réutilisation des microservices.
- Evolutivité.

IV.3 Défis soulevés par les microservices

Bien que l'architecture microservices soit la plus agile approche pour concevoir des applications, et elle présente certaines limites vue qu'elle nécessite des pré-requis. Parmi les pré-requis nous citons :

- Nécessité d'une équipe produit.

- Automatisation des tests obligatoires.
- Besoin d'automatisation du déploiement.

IV.4 Inconvénients de l'architecture microservices

Malgré leurs avantages, les architectures microservices font l'objet de plusieurs critiques et présente quelques inconvénients. Nous pouvons citer les plus récurrents :

- La complexité d'un système distribué.
- Un coût initial plus élevé.
- Un déploiement par microservices.
- Un monitoring essentiel et complexe.

Conclusion

L'objectif de ce chapitre consistait à éclaircir les concepts de base aidant à comprendre et mener à bien notre projet. Le prochain chapitre s'étalera sur les détails des spécifications fixées dans le cadre de refonte de l'outil BOTOOL et l'approche à suivre pour la migration.

Chapitre 3

Analyse et spécification des besoins et architecture en microservices proposée

Le présent chapitre représente notre sprint de démarrage au cours duquel nous spécifions les exigences des différents utilisateurs. Une étude des besoins globaux de ces acteurs est alors nécessaire. Ensuite, nous exposerons notre backlog de produit ainsi que les besoins non fonctionnels. Enfin, nous exposerons l'architecture en microservices proposée.

I | Capture des besoins globaux

L'étape de l'analyse des besoins est très importante puisque la réussite de toute application dépend de la qualité de son étude. Il faut donc bien déterminer les fonctionnalités attendues du système pour les différents acteurs.

I.1 Définition des acteurs

Avant d'analyser les besoins, nous avons identifié deux acteurs : L'administrateur de BOTOOL et l'utilisateur.

- L'administrateur : Un administrateur de BOTOOL est un collaborateur de l'opérateur télécom. dont le rôle est de gérer le fonctionnement de la plateforme.
- L'utilisateur : C'est un simple collaborateur de l'opérateur télécom dont les droits sont restreints.

I.2 Analyse des besoins globaux

Les besoins sont divisés en deux catégories, à savoir les besoins fonctionnels et les besoins non fonctionnels.

I.2.1 Besoins fonctionnels

Ce sont les actions et les réactions que le système doit faire suite à une demande d'un acteur principal. Tenant compte de la nature de l'application, on distingue les besoins par acteurs :

- Administrateur : L'application doit permettre à l'administrateur de :
 - Gérer les utilisateurs de BOTOOL.
 - Consulter la liste des utilisateurs de BOTOOL.
 - Gérer les rôles.
 - Consulter la liste des rôles.
 - Consulter la liste des écrans de BOTOOL : Un écran BOTOOL représente une interface Homme-Machine.
 - Configurer les écrans de BOTOOL .
 - Consulter les informations des clients de l'opérateur télécom.
 - Exporter les informations des clients sous format Excel.
 - Consulter le journal des opérations effectuées par les différents utilisateurs.
- Utilisateur : L'application doit permettre à l'utilisateur de :
 - Consulter les informations des clients de l'opérateur télécom.

- Exporter les informations des clients sous format Excel.

I.2.2 Besoins non fonctionnels

Les besoins non fonctionnels de notre application sont :

- Tolérance aux pannes : BOTOOL doit fonctionner même en cas de défaillance des microservices.
- Mise à l'échelle : BOTOOL doit permettre une mise à l'échelle ciblée, c'est-à-dire mettre à l'échelle que les microservices qui ont besoin d'y être.
- Temps de réponse : BOTOOL doit rendre une réponse dans un temps minimal.
- Maintenabilité : L'architecture de BOTOOL doit être évolutive et extensible vu que les besoins du client ne sont pas fixent.

I.3 Backlog de produit

Après avoir cité les besoins de notre application, nous décrivons dans cette partie le backlog du produit, illustré par le tableau 3.1, qui représente une liste de tâches exprimées sous forme de besoins.

TABLE 3.1 – Backlog du produit

ID	User stories	Priorité
1	En tant qu'utilisateur, je veux pouvoir me connecter à BOTOOL avec les bons droits.	Moyenne
2	En tant qu'administrateur, je veux pouvoir ajouter un nouveau utilisateur.	ELevée
3	En tant qu'administrateur, je veux pouvoir modifier un utilisateur.	ELevée
4	En tant qu'administrateur, je veux pouvoir supprimer un utilisateur.	ELevée

ID	User stories	Priorité
5	En tant qu'administrateur, je veux pouvoir consulter la liste des utilisateurs de BOTOOL.	Faible
6	En tant qu'administrateur, je veux pouvoir ajouter un nouveau rôle utilisateur.	Elevée
7	En tant qu'administrateur, je veux pouvoir modifier un rôle.	Elevée
8	En tant qu'administrateur, je veux pouvoir supprimer un rôle.	Elevée
9	En tant qu'administrateur, je veux pouvoir consulter la liste des rôles.	Faible
10	En tant qu'administrateur, je veux pouvoir consulter la liste des IHM de BOTOOL.	Moyenne
11	En tant qu'administrateur, je veux pouvoir configurer les IHM de BOTOOL.	Elevée
12	En tant qu'administrateur, je veux pouvoir consulter les informations des clients, soit par le code client, par le numéro de téléphone ou bien par le numéro de contrat.	Elevée
13	En tant qu'administrateur, je veux pouvoir exporter les informations des clients sous format Excel.	Moyenne
14	En tant qu'administrateur, je veux pouvoir Consulter le journal des opérations effectuées sur BOTOOL.	Moyenne
15	En tant qu'utilisateur, je veux pouvoir consulter les informations des clients, soit par le code client, par le numéro de téléphone ou bien par le numéro de contrat.	Moyenne
16	En tant qu'utilisateur, je veux pouvoir exporter les informations des clients sous format Excel.	Faible

II Spécification des besoins fonctionnels globaux

L'outil BOTOOL offre divers fonctionnalités. Le diagramme de cas d'utilisation global représenté par la figure 3.1, modélise les fonctionnalités des modules que nous avons migré. Nous détaillons dans les chapitres suivants les cas d'utilisation réalisés.

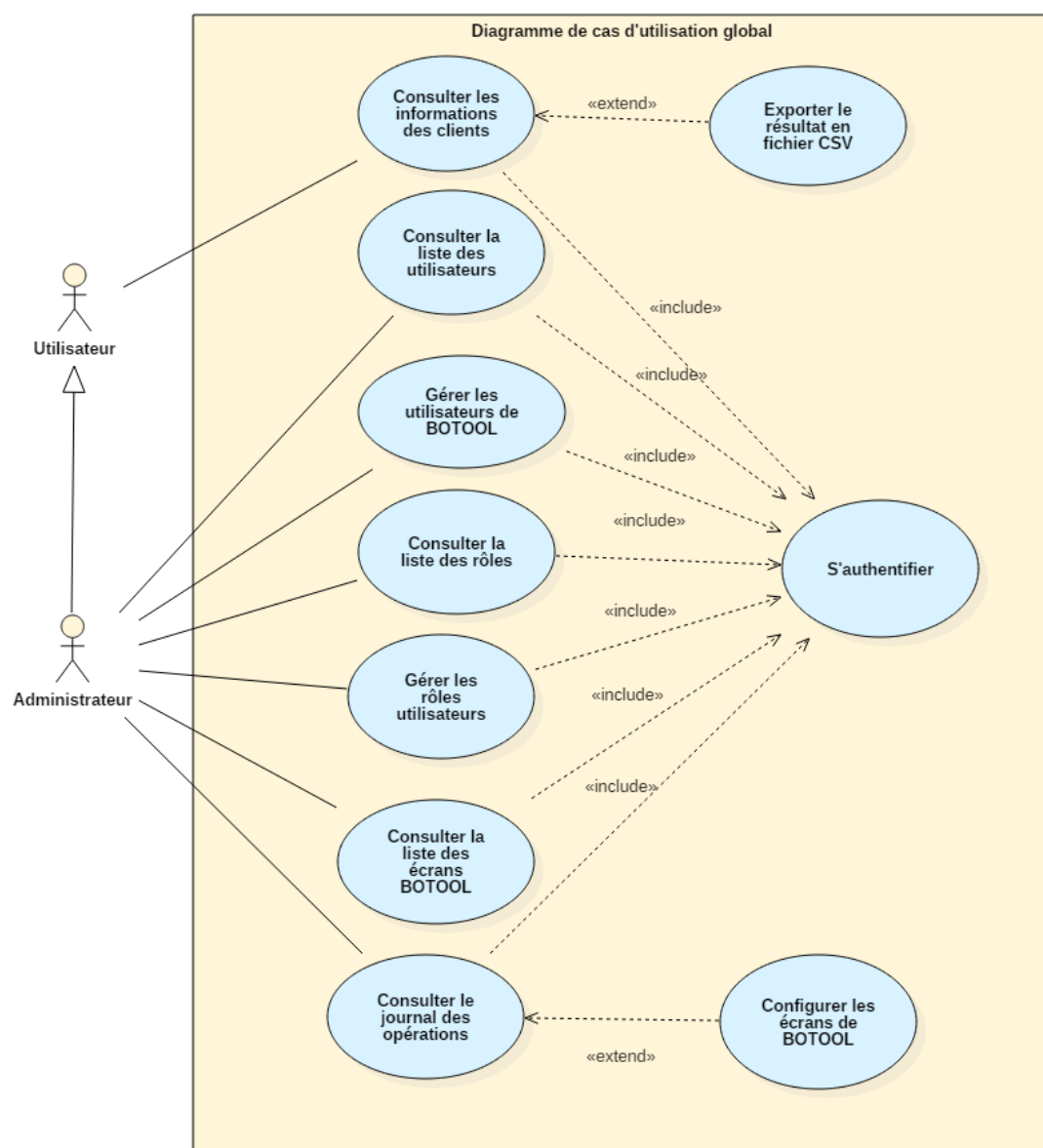


FIGURE 3.1 – Diagramme de cas d'utilisation global

III | Division de BOTOOL en micro-services

Pour passer d'une architecture monolithique vers une architecture en microservices, nous suivons le pattern des contextes bornés de la conception pilotée par le domaine, décrit dans le chapitre 2. Nous présentons dans La figure 3.2 ci-dessous, la carte de contexte de notre système.

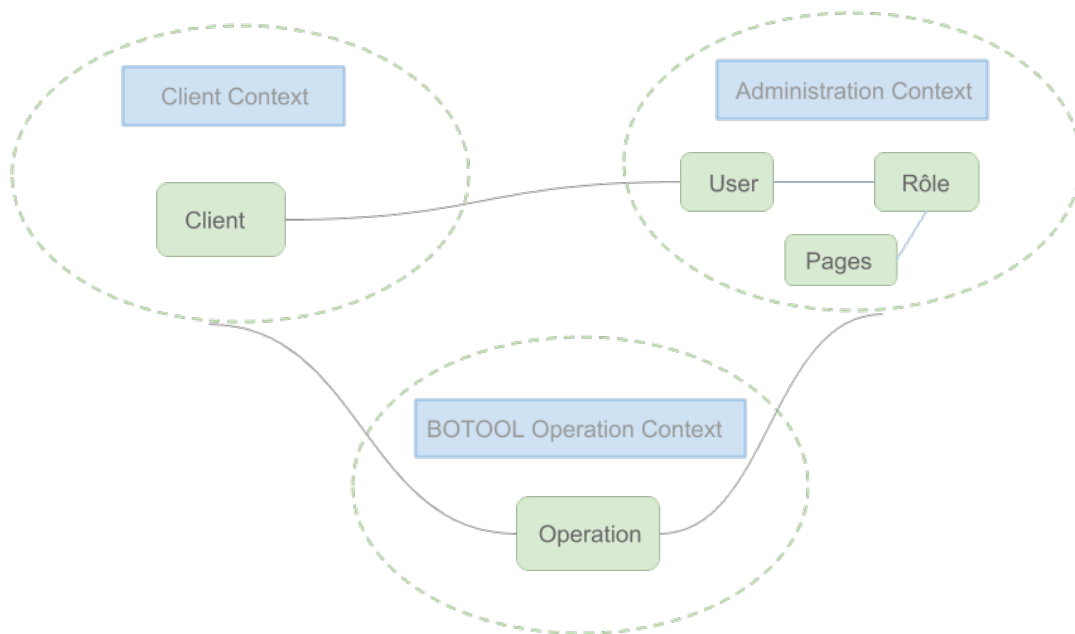


FIGURE 3.2 – Carte de contexte de BOTOOL

Le pattern bounded context nous a permis d'avoir trois microservices autonome et dont le couplage entre eux est faible :

- Microservice administration-service,
- Microservice client-service,
- Microservice journalisation-service.

En se basant sur notre backlog du produit et sur la décomposition de notre projet en microservices, nous divisons notre Release en 3 sprints comme le montre le tableau 4.2 ci-dessous.

TABLE 3.2 – Partitionnement des user stories par sprint

	Sprint 1 : Micro-service Administration	Sprint 2 : Microservice Client	Sprint 3 : Microservice journalisation
User Story ID	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	12, 13, 15, 16	14

IV | L'architecture de l'application en microservices

La figure 3.3 présente les composants primordiaux pour mettre en place une architecture microservices. Elle est composée généralement de :

- Un serveur de configuration encore appelé "Config Server" qui permet de centraliser les fichiers des configurations de chaque microservice dans un simple dépôt Git. Ceci permet d'avoir une configuration partagée et évolutive indépendamment des applications. Au démarrage, chaque microservice récupère ses propriétés et sa configuration auprès du serveur de configuration.
- Un service d'enregistrement "Service Registration" qui contient la liste de toutes les instances disponibles des microservices. Donc, après avoir récupéré leurs configurations, les microservices s'enregistrent dans le serveur d'enregistrement. Cela rend la découverte des microservices plus facile.
- Une passerelle appelée encore "API Gateway", qui présente le point d'entrée au système. Elle encapsule l'architecture du système interne et fournit des API adaptées pour chaque type de client. l'API Gateway encapsule un composant très important qui est l'équilibreur de charge, appelé "Load Balancer". Il gère le routage et répartition

de la charge entre les instances des microservices disponibles. Pour avoir la liste des instances disponibles, le load balancer consulte le serveur d'enregistrement.

- Un disjoncteur de circuit, "Circuit Breaker", ce composant est primordiale dans une architecture microservices. Il garantit une caractéristique très importante qui est la conception pour l'échec, que nous avons évoqué dans le chapitre 2. Le disjoncteur de circuit permet d'éviter l'échec de tout le système en cas de défaillance d'un microservice.

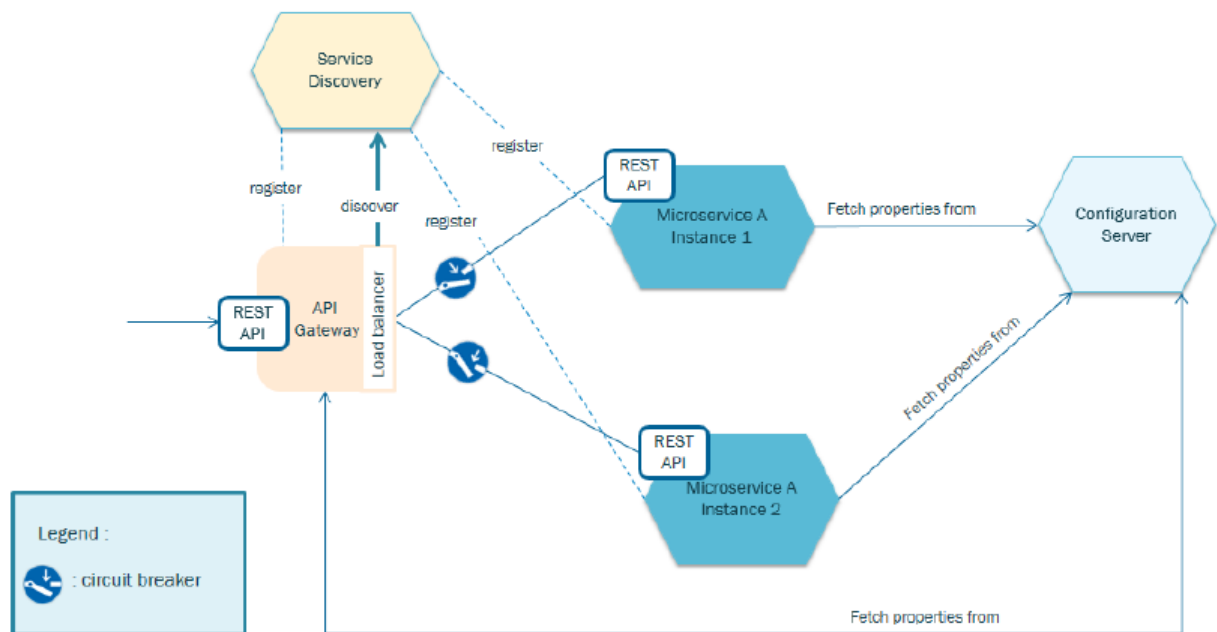


FIGURE 3.3 – Les composants d'une architecture microservices

Conclusion

Tout au long de ce chapitre, nous avons spécifié, tout d'abord, nos besoins ce qui nous a aidé à avoir une vision plus claire et plus profonde sur notre projet. Ensuite, nous avons présenté les microservices qui composent notre application et nous avons planifié nos sprints. Après, nous avons présenté l'architecture globale d'une application microservices. Dans le chapitre suivant, nous abordons notre premier sprint.

Chapitre 4

Sprint 1 : Microservice Administration

Après avoir analysé et spécifié les besoins globaux de notre client, nous détaillerons les différentes étapes effectuées durant ce premier sprint ayant pour objectif la réalisation du microservice Administration. Ce microservice permet de gérer les utilisateurs de BOTOOL, les rôles, ainsi que la configuration des écrans.

Nous commencerons, tout d'abord, par présenter le backlog du sprint suivi d'une analyse détaillée et la conception du microservice administration. Et nous présenterons par la suite, les interfaces homme-machine réalisées ainsi que la phase de test.

I | Sprint Backlog

TABLE 4.1 – Backlog du sprint 1

Exigence	Sous tâche
1.1	Afficher la liste des utilisateurs de BOTOOL.
1.2	Mettre à jour les utilisateurs.
1.3	Afficher la liste des rôles de BOTOOL.

1.4	Mettre à jour les rôles.
1.5	Afficher la liste des écrans de BOTOOL.
1.6	Configurer un écran BOTOOL.

II | Spécification fonctionnelle du microservice Administration

Pour la spécification des besoins, nous nous référerons aux diagrammes d’UML : les diagrammes de cas d’utilisation et les diagrammes de séquence.

II.1 Diagrammes de cas d’utilisation du microservice Administration

L’administrateur doit pouvoir gérer les utilisateurs de BOTOOL, les rôles ainsi que configurer les différents écrans de l’outil. Nous allons présenter les diagrammes de cas d’utilisation de gestion des utilisateurs et de gestion des rôles. Pour les cas d’utilisation consulter information client et consulter le journal des opérations, ils seront détaillés dans les chapitres suivants.

II.1.1 Diagrammes de cas d’utilisation relatif à la gestion des utilisateurs

La figure 4.1 ci-dessous, décrit les opérations relatives à la gestion des utilisateurs.

Ce diagramme de cas d’utilisation présente les différentes opérations que l’administration peut effectuer pour gérer les utilisateurs de BOTOOL. L’administrateur doit s’authentifier pour accéder à son espace. Il peut consulter la liste des collaborateurs de l’opérateur télécom qui utilise l’outil BOTOOL. Il peut aussi ajouter, modifier ou supprimer un compte utilisateur. Les opérations effectuées, sont enregistrées dans le journal des opérations.

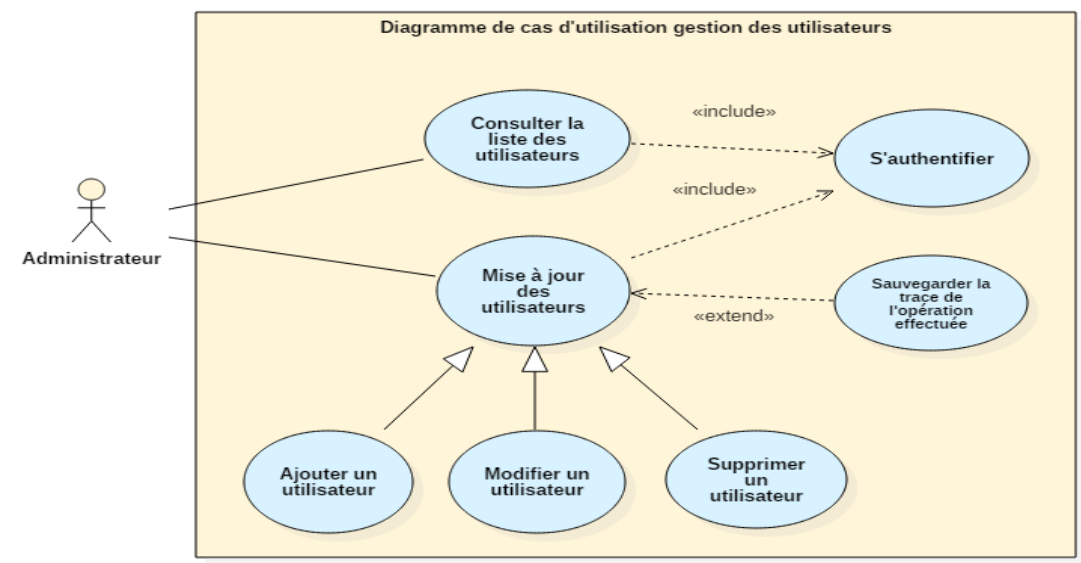


FIGURE 4.1 – Diagramme de cas d'utilisation gestion des utilisateurs

II.1.2 Diagrammes de cas d'utilisation relatif à la gestion des rôles

La figure 4.2 ci-dessous, décrit les opérations relatives à la gestion des rôles utilisateurs.

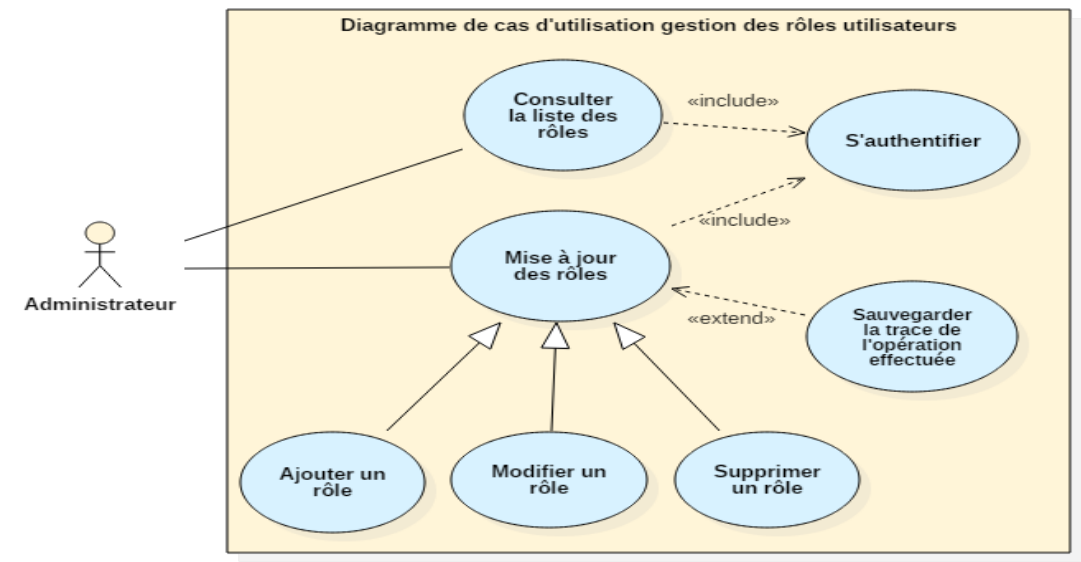


FIGURE 4.2 – Diagramme de cas d'utilisation gestion des rôles utilisateurs

Ce diagramme de cas d'utilisation illustre les différentes opérations que l'administration peut effectuer pour gérer les rôles utilisateurs. L'administrateur doit s'authentifier pour accéder à son espace. Il peut consulter la liste des rôles utilisateurs. Il peut aussi ajouter, modifier ou supprimer un rôle. Toute opération effectuée, est enregistrée dans le journal des opérations.

II.1.3 Diagrammes de cas d'utilisation relatif à la configuration des écrans

La figure 4.3 ci-dessous, décrit les opérations relatives à la configuration des écrans de BOTOOL.

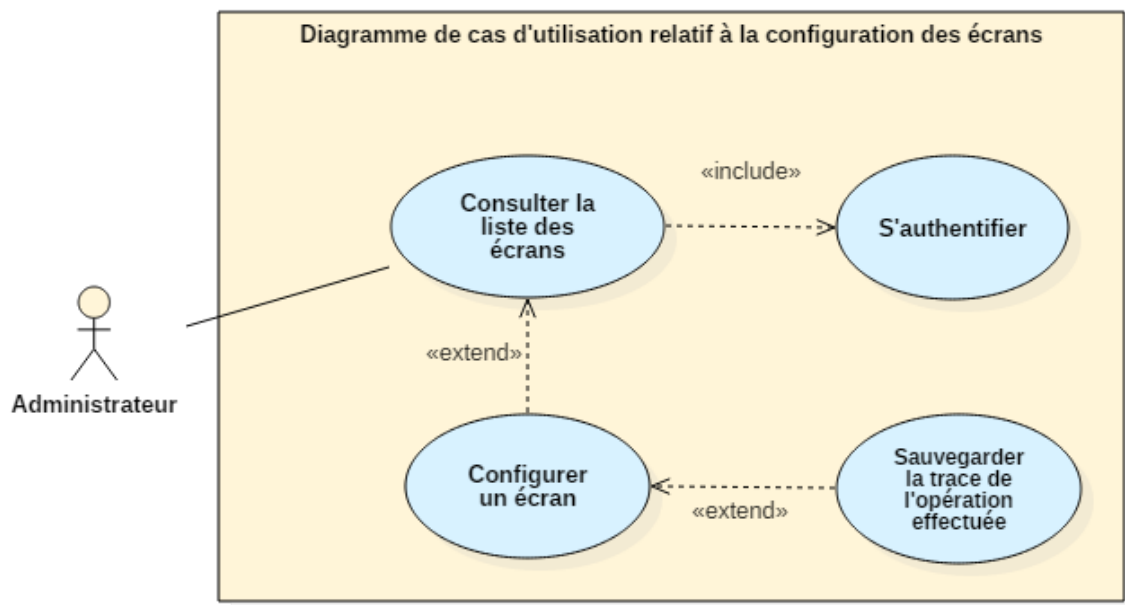


FIGURE 4.3 – Diagramme de cas d'utilisation configuration des écrans

Ce diagramme de cas d'utilisation représente l'opération de configuration des écrans de BOTOOL. Un écran est IHM Homme-Machine de l'outil BOTOOL. L'administrateur doit s'authentifier pour effectuer cette opération. Il peut consulter la liste de tous les écrans de BOTOOL. Il a aussi la possibilité de configurer un écran. L'opération de configuration des écrans est enregistrée dans le journal des opérations.

II.2 Description de quelques scénarii

II.2.1 Scénario du cas d'utilisation "S'authentifier"

La figure 4.9 ci-dessous représente le diagramme de séquence système du scénario d'authentification pour un administrateur.

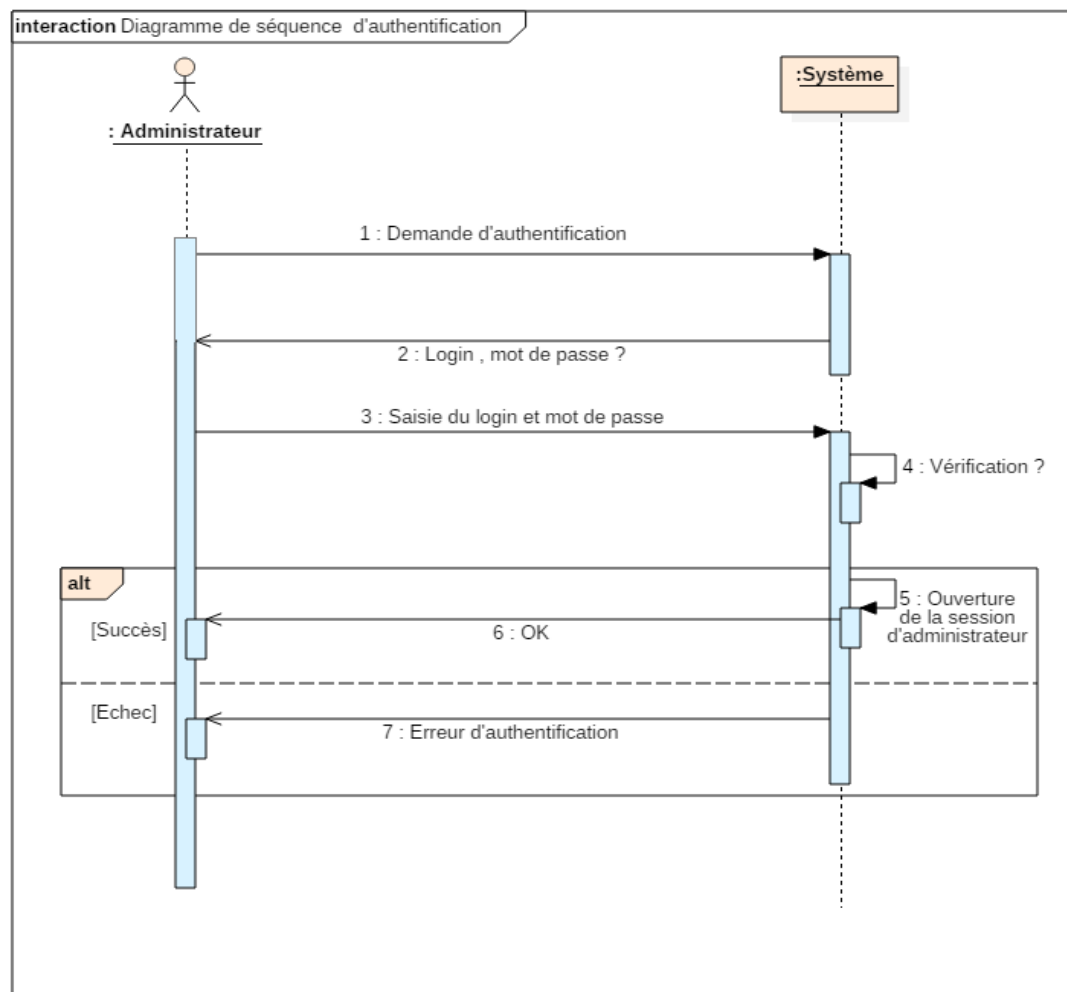


FIGURE 4.4 – Diagramme de séquence système d'authentification

Pour s'authentifier, un administrateur doit saisir son login et son mot de passe, si les données saisies sont correctes alors sa session sera ouverte et il sera redirigé automatiquement au dashboard de BOTOOL. Si les données sont erronées alors un message d'erreur apparaîtra lui demandant de saisir de nouveau le login et le mot de passe corrects.

II.2.2 Scénario du cas d'utilisation Modifier un rôle utilisateur

La figure 4.5 ci-dessous représente le diagramme de séquence système du scénario de modification d'un rôle.

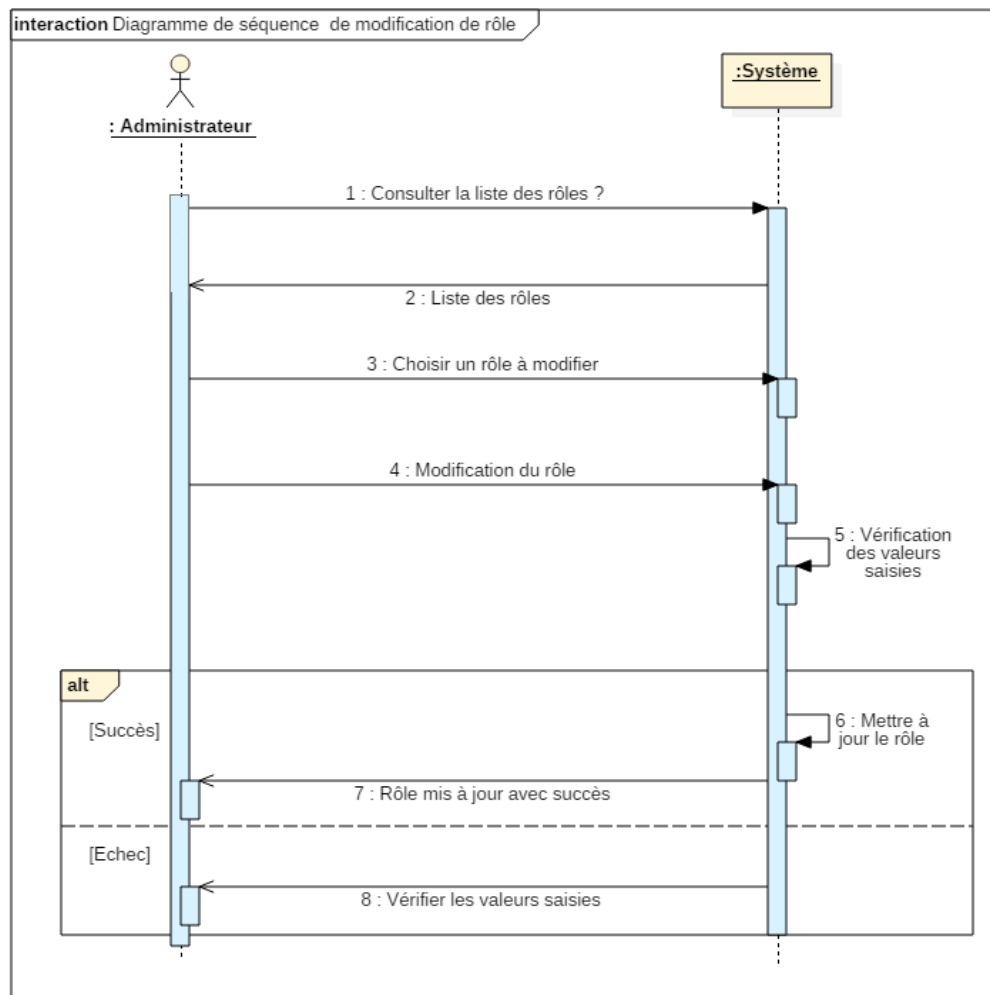


FIGURE 4.5 – Diagramme de séquence système de modification de rôle utilisateur

Pour modifier un rôle, l'administrateur consulte, tout d'abord, la liste des rôles qui existent. Puis il peut choisir un rôle parmi la liste et met à jour les différents champs. Si les valeurs des champs sont valides alors le rôle est mis à jour. Sinon si les valeurs ne sont pas valides alors un message d'erreur apparaîtra à l'administrateur pour l'avertir.

III | Conception du microservice Administration

Dans cette section nous présentons, la phase de conception de notre microservice Administration. Nous détaillerons l'architecture logicielle, le diagramme de package ainsi que le diagramme de classes.

III.1 Architecture logicielle du microservice Administration

La figure 4.6 présente l'architecture logicielle du microservice Administration qui est constituée de trois couches.

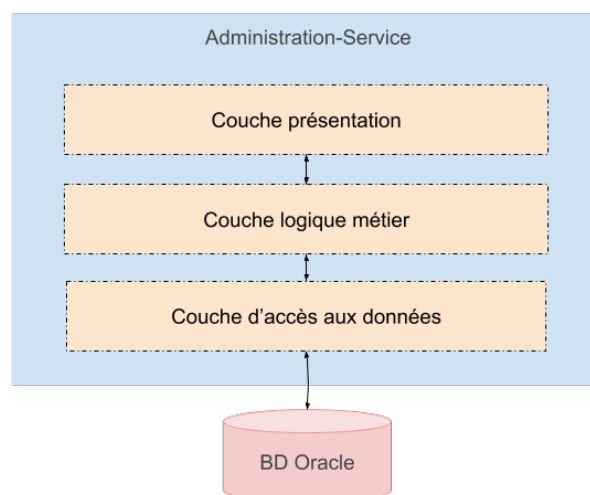


FIGURE 4.6 – Architecture logicielle du microservice Administration

- La couche présentation : Elle est la responsable de la communication avec les autres microservices du système et elle délègue le traitement à la couche couche logique métier.
- La couche logique métier : Elle est la responsable de la logique métier.
- La couche d'accès aux données : Elle est la responsable de la communication avec la base de données.

III.2 Diagramme de paquets du microservice Administration

Administration-service se compose de cinq paquets, qui sont présentés par la figure 4.7 ci-dessous.

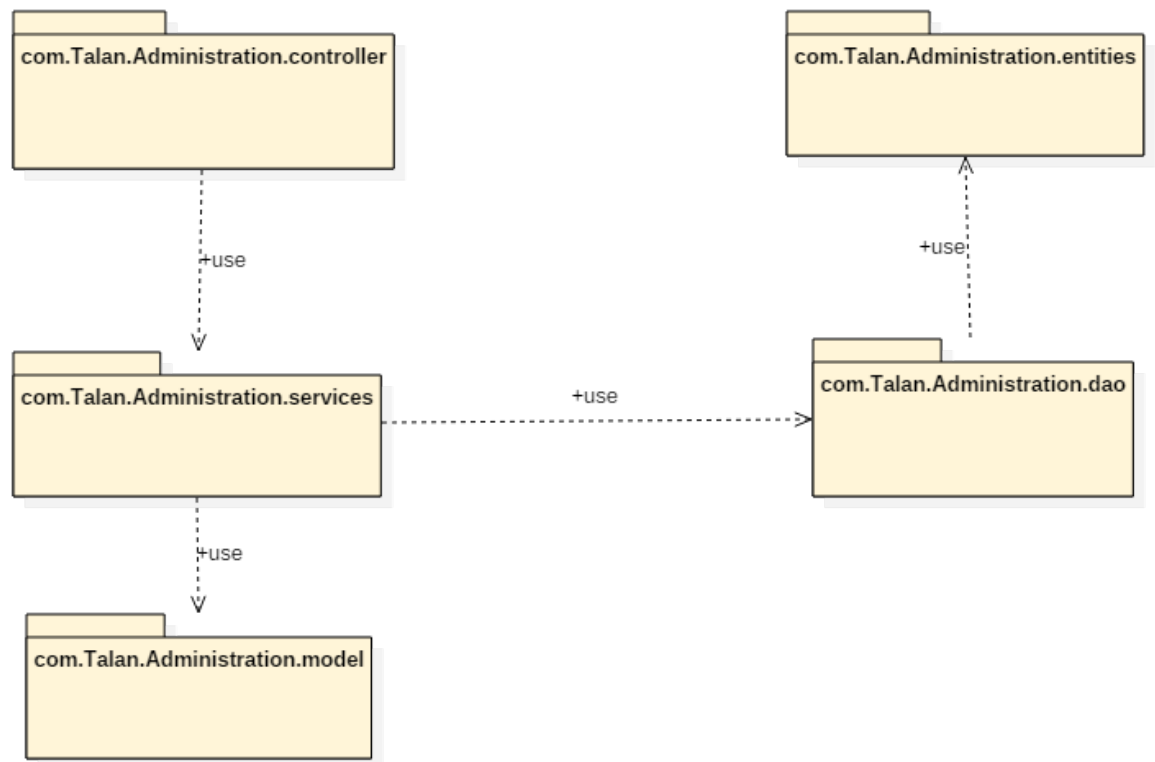


FIGURE 4.7 – Diagramme de paquets du microservice Administration

- `com.Talan.Administration.entities` : contient toutes les entités de notre microservice.
- `com.Talan.Administration.dao` : contient toutes les méthodes nécessaires pour l'accès à la base de données.
- `com.Talan.Administration.service` : présente notre logique métier.
- `com.Talan.Administration.controller` : présente notre REST API.
- `com.Talan.Administration.model` : contient le modèle de l'entité `BoToolOperationHistory`.

III.3 Diagramme de classes du microservice Administration

La figure 4.8 illustre le diagramme de classes de notre microservice et les relations qui existent entre ses différentes classes.

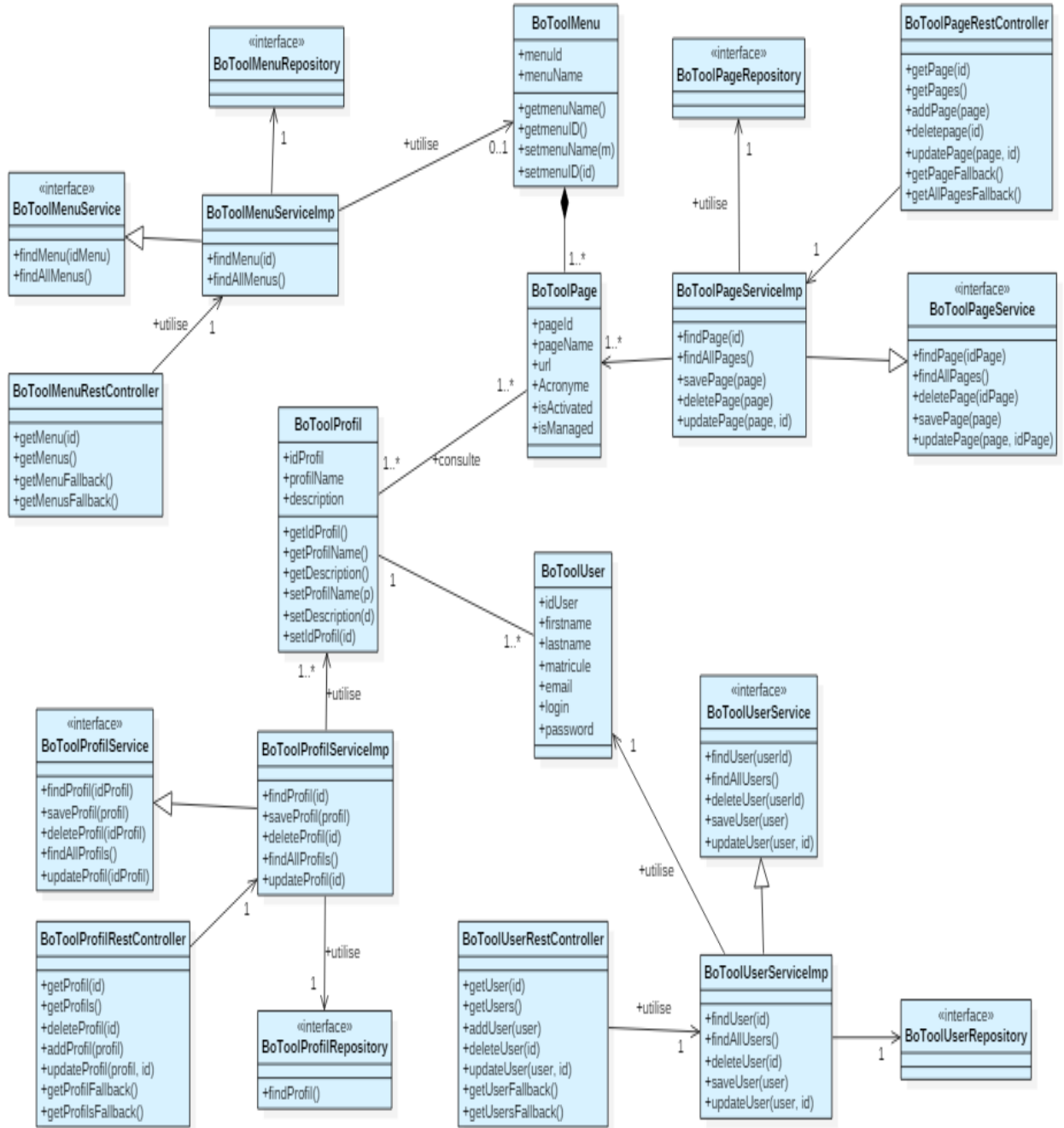


FIGURE 4.8 – Diagramme de classes du microservice Administration

TABLE 4.2 – Les principales classes conçues pour l’élaboration du microservice

Nom de la classe	Description
BoToolUser	Cette classe contient les informations sur les collaborateurs de l’opérateur télécom qui utilisent l’outil BOTOOL.
BoToolProfil	Cette classe contient les informations sur les collaborateurs de l’opérateur télécom qui utilisent l’outil BOTOOL.
BoToolUser	Cette classe contient les informations sur les rôles utilisateurs de l’outil BOTOOL.
BoToolMenus	Cette classe contient les informations sur les menus de BOTOOL.
BoToolPages	Cette classe contient les informations les écrans (IHM) de l’outil BOTOOL.
BoToolUserRepository BoToolPageRepository BoToolMenuRepository BoToolProfilRepository	Ce sont les interfaces de la couche DAO qui implémentent l’interface JPA Repository de Spring Data.
BoToolUserService BoToolPageService BoToolMenuService BoToolProfilService	Ces interfaces contiennent les opérations nécessaires de la couche service. Elles regroupent tous les services et la réalisation des cas d’utilisation de microservice.
BoToolUserServiceImp BoToolPageServiceImp BoToolMenuServiceImp BoToolProfilServiceImp	Ce sont les implémentations des interfaces de la couche service.
BoToolUserRestController BoToolPageRestController BoToolMenuRestController BoToolProfilRestController	Ce sont les classes de la couche contrôleur. Elles interceptent les interactions de l’utilisateur et qui retournent une réponse aux requêtes. Ces classes contiennent les services REST de notre microservice.

III.4 Description du scénario de mise à jour d'un compte utilisateur

La figure 4.9 ci-dessous représente le digramme de séquence de modification d'un utilisateur :

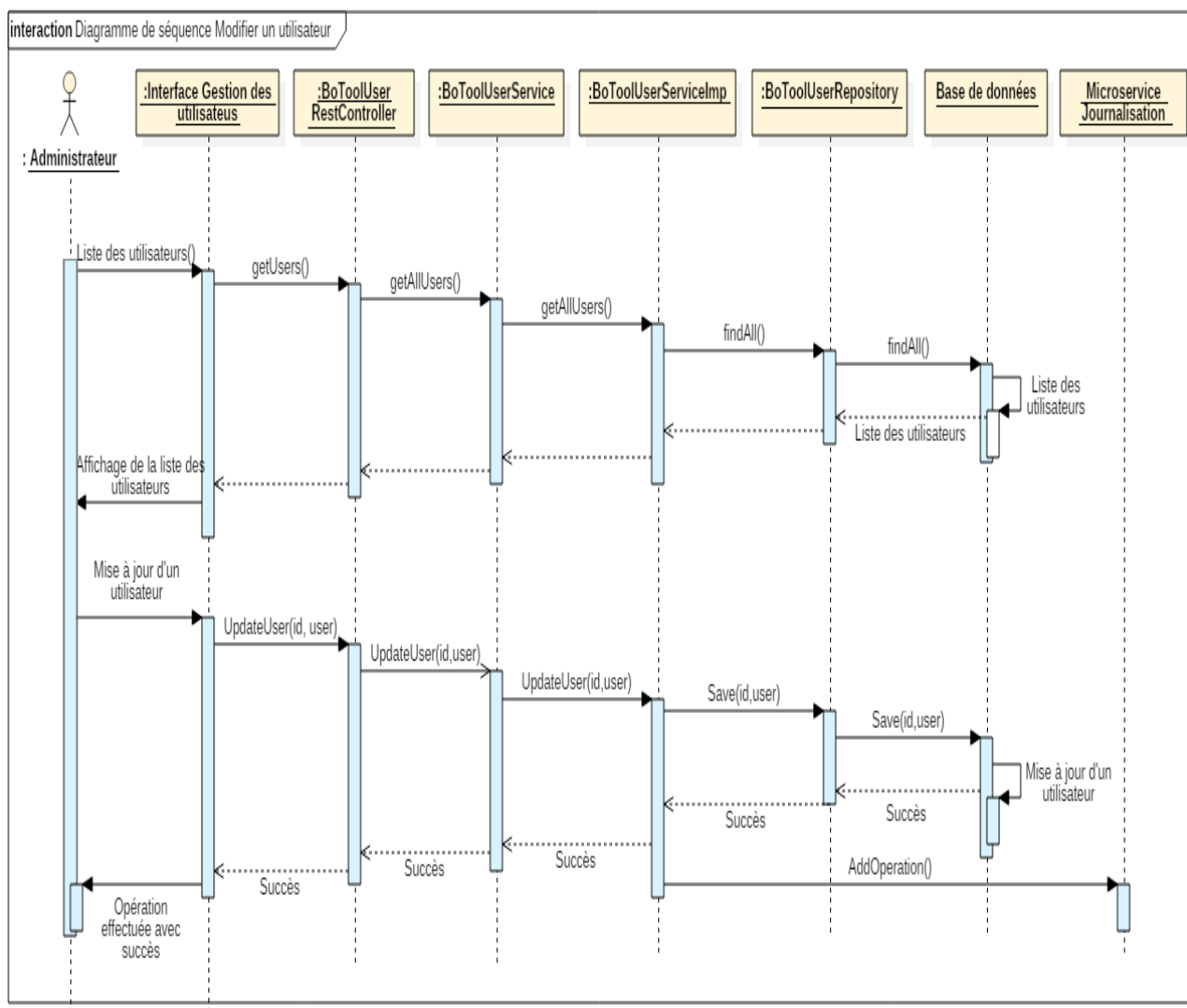


FIGURE 4.9 – Diagramme de séquence du scénario de mise à jour d'un utilisateur

Afin de modifier un les informations des utilisateurs, l'administrateur consulte tout d'abord, la liste de tous les utilisateurs de BOTOOL, choisit un utilisateur à modifier et met à jour les différents champs.

IV | Réalisation du microservice Administration

Nous exposerons dans cette section les interfaces Homme-Machine du microservice Administration. La figure 4.10 représente l'interface de authentification de l'outil BOTOOL.

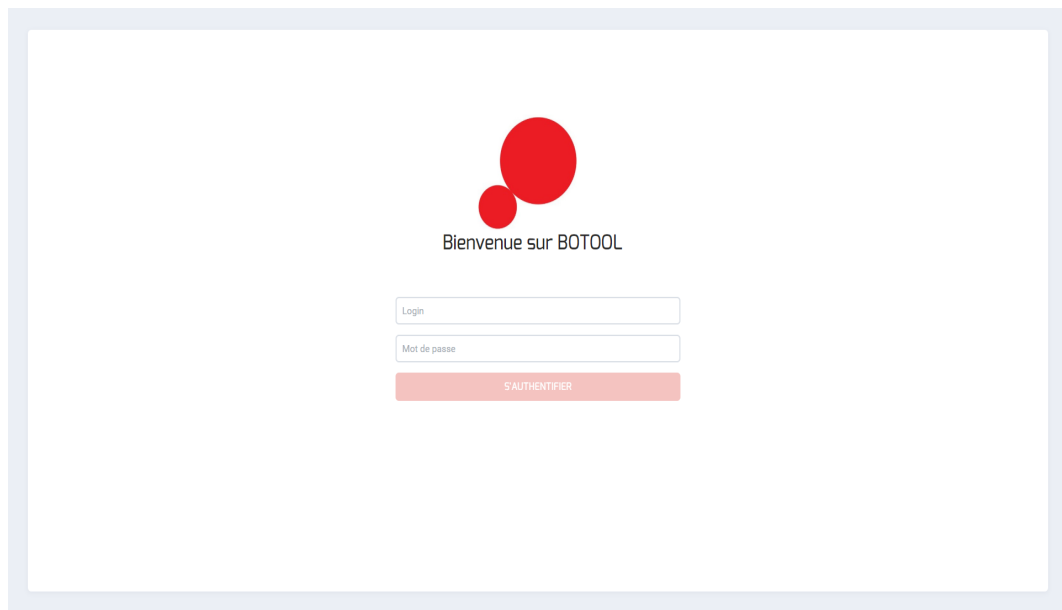
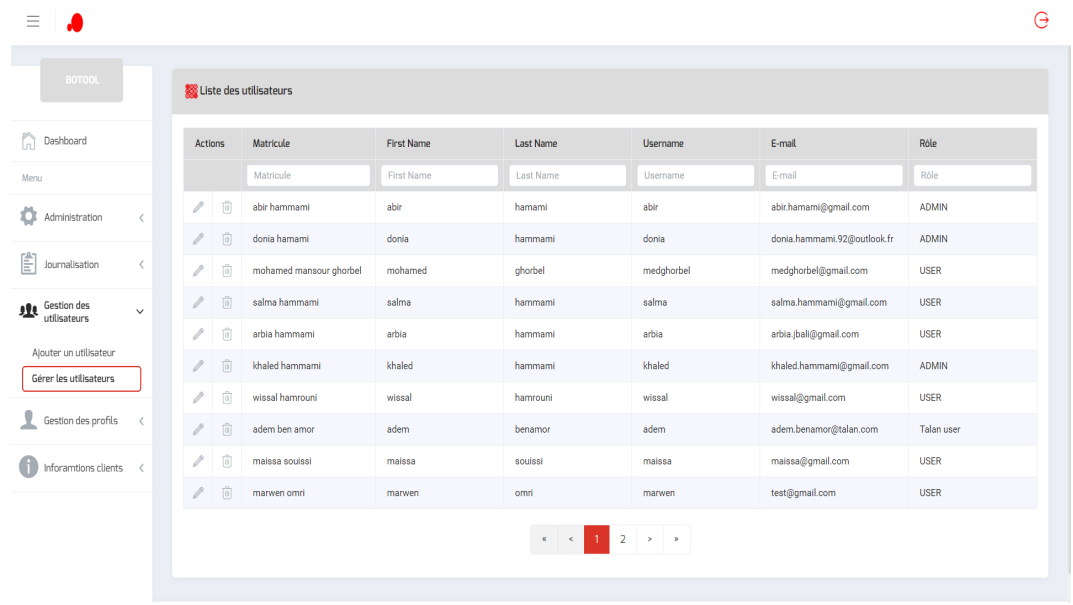


FIGURE 4.10 – Interface d'authentification de l'outil BOTOOL

Une fois authentifié, l'administrateur peut accéder à son espace. Il peut consulter la liste des utilisateurs de BOTOOL, comme il peut soit modifier ou supprimer un utilisateur via la liste affichée. Il peut aussi ajouter un nouveau utilisateur. Les figures 4.11 et 4.12 ci-dessous, illustrent ces opérations.

L'administrateur peut accéder à son espace. Il peut consulter la liste des rôles de BOTOOL, comme il peut gérer ceux qui existent déjà. La figure 4.13 ci-dessous, présente la liste rôles de BOTOOL.



The screenshot displays the 'Liste des utilisateurs' (List of users) page in the BOTOOL application. The left sidebar contains navigation links: Dashboard, Menu, Administration, Journalisation, Gestion des utilisateurs (highlighted), Gestion des profils, and Informations clients. Under 'Gestion des utilisateurs', there are links for 'Ajouter un utilisateur' and 'Gérer les utilisateurs' (highlighted with a red box). The main content area shows a table of users with columns for Actions, Matricule, First Name, Last Name, Username, E-mail, and Rôle. The table lists 10 users, including administrators and regular users. At the bottom of the table, there is a pagination control showing page 1 of 2.





















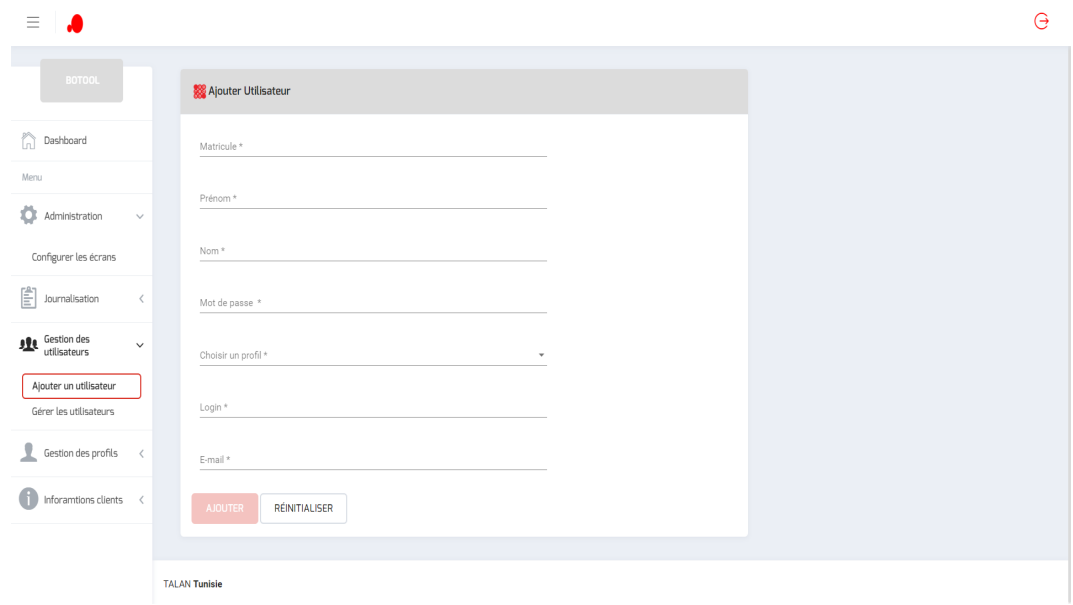
Actions	Matricule	First Name	Last Name	Username	E-mail	Rôle
 	abir hamami	abir	hamami	abir	abir.hamami@gmail.com	ADMIN
 	donia hamami	donia	hamami	donia	donia.hamami.92@outlook.fr	ADMIN
 	mohamed mansour ghorbel	mohamed	ghorbel	medghorbel	medghorbel@gmail.com	USER
 	salma hammami	salma	hammami	salma	salma.hammami@gmail.com	USER
 	arbia hammami	arbia	hammami	arbia	arbia.jball@gmail.com	USER
 	khaled hammami	khaled	hammami	khaled	khaled.hammami@gmail.com	ADMIN
 	wissal hamrouni	wissal	hamrouni	wissal	wissal@gmail.com	USER
 	adem ben amor	adem	benamor	adem	adem.benamor@talan.com	Talan user
 	maïssa souissi	maïssa	souissi	maïssa	maïssa@gmail.com	USER
 	marwen omri	marwen	omri	marwen	test@gmail.com	USER

FIGURE 4.11 – Liste des utilisateurs de BOTOOL



The screenshot displays the 'Ajouter Utilisateur' (Add User) page in the BOTOOL application. The left sidebar is identical to the previous figure, with 'Ajouter un utilisateur' highlighted under 'Gestion des utilisateurs'. The main content area shows a form with the following fields: Matricule *, Prénom *, Nom *, Mot de passe *, Choisir un profil *, Login *, and E-mail *. At the bottom of the form, there are two buttons: 'AJOUTER' and 'RÉINITIALISER'. The footer of the page indicates 'TALAN Tunisie'.

FIGURE 4.12 – Ajout d'un nouveau utilisateur de BOTOOL

La figure 4.14 ci-dessous, présente l'interface de modification d'un rôle existant.

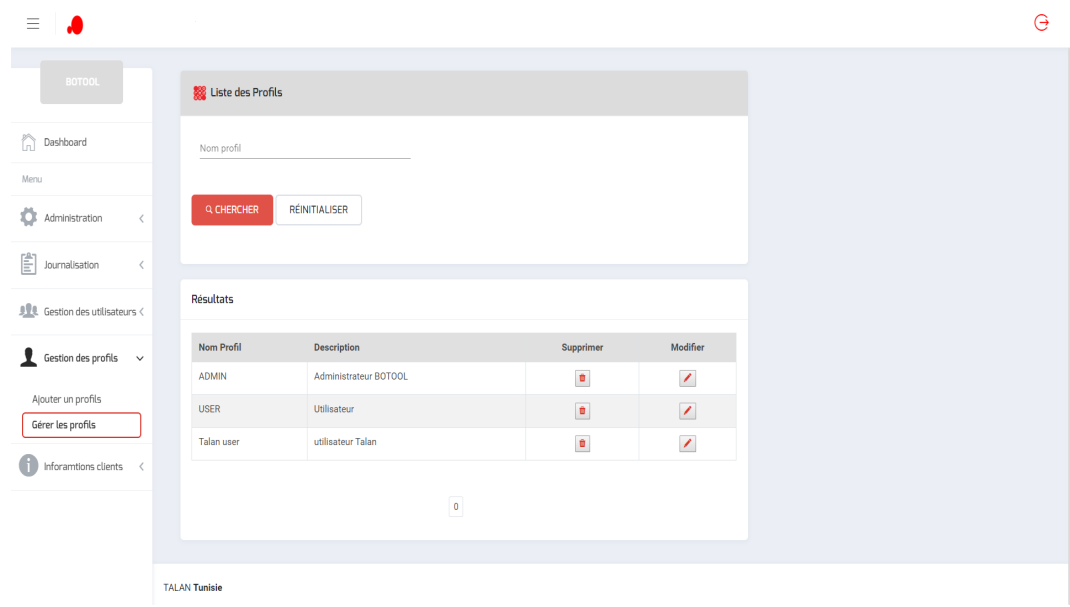


FIGURE 4.13 – Liste des rôles

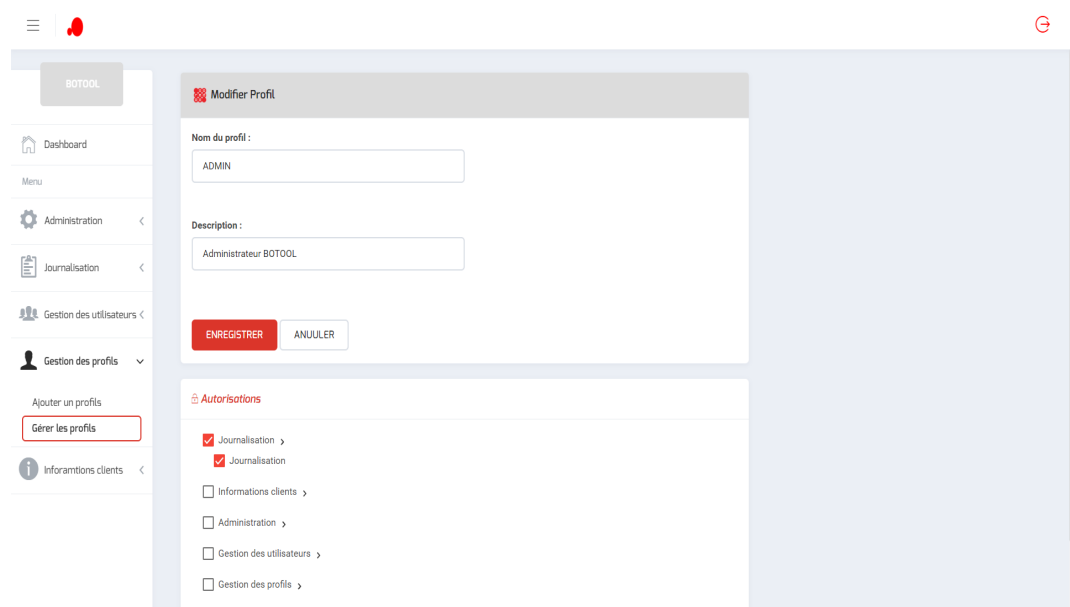


FIGURE 4.14 – Mettre à jour un rôle existant

La figure 4.15 ci-dessous, présente l’interface qu’obtient l’administrateur lorsqu’il effectue la recherche d’un rôle non existant.

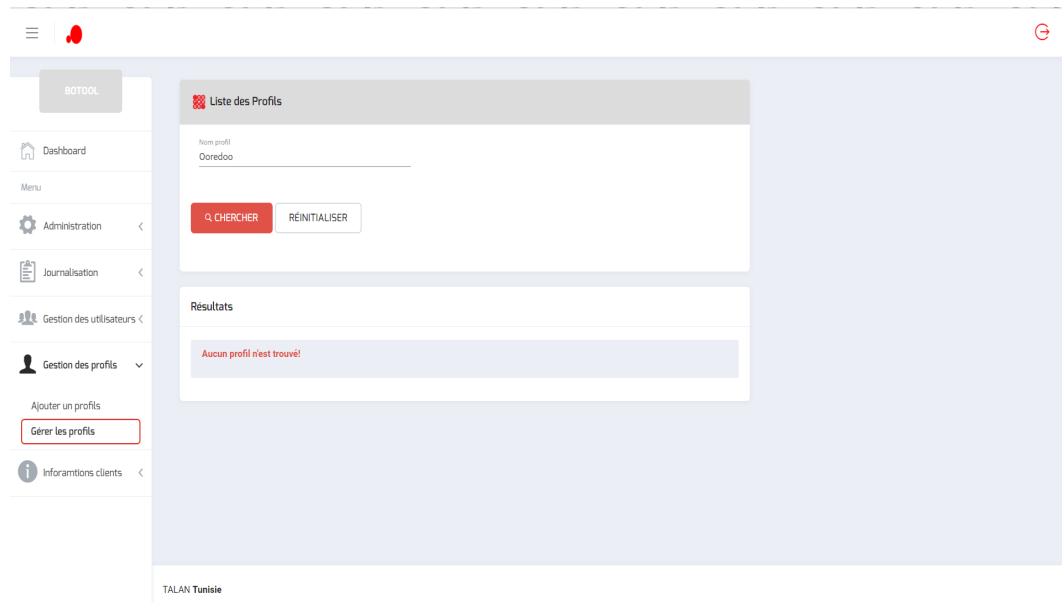


FIGURE 4.15 – Recherche d'un rôle inexistant

La configuration des écrans de BOTOOL (IHM) fait partie des responsabilités d'administrateur. Il a la possibilité de consulter la liste des menus et des pages de l'outil et configurer un écran particulier s'il le souhaite. Configurer un écran revient à rendre visible ou non cet écran aux autres utilisateurs selon leurs privilèges. Les figures 4.16, 4.17 et 4.18 ci-dessous illustrent ces opérations.

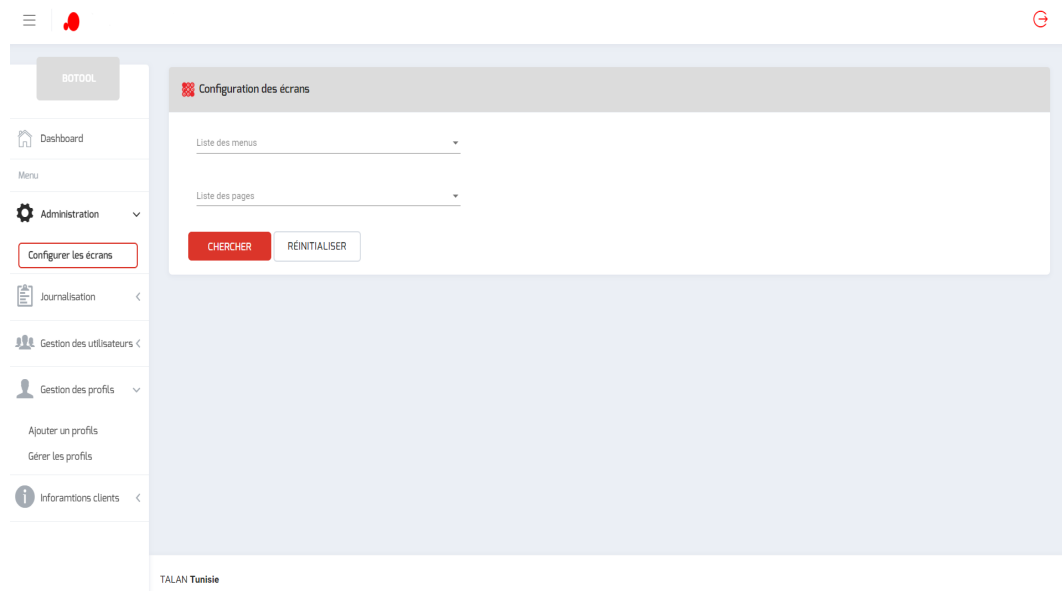


FIGURE 4.16 – Configuration des écrans de BOTOOL

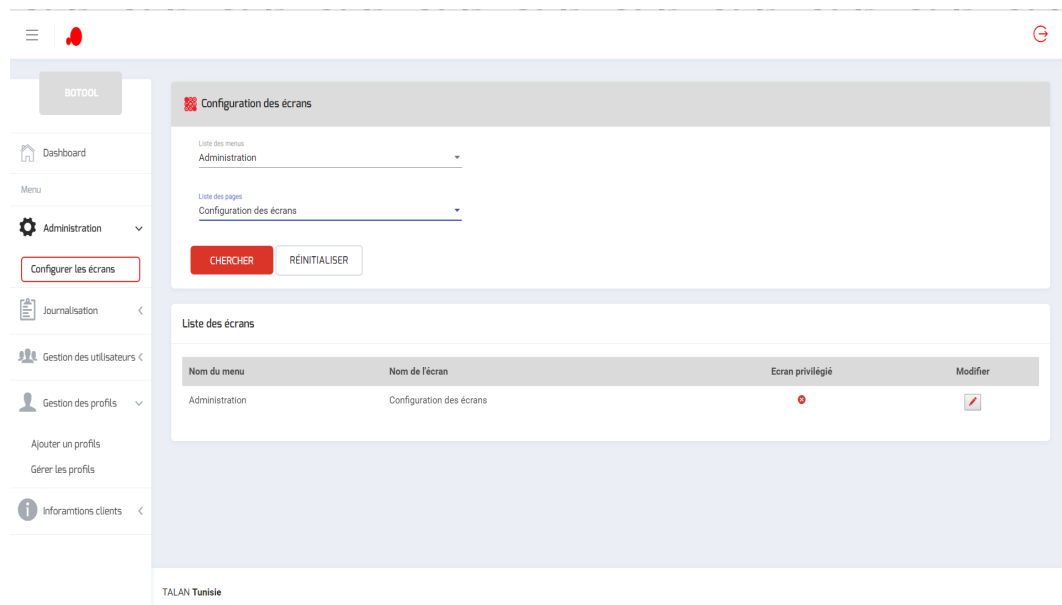


FIGURE 4.17 – Recherche d'un écrans de BOTool

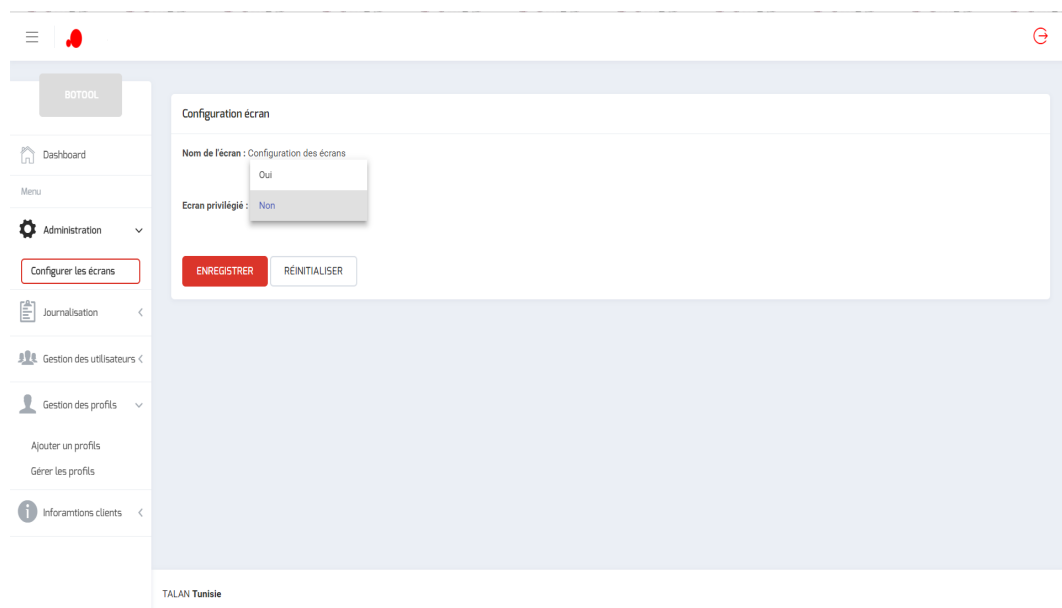


FIGURE 4.18 – Mise à jour de la configuration d'un écrans de BOTool

V Phase de test du microservice Administration

Les tests unitaires sont au centre de l'activité de programmation, il nous permettent de vérifier le bon fonctionnement d'une partie de l'application. Nous avons choisi le Fra-

mework JUnit pour l'automatisation des tests unitaires de notre application. JUnit est un cadre simple pour écrire des tests reproductibles. La figure 4.19 ci-dessous, nous donne un aperçu sur l'exécution des tests unitaires. Pour ce microservice, nous avons atteint un niveau de qualité plutôt satisfaisant.

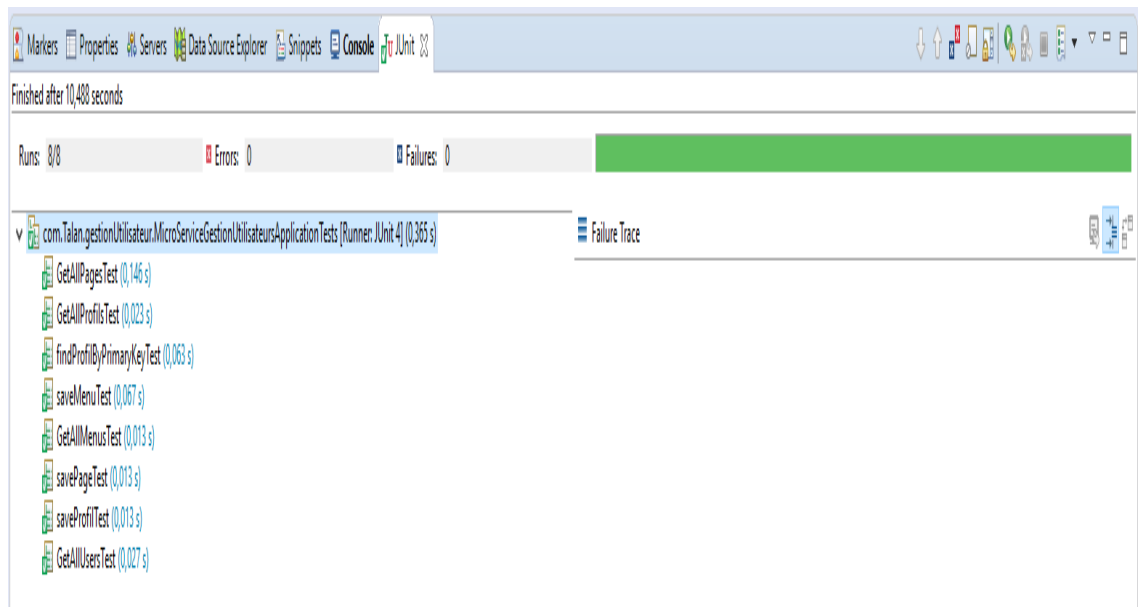


FIGURE 4.19 – Tests unitaires du microservice Administration

Conclusion

Dans ce sprint nous avons présenté le cycle de vie de notre premier microservice administration. Nous avons commencé par une spécification fonctionnelle grâce aux diagrammes de cas d'utilisation et des diagrammes de séquence système. Ensuite, nous avons détaillé la conception en présentant le diagramme de paquets du microservice ainsi qu'un diagramme de classes et un diagramme de séquence. Enfin, nous avons présenté quelques interfaces Homme-Machine de ce microservice ainsi les tests unitaires réalisés. Le chapitre suivant est consacré au microservice Client.

Chapitre 5

Sprint 2 : Microservice Client

Après avoir développé le premier microservice Administration, nous passons maintenant au développement d'un autre microservice qui permet de consulter les informations des clients de l'opérateur télécom. Ce chapitre illustre le cycle de vie du deuxième sprint à savoir le backlog du sprint, la spécification fonctionnelle, la conception, la réalisation ainsi que la phase de test.

I | Sprint Backlog

TABLE 5.1 – Backlog du sprint 2

Exigence	Sous tâche
2.1	Consulter informations client par MSISDN, code client ou par numéro de contrat.
2.2	Exporter le résultat en fichier CSV.

II | Spécification fonctionnelle du microservice Client

Pour la spécification des besoins, nous nous référerons aux diagrammes d’UML : les diagrammes de cas d’utilisation et les diagrammes de séquence.

II.1 Diagramme de cas d’utilisation du microservice Client

Tout utilisateur de l’outil BOTOOL peut consulter les informations relatifs aux clients et exporter le résultat en fichier CSV. La figure 5.1 ci-dessous représente le diagramme de cas d’utilisation du microservice Client.

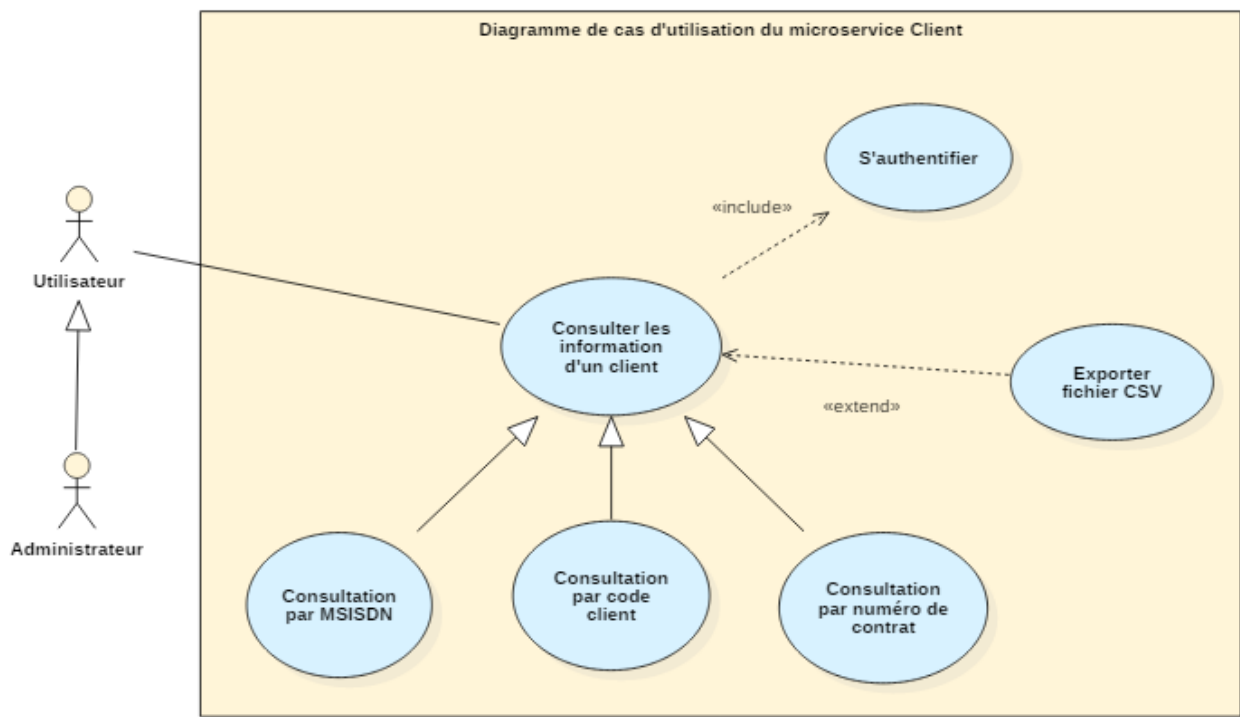


FIGURE 5.1 – Diagramme de cas d’utilisation consulter information client

Une fois authentifié, un utilisateur de BOTOOL peut consulter les informations relatifs à un client. Cette consultation peut se faire soit par la saisie du numéro MSISDN,

qui est le numéro de téléphone de l'utilisateur, soit par le code client ou par le numéro de contrat. L'utilisateur a aussi la possibilité d'exporter le résultat obtenu en fichier CSV.

II.2 Description de quelques scénarii

II.2.1 Scénario du cas d'utilisation consulter information client

La figure 5.2 ci-dessous représente le diagramme de séquence système du scénario de consultation des informations client.

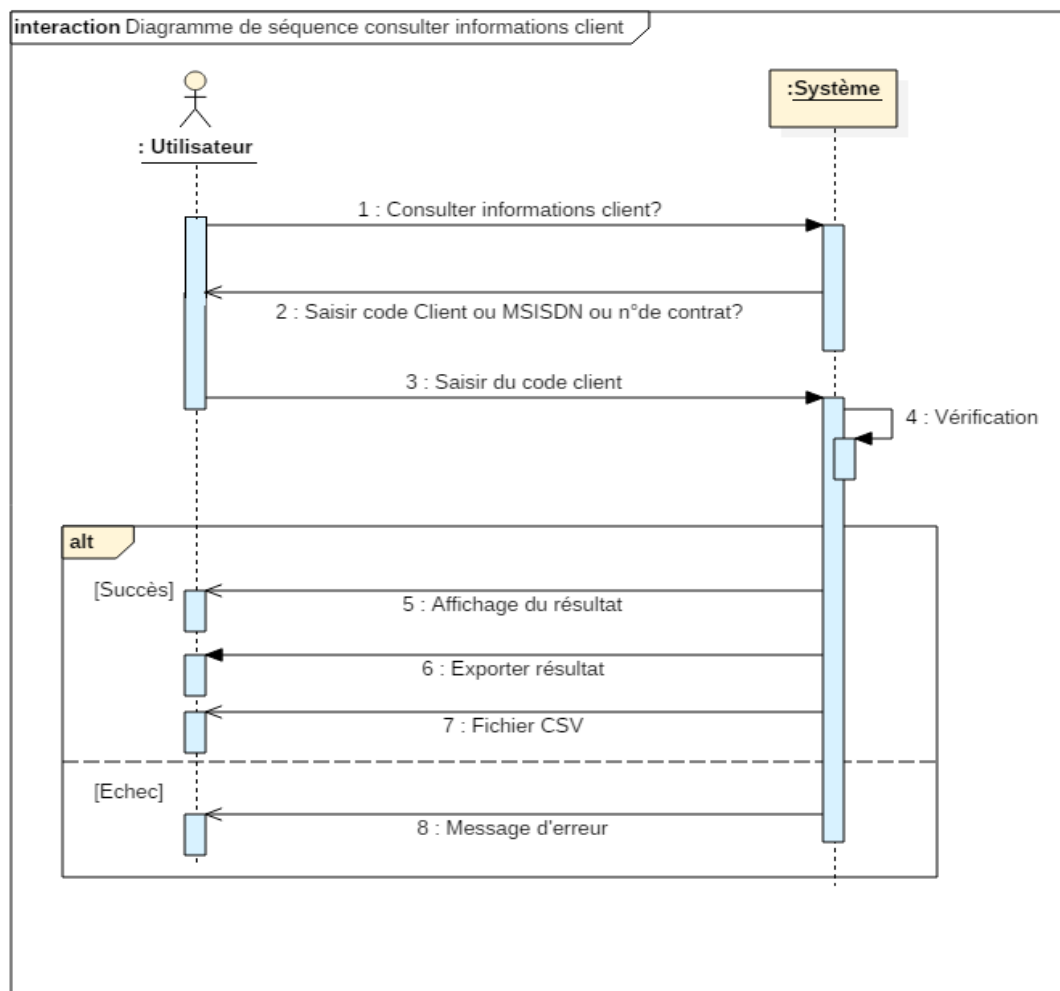


FIGURE 5.2 – Diagramme de séquence consulter information client

Pour consulter les informations d'un client, l'utilisateur saisit par exemple le code client. Si la code client saisi est valide alors le résultat est affiché et il peut l'exporter en fichier CSV. Sinon si le code saisi est soit non existant soit incorrecte alors un message d'erreur est affiché.

III Conception du microservice Client

Dans cette section nous présentons, la phase de conception du microservice Client. Nous présenterons l'architecture physique, le diagramme de paquets ainsi que le diagramme de classes.

III.1 Architecture physique du microservice Client

Pour illustrer le déploiement de notre microservice, nous avons utilisé le diagramme de déploiement, comme le montre la figure 5.3 ci-dessous, illustrant la disposition physique des différents matériels (ou noeuds) et la répartition des composants au sein des noeuds :

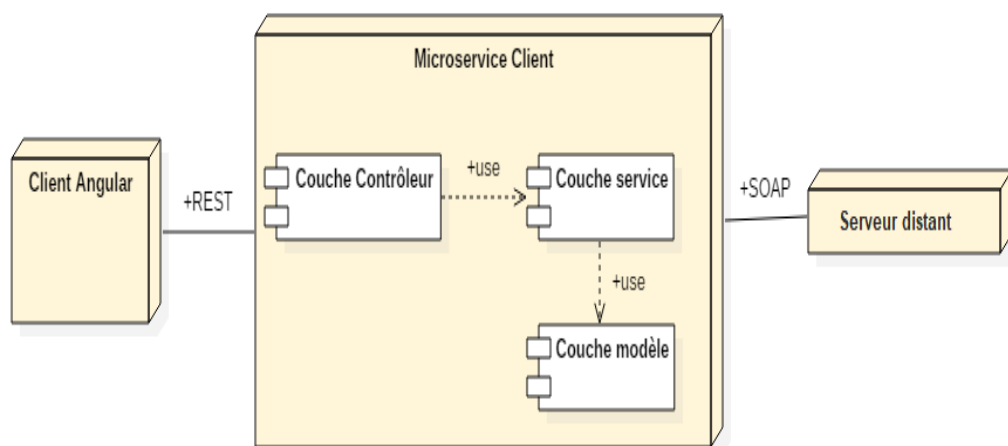


FIGURE 5.3 – Diagramme de déploiement du microservice Client

Notre diagramme de déploiement est constitué de trois nœuds principaux :

- Le client Angular qui sert d'outil de communication entre les utilisateurs de notre système et le reste des noeuds. Les utilisateurs lancent leurs demandes sous forme de requête REST et en reçoivent des réponses en format JSON.
- Le serveur web qui regroupe quatres composants à savoir :
 - La couche Contrôleur : Représente notre contrôleur Rest et contient la logique concernant les actions effectuées par l'utilisateur.
 - La couche Modèle : Contient une représentation des données que nous manipulons.
 - La couche Service : Représente les services offerts par le microservice.
- Le serveur distant qui contient les informations de tous les clients de l'opérateur télécom.

III.2 Diagramme de classes du microservice Client

La figure 5.4 présente la structure en classe du microservice Client.

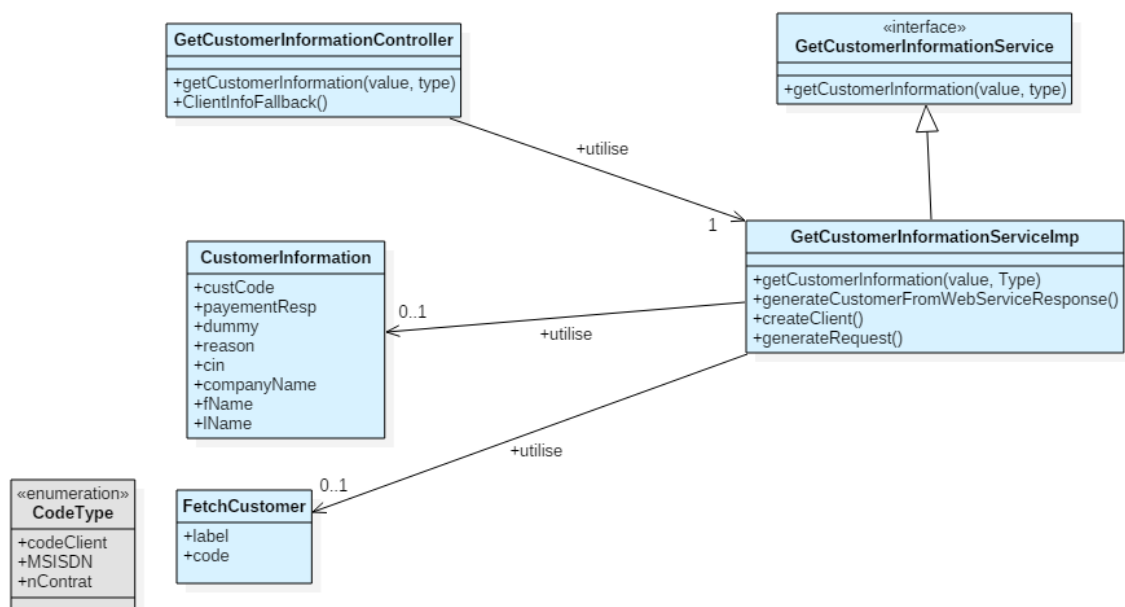


FIGURE 5.4 – Diagramme de classes du microservice Client

TABLE 5.2 – Les principales classes conçues pour l’élaboration du microservice

Nom de la classe	Description
CustomerInformation	Cette classe représente le modèle de données des informations sur les clients.
FetchCustomer	Cette classe contient le modèle de données de la requête de consultation informations clients. Le code Client peut être soit le numéro de téléphone du client (MSISDN), ou bien le code client ou bien le numéro de contrat.
GetCustomerInformationService	Cette interface contient les opérations nécessaires du web service consultation informations clients.
GetCustomerInformationServiceImp	Cette classe contient l’implémentation des opérations du web service.
GetCustomerInformationController	Cette classe intercepte les requêtes de l’utilisateur et retourne le service REST correspondant à la consultation des informations client.

III.3 Diagramme d’activités consultation informations client

Afin de consulter les informations d’un client, l’utilisateur doit effectuer une recherche soit par la saisie du numéro de téléphone (MSISDN), du code client ou par numéro de contrat. En cas d’échec un message d’erreur apparaît indiquant à l’utilisateur que la valeur saisie est incorrecte ou bien qu’il y a eu une exception côté serveur. La figure 5.5 ci-dessous, illustre le diagramme d’activités du microservice Client.

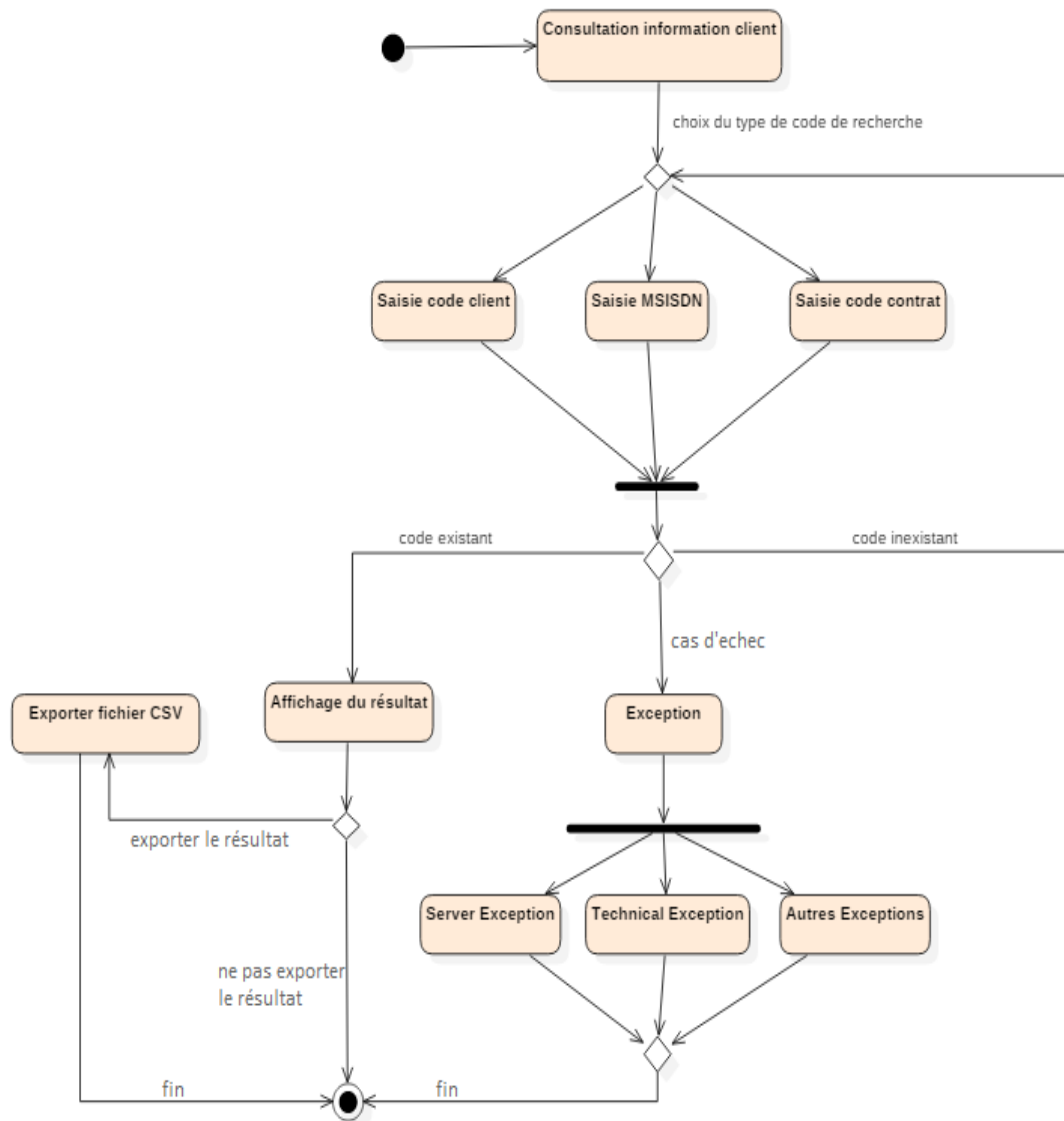


FIGURE 5.5 – Diagramme d'activités du microservice Client

IV Réalisation du microservice Client

Nous exposerons dans cette section les interfaces Homme-Machine du microservice Client. Les figures 5.6 et 5.7 représente l'interface de consultation informations client.

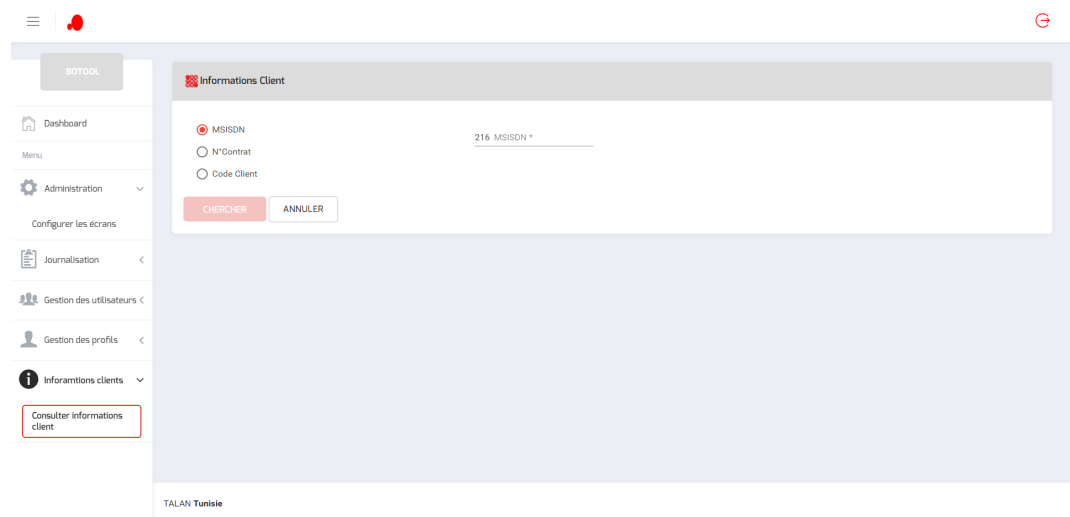


FIGURE 5.6 – Consulter informations client

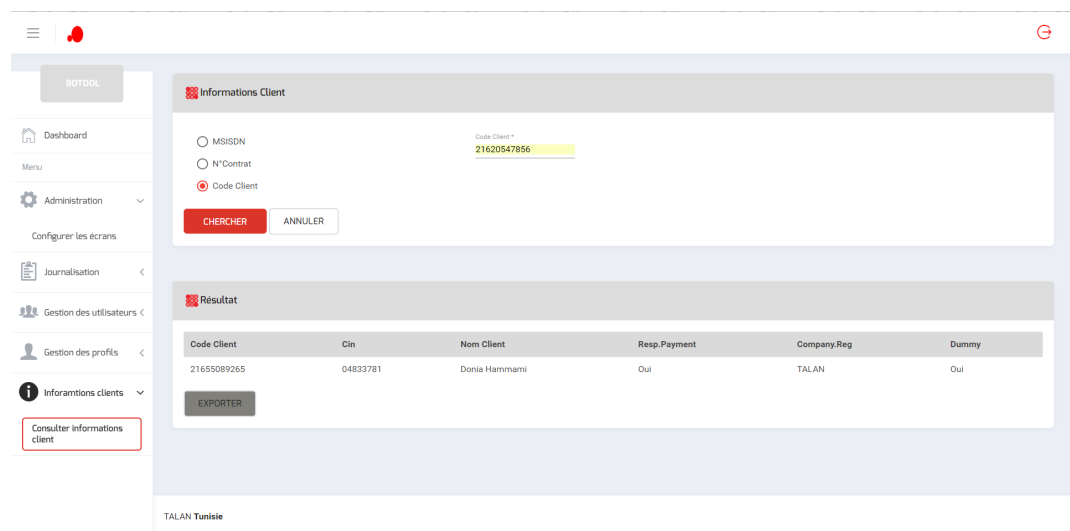


FIGURE 5.7 – Résultat de consultation informations client

La figure 5.8 représente le résultat obtenu en cas d'échec de l'opération de consultation.

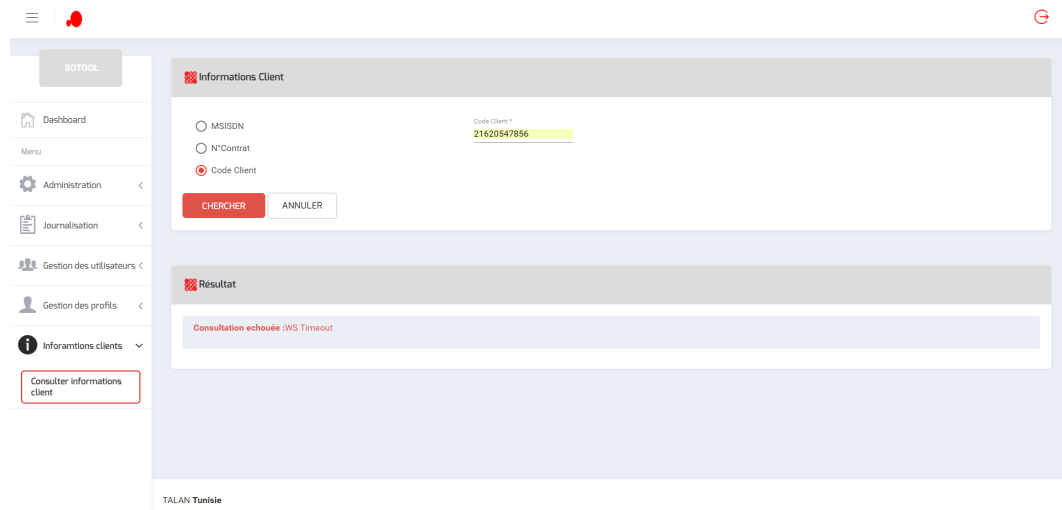


FIGURE 5.8 – Résultat de consultation informations client en cas d'échec

V Phase de test du microservice Client

Pour vérifier le bon fonctionnement du service du microservice Client nous avons réalisé les tests unitaires illustrés par la figure 5.9. Nous avons utilisés des Mocks pour suivre les états de la requêtes et simuler le comportement de consultation des informations client depuis le serveur distant de l'opérateur télécom.

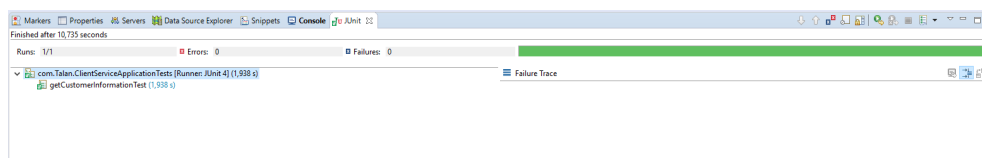


FIGURE 5.9 – Test unitaire de microservice Client

Conclusion

Dans ce chapitre nous avons présenté la construction de microservice Client. Nous avons mis l'accent sur sa spécification et sa conception puis le livrable sur lequel nous avons effectué les tests unitaires. Le chapitre suivant illustre le cycle de vie du microservice Journalisation.

Chapitre 6

Sprint 3 : Microservice

Journalisation

Après avoir achevé le développement des microservices Administration et Client, nous détaillerons dans ce chapitre le cycle de vie du dernier sprint. Nous présenterons, tout d'abord le backlog du sprint, par la suite nous détaillerons la spécification fonctionnelle du microservice Journalisation, la conception, la réalisation ainsi que la phase de test.

I | Sprint Backlog

TABLE 6.1 – Backlog du sprint 3

Exigence	Sous tâche
3.1	Consulter le journal des opérations.

II | Spécification fonctionnelle du microservice Journalisation

Pour la spécification des besoins, nous présenterons le diagramme de cas d'utilisation ainsi que le diagramme de séquences relatif au microservice Journalisation.

II.1 Diagramme de cas d'utilisation du microservice Journalisation

La figure 6.2 ci-dessous, illustre le diagramme de cas d'utilisation de consultation du journal d'opérations effectuées par les différents utilisateurs.

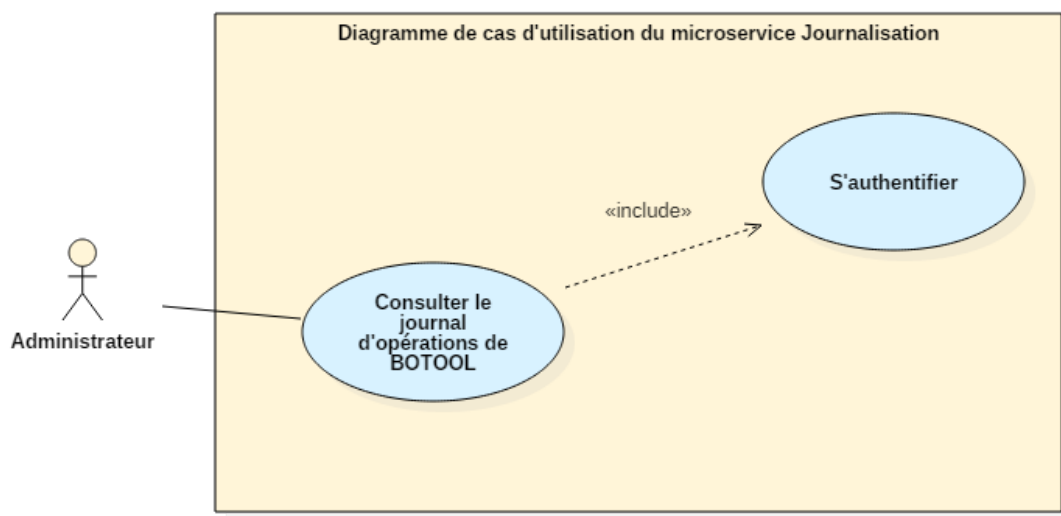


FIGURE 6.1 – Diagramme de cas d'utilisation du microservice Journalisation

II.2 Scénario du cas d'utilisation consulter le journal d'opérations

La figure 6.2 ci-dessous, illustre le scénario du cas d'utilisation de consultation du journal d'opération de BOTool.

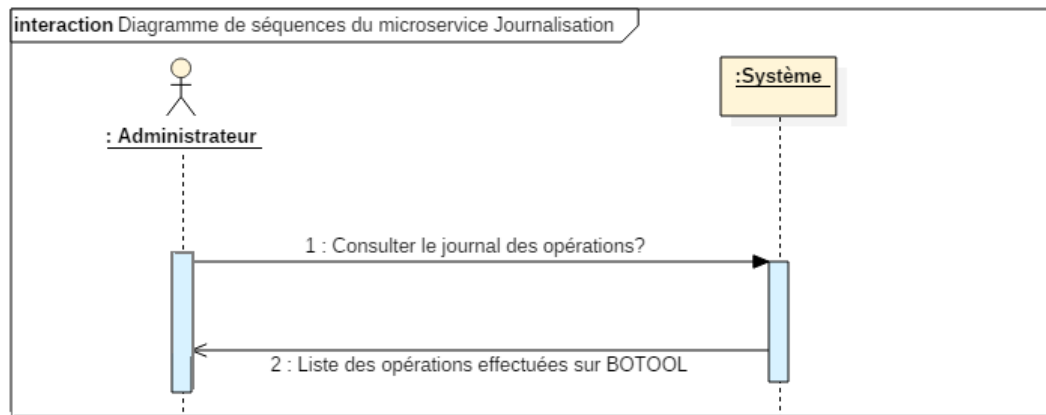


FIGURE 6.2 – Diagramme de séquences du microservice Journalisation

III | Conception du microservice Journalisation

Cette section présente la phase de conception du microservice Journalisation. Nous présenterons l'architecture logicielle, le diagramme de paquets ainsi que le diagramme de classes.

III.1 Architecture logicielle du microservice Journalisation

La figure 4.6 présente l'architecture logicielle du microservice Journalisation qui est constituée de trois couches :

- Couche présentation : c'est la couche qui permet d'assurer la communication entre le microservice et l'environnement externe..
- Couche logique métier : c'est la couche qui encapsule la logique métier du microservice.
- Couche persistance de données.

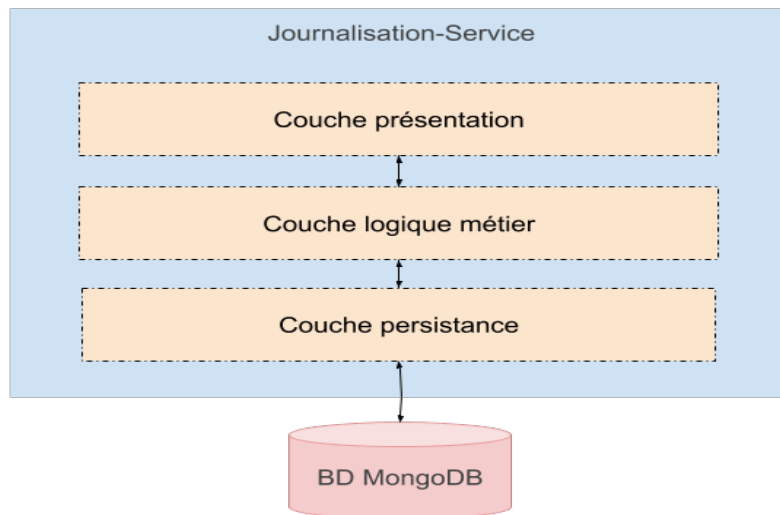


FIGURE 6.3 – Architecture logicielle du microservice Journalisation

III.2 Diagramme de paquets du microservice Journalisation

La figure 6.4 ci-dessous illustre le diagramme de paquet de notre microservice.

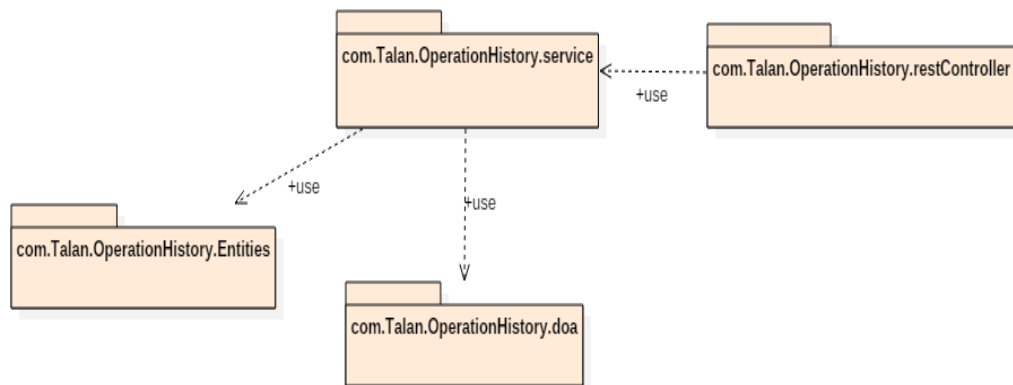


FIGURE 6.4 – Diagramme de paquets du microservice Journalisation

- `com.Talan.OperationHistory.entities` : contient l'entité `BOTOOLHistory` de notre microservice.
- `com.Talan.OperationHistory.dao` : contient toutes les méthodes nécessaires pour l'accès à la base de données.
- `com.Talan.OperationHistory.service` : présente notre logique métier.
- `com.Talan.OperationHistory.controller` : présente notre REST API.

III.3 Diagramme de classes du microservice Journalisation

La figure 6.5 ci-dessous présente le diagramme de classes relatif au microservice Journalisation.

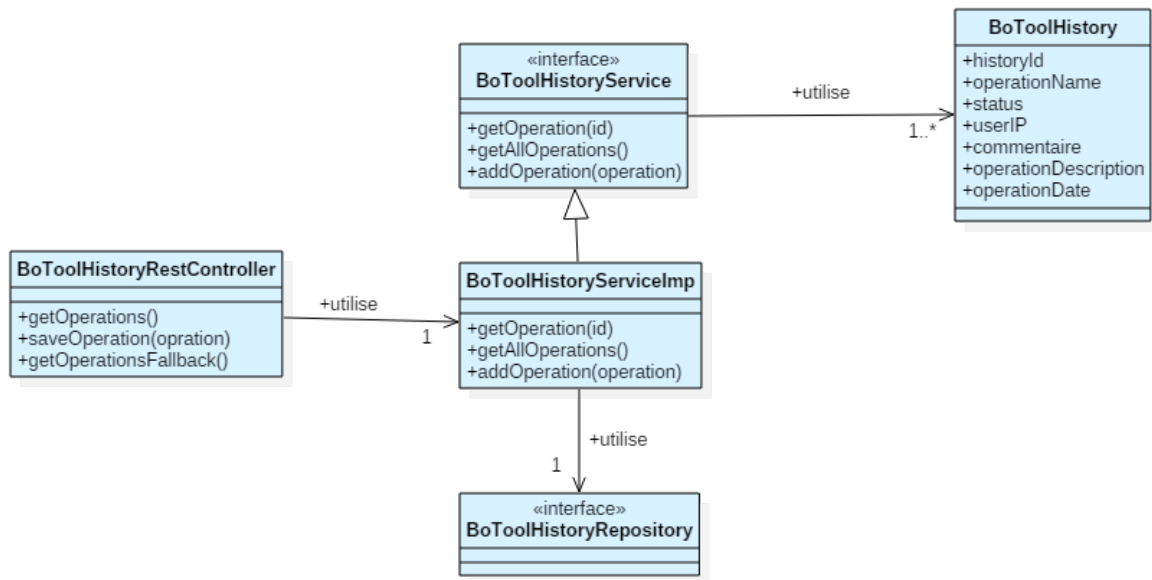


FIGURE 6.5 – Diagramme de classes du microservice Journalisation

TABLE 6.2 – Les principales classes conçues pour l’élaboration du microservice

Nom de la classe	Description
BoToolHistory	Cette classe représente l’entité de journalisation.
BoToolHistoryRepository	C’est l’interface de la couche DAO qui implémentent l’interface JPA Repository de Spring Data.
BoToolHistoryynService	Cette interface contient les opérations nécessaires du service de journalisation.
BoToolHistoryServiceImp	Cette classe contient l’implémentation des opérations de l’interface service.

BoToolHistoryController	Cette classe intercepte les requêtes de l'utilisateur et retourne le service REST correspondant à la consultation du journal et de sauvegarde de la trace des opérations effectuées.
--------------------------------	--

IV | Réalisation du microservice Journalisation

Toute opération effectuée sur BOTOOL a une trace sauvegardée par le microservice Journalisation. La figure 6.6 ci-dessous, représente la liste des opérations effectuées. Pour chaque opération, nous sauvegardons le nom de l'utilisateur, son adresse IP, la date, une description de l'opération effectuée ainsi que son statut : OK si l'opération est effectuée avec succès et KO sinon.

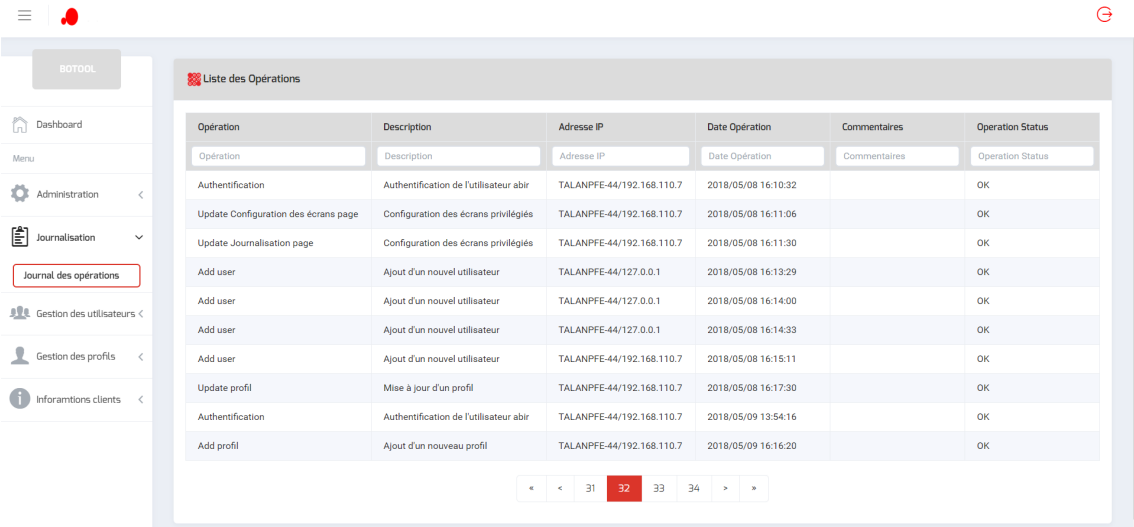


FIGURE 6.6 – Liste des opérations effectuées

V Phase de test du microservice Client

Pour vérifier le bon fonctionnement du service du microservice Journalisation, nous avons réalisé les tests unitaires illustrés par la figure 6.7.

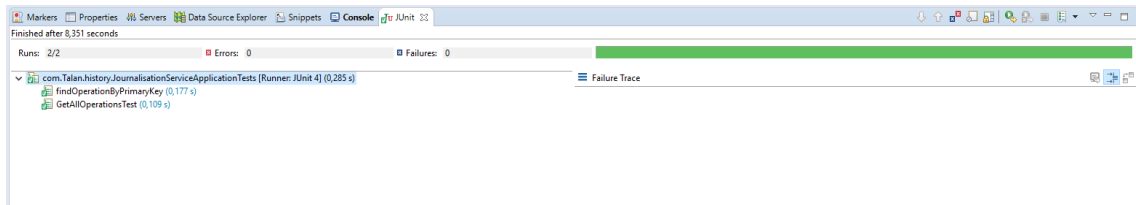


FIGURE 6.7 – Test unitaire de microservice Journalisation

Conclusion

Ce chapitre illustre le dernier microservice de l'outil BOTOOL. Nous avons présenté tout d'abord, la spécification fonctionnelle et la conception. Ensuite, nous avons présenté le livrable testé et validé. Dans le chapitre suivant nous détaillerons la vue globale du projet et l'intégration des différents microservices.

Chapitre 7

Intégration des microservices de l'outil BOTOOL

Dans ce chapitre, nous exposerons dans un premier temps l'environnement de travail matériel et logiciel utilisé pour la mise en place de notre écosystème. Dans un deuxième temps, nous présenterons la phase d'intégration afin de formuler une application composée de microservices qui va apparaître à l'utilisateur comme étant un composant unique.

I | Environnements de travail

Tout au long de la réalisation de notre projet, nous avons utilisé des matériels et des logiciels bien particuliers que nous présentons dans ce qui suit.

I.1 Environnements de développement matériel

Pour mener à bien la réalisation, nous avons utilisé comme environnement matériel, un poste de travail ayant les caractéristiques suivantes :

- Système d'exploitation : Windows 10

- Disque dur : 8 Go
- Ram : 16 Go
- Processeur : Intel(R) Core(TM)i5-6500U CPU 3.20GHz

I.2 Environnements de développement logiciel

Dans cette partie, nous nous intéressons aux langages, aux bibliothèques et aux techniques de programmation utilisées tout au long de la réalisation de notre application en justifiant notre choix.

I.2.1 Spring Boot

Spring Boot est un framework avancée qui simplifie le démarrage et le développement de nouvelles applications Java EE. Les configurations sont atténuées avec Spring Boot, qui soutient des conteneurs embarqués. Cela permet à des applications web de s'exécuter indépendamment et sans déploiement sur le serveur web.[\[7\]](#)

I.2.2 Spring Cloud

Spring Cloud, basé sur Spring Boot, offre une boîte à outils permettant de créer un système d'applications distribuées rapidement.[\[8\]](#)

I.2.3 Spring Data JPA

Spring Data JPA vise à améliorer de manière significative la mise en œuvre des couches d'accès aux données en réduisant la quantité de code à écrire. Les développeurs n'ont à écrire que leurs interfaces de référentiel, y compris les méthodes de recherche personnalisées, et Spring fournira automatiquement l'implémentation.

I.2.4 Spring Data MongoDB

Pour une utilisation simplifiée de la base de données NOSQL MongoDB nous avons opté pour le framework Spring Data MongoDB pour l'implémentation de la couche dao du microservice journalisation, qui propose une certaine unification pour les accès aux bases de données NOSQL. Spring Data MongoDB est une implémentation qui fournit une couche d'abstraction au client Java fourni par Mongo et offre les fonctionnalités nécessaires en s'intégrant avec le framework Spring.

I.2.5 Spring Cloud Config

Spring Cloud Config permet de centraliser les configurations du système distribué. Cela dans le but de rendre plus aisé la maintenance des microservices. [8]

I.2.6 Spring Eureka Service

Eureka, faisant partie du projet Netflix Open Source Software, est une application permettant la localisation d'instances de services. Elle se caractérise par une partie serveur et une partie cliente. Ces services doivent être créés en tant que clients Eureka, ayant pour objectif de se connecter et s'enregistrer sur un serveur Eureka. [9]

I.2.7 Zuul Proxy Service

Zuul, faisant partie de la stack Netflix OSS, joue le rôle de point d'entrée à notre système. Il se place donc en entrée de l'architecture et permet de réaliser des opérations sur les requête ainsi que sur leurs retour. [10]

I.2.8 Hystrix Circuit breaker

Hystrix, logiciel open source faisant partie de la stack Netflix OSS. Ce logiciel permet d'arrêter rapidement un service qui ne peut plus fonctionner, afin éviter des problèmes de surcharge. Il implémente aussi un système de "circuit breaker" et de "fallback" qui permet

de rompre la connexion entre la requête et le service défaillant.[\[11\]](#)

I.2.9 Ribbon Load Balancer

Ribbon est un sous-projet Spring Cloud qui prend en charge la fonctionnalité d'équilibrage de charge entre les différentes instances des microservices. Il peut fonctionner comme un composant autonome, ou s'intégrer et travailler en toute transparence avec Zuul pour le routage du trafic.[\[12\]](#)

I.2.10 Base de données NOSQL : MongoDB

MongoDB est un système de gestion de bases de données NoSQL, orienté document classé parmi les «leader» de systèmes de gestion de base de données. Les données sont stockées dans des collections, qui sont l'équivalent des tables dans une base relationnelle.

I.2.11 Angular 5

Angular 5 est l'un des frameworks Javascript les plus avancés pour le web, créé par Google. Il est le résultat d'amélioration d'Angular 4 et Angular 2, développé avec TypeScript. Il offre des performances améliorées, une modularité amplifiée, un respect des nouveaux standards du web et un code plus expressif.

II | Intégration des microservices

Après avoir effectué nos choix technologiques pour mettre en place notre système, nous pouvons donner maintenant une image globale sur l'architecture cible et son fonctionnement. La figure 7.1 ci-dessous présente l'architecture résumant notre étude technique évoquée précédemment. Cette architecture se compose d'une partie front-end développée avec Angular5 et d'une partie back-end développé avec Spring boot. Au début, les microservices consultent le Config Server afin de récupérer leurs configurations. Une fois que le microservice soit démarré, le service d'enregistrement Eureka le détecte et l'ajoute

à l'annuaire. Pour que Zuul Proxy Server redirige une requête vers un microservice, il consulte le service de découverte Eureka afin de connaître toutes les instances disponibles du microservice concerné.

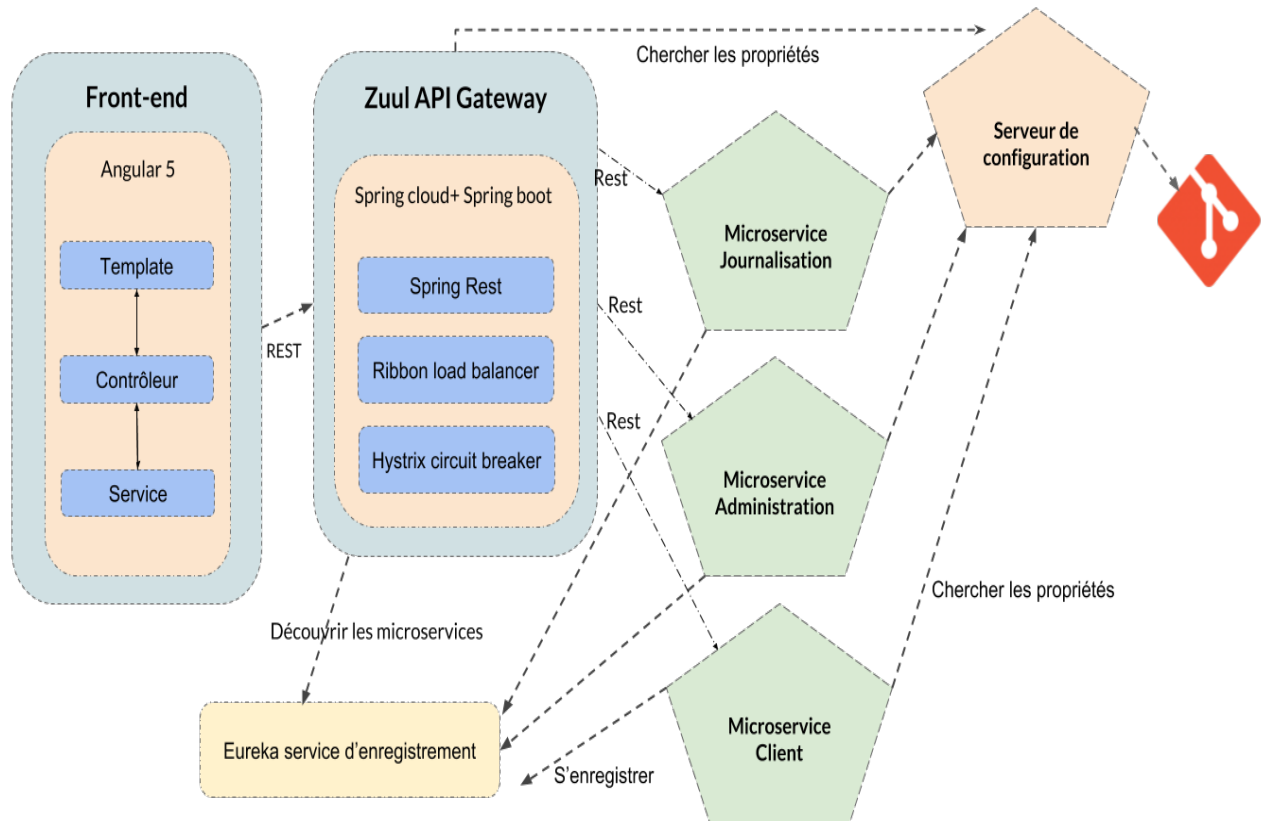


FIGURE 7.1 – Architecture technique en microservices

II.1 Service de découverte Eureka

Dans notre architecture nous avons opté pour le service de découverte Eureka de la stack Netflix. La figure 7.2 ci-dessous, représente l'enregistrement des microservices dans le service de découverte.

La figure 7.3 ci-dessous représente le dashboard de l'annuaire de service Eureka, dans lequel nous pouvons visualiser toutes les instances de nos microservices.

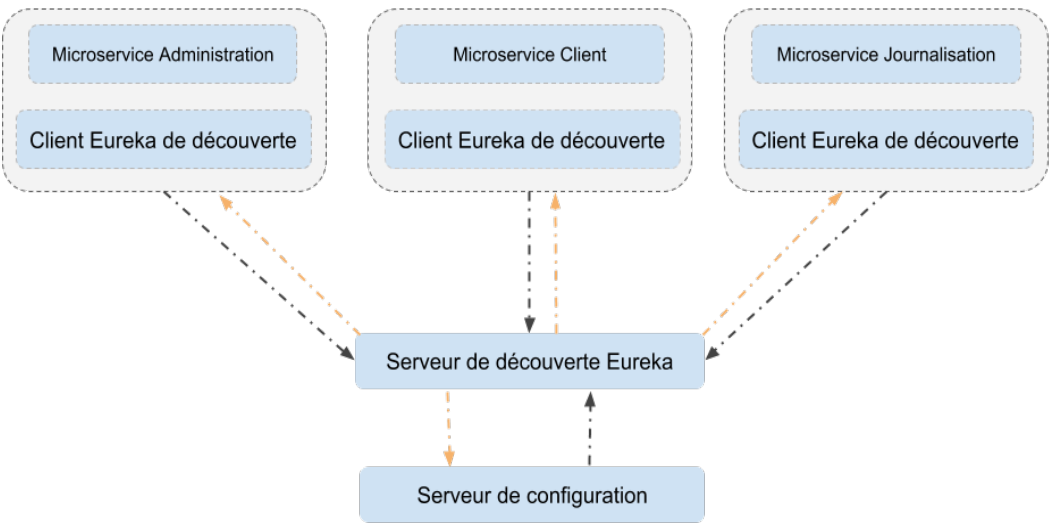


FIGURE 7.2 – Enregistrement des microservices de BOTOOL dans l’annuaire

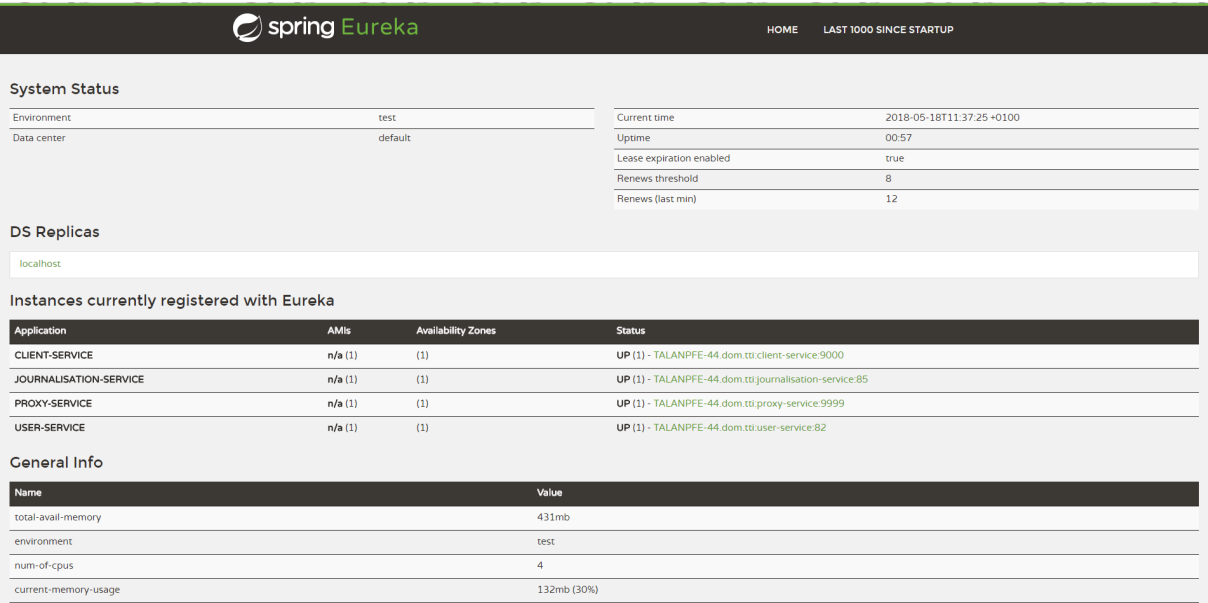


FIGURE 7.3 – Dashboard Eureka

II.2 L’API Gateway Zuul

Dans un style architectural en microservices, les services changent leurs emplacements très souvent. Afin d’assurer une transparence au client, nous avons utilisé l’API Gateway Zuul, qui représente le point d’entrée à notre application.

Zuul offre une seule url au client lui permettant de naviguer entre les différents modules de BOTOOL. Afin de router les requêtes aux microservices concernés, Zuul consulte

l'annuaire des services Eureka pour connaître les adresses des instances disponibles. La figure 7.4 illustre le fonctionnement du proxy.

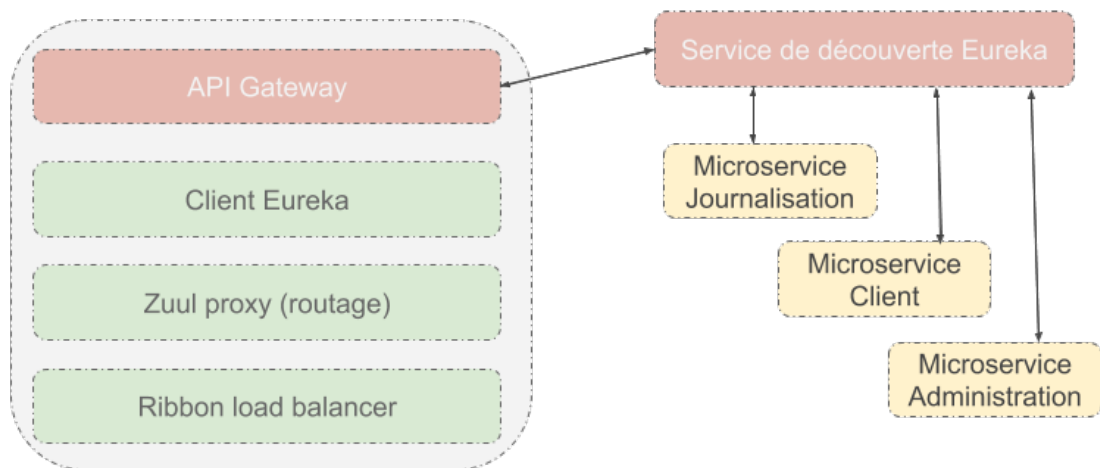


FIGURE 7.4 – Fonctionnement de l'API Gateway Zuul

II.2.1 Tolérance aux pannes au niveau de l'API Gateway

De chaque microservice, nous avons plusieurs instances dupliquées, si l'instance appelée par le proxy est défaillante, alors il appelle une autre instance inscrite dans l'annuaire de service Eureka. Dans le but d'assurer la tolérance aux pannes, l'utilisation du Circuit Breaker, implémenté par l'outil Hystrix, est primordiale. La figure 7.5 ci-dessous illustre la tolérance aux pannes pour le microservice Administration.

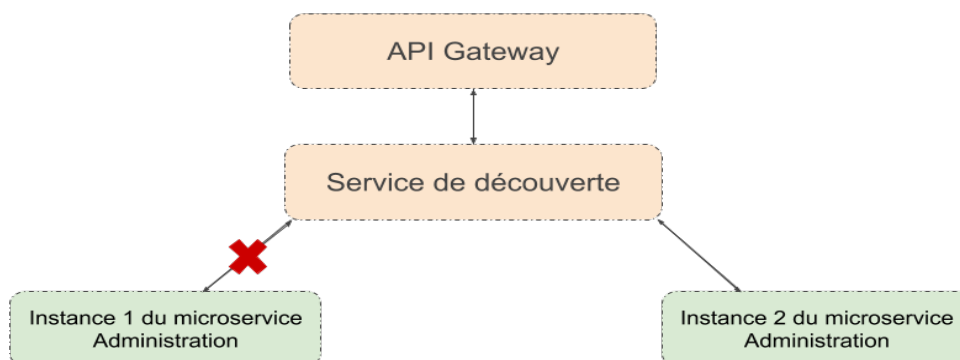


FIGURE 7.5 – Tolérance aux pannes au niveau de l'API Gateway

II.2.2 Mise à l'échelle des microservices via Ribbon Netflix

Pour mettre nos microservices à l'échelle, nous avons utilisé Ribbon Load Balancer afin de répartir la charge entre toutes les instances des microservice. Il est intégré d'une façon transparente dans l'API Gateway Zuul. Ribbon répartit la charge entre les différents instances des microservices inscrits chez le service de découverte selon plusieurs algorithmes et règles.

II.3 Communication entre microservices

Toute opération effectuée sur BOTOOL doit garder une trace dans le journal des opérations. Ce besoin est traduit en une communication à travers des appels REST entre les différents microservices. Cependant, les appels à distance peuvent échouer parfois, ou on risque d'attendre sans réponse jusqu'à ce qu'un délai d'expiration. Donc dans le but d'assurer une tolérance à ce problème, nous avons opté pour la solution le circuit breaker hystrix. Un dashbord est offert par hystrix afin de faire le suivi des microservices en cours d'exécution.

En cas de défaillance de toutes les instances des microservices, il existe une méthode de fallback qui offre des résultats par défaut. Le circuit breaker réalisé avec Hystrix enveloppe cette méthode et surveille les défaillances.

III | Chronogramme

Ce travail a été réalisé durant une période de six semaines. La répartition des tâches durant notre stage est illustrée par le diagramme de Gantt réel de la figure 7.6 ci-dessous :

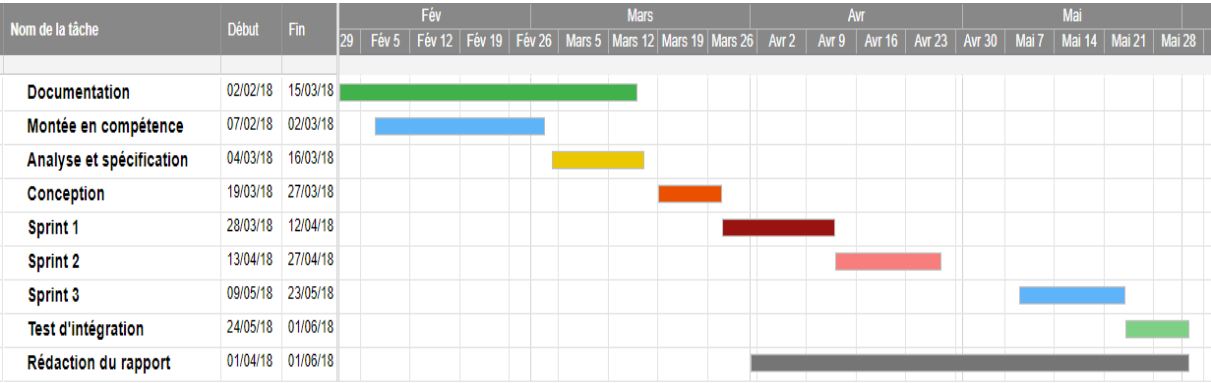


FIGURE 7.6 – Diagramme de gantt réel

Conclusion

Ce chapitre représente une récapitulation de tout le travail élaboré pendant notre stage. En effet, nous avons décrit les environnements matériels et logiciels sur lesquels nous avons travaillé. Nous avons ensuite détaillé notre architecture selon les technologies choisies et nous avons clôturé ce chapitre par la présentation du chronogramme des tâches.

Conclusion générale et perspectives

Adopter l'approche microservices n'est pas un sujet de prospective mais bien une réalité pour les entreprises, surtout que les géants du web comme Netflix et Amazon la trouvent efficace et fiable. En effet, les entreprises doivent désormais comprendre que le choix architectural est un enjeu très important qui conditionne la performance de leurs activités.

Le projet de refonte architecturale de l'outil « Back Office Tool » (BOTOOL) réalisé par Talan Tunisie, constitue un pas en avant vers une agilité de bout en bout. L'objectif ultime est issu d'une pure volonté stratégique, vise à faire évoluer son organisation, mais aussi à apporter des nouveaux projets à l'entreprise plus robustes et plus performants.

Pour mener à terme ce projet, nous avons procédé par une suite d'étapes bien étudiées. Nous étions responsables de la mise en place de l'architecture de la nouvelle solution ainsi que la conception, la réalisation et les tests des microservices. En se basant sur une architecture en microservices, nous avons résolu plusieurs problèmes présentés par l'architecture monolithique de BOTOOL.

Dans le présent rapport, nous avons détaillé les étapes par lesquelles nous sommes passés pour réaliser notre solution. Afin d'aboutir à ce résultat, nous avons tout d'abord commencé par présenter le cadre général de notre travail. Puis, nous avons présenté les concepts clés liés à notre solution. Nous avons détaillé aussi les différents besoins et les exigences relevées. Ensuite, nous avons abordé la spécification, la conception, la réalisation et le test des différents microservices. Finalement, l'étape d'intégration des microservices, au cours de laquelle nous avons présenté le fonctionnement de l'outil BOTOOL via les microservices.

Durant ce projet, nous avons été confrontés à plusieurs problèmes et obstacles au niveau développement. En effet, nous avons fait face aux défis exigés par l'architecture microservice comme la recherche des microservices en utilisant le service de découverte et la communication entre les différents microservices.

Afin d'améliorer certains aspects, nous proposons d'enrichir cette application en s'intéressant à certains points. Nous envisageons de déployer les des microservices réalisés sur le cloud, migrer quelques autres modules en architecture en microservices ainsi que l'automatisation des tests d'intégration entre les microservices.

Bibliographie

[B1]C. Posta, Microservices for Java Developers. O'Reilly(2016). 11, 24

[B2]S. Newman, Building Microservices. O'Reilly(2015). 11

[B3]M. Hariati et D. Meslati, Les Composants Logiciels et La Séparation Avancée des Préoccupations : Vers une nouvelle Approche de Combinaison, janvier 2009

Netographie

- [1] <http://www.talan.tn/> (consulté le 22/02/2018)
- [2] <https://www.pentalog.fr/notre-demarche/methode-agile-scrum.htm> (consulté le 12/03/2018)
- [3] <http://www.nutcache.com/fr/blog/sprint-agile/> (consulté le 12/03/2018)
- [4] <https://martinfowler.com/microservices/> (consulté le 12/03/2018)
- [5] <http://martinfowler.com/bliki/BoundedContext.html> (consulté le 12/03/2018)
- [6] <http://blog.zenika.com/2012/09/27/la-persistance-polyglotte-avec-grails/> (consulté le 12/03/2018)
- [7] <https://o7planning.org/fr/11267/le-tutoriel-de-spring-boot-pour-les-debutants> (consulté le 02/04/2018)
- [8] <http://blog.ippon.fr/2015/10/16/rex-architecture-orientee-microservices-avec-netflix-oss-et-spring-article-2/> (consulté le 02/04/2018)
- [9] <https://www.supinfo.com/articles/single/3638-spring-cloud-architecture-micro-services> (consulté le 02/04/2018)
- [10] <http://blog.ippon.fr/2015/10/20/rex-architecture-orientee-microservices-avec-netflix-oss-et-spring-article-3/> (consulté le 02/04/2018)
- [11] <http://blog.ippon.fr/2015/11/04/rex-architecture-orientee-microservices-avec-netflix-oss-et-spring-article-4/> (consulté le 02/04/2018)

[12] <https://fr.linkedin.com/pulse/mise-en-place-dune-architecture-microservice-avec-sydney-gael> (consulté le 02/04/2018)

ملخص

يلخص هذا التقرير الأعمال المنجزة في إطار تربص نهاية الدراسة للحصول على دبلوم مهندس وطني في الإعلامية داخل المؤسسة تالون تونس. ويتمثل العمل في إعادة تصميم تطبيق واب يقوم بإدارة موارد مشغل الاتصالات وفق الميكروسرفيس وإيجاد بديل لبعض التكنولوجيات القديمة التي استوفت حدودها

المفاتيح : ميكروسرفيس، سبرينغ بوت، سبرينغ كلاود، نات فليكس أسس

Résumé

Le présent rapport synthétise le travail effectué dans le cadre du projet de fin d'études pour l'obtention du diplôme national d'ingénieur en informatique au sein de l'entreprise Talan Tunisie Consulting. Le travail consiste à une refonte d'une application web de gestion de ressources d'un opérateur télécom en se basant sur une architecture en microservices ainsi que trouver des alternatives pour quelques technologies qui ont présenté leurs limites.

Mots clés : Microservices, Spring Boot, Spring Cloud, Netflix OSS

Abstract

This report summarizes the work carried out as part of the final project to obtain the national software engineer degree in Talan Consulting Company. The work consists of a redesign of a web application of management of the resources for a telecom operator based on microservices architecture as well as finding alternatives for some technologies that have presented their limits.

Key words : Microservices, Spring Boot, Spring Cloud, Netflix OSS