
Dédicaces

*A celle qui m'a mis sur la bonne route, qui m'a entouré d'amour, d'affection et qui fait tout pour
ma réussite, que dieu la garde ma chère maman Fatma*

*A mon père Ammar et mon frère Kadhem que Dieu vous bénisse et j'espère que vous seriez fiers de
moi,*

*A celles qui m'apportent la joie et le bonheur, A mes chères adorables sœurs Khouloude & Raja,
A tous les moments d'enfance passés ensemble,*

A celle qui m'a aimé toujours et m'a gâté le plus, A ma tante et ma deuxième mère Mabrouka

*A celle qui était toujours là pour moi, qui m'a assisté dans les moments difficiles et m'a pris
doucement par la main pour traverser ensemble des épreuves pénibles, A mon amie Sarra,*

A ma cousine Soulef et son mari pour leur support et bienveillance,

A celui qui me considère comme sa fille... A mon cher EL Hedi,

A ceux qui je considère comme ma deuxième famille... A mes adorables amis,

*A ceux qui ont toujours été mes sources d'inspiration... A mes professeurs pendant mon parcours
éducatif de l'école primaire vers l'ENSIT*

A toutes les personnes qui me sont chères, et à toutes celles qui veulent partager mon bonheur,

Je dédie ce modeste travail, symbole de ma profonde gratitude et couronnement de leur assistance.

Remerciements

C'est avec plaisir que je réserve ces quelques lignes en signe de gratitude et de profonde reconnaissance à tous ceux qui m'ont écouté, conseillé, critiqué, encadré et contribué d'une façon ou d'une autre à l'aboutissement de ce travail.

Je tiens à remercier :

Madame Ines BAYOUDH, mon enseignante à l'ENSIT, pour avoir accepté de m'encadrer et qui, avec ses conseils, sa collaboration et sa disponibilité, m'a assuré le bon déroulement de mon projet.

Je remercie bien évidemment mes encadrantes à Busines & Decision Tunisie Madame Emna MARZOUK et Madame Marwa DRIDI, pour leur disponibilité, ses remarques pertinentes, ses conseils constructives et l'aide qu'elles m'ont accordée pendant l'élaboration de ce projet.

J'exprime ma profonde reconnaissance à tout le cadre professoral de l'ENSIT pour la formation d'excellence qu'ils nous ont offert, pour leurs conseils et leurs contributions pour que nous entamons notre carrière professionnelle avec des bases solides.

Je remercie aussi les membres du jury tout en espérant qu'ils trouvent dans ce rapport les qualités de motivation et de clarté qu'ils espèrent.

Tables des matières

Introduction générale	1
Présentation Générale	3
1. Présentation de l'organisme d'accueil.....	3
1.1. Business & Descision Group	3
1.2. Business & Decision Tunisie	4
2. Contexte général du projet	5
2.1. Architecture monolithique	5
2.2. Architecture microservices.....	6
2.3. Architecture microfrontends.....	8
3. Etude de l'existant et problématique	10
3.1. Asana	10
3.2. Bitrix24	11
3.3. Trello	12
3.4. Jira Software.....	12
3.5. Synthèse	13
4. Solution proposée	14
5. Méthodologie de travail	14
5.1. Manifeste agile	14
5.2. SCRUM.....	15
5.3. Principe.....	15
5.3.1. Répartition des rôles	15
5.3.2. Les événements	16
5.3.3. Les artéfacts.....	17
Analyse et planification.....	18
1. Spécification des besoins	18
1.1. Identification des acteurs	18
1.2. Backlog de produit de l'application.....	18
1.3. Les besoins fonctionnels	21

1.4. Les besoins non fonctionnels	22
2. Diagramme de cas d'utilisation global	22
3. Planification des sprints	24
4. Environnement de travail	24
4.1. Environnement matériel	25
4.2. Environnement logiciel.....	25
4.2.1. Outils de développement et modélisation	25
4.2.2. Technologies utilisées.....	26
5. Architecture globale du projet	27
5.1. Architecture physique	27
5.2. Architecture logique.....	28
<i>Etude et mise en place de l'architecture de projet.....</i>	<i>30</i>
1. Planification du release 1.....	30
2. Sprint 1.1 Etude et mise en place de l'architecture microservices.....	30
2.1. Caractéristiques d'une architecture microservices.....	30
2.1.1. Division en composants via les services	31
2.1.2. Organisation autour des capacités métiers.....	31
2.1.3. Produits, pas projets.....	31
2.1.4. Extrémités Intelligentes et canaux stupides.....	31
2.1.5. Une gouvernance décentralisée	32
2.1.6. Gestion des données décentralisée.....	32
2.1.7. Automatisation de l'infrastructure.....	32
2.1.8. Tolérance aux pannes.....	32
2.1.9. Conception évolutive.....	33
2.2. La conception pilotée par le domaine	33
2.3. Architecture de Spring Cloud	34
2.3.1. API Gateway	35
2.3.2. Service de découverte	36
2.3.3. Distributed tracing.....	37
2.3.4. Protocole de communication	37
2.4. Urbanisation fonctionnelle en microservices.....	39
2.5. Synthèse	40
2. Sprint 2 Mise en place de l'architecture microfrontends	41
2.1. Approches d'intégration des microfrontends	42

2.1.1. Build time integration.....	42
2.1.2. Server-side integration	43
2.1.3. Client-side Integration	43
2.2. Etude comparative des solutions de l'approche « client-side composition »	44
2.3. Single spa.....	45
2.3.1. Aperçu architectural	45
2.3.2. Architecture proposée.....	45
Configuration du projet.....	48
1. Planification du release 2.....	48
2. Sprint 2.1 Gestion de ressources	48
2.1. Backlog de Sprint 2.1	49
2.2. Analyse des besoins de sprint 2.1	50
2.2.1. Diagramme de cas d'utilisation du sprint 2.1	50
2.2.2. Raffinement des cas d'utilisation	51
2.3. Conception du sprint 2.1	53
2.3.1. Diagramme de classe du sprint 2.1.....	53
2.3.2. Diagrammes de séquence	54
2.4. Réalisation du sprint 2.1.....	55
3. Sprint 2.2 Création du projet et construction de son équipe	57
3.1. Backlog de sprint 2.2	57
3.2. Analyse des besoins du sprint 2.2	58
3.2.1. Diagramme de cas d'utilisation du sprint 2.2	59
3.2.2. Raffinement des cas d'utilisation	59
3.3. Conception du sprint 2.2	61
3.3.1. Diagramme de classe du sprint 2.2.....	61
3.3.2. Diagrammes de séquence	61
3.4. Réalisation du sprint 2.2.....	63
Planification et suivi de projet	67
1. Planification du release 3.....	67
2. Sprint 3.1 Gestion des phases	67
2.1. Backlog de sprint 3.1	67
2.2. Analyse des besoins du sprint 3.1	69
2.2.1. Diagramme de cas d'utilisation du sprint 2.2	69
2.2.2. Raffinement des cas d'utilisation	69

2.3. Conception du sprint 3.1	71
2.3.1. Diagramme de classe du sprint 3.1.....	71
2.3.2. Diagramme de séquence	72
2.4. Réalisation du sprint 3.2.....	73
3. Sprint 3.2 Gestion des tickets et des événements	77
3.1. Backlog de sprint 3.2	77
3.2. Analyse des besoins du sprint 3.2	77
3.2.1. Diagramme de cas d'utilisation du sprint 3.2.	77
3.2.2. Raffinement des cas d'utilisation	78
3.3. Diagramme de classe de sprint 3.2	79
3.4. Réalisation du sprint 3.2.....	80
4. Sprint 3.3 : Imputation et statistiques	81
4.1. Backlog de sprint 3.3	81
4.2. Analyse des besoins de sprint 3.3	81
4.2.1. Diagramme de cas d'utilisation de sprint 3.3	81
4.2.2. Raffinement des cas d'utilisation de sprint 3.3	81
4.3. Réalisation de sprint 3.3.....	81
Conclusion Générale.....	84

Tables des figures

Figure 1 : Logo du groupe Business & Decision[1]	3
Figure 2 : Organisation de Business & Decision Tunisie	4
Figure 3 : Architecture monolithique [3]	5
Figure 4 : Exemple d'une architecture microservices [3]	7
Figure 5 : Evolution de l'architecture du frontend [5]	8
Figure 6 : End to end teams [4]	9
Figure 7 : Interface de gestion de tâches de l'outil Asana [6]	11
Figure 8 : Dashboard de l'outil Bitrix24 [6]	11
Figure 9 : Interface de gestion de tâches de Trello [6]	12
Figure 10 : Dashboard de gestion de tâches pour Jira [6]	13
Figure 11 : Diagramme de cas d'utilisation général	23
Figure 12 : Architecture 3 tiers	28
Figure 13: Architecture en microservices [22]	29
Figure 14 : Carte de contexte	34
Figure 15 : Composants de l'architecture de spring cloud [24]	35
Figure 16 : Reverse-proxy [26]	35
Figure 17 : Service discovery avec Eureka	37
Figure 18 : Communication avec openFeign	39
Figure 19: Modèle métier de la plateforme de gestion de projet	39
Figure 20 : Carte de contexte de plateforme de gestion de projet	40
Figure 21 : Architecture microservices proposée	40
Figure 22 : Diagramme de package du microservice MS-NAME	41

Figure 23: Exemple d'un package.json d'une applications divisée en microfrontends basée sur l'approche Build time integration.....	42
Figure 24 : Server side integration.....	43
Figure 25 : Client Side Integration.....	44
Figure 26 : Architecture microfrontend	46
Figure 27 : Api d'enregistrement des applications avec single-spa	46
Figure 28 : Diagramme de cas d'utilisation de sprint 2.1 pour l'acteur "Administrateur"	51
Figure 29 : Diagramme de cas d'utilisation de gestion de compte pour chaque utilisateur de la plateforme	51
Figure 30 : Diagramme de classes du sprint 2.1	54
Figure 31 : Diagramme de séquence de cas d'utilisation "Ajouter une ressource"	55
Figure 32 : Interface de login.....	56
Figure 33 : Interface d'ajout d'une ressource	56
Figure 34 : Interface de mise à jour d'un compte.....	57
Figure 35 : Diagramme de cas d'utilisation du sprint 2.2	59
Figure 36 : Raffinement de cas d'utilisation "Construire l'équipe projet"	60
Figure 37: diagramme de classe de sprint 2.2.....	61
Figure 38 : Diagramme de séquence Supprimer un projet	62
Figure 39 : Diagramme de séquence "Inviter un nouveau membre pour rejoindre l'équipe projet" ...	63
Figure 40 : Interface de création d'un projet	64
Figure 41 : Interface d'affichage de la liste des projets.....	64
Figure 42 : Interface du filtre des ressources par département du microservice Resource-MS suivi d'un filtre par profil.....	65
Figure 43 : Invitation envoyée	65
Figure 44 : Equipe projet	66
Figure 45 : Diagramme de cas d'utilisation du sprint 3.1	69

Figure 46 : Diagramme de classe de sprint 3.1	72
Figure 47 : Diagramme de séquence "Affecter une tâche"	73
Figure 48 : Interface d'ajout d'une phase	74
Figure 49 : Interface d'affichage des phases avec leurs tâches	74
Figure 50 : Interface de suppression d'une phase.....	75
Figure 51 : Interface de mise à jour d'une phase.....	75
Figure 52 : Interface d'affectation d'une tâche à un membre de l'équipe	76
Figure 53 : Task Board pour un membre de projet.....	76
Figure 54: Diagramme de classe de sprint 3.2	78
Figure 55 : Diagramme de classe de sprint 3.2	80
Figure 56 : Calendrier des événements	80
Figure 57 : Dashboard de statistiques	82
Figure 58 : Export PDF des tâches	82

Liste des tableaux

Tableau 1 : BackLog de produit.....	19
Tableau 2 : Planification des releases	24
Tableau 3 : Outils logiciels de développement	25
Tableau 4 : Planification Release 1	30
Tableau 5 : Comparaison entre Zuul et Spring Cloud Gateway	36
Tableau 6 : Comparaison des solutions de l'approche "Client Side Composition"	44
Tableau 7: Planning du release 2	48
Tableau 8 : Backlog du sprint 2.1	49
Tableau 9 : Description textuelle de cas d'utilisation "Ajouter une ressource"	52
Tableau 10 : Description textuelle du cas d'utilisation "Mettre à jour un compte"	52
Tableau 11 : Baklog du sprint 2.2.....	58
Tableau 12 : Description textuelle du cas utilisation "Créer un projet".....	59
Tableau 13 : Description textuelle du cas d'utilisation "Construire l'équipe projet"	60
Tableau 14 : Planification du release 3	67
Tableau 15 : Backlog de sprint 3.1	68
Tableau 16 : Description textuelle du cas utilisation " Mettre à jour une phase "	70
Tableau 17 : Description textuelle du cas d'utilisation "Supprimer une phase"	70
Tableau 18 : Description textuelle du cas utilisation "Modifier l'état d'une tâche"	71
Tableau 19: Backlog de produit de sprint 3.2.....	77
Tableau 20 : Description textuelle du cas d'utilisation "Créer événement"	78
Tableau 21 : Description textuelle du cas d'utilisation " Créer un ticket "	79

Introduction générale

Dans un environnement de plus en plus concurrentiel, les entreprises sont séduites par des méthodes de travail dites en “mode projet”. La gestion de projet s’impose alors dans les structures de toute taille comme un mode d’organisation particulièrement efficace. Elle peut être définie comme une méthode de travail collaborative qui aide les entreprises à coordonner et harmoniser les diverses tâches exécutées dans le cadre du projet et à améliorer leur productivité, leur réactivité et agilité afin de satisfaire les besoins des clients.

Il n'y a pas si longtemps, de nombreux projets sont suivis de manière artisanale, par mails et par tableaux Excel notamment. Toutefois, dès qu’il y a plusieurs participants et des échanges d’informations, ces outils ne conviennent pas car ils ne sont pas adaptés au travail collaboratif. Alors l’utilisation d’un outil de gestion de projet devient indispensable pour une entreprise afin de structurer et poser les différents jalons de ses projets pour s’assurer de les mener à terme.

En développement logiciel classique les applications sont généralement créées de manière monolithique. L’inconvénient majeur, c’est que plus l’application devient volumineuse, plus il devient difficile de l’enrichir de fonctions et de traiter rapidement les problèmes qui surviennent. De plus, elles sont en fait un obstacle aux déploiements fréquents c’est à dire afin de mettre à jour un composant, nous devons déployer de nouveau toute l’application. Pour pallier à ce problème des nouveaux styles architecturaux ont révolutionné le monde de développement des systèmes d’information. Ces styles visent principalement à diviser le monolithe backend ainsi que frontend en des petites applications faciles à gérer, maintenir et indépendamment déployables.

Dans le cadre de besoin d’un outil de gestion de projet combiné au besoin d’adopter des architectures qui garantissent la scalabilité, la fiabilité, la liberté de choix et la mise à niveau des technologies de développement s’inscrit notre projet de fin d’études intitulé « Conception et développement d’une plateforme de gestion de projet basée sur une architecture microservices et microfrontends ».

Le présent rapport présentera les différentes étapes de la réalisation de ce projet et s’étalera sur cinq chapitres :

Le premier chapitre « Présentation Générale » est un chapitre introductif dans lequel nous effectuons une brève description de l’entreprise. Ensuite, nous exposons le cadre général du projet, une étude et critique de l’existant et la solution proposée.

Le deuxième chapitre « Analyse et planification » fait l’objet d’une spécification des besoins fonctionnels dans un backlog de produit ainsi que les besoins non fonctionnels. Nous définissons par

la suite un planning des sprints, nous décrivons l'environnement matériel et logiciel et l'architecture générale de notre projet.

Les trois chapitres suivants seront dédiés aux trois releases de notre application où nous nous intéressons à la réalisation des sprints répartis chacun en 3 modules, analyse, conception et réalisation.

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

Présentation Générale

Introduction

Dans ce chapitre introductif, nous mettons le projet dans son cadre général. Nous présentons en premier lieu l'organisme d'accueil et le contexte du projet. Ensuite, nous exposons une étude critique de l'existant afin de fixer les objectifs. Nous clôturons ce chapitre par une présentation de la méthodologie de travail adoptée durant le stage

1. Présentation de l'organisme d'accueil

Cette partie est dédiée à la présentation de l'organisme d'accueil le long de notre projet de fin d'études.

1.1. Business & Decision Group

Business & Decision est un Groupe international de consulting et d'intégration de systèmes, leader de la Business Intelligence (BI), acteur majeur de l'e-Business, et du Customer Relationship Management (CRM). Le Groupe contribue à la réussite des projets à forte valeur ajoutée des entreprises et accompagne ses clients dans des domaines d'innovation tels que le Big Data et le Digital.

Il est reconnu pour son expertise fonctionnelle et technologique par les plus grands éditeurs de logiciels du marché avec lesquels il a noué des partenariats. Fort d'une expertise unique dans ses domaines de spécialisation, Business & Decision offre des solutions adaptées à des secteurs d'activité ainsi qu'à des directions métiers. Présent dans 11 pays, Business & Decision emploie plus de 2 400 personnes en France et dans le monde. La figure 1 présente le logo du groupe Business & Decision.



Figure 1 : Logo du groupe Business & Decision[1]

1.2. Business & Decision Tunisie

Business & Decision Tunisie est la filiale tunisienne du groupe Business Décision, créée en 2000. Elle est considérée comme un leader de la Business Intelligence (BI) et du Customer Relationship Management (CRM), acteur majeur de l'e-business, la Business performance Management (EPM) ainsi que du Management Consulting. La société contribue à la réussite des projets à forte valeur ajoutée des entreprises et à leur amélioration :

- En délivrant des conseils innovants pour accompagner la transformation des entreprises et la mise en oeuvre opérationnelle (cadrage des besoins, définition de la feuille de route, assistance aux choix d'outils, schémas d'amélioration continu)
- En mettant en oeuvre des systèmes informatiques pour le pilotage des structures et de la performance des entreprises (reporting, tableau de bord, consolidation, etc.)
- En connaissant et gérant les clients (outils pour les forces de ventes, centres d'appels, gestion de campagnes, CRM analytique)
- En gérant les relations via le web (portail collaboratif où d'entreprise, annuaires et méta-annuaires, e-commerce, knowledge management, etc.).

L'organisation de Business & Decision Tunisie est illustré par la figure 2. Nous effectuons notre stage au pôle Digital (Digital Business Unit).

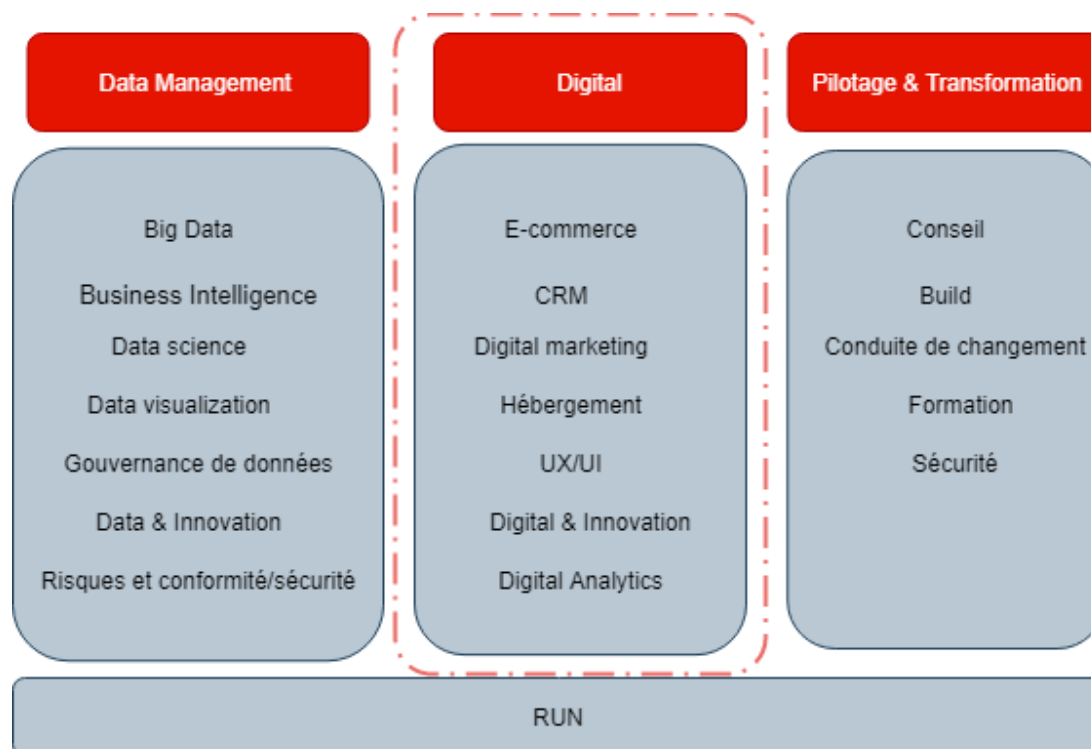


Figure 2 : Organisation de Business & Decision Tunisie

2. Contexte général du projet

Dans cette partie, nous allons définir quelques mots clés de notre sujet de fin d'études afin de faciliter l'explication tout au long de ce rapport.

2.1. Architecture monolithique

Un monolithe est conçu comme un grand système possédant une base de code unique et déployé comme une seule unité derrière un répartiteur de charge. Il dépend généralement d'une base de données unique [2]. Le monolithe se compose de quatre éléments principaux : une interface utilisateur, des logiques métiers, une interface de données et une base de données comme il est illustré dans la figure 3.

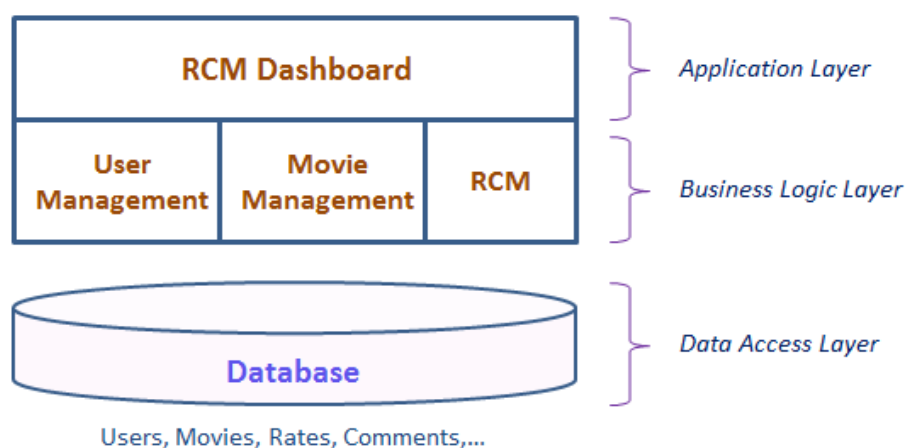


Figure 3 : Architecture monolithique [3]

Les architectures monolithiques sont simples à construire, à tester et à déployer. Les applications qui en dépendent peuvent être mises à l'échelle horizontalement, en exécutant plusieurs copies d'une application derrière un répartiteur de charge.

Par contre elles se trouvent face à plusieurs problématiques :

- **Flexibilité** : Nous ne pouvons pas utiliser différentes technologies. La pile technologique est décidée au départ et suivie tout au long. Une fois le développement arrivé à maturité, il devient parfois difficile de mettre à niveau les versions de la pile technologique, et encore moins d'adopter progressivement une nouvelle technologie.
- **Fiabilité** : Si une fonctionnalité tombe en panne, l'application entière peut tomber en panne.
- **Vitesse de développement** : le développement est vraiment lent dans une architecture monolithique. Il est difficile pour les nouveaux membres de l'équipe de comprendre et de

modifier le code d'une grande application monolithique. La qualité du code diminue avec le temps. Avec la taille croissante de la base de code, l'environnement de développement intégré (EDI) est surchargé et devient plus lent. Plus l'application est grande, plus il faut de temps pour démarrer. Tous ces facteurs ont un impact énorme sur la productivité des développeurs.

- **Scalabilité** : les applications monolithiques sont difficiles à mettre à l'échelle une fois qu'elles deviennent plus grandes. Nous pouvons créer de nouvelles instances du monolithe et demander à l'équilibreur de charge de distribuer le trafic vers les nouvelles instances, mais l'architecture monolithique ne peut pas évoluer avec une charge croissante. Chaque copie de l'instance d'application accède à toutes les données, ce qui rend la mise en cache moins efficace et augmente la consommation de mémoire et le trafic d'entrée sortie. En outre, différents composants d'application ont des besoins en ressources différents - l'un peut être gourmand en CPU tandis qu'un autre peut consommer beaucoup de mémoire. Avec une architecture monolithique, nous ne pouvons pas mettre à l'échelle chaque composant indépendamment.
- **Déploiement continu** : Les grandes applications monolithiques sont en fait un obstacle aux déploiements fréquents. Afin de mettre à jour un composant, nous devons déployer de nouveau toute l'application.

Pour neutraliser les limites de l'architecture monolithique une nouvelle architecture vient de révolutionner le style de développement appelé l'architecture microservices.

2.2. Architecture microservices

Le style architectural des microservices est une approche pour développer une seule application en tant que suite de petits services, chacun s'exécutant dans son propre processus. Chaque microservice implémente un besoin métier unique, est conteneurisé et indépendamment déployable. La figure 4 décrit un exemple d'une architecture microservice.

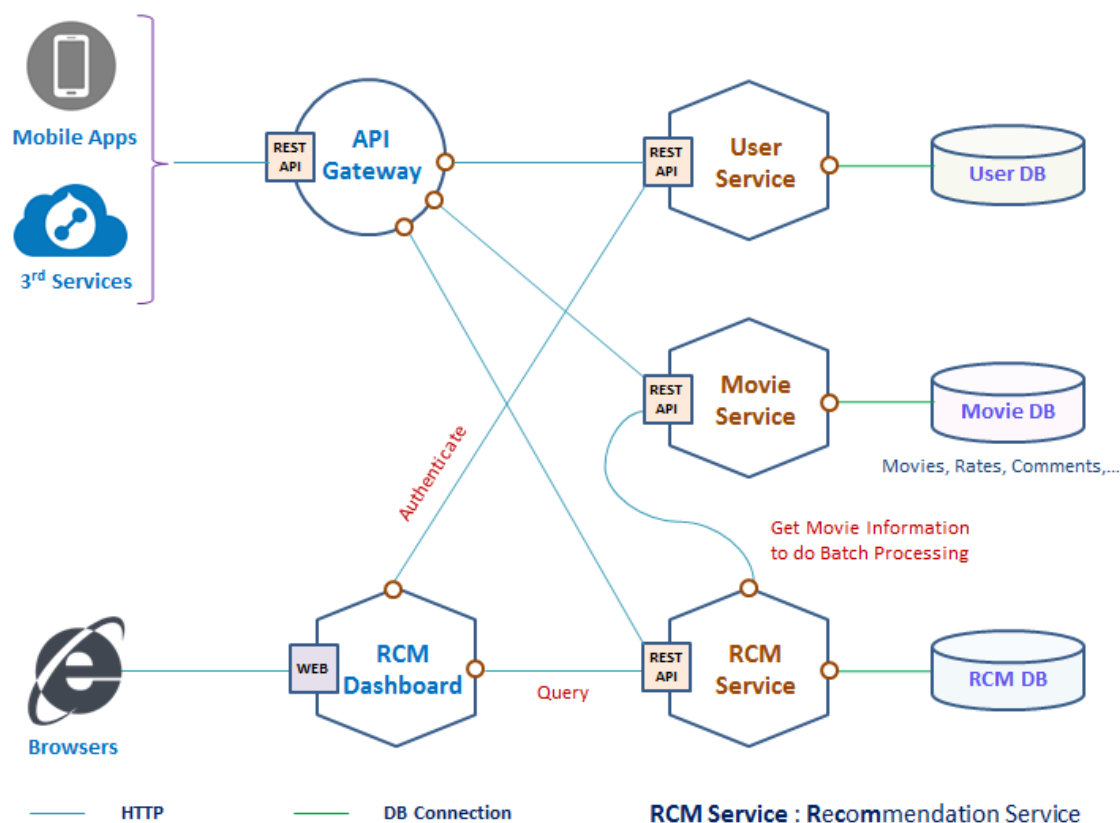


Figure 4 : Exemple d'une architecture microservices [3]

Cette approche apporte plusieurs avantages qui servent à surmonter les difficultés rencontrées dans l'approche traditionnelle :

- **Flexibilité** : Différents microservices peuvent être développés avec différentes technologies. Étant donné qu'un microservice est plus petit, la base de code est minimale, il n'est donc pas si difficile de mettre à niveau les versions de pile technologique.
- **Fiabilité** : Si une fonctionnalité tombe en panne, l'application entière reste fonctionnelle. Nous pouvons résoudre le problème dans le microservice correspondant et le déployer immédiatement.
- **Scalabilité** : Chaque microservice peut être mis à l'échelle individuellement. Étant donné que les microservices individuels sont beaucoup plus petits, la mise en cache devient très efficace.
- **Vitesse de développement** : Étant donné que le volume de code est beaucoup moins important pour un microservice, il n'est pas difficile pour les nouveaux membres de l'équipe de comprendre et de modifier le code. Ils deviennent productifs dès le départ. La qualité du code est bien maintenue. L'EDI est beaucoup plus rapide. Tous ces facteurs augmentent considérablement la productivité du développeur.

- Création des applications complexes : Si les fonctionnalités de l'application sont analysées correctement, nous pouvons la décomposer en composants indépendants qui peuvent être déployés indépendamment. Aussi, On peut même décomposer un microservice en sous microservices qui seront d'ailleurs déployés indépendamment.

2.3. Architecture microfrontends

Le terme Microfrontends est apparu pour la première fois dans ThoughtWorks Technology Radar fin 2016. Il étend les concepts de micro-services au frontend. La tendance actuelle est de créer une application qui s'exécute dans le navigateur, puissante et riche en fonctionnalités, également appelée application à page unique (Single Page Application SPA), qui sera en interaction avec une architecture de micro-service. Au fil du temps, la couche frontale, souvent développée par une équipe spécialiste, se développe et devient plus difficile à maintenir. C'est ce que nous appelons un monolithe de frontend.

L'idée derrière les MicroFrontends est de considérer un site Web ou une application Web comme une composition de fonctionnalités développées par des équipes indépendantes. Chaque équipe a un domaine d'activité ou une mission qu'elle se soucie et se spécialise. Une équipe est inter fonctionnelle et développe ses fonctionnalités de bout en bout, de la base de données à l'interface utilisateur [4].

La figure 5 décrit l'évolution de l'architecture d'une application web : du monolithe un seul bloc pour tout vers la séparation frontend backend et finalement vers les microservices.

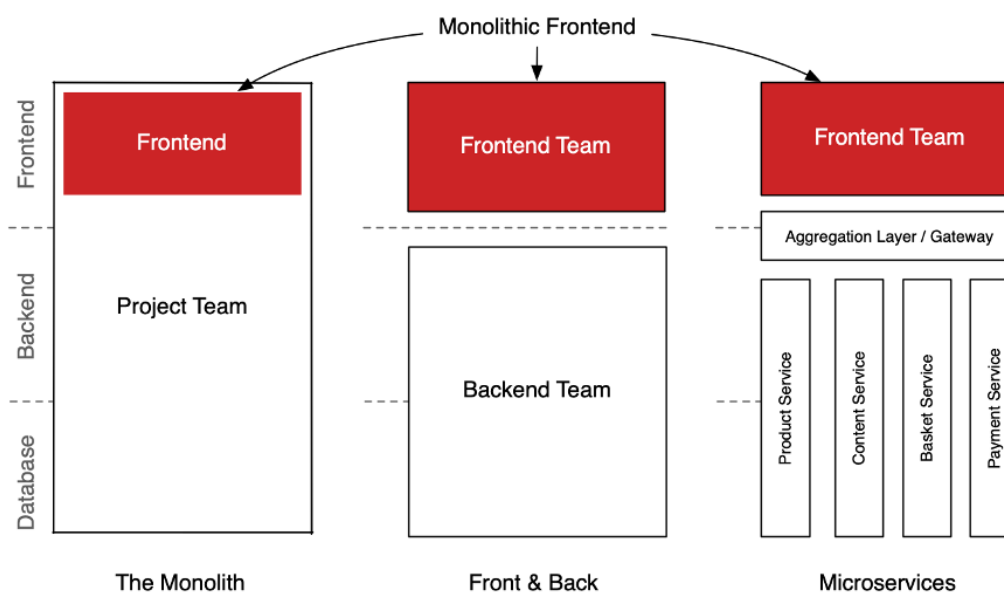


Figure 5 : Evolution de l'architecture du frontend [5]

L'idée de microfrontends est basée sur une organisation verticale pour les équipes en termes d'unité fonctionnelle appelé en anglais "end to end teams" (Figure 6).

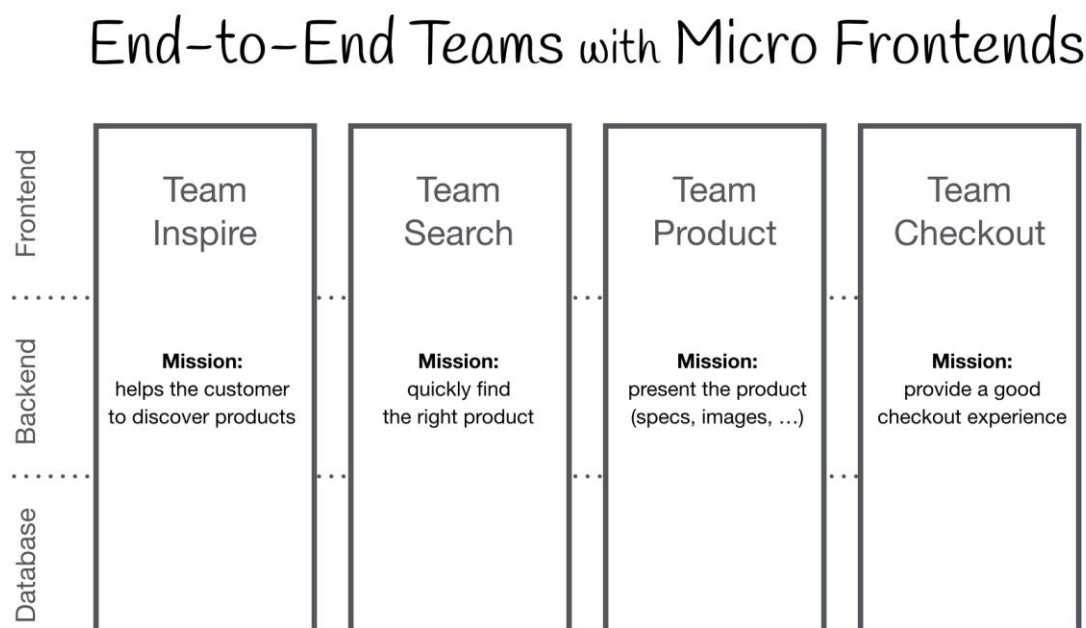


Figure 6 : End to end teams [4]

Cette nouvelle organisation a apporté plein d'avantages pour le développement frontend nous allons alors présenter son influence au niveau de l'équipe de projet ainsi que sa puissance de résoudre les problèmes de maintenance des applications :

Des équipes indépendantes : En travaillant dans plusieurs équipes la communication sera obligatoire régulièrement et le déploiement sera impossible si nous ne sommes pas sur la même ligne d'avancement. Avec les microfrontends, l'évolutivité en termes de ressources de développement devient beaucoup plus facile à gérer. Généralement, chaque fonctionnalité peut être développée par une équipe indépendante. Chaque équipe peut publier de manière autonome ses fonctionnalités sans aucun alignement requis.

Ne faire qu'une chose et le faire bien : Avoir une capacité ou une fonctionnalité métier unique pour une seule équipe responsable est certainement un avantage. L'équipe va regrouper toutes les compétences pour développer le besoin et aucune coordination avec les autres équipes frontend et backend n'est requise. Elle permettra de concentrer sur la qualité du microfrontend et du microservice concernés en une durée minimale. Et finalement, chaque microfrontend sera déployable indépendamment.

Liberté de choix de stack technologique : On peut utiliser plusieurs technologies ou frameworks pour plusieurs microfrontends, ce qui va faciliter les mises à niveau du frontend : chaque équipe possède sa pile complète du frontend à la base de données. Les équipes peuvent décider de mettre à jour ou de changer leur technologie frontend indépendamment. Ainsi nous ne partageons pas un runtime, même si toutes les équipes utilisent le même framework elles créent des applications indépendantes conteneurisées.

3. Etude de l'existant et problématique

Un logiciel de gestion de projet se définit comme un outil permettant de planifier et de conduire de manière efficiente un ou plusieurs projets, tout en mettant l'accent sur la collaboration, la communication et la simplification des processus.

Pour ce faire, les logiciels de gestion de projet offrent de nombreuses fonctionnalités, parmi lesquelles nous pouvons citer :

- La planification et le suivi des temps,
- La gestion et le suivi des tâches
- L'attribution des ressources
- Le suivi budgétaire
- Le partage de documents
- Le diagramme de Gantt, etc.

Plusieurs outils de gestion de projet disponibles en des versions gratuites et payantes. Nous allons présenter et étudier quelques exemples en présentant leurs fonctionnalités principales et leurs limites.

3.1. Asana

Asana est un excellent outil de gestion de tâches. Il permet avant tout de répartir les tâches à faire entre tous les acteurs de l'entreprise, de collaborer, d'échanger dans le but de travailler plus efficacement. En somme, chacun sait ce qu'il a à faire en arrivant au bureau ou en sortie de réunion : fini les to-do List sur le coin du bureau ou les réunions qui ne débouchent pas sur des actions concrètes.

Asana pose en revanche quelques problèmes d'usage : suppression trop facile de tâches (que vous ne pouvez pas récupérer), impossibilité de personnaliser les champs des tâches, etc.

Asana est donc excellent pour la gestion classique de petits projets collaboratifs, mais ne permet pas de gérer des projets complexes [6] (Figure 7).

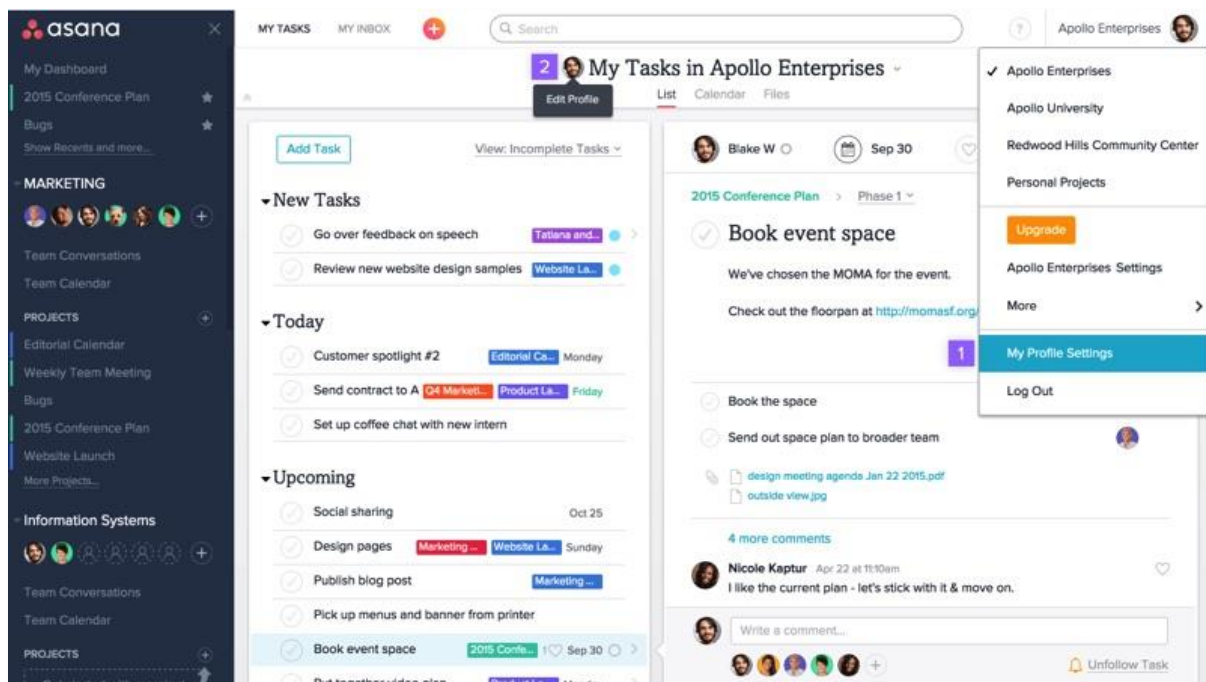


Figure 7 : Interface de gestion de tâches de l'outil Asana [6]

3.2. Bitrix24

Un outil de gestion de projet ou en d'autres termes un espace de travail collaboratif. Ce logiciel permet seulement de planifier des tâches ou de discuter avec les membres d'une équipe (Figure 8).

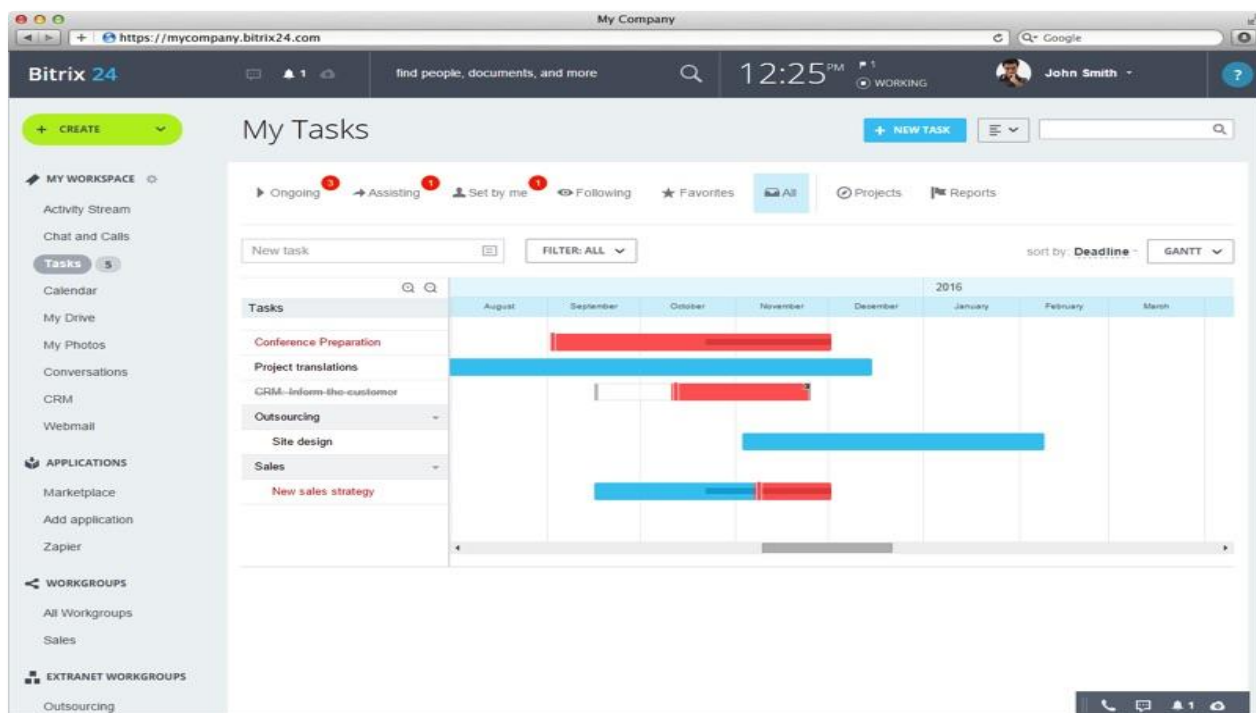


Figure 8 : Dashboard de l'outil Bitrix24 [6]

3.3. Trello

Trello a compris que la gestion de projet passait avant tout par la collaboration et la visualisation de l'avancement des tâches en cours. Comme le montre la capture d'écran ci-dessus, la solution permet de créer des projets qui prennent la forme de tableaux en plusieurs colonnes (personnalisables). Dans chaque colonne se trouvent les tâches (ou les idées), visualisées sous forme de cartes, que les collaborateurs font progresser de gauche à droite au fur et à mesure de leur avancement.

Trello est entièrement gratuit dans sa première version et dispose d'une application mobile très bien conçue. En revanche, cette version gratuite pose un problème de confidentialité et de sécurité des données. L'éditeur avoue lui-même proposer "Plus de sécurité" dans l'offre payante.

Enfin Trello s'apparente plus à un gestionnaire de tâches qu'à un outil de gestion de projet à proprement parlé. En effet, il fait l'impasse sur la gestion des coûts, des temps, des budgets et les fonctions d'administration (Figure 9).

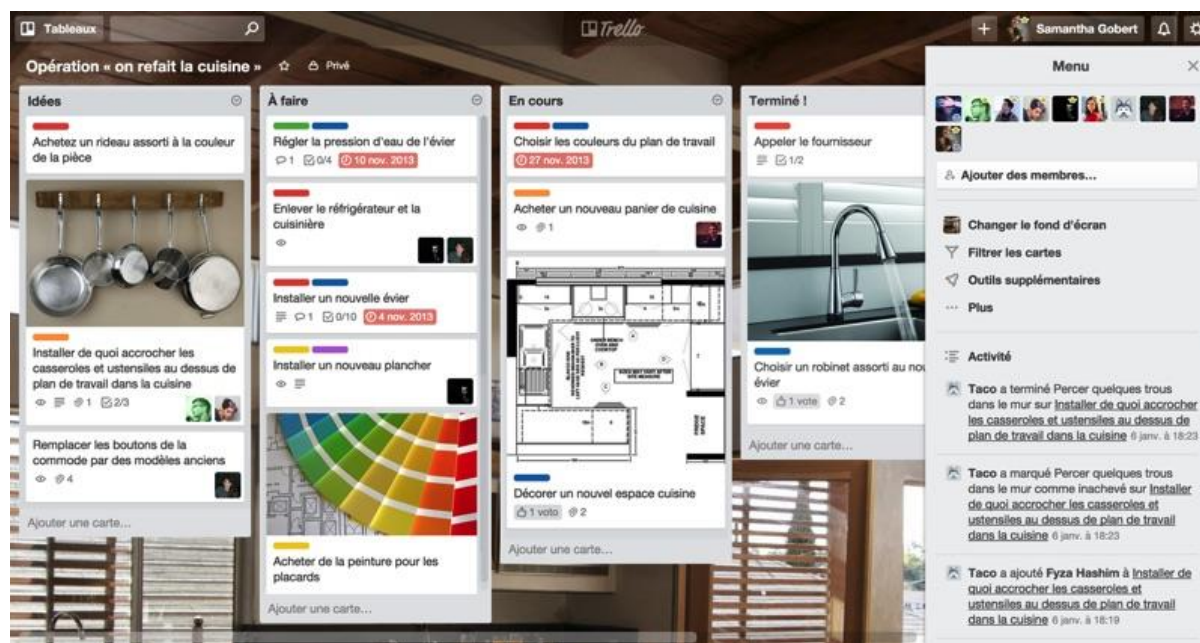


Figure 9 : Interface de gestion de tâches de Trello [6]

3.4. Jira Software

Un outil de gestion de projet qu'offre à ses utilisateurs une bonne gamme de fonctionnalités pour gérer leurs projets ainsi que suivre leurs avancements.

Mais l'utilisation de cet outil est mieux adaptée aux grands projets avec des grandes équipes. En outre, le Dashboard proposé n'est pas aussi détaillé qu'aucun le souhaite (Figure 10).

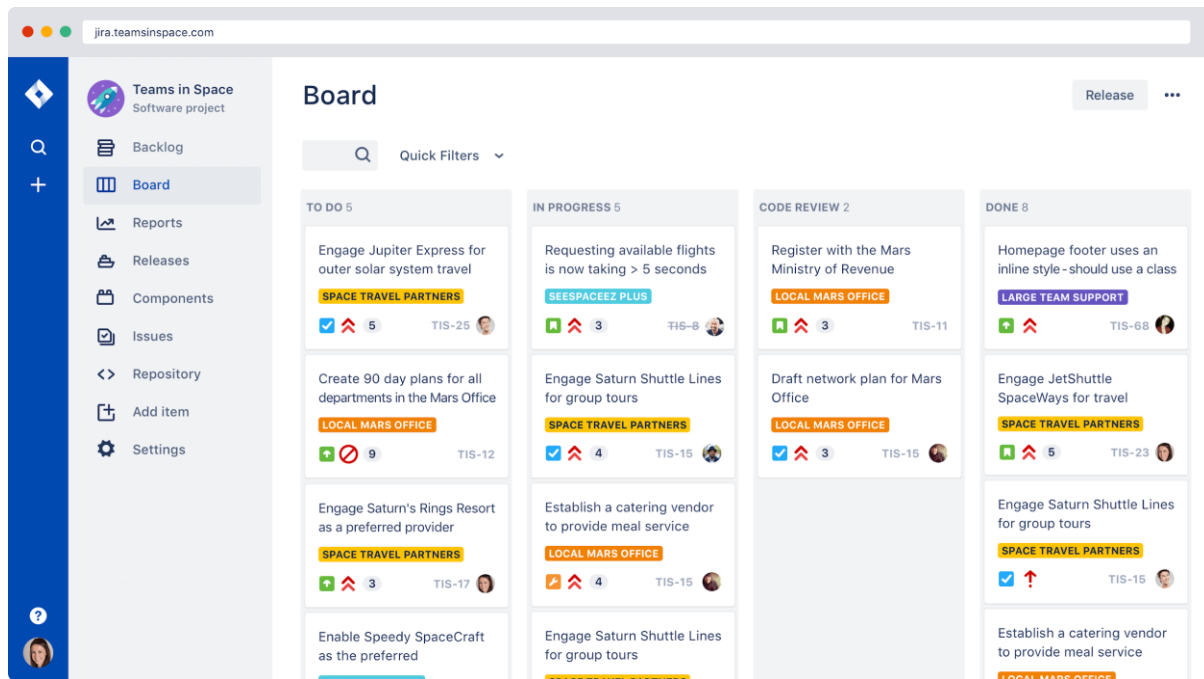


Figure 10 : Dashboard de gestion de tâches pour Jira [6]

3.5. Synthèse

Après avoir décrit des échantillons d'outils de gestion de projet les plus connus sur le marché. Vu leur richesse nous a vu que la contrepartie de la gratuité est souvent cachée, la sécurité et la confidentialité des données ne sont pas toujours garanties, le nombre d'utilisateurs est souvent limité, les fonctionnalités sont généralement limitées et ne satisfaisant pas les besoins spécifiques pour chaque entreprise comme il est le cas de Business & Decision Group.

En plus, en étudiant les problèmes de l'architecture monolithique quel que soit en frontend ou en backend comme il est détaillé dans la partie précédemment nous pouvons dire :

Les applications monolithiques ont généralement un code source assez volumineux, ce qui est souvent intimidant voir même rebutant pour certains développeurs. Lorsque des nouveaux développeurs rejoignent l'équipe de projet, il leur faut alors passer beaucoup de temps pour se familiariser avec le code source de l'application et comprendre les dépendances entre les modules. Avec une architecture monolithique, il est difficile d'apporter des améliorations à l'application sans prendre le risque d'affecter les autres fonctionnalités. Toute modification de l'application entraîne plusieurs révisions et approbations qui augmentent le temps du cycle de déploiement. D'où l'intérêt de notre projet de fin d'étude.

4. Solution proposée

Après avoir étudié les différents outils de gestion de projet dispersés sur le marché, nous avons constaté que ce n'est pas conformes à nos besoins, pour garantir la sécurité de nos données ainsi pour permettre un accès non limité par le nombre d'utilisateurs, Notre travail consiste alors à concevoir, mettre en place une architecture microservices et microfrontends et développer un outil de gestion de projet offrant les fonctionnalités suivantes :

- L'authentification
- La gestion des ressources
- La gestion des rôles et droits d'accès à notre plateforme
- La gestion des compétences et des profils
- La création des projets
- La gestion des différentes phases du projet
- La gestion des tâches et leur affectation
- L'invitation et le mailing
- Le Reporting et les statistiques

5. Méthodologie de travail

Dans la présente section nous allons décrire le manifeste agile et la méthodologie de travail adopté au cours de notre stage.

5.1. Manifeste agile

La méthode Agile se base sur un cycle de développement qui porte le client au centre. Le client est impliqué dans la réalisation du début à la fin du projet. Grâce à la méthode agile le demandeur obtient une meilleure visibilité de la gestion des travaux qu'avec une méthode classique.

L'implication du client dans le processus permet à l'équipe d'obtenir un feedback régulier afin d'appliquer directement les changements nécessaires.

Cette méthode vise à accélérer le développement d'un logiciel. De plus, elle assure la réalisation d'un logiciel fonctionnel tout au long de la durée de sa création.[7]

La méthode agile nommée Manifeste Agile repose sur quatre grands principes :

- **Collaboration** : Communication et cohésion d'équipe passent avant les outils et les processus.

- **Equipe** : Le privilège de la relation équipe/client est mis en avant plutôt que la négociation contractuelle.
- **Application** : Préférer une application bien construite à une documentation détaillée.
- **Acceptation** : Le choix de l'acceptation du changement et de la flexibilité au détriment d'un plan rigide.

5.2. SCRUM

Scrum est la méthodologie la plus utilisée parmi les méthodes agiles existantes. Le terme Scrum (qui signifie mêlée) apparaît pour la première fois en 1986 dans une publication de Hirotaka Takeuchi et Ikujiro Nonaka qui décrit une nouvelle approche plus rapide et flexible pour le développement de nouveaux produits. Le principe de base étant que l'équipe avance ensemble et soit toujours prête à réorienter le projet au fur-et-à-mesure de sa progression, tel un ballon de rugby qui doit passer de main en main jusqu'à marquer un essai [8].

5.3. Principe

L'approche SCRUM suit les principes de la méthodologie Agile, c'est-à-dire l'implication et la participation active du client tout au long du projet.

Considéré comme un framework de gestion de projet, Scrum se compose de plusieurs éléments fondamentaux : rôles, événements, artefacts.

5.3.1. Répartition des rôles

L'équipe Scrum se compose de :

Le Scrum Master est responsable de la compréhension, de l'adhésion et de la mise en œuvre de la méthode Scrum qu'il maîtrise parfaitement. Il veille à ce que les principes et les valeurs de la méthodologie sont respectés. C'est un facilitateur qui aide à améliorer la communication au sein de l'équipe et cherche à maximiser la productivité et le savoir-faire de celle-ci. Il est considéré comme le coach de l'équipe de développement.

Le Product Owner porte la vision du produit à réaliser. Il travaille en interaction avec l'équipe de développement qui doit suivre ses instructions. C'est lui qui établit la priorité des fonctionnalités à développer ou à corriger, et qui valide les fonctionnalités terminées. Il est responsable de la gestion du product backlog.

L'équipe de développement est chargée de transformer les besoins définis par le Product Owner en fonctionnalités utilisables. Elle est pluridisciplinaire et possède toutes les compétences nécessaires pour réaliser le projet, sans faire appel à des prestations externes. Parmi ses membres, on trouve un architecte, un développeur, un testeur, etc. La taille idéale de l'équipe de développement est de 3 à 9 personnes. Il n'y a pas de notion de hiérarchie, toutes les décisions sont prises ensemble [8].

5.3.2. Les événements

La vie d'un projet Scrum est rythmée par un ensemble de réunions définies avec précision et limitées dans le temps.

Le Sprint : Un Sprint est une itération. Il s'agit d'une période de 2 à 4 semaines maximum pendant laquelle une version terminée et utilisable du produit est réalisée. Un nouveau sprint commence dès la fin du précédent. Chaque sprint a un objectif et une liste de fonctionnalités à réaliser.

Planification d'un Sprint : Les tâches à accomplir pendant le Sprint sont déterminées par l'ensemble de l'équipe Scrum lors de la réunion de planification de Sprint. La durée de cette réunion est limitée à 8 heures pour les Sprints d'un mois. Cette réunion permet à l'équipe d'établir les éléments qu'elle traitera au cours de ce Sprint et comment elle procédera.

Revue du Sprint : Il s'agit du bilan du Sprint réalisé. Une fois le Sprint terminé, l'équipe Scrum et les parties prenantes se réunissent pour valider ce qui a été accompli pendant le Sprint. Cette réunion dure 4 heures maximum.

Rétrospective du Sprint : Cette réunion est interne à l'équipe Scrum et dure 3 heures pour un Sprint d'un mois. Le but est l'adaptation aux changements qui peuvent survenir et l'amélioration continue du processus de réalisation. L'équipe passe en revue le Sprint terminé afin de déterminer ce qui a bien fonctionné et ce qu'il faut améliorer.

Mêlée quotidienne : Cette réunion quotidienne de 15 minutes est très importante. Elle se fait debout (d'où son nom anglais de “stand-up meeting”) afin d'éviter de s'éterniser. Le but est de faire un point sur la progression journalière du Sprint. Elle permet à l'équipe de synchroniser ses activités et de faire un plan pour les prochaines 24 heures [8].

La mêlée a lieu à la même heure et au même endroit chaque jour. Chaque membre de l'équipe de développement doit répondre à ces trois questions :

- Qu'est-ce qu'ils ont réalisé la veille ?
- Qu'est-ce qu'ils vont accomplir aujourd'hui ?

- Quels sont les obstacles qui les retardent ?

5.3.3. Les artéfacts

Le product backlog : Il s'agit d'une liste hiérarchisée des exigences initiales du client concernant le produit à réaliser. Ce document évolue sans cesse durant le projet, en fonction des besoins du client. Le product owner est responsable du product backlog.

Le Sprint backlog : C'est le plan détaillé de la réalisation de l'objectif du Sprint, défini lors de la réunion de planification du Sprint. Le Sprint backlog est mis à jour régulièrement par l'équipe afin d'avoir une vision précise de la progression du Sprint.

L'incrément : Il s'agit de l'ensemble des éléments terminés du product backlog pour le Sprint en cours, ainsi que ceux des Sprints précédents. L'incrément doit fonctionner et être utilisable.

Le Burndown Chart (ou graphique d'avancement) : Ce graphique simple indique l'état d'avancement dans la réalisation des tâches du Sprint backlog. Il s'agit du tracé de la charge de travail restante (exprimée généralement en heures) en fonction du temps (en jours). Le Burndown Chart est actualisé tous les jours par le Scrum Master après la mêlée quotidienne [8].

Conclusion

Tout au long de ce chapitre, nous avons présenté l'organisme d'accueil Business & Decision. Par ailleurs, nous avons pu dégager le contexte général du projet, exposer le choix de la méthodologie du travail.

À ce stade, nous pouvons désormais passer au prochain chapitre, qui va porter sur la planification du projet et l'identification des fonctionnalités à satisfaire.

Analyse et planification

Introduction

Dans ce chapitre, nous présentons les principaux axes se rapportant à la planification de notre projet. Tout d'abord, nous élaborons une étude contextuelle où nous dégagons les acteurs de notre application, le backlog produit, les besoins fonctionnels et les besoins non fonctionnels. Par la suite, nous établissons une analyse globale en présentant des modèles descriptifs de notre application.

1. Spécification des besoins

Commençons par l'identification des acteurs de notre plateforme.

1.1. Identification des acteurs

Un acteur est une entité externe qui interagit avec le système (opérateur, centre distant, autre système...). En réponse à l'action d'un acteur, le système fournit un service qui correspond à son besoin. Nous identifions trois acteurs pour notre système :

Administrateur de la plateforme : Celui qui gère les utilisateurs, les affecte à leur équipe de travail et gère leur droit d'accès à la plateforme.

Responsable du projet : qui va créer le projet et gérer toutes les activités reliées à ce projet. Son rôle est l'administration du projet

Membre de l'équipe : qui sera invité à rejoindre un projet avec un rôle prédéfini, possède son propre espace où il va modifier l'état des tâches qui lui sont affectées participer aux événements planifiés au niveau de l'espace collaboratif.

1.2. Backlog de produit de l'application

Le backlog de produit est la liste des fonctionnalités attendues d'un produit. Il contient tous les éléments qui vont nécessiter du travail pour l'équipe appelés «user stories ».

Après des réunions successives avec le Product Owner nous avons pu construire notre product backlog qui est résumé dans le tableau ci-dessous.

Chapitre 2

Chaque cas d'utilisation ou user story possède un effort qui est l'estimation initiale sur la quantité de travail nécessaire pour implémenter cette exigence. En effet, un point correspond à une journée de travail. Le tableau 1 détaille notre backlog de produit.

Tableau 1 : BackLog de produit

User Story	Description	Module	Priorité	Effort
Gestion des départements	En tant qu'administrateur, je peux définir des équipes travaillant sur différents secteurs.	Département	Moyenne	3
Gestion de ressources	En tant qu'administrateur, je peux gérer les ressources de la plateforme.	Ressource	Elevée	5
Affectation ressource-rôle, ressource-équipe	En tant qu'administrateur, je peux attribuer des rôles à chaque ressource et l'affecter à une équipe ou bien un département.	Ressource	Elevée	3
Invitation membre de la plateforme	En tant qu'administrateur, je peux envoyer un mail contenant l'identifiant et le mot de passe à chaque ressource ajoutée.	Ressource	Elevée	6
Gestion de profil	En tant qu'utilisateur de la plateforme je peux gérer mon profil.	Ressource	Moyenne	8
Authentification	En tant qu'utilisateur, je veux m'authentifier pour accéder à la plateforme.	Ressource	Elevée	10
Gestion des Compétences	En tant qu'administrateur, je peux ajouter, supprimer des compétences et affecter une ou plusieurs à chaque membre inscrit dans la plateforme	Compétences	Moyenne	4
Gestion de projet	En tant que chef de projet, je peux créer, mettre à jour, supprimer ou filtrer un projet.	Projet	Elevée	5
Construction équipe projet	En tant que chef de projet, je peux construire mon équipe projet.	Projet	Elevée	10
Consultation équipe	En tant que chef de projet, je peux consulter mon équipe projet ou supprimer un membre.	Projet	Moyenne	4
Planifier des phases	En tant que chef de projet, je peux ajouter des phases à mon projet	Phase	Elevée	6
Manipulation des phases	En tant que chef de projet, je peux mettre à jour les dates de début et fin de chaque phase ou supprimer une phase.	Phase	Elevée	3

Chapitre 2

Ajout des tâches	En tant que chef de projet, je peux ajouter les tâches à chaque phase déjà créée	Tâche	Elevée	5
Manipulation des tâches	En tant que chef de projet, je peux mettre à jour/ supprimer une tâche.	Tâche	Elevée	4
Affectation des tâches	En tant que chef de projet, je peux affecter une tâche à un membre de l'équipe projet.	Tâche	Elevée	4
Tâche-owner	En tant que membre de l'équipe projet, je peux consulter mon « task board » ou je peux mettre à jour l'état des tâches qui me sont affectées.	Tâche	Elevée	8
Gestion des événements	En tant que chef de projet, je peux ajouter et mettre à jour un événement (réunion/livraison ...)	Evénement	Moyenne	9
Calendrier	En tant que chef de projet ou membre de l'équipe, je peux consulter mon calendrier pour voir les différents événements du projet que je me suis inscrit.	Evénement	Moyenne	6
Manipulation des tickets	En tant que membre du groupe je peux créer un ticket (exemple bug ou incident identifié au cours de la phase) et inviter le chef de projet à intervenir.	Ticket	Faible	4
Export événement	En tant que chef de projet ou membre de l'équipe, je peux exporter en format PDF tous les événements d'un tel projet	Reporting	Faible	2
Export tâches	En tant que chef de projet ou membre de l'équipe, je peux exporter en format PDF les tâches incluses dans un projet	Reporting	Moyenne	2
Pourcentage Avancement dans le projet	En tant que chef de projet, je peux consulter l'état d'avancement du projet	Statistiques	Elevée	3
Etat des tâches	En tant que chef de projet je peux visualiser l'état des différentes tâches dans mon projet ainsi que leur priorité	Statistiques	Moyenne	4

À partir de ces “User Stories”, nous identifions donc les besoins fonctionnels et non fonctionnels de notre application.

1.3. Les besoins fonctionnels

L'établissement des besoins fonctionnels est une étape primordiale pour définir les bases des fonctionnalités attendues du système.

Authentification : l'utilisateur s'authentifie auprès du système pour être reconnu et pour accéder aux fonctionnalités permises en saisissant le mot de passe et le nom utilisateur.

Gestion des ressources (utilisateurs) : L'administrateur de la plateforme est le seul qui a le droit d'ajouter des nouvelles ressources y compris la modification ou la suppression d'une ressource.

Gestion des rôles : L'administrateur de la plateforme peut créer des rôles et affecter un rôle à chaque ressource.

Gestion des départements : L'administrateur de la plateforme peut créer des équipes qui représentent généralement les différents départements existants dans l'entreprise et affecter les ressources au département métier.

Gestion de profil : Chaque ressource ajoutée par l'administrateur, après son authentification, elle peut changer sa photo de profil, son numéro de téléphone ou bien son mot de passe.

Gestion de projet : Le chef de projet peut créer un nouveau projet, mettre à jour ou bien le supprimer.

Construction de l'équipe projet : Le chef de projet peut filtrer les ressources existant par département ou par compétence, vérifier leur disponibilité et l'inviter à rejoindre son équipe projet.

Gestion des phases : Le chef de projet peut ajouter au projet un ou plusieurs phases ainsi que la possibilité de mettre à jour ou bien supprimer une phase.

Gestion des tâches : Le chef de projet ajoute une ou plusieurs tâches à chaque phase du projet, modifier son état ou priorité et finalement l'affecter à un membre de son groupe.

Le membre de l'équipe, auquel la tâche est affectée, a le droit de visualiser l'ensemble de ses tâches par projet ainsi que la possibilité de modifier son état.

Gestion des événements : Le chef de projet peut gérer les événements inclus dans un projet ainsi que la possibilité de les exporter en format pdf. Chaque membre du projet peut visualiser un calendrier contenant les différents événements d'un projet.

Consultation de statistiques : Le chef de projet peut consulter le pourcentage de réalisation des tâches, leur état et priorité ainsi que l'avancement dans le projet.

1.4. Les besoins non fonctionnels

Outre les besoins fonctionnels précédemment établis, l'application devra respecter une liste de besoins non fonctionnels qui représentent les exigences implicites auxquels notre projet doit répondre. Parmi ces besoins nous citons :

La sécurité : L'application doit respecter la confidentialité des données et chaque utilisateur doit posséder son propre compte.

La scalabilité : L'évolutivité est l'aspect essentiel des microservices. Chaque service étant un composant distinct, vous pouvez mettre à l'échelle une fonction ou un service sans avoir à mettre à l'échelle l'ensemble de l'application. Les services critiques de l'entreprise peuvent être déployés sur plusieurs serveurs pour une disponibilité et des performances accrues sans affecter les performances des autres services.

Ergonomie du produit : L'application réalisée doit fournir des interfaces faciles à utiliser.

La tolérance aux pannes (résistance) : L'application doit être fonctionnelle même qu'il existe des micro-services en panne. Avec micro-services, l'ensemble de votre application est décentralisée et découplée en services qui agissent en tant qu'entités distinctes. L'impact d'une défaillance à l'aide de micro-services est minimal. Même lorsque plusieurs systèmes sont mis hors service pour maintenance, vos utilisateurs ne le remarquent pas.

2. Diagramme de cas d'utilisation global

La plateforme de gestion de projet offre plusieurs fonctionnalités aux différents acteurs, et la figure 11 présente une vue globale du système via un diagramme de cas d'utilisation global.

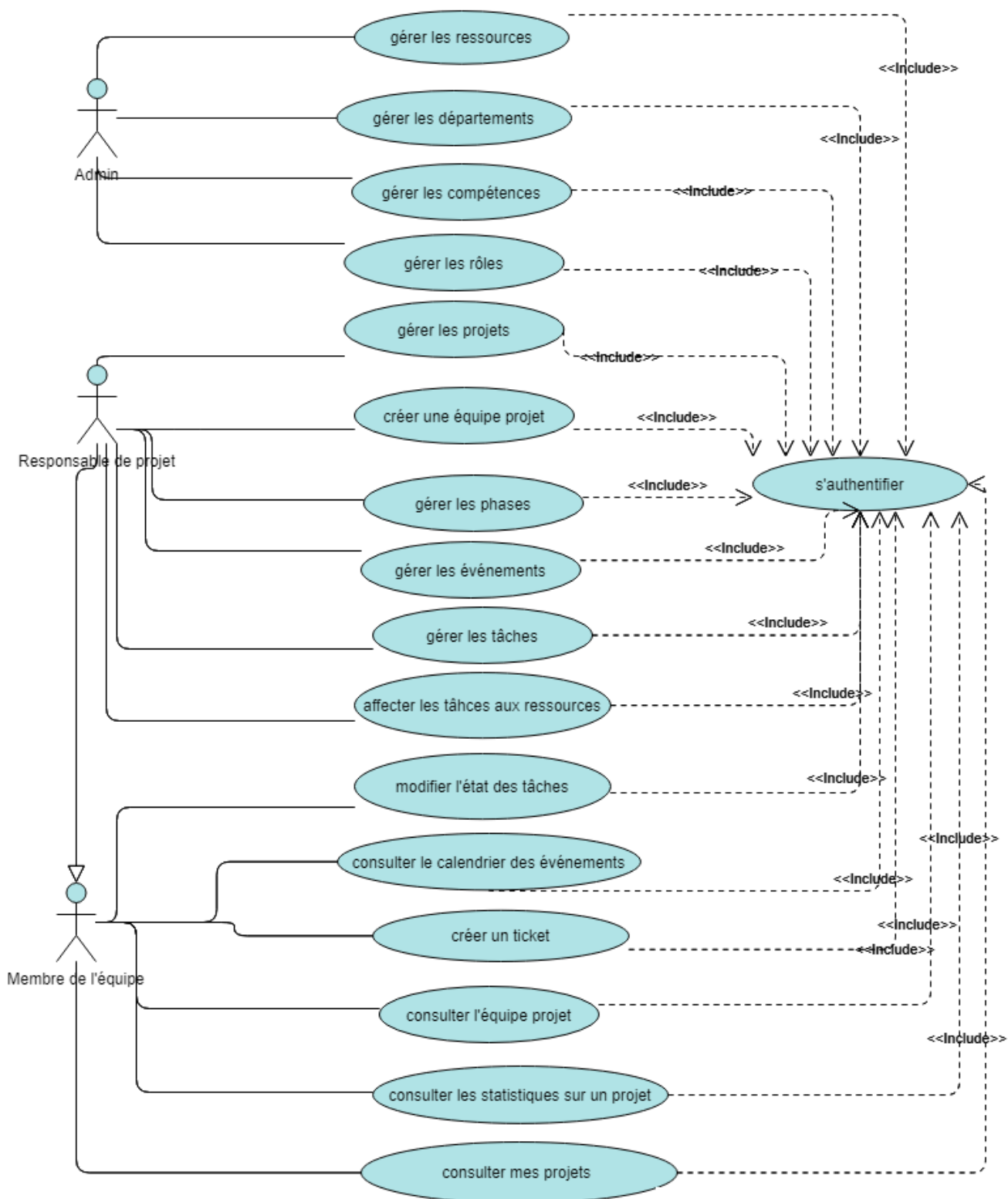


Figure 11 : Diagramme de cas d'utilisation général

3. Planification des sprints

Planifier un projet, c'est avoir une visibilité générale sur le déroulement du projet tout au long de son cycle de vie. Nous avons décidé alors de diviser le travail en 3 releases regroupant à leurs tours un ensemble de sprints (Tableau 2).

Tableau 2 : Planification des releases

Release 1 : Etude et mise en place de l'architecture de projet De 15/02/20 à 20/04/2020	Sprint 1 : Etude et mise en place de l'architecture microservices
	Sprint 2 : Etude et mise en place de l'architecture microfrontends
Release 2 : Configuration du projet De 23/04/2020 à 12/06/2020	Sprint 3 : Gestion des ressources : Comptes, Départements, Ressources, Droits d'accès
	Sprint 4 : Création et manipulation de projet et de son équipe
Release 3 : Planification et suivi de projet De 13/06/ 2020 à 31/08/2020	Sprint 5 : Gestion des phases : Phases, Tâche, Affectation des tâches
	Sprint 6 : Gestion des événements
	Sprint 7 : Suivi de projet : Statistiques et Imputation

4. Environnement 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.

4.1. Environnement matériel

Pour réaliser notre travail, nous avons utilisé comme environnement matériel, un ordinateur ayant les caractéristiques suivantes :

- Ordinateur portable : ASUS
- Système d'exploitation : Windows 10 PRO 64 bits.
- Processeur : Intel R core i5
- RAM : 8Go
- Disque Dur : 1To HDD




4.2. Environnement logiciel




Nous allons présenter dans cette section les outils et les technologies utilisés au cours de notre stage.

4.2.1. Outils de développement et modélisation

Le tableau 3 décrit les outils de développement et modélisation utilisées.

Tableau 3 : Outils logiciels de développement

	IntelliJ Idea 2019.3.3 est un IDE java commercial développé par JetBrains. Il est fréquemment appelé par le simple nom « IntelliJ » ou « IDEA ». Une licence « Ultimate de ce produit nous a été fourni par ENSIT. Chaque aspect d'intelliJ a été conçu pour maximiser la productivité des développeurs. Ensemble d'aide au codage intelligente et la conception ergonomique rendent le développement non seulement productif mais aussi agréable [9].
	Visual Code 1.49.1 est un éditeur de code source léger mais puissant qui s'exécute sur votre bureau et est disponible pour Windows, macOS et Linux. Il supporte la plupart des langages connus comme JavaScript, TypeScript et Node.js et dispose d'un riche écosystème d'extensions pour d'autres langages (tels que C ++, C #, Java, Python, PHP, Go) et des environnements d'exécution (tels que .NET et Unity) [10].
 POSTMAN	Postman est un environnement de développement d'API qui aide les utilisateurs à créer, tester, documenter, surveiller et publier la documentation de leurs API [11].

	<p>Gitlab, c'est une plateforme permettant d'héberger et de gérer des projets web de A à Z. Présentée comme la plateforme des développeurs modernes, elle offre la possibilité de gérer ses dépôts Git et ainsi de mieux appréhender la gestion des versions de vos codes sources [12].</p>
	<p>MongoDB Compass est l'interface graphique de MongoDB. Compass vous permet d'analyser et de comprendre le contenu de vos données sans connaissance formelle de la syntaxe de requête MongoDB[13].</p>
	<p>Draw.io est une application gratuite en ligne, accessible via son navigateur (protocole https) qui permet de dessiner des diagrammes ou des organigrammes. Cet outil vous propose de concevoir toutes sortes de diagrammes, de dessins vectoriels, de les enregistrer au format XML puis de les exporter [14].</p>

4.2.2. Technologies utilisées

Spring Cloud : Spring Cloud fournit des outils aux développeurs pour créer rapidement certains des modèles courants dans les systèmes distribués (par exemple, la gestion de la configuration, la découverte de services, les disjoncteurs, le routage intelligent, le micro-proxy, le bus de contrôle, les jetons à usage unique, les verrous globaux, l'élection du leadership, la distribution sessions, état du cluster). Les développeurs fonctionneront bien dans n'importe quel environnement distribué, y compris leur propre ordinateur portable, les data center bare metal et les plates-formes telles que Cloud Foundry [15].

Spring Boot : C'est un framework Java open source utilisé pour créer un micro service. Il est développé par Pivotal Team et est utilisé pour créer des applications standalone et prêtes pour la production [16].

JSON Web Token : C'est un standard ouvert qui permet l'échange sécurisé de jetons (tokens) entre plusieurs parties. Cette sécurité de l'échange se traduit par la vérification de l'intégrité des données à l'aide d'une signature numérique [17].

MapStruct : C'est un générateur de code qui simplifie considérablement l'implémentation des mappages entre les types de bean Java basés sur une approche de configuration par convention.

Le code de mappage généré utilise des invocations de méthode simples et est donc rapide, sûr de type et facile à comprendre [18].

Single-spa : C'est un framework pour rassembler plusieurs microfrontends JavaScript dans une application frontend. Architecturer votre frontend à l'aide de single spa offre de nombreux avantages, tels que

- Utilisez plusieurs frameworks sur la même page sans rafraîchir la page (React, AngularJS, Angular, Ember etc)
- Déployez vos microfrontends indépendamment.
- Écrivez du code à l'aide d'un nouveau framework, sans réécrire votre application existante.
- Lazy loading pour améliorer le temps initial de chargement [19].

Angular : C'est un Framework JavaScript côté client développé par Google. Il permet de rendre dynamique l'affichage d'une page web, mais également de faire des appels asynchrones à des services web. Angular ajoute des balises ou des attributs au HTML, et ensuite les interprète [20].

Spring Data MongoDB : Le projet Spring Data MongoDB fournit une intégration avec la base de données de documents MongoDB. Les domaines fonctionnels clés de Spring Data MongoDB sont un modèle centré POJO pour interagir avec un MongoDB DBCollection et écrire facilement une couche d'accès aux données de style Repository [21].

5. Architecture globale du projet

Cette partie est dédiée à la présentation de l'architecture physique et logique de notre projet.

5.1. Architecture physique

Notre solution proposée utilise l'architecture opérationnelle trois tiers. En effet, le travail nécessite la réalisation de trois groupes de fonctions : le stockage des données, la logique applicative et la présentation (Figure 12).

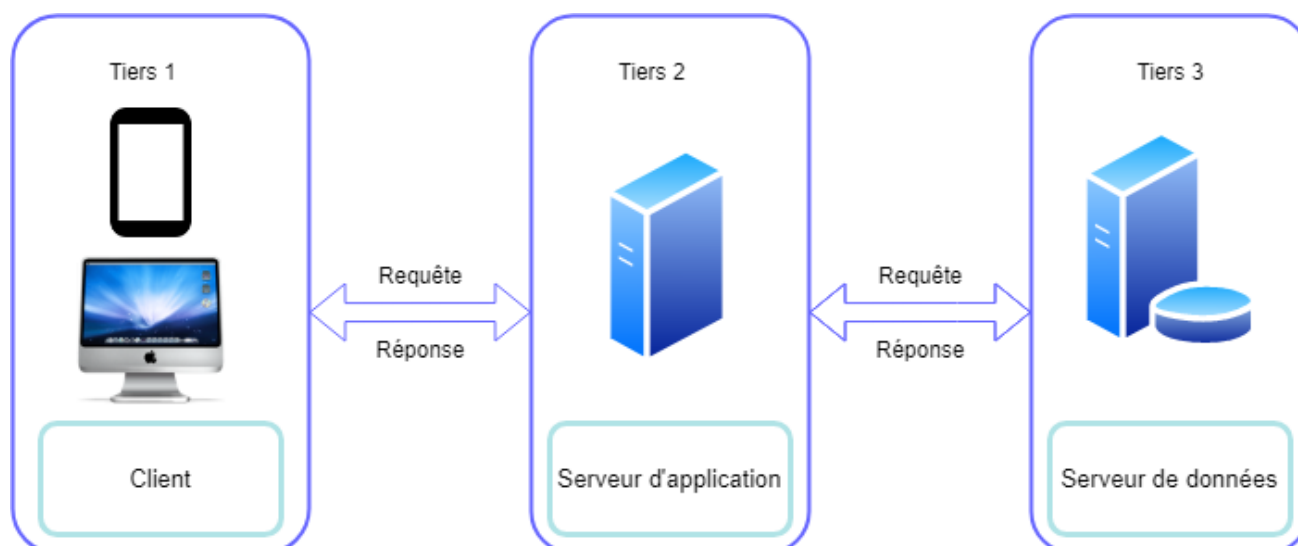


Figure 12 : Architecture 3 tiers

Comme la figure 12 l'indique, on identifie les trois niveaux (tiers) comme suit :

- **Client** : La partie présentation de l'application. C'est la partie interactive et visible de l'application. A travers cette couche, le client peut accéder aux données.
- **Serveur d'application** : La couche métier qui s'occupe du traitement de l'information. Cette couche contient les différentes règles de gestion et de contrôle du système.
- **Serveur de données** : la partie accès et stockage des données. Elle correspond à la partie qui gère l'accès aux données de l'application.

5.2. Architecture logique

Pour le backend, nous avons adopté une architecture en microservices qui est une méthode distinctive de développement de logiciels. Elle est une variante du style architectural de l'architecture orientée services (SOA) qui structure une application comme un ensemble de services faiblement couplés. Le point d'entrée de cette architecture est l'api gateway qui s'occupe de redirection des requêtes clients au microservices concernées (Figure 13).

Pour le frontend, nous avons adopté une architecture en microfrontends qui présente la continuité des microservices en frontend. Elle consiste alors à découper le frontend en des sous applications autonomes, développés et déployés indépendamment. En d'autres termes l'interface utilisateur marquée sur la figure 13 sera à son tour divisée en des microfronts gérés par une application conteneur appelée root. Dans le release 1, nous allons étudier ces architectures plus profondément.

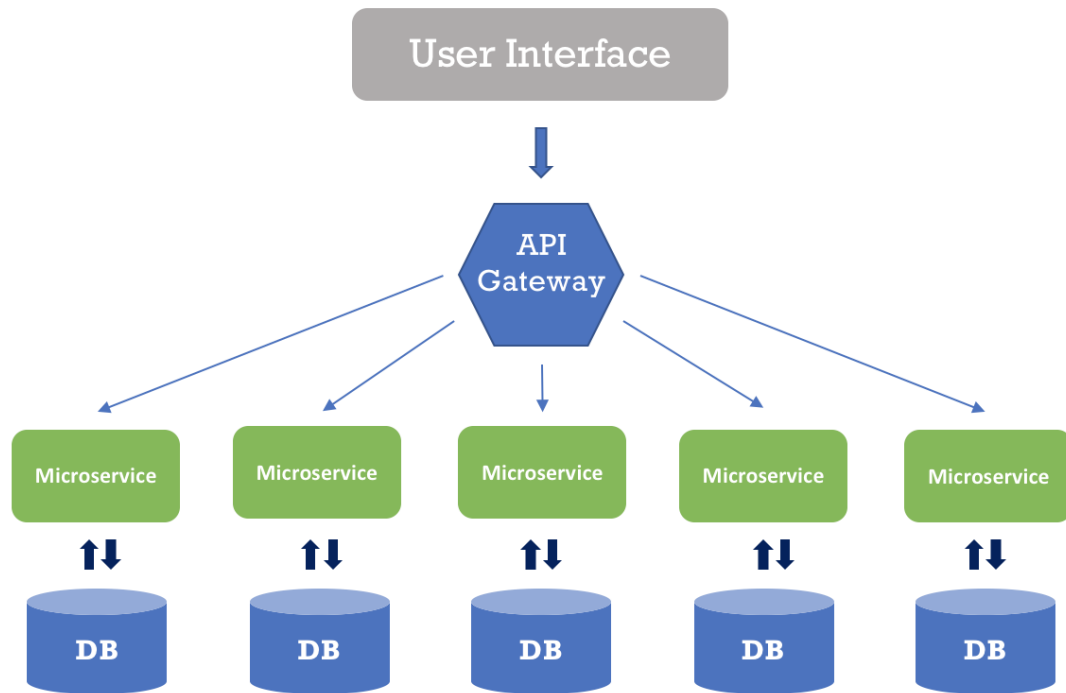


Figure 13: Architecture en microservices [22]

Conclusion

Ce chapitre nous a permis de déterminer les acteurs de notre application ainsi que les différents besoins fonctionnels et non fonctionnels, le diagramme de cas d'utilisation et le backlog de produit nous a permis de dégager les différentes fonctionnalités de manière détaillée. Et finalement, nous avons présentés l'environnement de travail et l'architecture globale de notre projet.

Le prochain chapitre sera dédié au release 1 qui est l'étude et la mise en place de l'architecture projet.

Etude et mise en place de l'architecture de projet

Introduction

Ce chapitre consiste dans sa première partie à modéliser le premier Release en présentant la phase de planification de notre travail réalisé. Dans la deuxième partie, nous allons déterminer les différents sprints de notre release tout en spécifiant les étapes de chaque sprint.

1. Planification du release 1

Nous avons divisé notre première release en deux sprints. Le premier consiste à étudier et mettre en place l'architecture microservices et le deuxième est dédié à l'étude et la mise en place de l'architecture microfrontends (Tableau 4).

Tableau 4 : Planification Release 1

Release 1	
Sprint 1.1	Sprint 1.2
Etude, prototypage, choix technologique et mise en place de l'architecture microservices. De 15/02/2020 à 18/03/2020	Etude, choix technologique et mise en place de l'architecture microfrontends. De 19/03/2020 à 20/04/2020

2. Sprint 1.1 Etude et mise en place de l'architecture microservices

Au cours de ce sprint nous détaillons les concepts clé autour les microservices, le choix technologique et la démarche à suivre pour mettre en place notre architecture.

2.1. Caractéristiques d'une architecture microservices

Dans un premier lieu, nous allons décrire les caractéristiques d'une architecture microservices présentés dans l'article de Martin Fowler qui est l'un des textes fondateurs de cette architecture [22] ces concepts feront par la suite notre base pour diviser notre backlog en microservices

2.1.1. Division en composants via les services

Par définition un logiciel développé en microservices peut être décomposé en plusieurs services utilisés comme des composants. Ceci permet de déployer, redéployer, modifier et remplacer un service sans affecter l'intégrité de l'application. En plus, ceci offre une interface de composants plus explicite. En effet, la plupart des langages sont incapables de fournir un bon mécanisme pour définir une interface publiée explicitement. Souvent, la documentation et la discipline empêchent les clients de rompre l'encapsulation d'un composant, ce qui entraîne un couplage fort entre les composants. Les services résolvent ce problème grâce à l'utilisation des mécanismes d'appel distants explicites [22].

2.1.2. Organisation autour des capacités métiers

La décomposition classique des applications logicielles consiste à décomposer l'application selon les couches techniques, ceci permet d'avoir trois équipes (équipe interface utilisateur, équipe développement métier et équipe base de données). Cependant, une application en microservices est décomposée en des services centrés sur des capacités métiers et où les équipes sont inter-fonctionnelles, avec tous les niveaux de compétences (UI, stockage, gestion de projet) [22].

2.1.3. Produits, pas projets

Avec les microservices, une équipe est responsable d'un produit tout au long de son cycle de vie. Une équipe de développement assume la pleine responsabilité du logiciel en production. Ceci mène les développeurs à rester au courant du comportement de leurs produits en production et augmente le contact avec le client vu qu'ils doivent prendre une partie de la charge du support [22].

2.1.4. Extrémités Intelligentes et canaux stupides

La communauté microservices favorise l'utilisation des canaux de communication stupides et des extrémités intelligents. Les applications en microservices visent à être aussi découplées et aussi cohérentes. Elles reçoivent une demande, appliquent la logique appropriée et produisent une réponse. Celles-ci sont chorégraphiées en utilisant des protocoles REST simples plutôt que des protocoles complexes tels que WS-Choreography ou BPEL où l'orchestration est effectuée par un outil central. Les deux protocoles souvent utilisés sont le HTTP et le messaging avec des bus de messagerie asynchrones et légers. L'infrastructure pour le bus de messaging est typiquement stupide, l'intelligence est concrétisée toujours dans les extrémités qui produisent et consomment le message (dans les services) [22].

2.1.5. Une gouvernance décentralisée

L'une des conséquences de la gouvernance centralisée est la normalisation de l'application sur une seule plateforme technologique. L'expérience montre que cette approche présente des limites, en effet "pas tous les problèmes sont des clous et pas toutes les solutions sont des marteaux ", il est donc difficile de trouver une seule technologie qui résout tous les problèmes. Avec une architecture microservices, nous sommes capables de développer chaque service en utilisant une technologie, un langage ou une plateforme différente ce qui permet de résoudre le problème d'une façon efficace. Un autre aspect de la gouvernance décentralisée concerne les équipes qui construisent les microservices. Ces équipes préfèrent l'idée de produire des outils utiles afin que d'autres développeurs puissent les utiliser pour résoudre des problèmes similaires. Ceci favorise l'idée du partage et de l'open source [22].

2.1.6. Gestion des données décentralisée

Avec une architecture microservices, chaque service a un model conceptuel différent. De plus, les microservices décentralisent également les décisions de stockage de données. En effet, pour une application monolithique, les entreprises préfèrent une base de données logique unique pour les données persistantes alors que pour les microservices chaque service gère sa propre base de données. Ceci est réalisé soit avec des instances différentes de la même technologie de base de données, soit avec des systèmes de base de données entièrement différents, nous parlons ici de "polygot Persistence" [22].

2.1.7. Automatisation de l'infrastructure

Les techniques d'automatisation de l'infrastructure ont considérablement évolué au cours des dernières années. L'évolution du cloud a réduit la complexité opérationnelle de la construction, du déploiement et de l'exploitation de microservices. Les équipes qui développent des applications en microservices ont une expérience considérable dans la livraison continue et l'intégration continue. Ces équipes utilisent des techniques d'automatisation des infrastructures [22].

2.1.8. Tolérance aux pannes

Les applications en microservices sont conçues de manière à pouvoir tolérer l'échec des services. Tout appel de service pourrait échouer en raison de l'indisponibilité du fournisseur. Ceci ne doit jamais affecter le fonctionnement de client. Par conséquent, il est nécessaire que les équipes de microservices réfléchissent constamment sur la manière dont les pannes du service affectent l'expérience de

l'utilisateur. Alors, nous constatons l'utilisation des techniques de monitoring en temps réel pour détecter les échecs et la restauration du service est réalisée automatiquement [22].

2.1.9. Conception évolutive

Avec une application monolithique, toute modification nécessite une compilation et un déploiement de toute l'application. Avec les microservices, et grâce à leurs décompositions en services, il suffit de redéploier les services modifiés. Ceci simplifie et accélère le processus de publication et rend l'évolution de l'application plus fluide.

Les microservices sont une approche conceptuelle qui se traite différemment selon chaque langage. C'est l'un des avantages de cette architecture, car les développeurs peuvent utiliser le langage qu'ils maîtrisent le mieux. Dans notre cas nous orienterons au langage java, sa syntaxe d'annotation est le facteur clé qui le fait un excellent langage de programmation pour le développement de microservices. Par ailleurs, il existe trois plusieurs frameworks populaires pour développer une application en microservices comme Micron ut, Ballerine, Drop Izard. Nous intéressons au Spring Boot et Spring cloud ce sont les leaders pour la création d'applications microservices en Java grâce à leur maturité, documentation open-source et richesse en fonctionnalités avec une vaste communauté vers laquelle on peut se référer pour obtenir de l'aide. Les microservices Spring Boot peuvent être facilement déployés sur diverses plates-formes.

2.2. 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 contexte borné appelé en anglais « Bounded Context » pour les définir. D'après Martin Fowler [22] 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 (modèle dans un contexte) et leurs liaisons [23]. Cette carte est illustrée par la figure 14 :

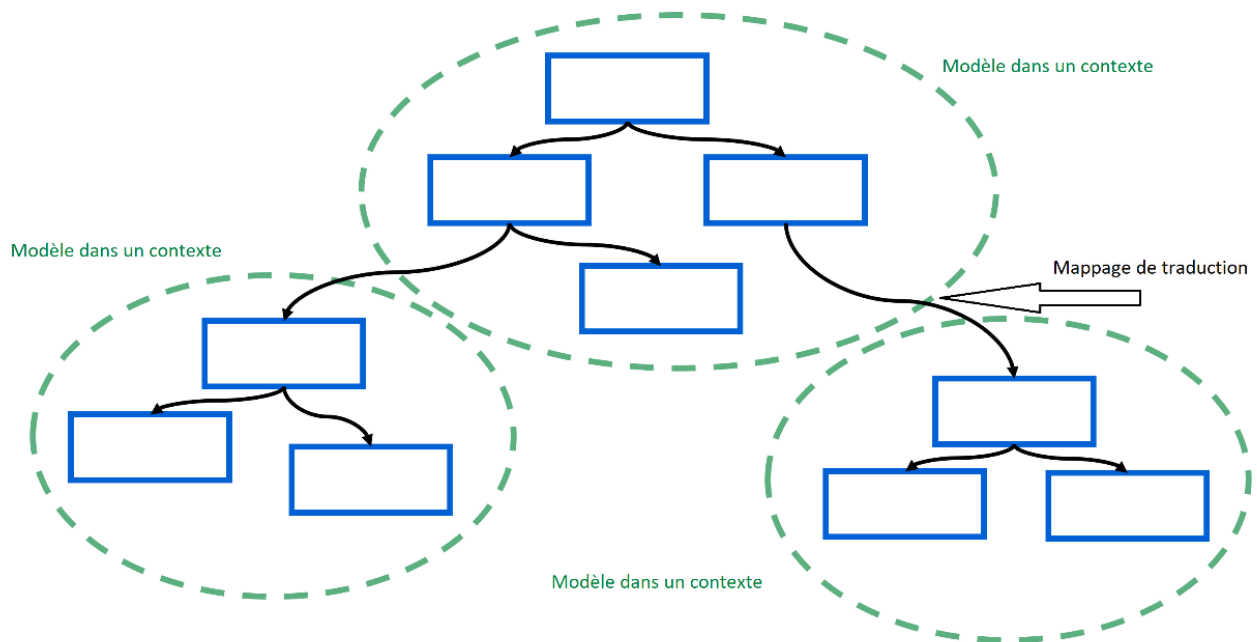


Figure 14 : Carte de contexte

2.3. Architecture de Spring Cloud

Dans cette partie nous allons décrire l'architecture de Spring cloud et voir comment choisir chaque composant pour obtenir un système final sein.

Spring cloud offre plusieurs services techniques tel que l'enregistrement de service de découverte « service registry », la passerelle API « API Gateway », le traçage distribué « distributed tracing ». Ces composants sont illustrés par la figure 15 et dans la partie qui suit nous allons fournir une description détaillée pour chaque composant.

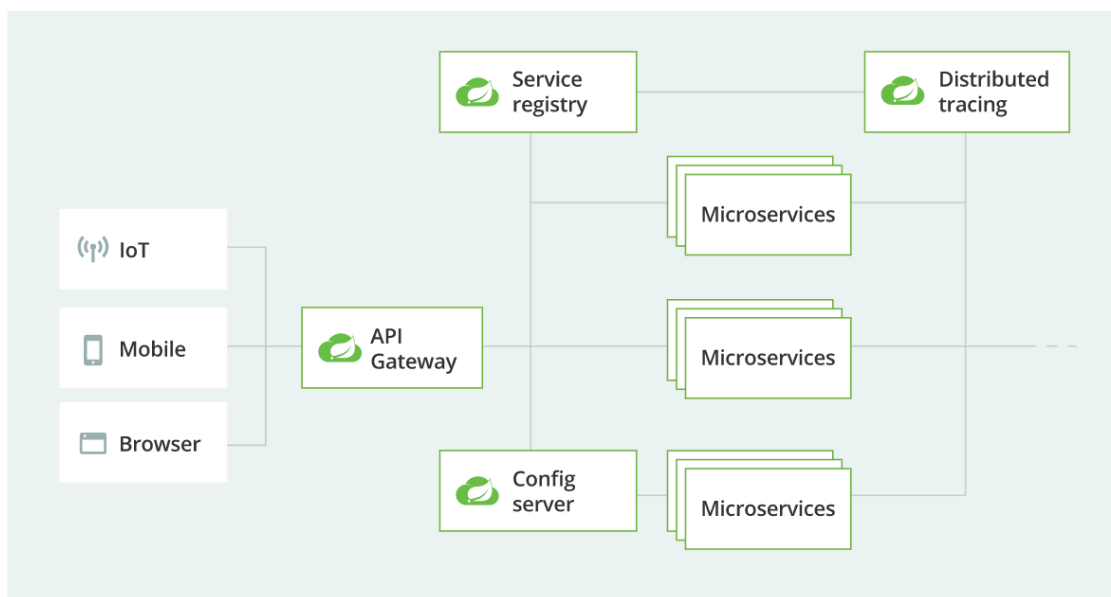


Figure 15 : Composants de l'architecture de spring cloud [24]

2.3.1. API Gateway

Spring cloud gateway est un proxy inverse (un proxy inverse est la seule connexion entre Internet et le réseau privé. Toutes les requêtes du serveur d'arrière-plan au réseau local passent par la même interface de communication avant d'être transférées aux systèmes cibles) (Figure 16) amélioré avec des fonctionnalités plus avancées y compris l'orchestration, la sécurité et le monitoring, agissant pour gérer une demande client en appelant un ou plusieurs microservices et en agrégeant les résultats pour identifier le meilleur chemin afin d'offrir une expérience fluide à l'utilisateur.

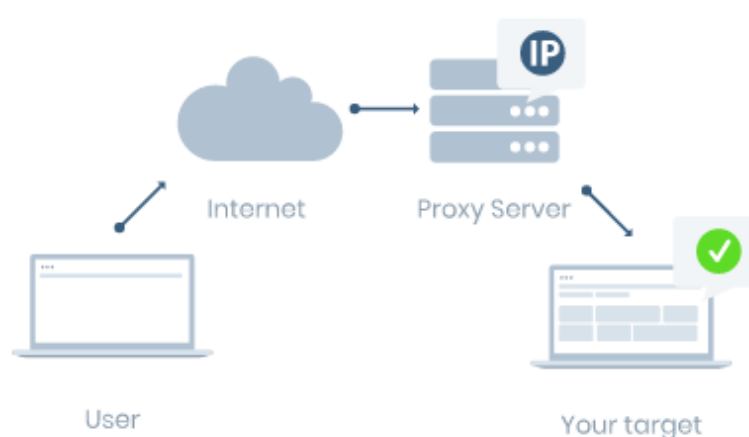


Figure 16 : Reverse-proxy [26]

Pour ce composant nous trouvons plein de passerelles, nous avons opté à comparer entre les deux choix les plus recommandés qui sont Netflix Zuul Proxy et Spring cloud gateway par le temoinage du tableau 5.

Tableau 5 : Comparaison entre Zuul et Spring Cloud Gateway

Zuul	Spring Cloud Gateway
Zuul se repose sur un modèle bloquant c'est-à-dire elle utilise autant de threads que le nombre de requêtes entrantes qui seront servi de manière synchrone.	Spring Cloud Gateway se repose sur un modèle non bloquant c'est à dire un thread est toujours disponible pour traiter une requête entrante de manière asynchrone.
Zuul n'était pas réactif à l'origine, mais Zuul 2 est une réécriture radicale pour le rendre réactif. Malheureusement, Spring Cloud ne prend pas en charge Zuul 2 et il ne le sera probablement jamais.	Il est basé sur Spring 5, Reactor et Spring WebFlux. Non seulement cela, il inclut également l'intégration de disjoncteurs, la découverte de services avec Eureka.

Vu sa base non bloquante pour les requêtes clients et sa facilité d'intégration avec spring cloud nous avons choisi comme API Gateway pour nos microservices Spring Cloud Gateway.

2.3.2. Service de découverte

La découverte de services est le processus de détection automatique des périphériques ou des services sur un réseau ce qui permet de réduire l'effort de passer par un long processus d'installation ou de configuration. Il n'est pas nécessaire pour la maintenance des serveurs physiques (un fichier de configuration satisfera principalement à cette exigence dans une application monolithique). Cependant, la découverte de services devient incontournable dans un environnement de microservices. Si nous écrivons du code qui appelle un service qui a, par exemple, une API Thrift ou REST, notre code devra connaître l'emplacement réseau (à la fois l'adresse IP et le port) d'une instance de service. Dans une application de microservices, cela sera difficile car les instances de service ont des emplacements réseau attribués de manière dynamique. La maintenance manuelle d'un fichier de configuration n'est pas possible lorsque vos services ont des emplacements réseau dynamiques en raison du redémarrage, des mises à niveau, des pannes et de la mise à l'échelle automatique.

Pour assurer la découverte automatique de nos microservices on a la possibilité d'utiliser Zookeeper, Consul ou Eureka.

- **Zookeeper** est un service centralisé pour maintenir les informations de configuration et synchroniser les systèmes distribués et fournir des services de groupe.

- **Consul** est un groupement de services offrant un plan de contrôle complet avec des fonctionnalités de découverte, de configuration et de segmentation des services.
- **Eureka** est un service basé sur REST qui est principalement utilisé dans le cloud AWS pour localiser des services afin d'équilibrer la charge entre eux c'est-à-dire assurer la haute disponibilité.

Zookeeper est un système distribué à son tour ce qui ajoutera une couche de complexité, Consul offre un stockage distribué clé valeur et une possibilité de découverte via dns mais notre critère de choix était la facilité d'intégration avec Spring cloud ce qui représente le point fort pour Eureka qui sera alors notre service de découverte. Les microservices dotant d'un client Eureka s'enregistrent dans le serveur de découverte comme il est indiqué sur la figure 17.

DS Replicas			
Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
CONTACT	n/a (1)	(1)	UP (1) - localhost:contact:8083
GATEWAYSERVICE	n/a (1)	(1)	UP (1) - localhost:gatewayService:8885
PROJMANAGEMENT	n/a (1)	(1)	UP (1) - localhost:projManagement:8084
RESOURCEMANAGEMENT	n/a (1)	(1)	UP (1) - localhost:resourceManagement:8082

Figure 17 : Service discovery avec Eureka

2.3.3. Distributed tracing

Le débogage des applications distribuées peut être complexe et prendre du temps. Pour toute défaillance donnée, nous devons peut-être rassembler des traces d'informations provenant de plusieurs services indépendants.

Spring Cloud Sleuth peut instrumenter les applications de manière prévisible et reproductible. Et lorsqu'il est utilisé en conjonction avec Zipkin, nous pouvons nous concentrer sur tous les problèmes de latence que vous pourriez avoir.

2.3.4. Protocole de communication

Un client et des services peuvent communiquer via de nombreux types différents de communication, chacun d'eux ciblant un scénario et des objectifs différents. Au départ, ces types de communication peuvent être classés selon deux axes.

Le premier axe définit si le protocole est synchrone ou asynchrone :

- **Protocole synchrone** : HTTP est un protocole synchrone. Le client envoie une requête et attend une réponse du service. Cela est indépendant de l'exécution du code client, qui peut être

synchrone (le thread est bloqué) ou asynchrone (le thread n'est pas bloqué, et la réponse atteint finalement un rappel). Le point important ici est que le protocole (HTTP/HTTPS) est synchrone et que le code client peut continuer sa tâche seulement quand il reçoit la réponse du serveur HTTP.

- **Protocole asynchrone :** D'autres protocoles, comme AMQP (un protocole pris en charge par de nombreux systèmes d'exploitation et environnements cloud), utilisent des messages asynchrones. Le code client ou l'expéditeur du message n'attend généralement pas de réponse. Il envoie simplement le message comme lors de l'envoi d'un message à une file d'attente RabbitMQ ou à tout autre service broker de messages.

Le deuxième axe définit si la communication a un ou plusieurs destinataires :

- **Destinataire unique :** Chaque demande doit être traitée par exactement un récepteur ou un service.
- **Plusieurs destinataires :** Chaque demande peut être traitée par zéro à plusieurs destinataires. Ce type de communication doit être asynchrone.

Une application basée sur des microservices utilise souvent une combinaison de ces styles de communication. Le type le plus courant est une communication avec un seul destinataire, avec un protocole synchrone comme HTTP/HTTPS lors de l'appel d'un service web HTTP d'API ordinaire. Les microservices utilisent aussi en général des protocoles de messagerie pour la communication asynchrone inter-microservices.

Communication synchrone : Pour ce type de communication spring cloud offre deux alternatives :

- **RestTemplate :** Client synchrone pour effectuer des requêtes HTTP, exposant une API de méthode de modèle simple sur des bibliothèques client HTTP sous-jacentes telles que JDK HttpURLConnection, Apache HttpComponents [27]. Le problème avec RestTemplate c'est qu'elle sera obsolète dans une version future (> 5.0) et n'aura pas de nouvelles fonctionnalités majeures ajoutées à l'avenir.
- **Spring Cloud OpenFeign :** Feign est un client de service Web déclaratif. Cela facilite l'écriture des clients de services Web. Pour utiliser Feign, Il suffit d'annoter une interface comme il est illustré par la figure 18.


```

@FeignClient(name = "resourceManagement", url = "localhost:8082")
public interface ProxyResourceProject {

    // get a list of resources from the resourceManagement-MS
    @GetMapping("resources/all")
    List<Ressource> getAllResources();
}

```

Figure 18 : Communication avec openFeign

Nous avons choisi d'utiliser Spring cloud OpenFeign pour toute communication synchrone inter-microservices.

2.4. Urbanisation fonctionnelle en microservices

La figure 19 présente le modèle métier global de notre plateforme de gestion de projet qui met l'accent sur les différents composants métiers ainsi que les liens qui les attachent.

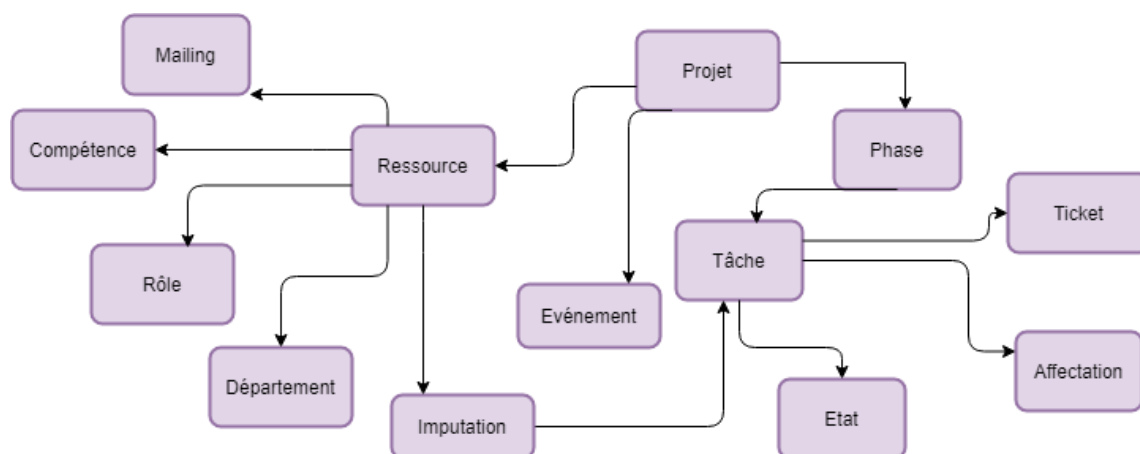


Figure 19: Modèle métier de la plateforme de gestion de projet

En suivant le pattern « Bounded Contexts » du « Domain Driven Design », expliqué au début de ce chapitre, nous divisons notre modèle métier en trois sous modèles. La carte de contexte de la figure 20 présente les limites des contextes et deux exemples de dépendances entre eux.

Chaque contexte présente un microservice autonome et le couplage entre les microservices est faible.

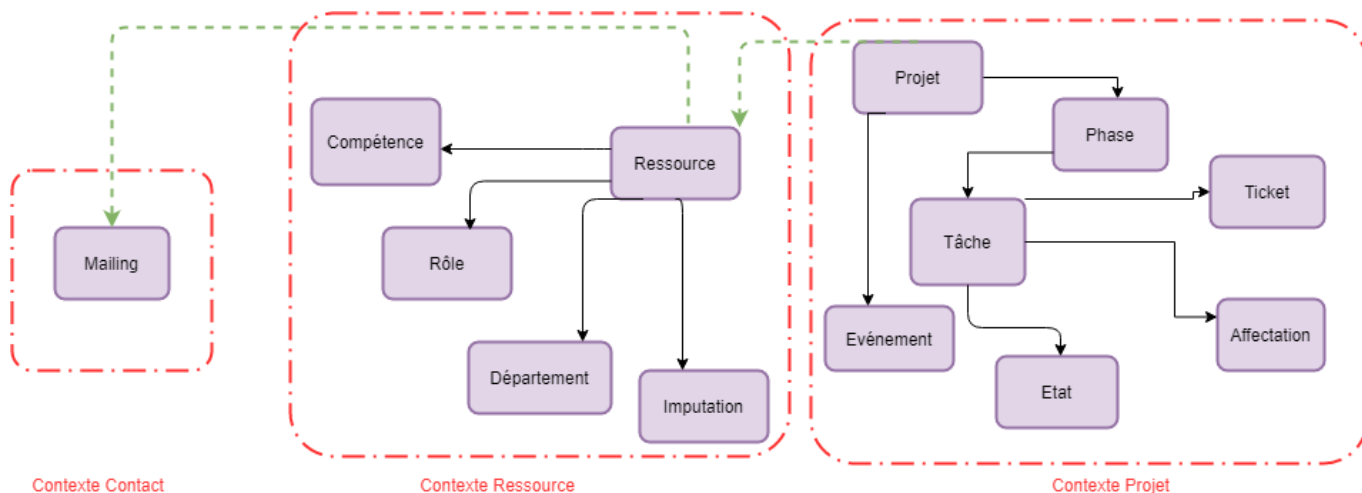


Figure 20 : Carte de contexte de plateforme de gestion de projet

2.5. Synthèse

L'ensemble des choix effectués est illustré par la figure 21.

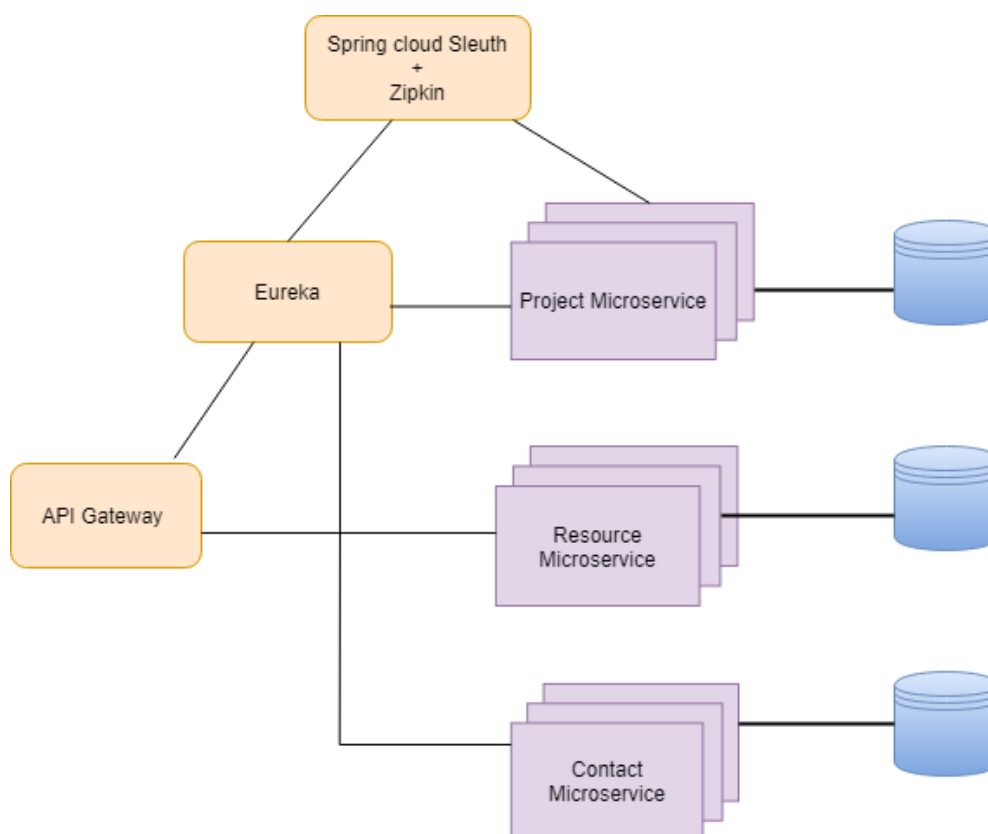


Figure 21 : Architecture microservices proposée

Nous structurons nos microservices développés avec Spring boot en des packages :

- **Model** : regroupe les différentes tables de la base de données.

- **Dto** : contient la correspondance en classe des models (en spring boot càd elle n'est pas annotée avec @Document ou @Entity)
- **Mapper** : regroupe les interfaces qui contiennent chacune des méthodes qui transforment un Model en Dto, une liste de model en Dtos et vice versa.
- **Repository** : regroupe des interfaces héritant de l'interface Repository. Leur Objectif consiste à rendre la création de la couche d'accès aux données plus rapides.
- **Exception** : contient les exceptions
- **Service** : regroupe des interfaces contenant chacune les signatures d'un ensemble de fonctions qui seront redéfinies par la suite.
- **Service implémentations** : regroupe les classes qui implémentent les services
- **Controller** : un ensemble de classes qui contiennent les APIs GET, POST, PUT et DELETE

La figure 22 représente le diagramme de package d'un MS-NAME :

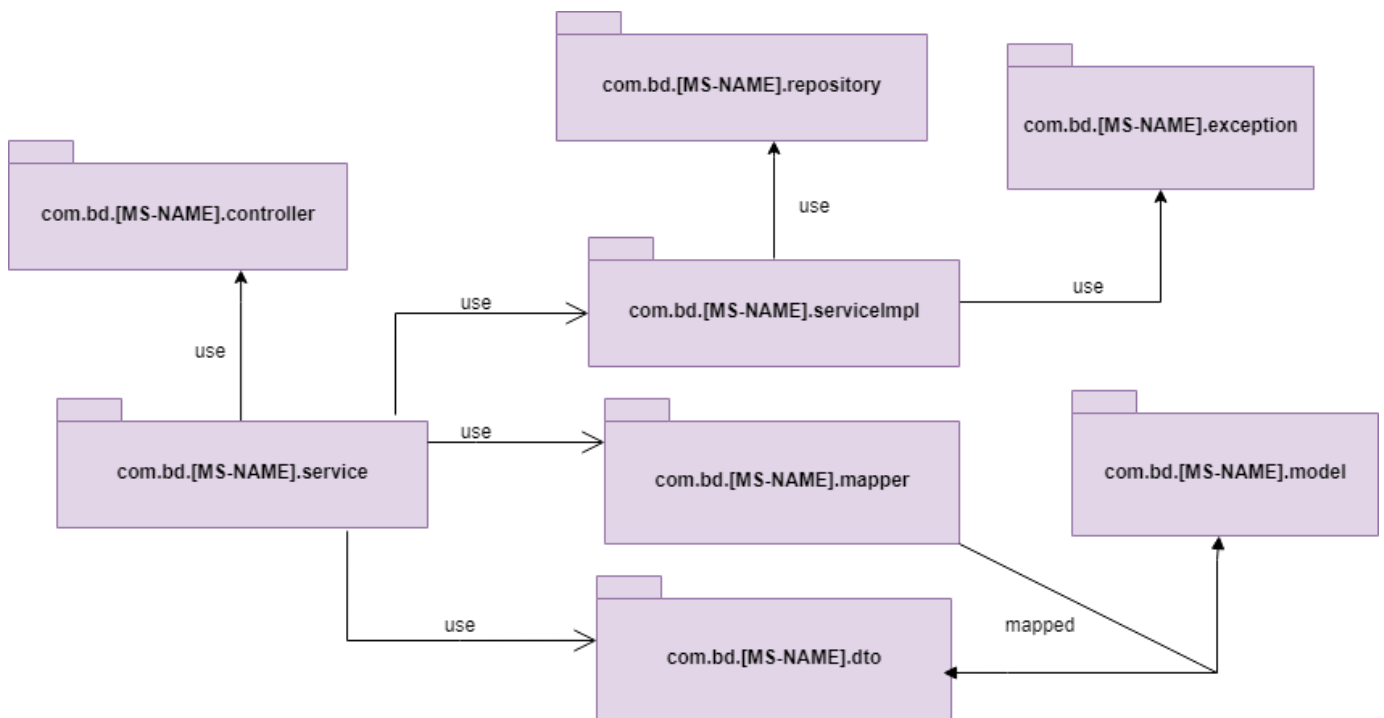


Figure 22 : Diagramme de package du microservice MS-NAME

2. Sprint 2 Mise en place de l'architecture microfrontends

Tout comme pour le back, il fallait donc trouver une solution pour éviter le fourre-tout. Il nous fallait un moyen de découper une application frontend en domaines indépendants et réutilisables et les intégrer dans les différentes applications.

Pour permettre cela, les développeurs se sont orientés vers la componentization. Chaque composant a ainsi une responsabilité qui lui est propre, et qui pourrait être utilisé dans différents contextes. Notre frontend sera alors un groupement de plusieurs composant chargé sur la même page web et qui fait sentir l'utilisateur final que c'est la même application. En Web moderne, il y'a plusieurs moyens de le faire et tous ont leurs avantages et inconvénients.

2.1. Approches d'intégration des microfrontends

Les solutions d'intégration des microfrontends se répartissent sur 3 catégories distinctes :

- Build time Integration
- Server-side integration
- Client-side integration

2.1.1. Build time integration

Une approche qui consiste à publier chaque micro-frontend en tant que package et de faire en sorte que l'application conteneur les inclue tous en tant que dépendances. La figure 23 présente le « package.json » du conteneur d'une application nommée « feed-me » qui est divisé en 3 microfrontends « browse-restaurants », « order-food » et « user-profile ».

```
{
  "name": "@feed-me/container",
  "version": "1.0.0",
  "description": "A food delivery web app",
  "dependencies": {
    "@feed-me/browse-restaurants": "^1.2.3",
    "@feed-me/order-food": "^4.5.6",
    "@feed-me/user-profile": "^7.8.9"
  }
}
```

Figure 23: Exemple d'un « package.json » d'une application divisée en microfrontends basée sur l'approche Build time integration

Au début, cela semble logique. Il produit un seul bundle Javascript déployable nous permettant de dédupliquer les dépendances communes de nos différentes applications. Cependant, cette approche signifie que nous devons recompiler et déployer chaque micro-frontend afin de publier un changement dans une seule partie du produit

2.1.2. Server-side integration

L'application est intégrée pendant que les pages sont servies, le navigateur de l'utilisateur obtient l'application complète (Figure 24).

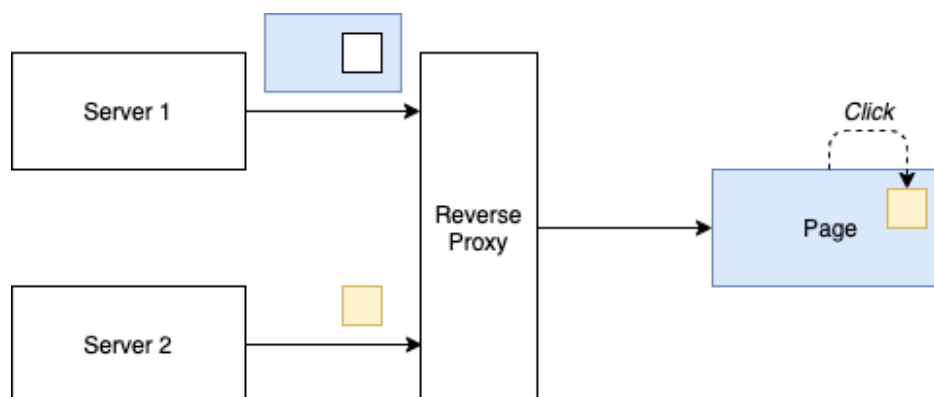


Figure 24 : Server side integration

La complexité de cette solution réside totalement dans la couche proxy inverse. La manière dont les différents microsites sont combinés en un seul site peut être délicate. En particulier, nous trouvons des difficultés comme les règles de mise en cache et le suivi.

Dans cette approche nous trouvons les solutions suivantes : Server Side Includes(SSI), Edge Side Includes (ESI) et project Mosaic

2.1.3. Client-side Integration

La construction de micro-frontends intègre l'application dans le navigateur Web de l'utilisateur. Chaque partie de l'application est livrée au navigateur indépendamment, puis chaque microfrontend est chargé à la demande.

Avec cette approche, aucune infrastructure supplémentaire n'est nécessaire pour construire l'application au moment de l'exécution et elle semble être la plus flexible. Les composants de l'application peuvent partager un certain contexte utilisateur et donc agir comme celui intégré pendant la construction sans compromettre les autres exigences pour les micro-frontends (Figure25).

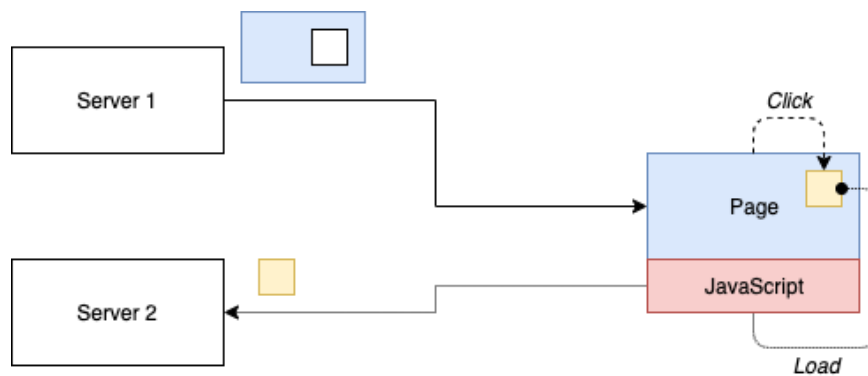


Figure 25 : Client Side Integration

Récapitulatif :

- Build time integration est sûrement n'est pas recommandée car après le build l'application revient monolithique c'est elle va être déployé comme un seul bloc du coût on va perdre les avantages des microfrontends déjà détaillés dans le chapitre 1.
- Server Side Integration offre une architecture complexe et vue que nous allons l'intégrer avec une architecture microservice la complexité devient énorme.
- Alors nous adoptons le troisième choix qui est l'approche de Client-side composition et nous abordons dans la partie suivante une étude comparative sur les technologies utilisées dans cette approche.

2.2. Etude comparative des solutions de l'approche « client-side composition »*Tableau 6 : Comparaison des solutions de l'approche "Client Side Composition"*

Technique	Point Forts	Points faible
Iframes	<ul style="list-style-type: none"> ▪ Isolation des composants ▪ Facile à mettre en place, aucun outil supplémentaire n'est nécessaire 	<ul style="list-style-type: none"> ▪ En termes de performances, chaque Iframe doit tout charger (assets, libraries, la réutilisation des actifs, des bibliothèques etc ▪ Difficile à mettre en œuvre des « deep linking » ▪ Difficile d'implémenter le RWD (Responsive Web Design)
Single SPA	<ul style="list-style-type: none"> ▪ Intégration parfaite entre les microfrontends et l'application conteneur ▪ Support pour la majorité des frameworks JS 	<ul style="list-style-type: none"> ▪ Documentation n'est pas assez riche

	▪ Lazy loading	
--	----------------	--

2.3. Single spa

Après avoir choisi single-spa comme une méthode d'intégration pour nos microfrontends

2.3.1. Aperçu architectural

Single-spa s'inspire des cycles de vie des composants des frameworks modernes (comme angular, react etc ..) et l'adapte à des application entières.

Les applications basées sur Single-spa se composent de :

- **Applications**, dont chacune est un SPA. Chaque application peut répondre aux événements de routage d'URL et doit savoir comment démarrer, se charger et se décharger à partir du DOM. La principale différence entre un SPA traditionnel et des applications à spa unique est qu'elles doivent pouvoir coexister avec d'autres applications et qu'elles n'ont pas chacune leur propre page HTML. Par exemple, vos SPAs React ou Angular sont des applications. Lorsqu'ils sont actifs, ils écoutent les événements de routage d'URL et placent le contenu sur le DOM. Lorsqu'ils sont inactifs, ils n'écoutent pas les événements de routage d'URL et sont totalement supprimés du DOM.
- **Single-spa-config** contenant la page HTML et le javascript qui registre les applications avec single spa via {nom, fonction qui charge le code de l'application, fonction qui détermine l'activation et le retrait d'une application}.

2.3.2. Architecture proposée

Tel est le cas pour les microservices nous avons découpé nos microfrontends selon le domaine métier. Nous avons opté à donner à chaque microservice son microfrontend(Planning-microfrontend pour projectManagement-MS et Resource-Microfrontend pour resourceManagement-MS).

Nous avons séparé les microfrontend partagé par tous les autres comme le « Navbar-Microfrontend » et « Footer-Microfrontend ». Et finalement « Authentication-MF » pour garantir la réutilisabilité (Figure27).

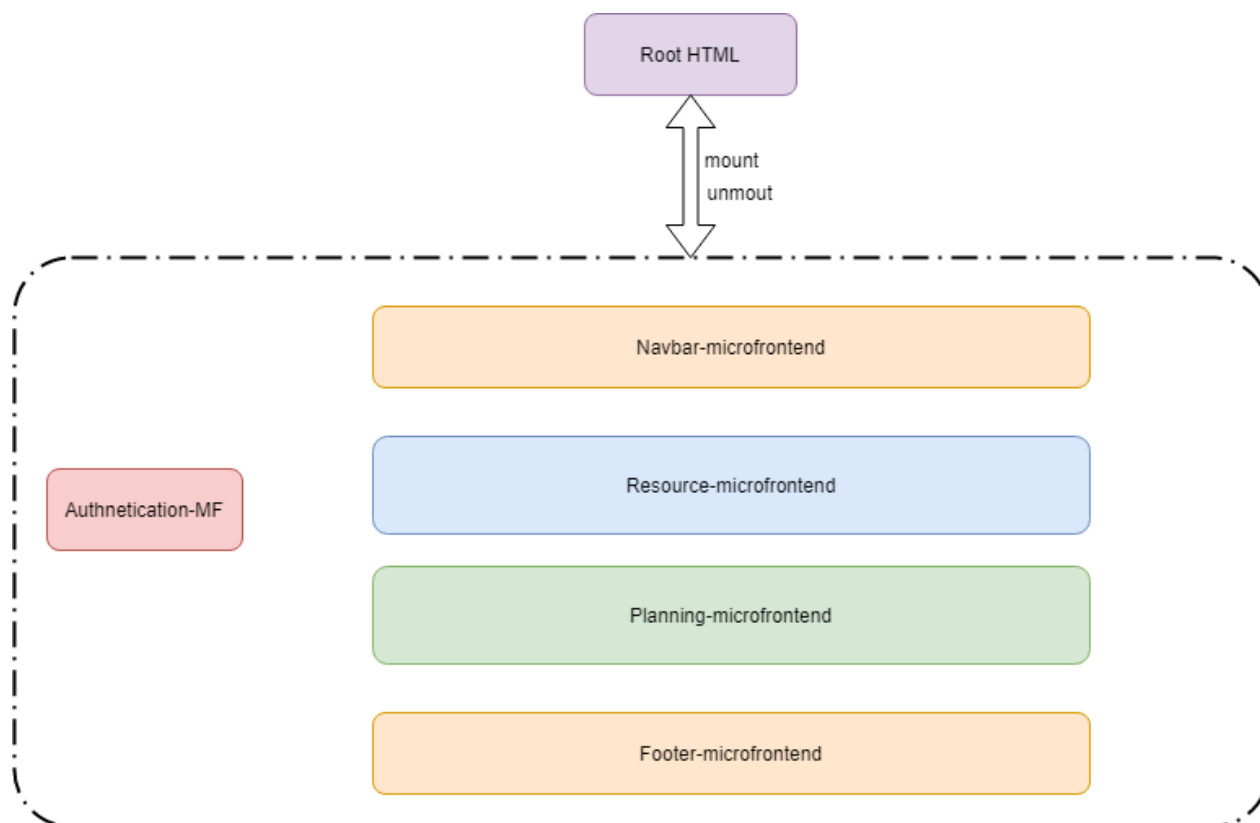


Figure 26 : Architecture microfrontend

L'application root est responsable de la coordination de l'activation ainsi que la désactivation des microfrontends. En d'autres termes elle contrôle si une application sera chargée dans le dom ou bien déchargée via une fonction appelé fonction d'enregistrement (Figure 26). Elle est dotée de 3 arguments le nom de microfrontend qui va s'enregistrer par exemple *planning-microfrontend*

« loadingFunction » importe le bundle.js de l'application concernée (() => import('src/planning-microfrontend/main.js'))

« activityFunction » l'application sera monté lorsque un chemin est routé ((location) => location.pathname.startsWith('/planning-microfrontend'))

```
registerApplication(name, loadingFunction, activityFunction)
```

Figure 27 : Api d'enregistrement des applications avec single-spa

Conclusion

Dans ce chapitre, après l'étude des concepts de base de l'architecture microservices et microfrontends nous avons argumenter notre démarche qui a réussi à aboutir la mise place de l'architecture de notre projet. Dans le prochain release nous allons passer à l'aspect fonctionnel pour injecter les besoins métier dans notre architecture.

Configuration du projet

Introduction

Après avoir mis en place l'architecture de notre projet, dans ce chapitre, nous allons présenter le travail réalisé lors du deuxième release appelé configuration du projet qui vise à préparer les ressources d'un projet. Nous commençons d'abord par une phase de planification dans laquelle nous soulignons les différents sprints à élaborer. Ensuite, pour chaque sprint, nous procédons à la présentation du backlog produit, l'analyse des besoins, la conception et enfin la réalisation.

1. Planification du release 2

Notre deuxième release se compose de deux sprints, le tableau 7 décrit leur planning :

Tableau 7: Planning du release 2

Release 2	
Sprint 2.1	Sprint 2.2
Gestion des ressources y compris l'authentification, la gestion de rôles, profils, compétences et départements. De 23/04/2020 à 15/05/2020	Création de projet et construction de son équipe. De 15/05/2020 à 12/06/2020

2. Sprint 2.1 Gestion de ressources

Ce sprint regroupe un ensemble de fonctionnalités relié à la gestion des essentiels paramètres d'un projet qui sont les ressources et dans notre cas les « ressources » représentent l'ensemble d'individus qui vont effectuer le travail demandé au cours d'un projet donné.

L'acteur principal dans ce sprint l'administrateur de la plateforme qui sera un membre dans le département IT de Business & Decision. En fait, ce dernier se connecte avec son compte déjà créé et il peut ajouter des nouvelles ressources possédantes chacune un rôle bien défini, l'affecter à un département, lui confier une liste de compétences. Chaque ressource inscrite dans la plateforme peut faire une mise à jour de son profil déjà créé (éditer sa photo de profil, son mot de passe...)

2.1. Backlog de Sprint 2.1

Le tableau 8 décrit le backlog de produit du sprint 2.1.

Tableau 8 : Backlog du sprint 2.1

User Story	Tâches	Estimation
En tant qu'utilisateur, je veux m'authentifier pour accéder à la plateforme.	[Backend] Implémentation de la partie métier de l'authentification [Frontend] Réalisation de l'interface d'authentification et intégration avec la partie métier	5
En tant qu'administrateur, je peux définir des équipes travaillant sur différents secteurs.	[Backend] Implémentation du service d'ajout d'un département. [Backend] Implémentation du service suppression d'un département. [Frontend] Préparation de l'interface d'ajout et d'affichage des départements et intégration avec la partie métier.	2
En tant qu'administrateur, je peux gérer les compétences	[Backend] Implémentation du service d'ajout d'une nouvelle compétence. [Backend] Implémentation du service de suppression d'une compétence. [Frontend] Préparation de l'interface d'ajout, d'affichage des compétences et intégration avec la partie métier.	2
En tant qu'administrateur, je peux gérer les ressources de la plateforme en affectant à chacun un rôle,	[Backend] Implémentation du service d'ajout une ressource. [Backend] Implémentation du service de mise à jour d'une ressource. [Backend] Implémentation du service de suppression d'une ressource.	7

département et des compétences.	<p>[Backend] Implémentation du service de recherche d'une ressource.</p> <p>[Backend] Implémentation du service d'envoi de mail pour transmettre le login et mot de passe de la ressource</p> <p>[Frontend] Préparation de l'interface d'ajout et mise à jour d'une ressource et intégration avec la partie métier.</p> <p>[Frontend] Préparation de l'interface d'affichage de toutes les ressources, la recherche d'une ressource particulière et intégration avec la partie métier.</p>	
En tant qu'utilisateur de la plateforme je peux gérer mon profil.	<p>[Backend] Implémentation des services de mise à jour de profil.</p> <p>[Frontend] Réalisation de l'interface de consultation et mise à jour de profil puis intégration avec la partie métier.</p>	4

2.2. Analyse des besoins de sprint 2.1

Une fois nous avons présenté le backlog de notre sprint, nous entamons alors la phase de la spécification fonctionnelle. Nous présentons cette dernière par les diagrammes de cas d'utilisation ainsi que des descriptions textuelles des différents cas.

2.2.1. Diagramme de cas d'utilisation du sprint 2.1

Le diagramme de cas d'utilisation illustré dans la figure 25 illustre la gestion des ressources qui est effectué par l'administrateur de la plateforme.

Cet acteur va en premier lieu ajouter les départements ou bien les secteurs (Business Unit) de son entreprise, définir des rôles, des compétences puis à chaque fois il ajoute une nouvelle ressource il va l'affecter alors à un département existant, le donner un rôle et l'attribuer un ensemble de compétences.

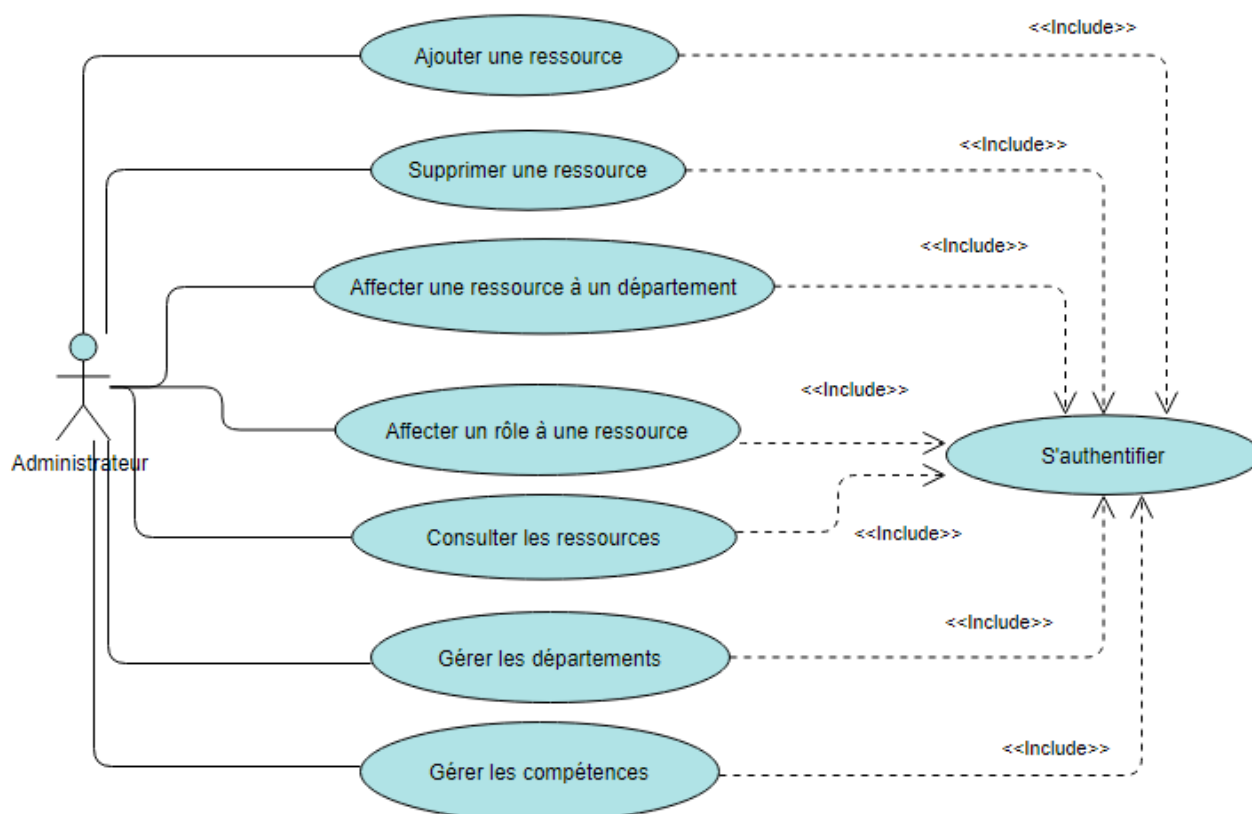


Figure 28 : Diagramme de cas d'utilisation de sprint 2.1 pour l'acteur "Administrateur"

La figure 26 illustre la partie gestion de compte. En fait, chaque utilisateur de la plateforme qui peut être un « Administrateur » ou « Chef de projet » ou un « Membre de la plateforme », va s connecter à la plateforme pour accéder à son espace, consulter son compte et le modifier :

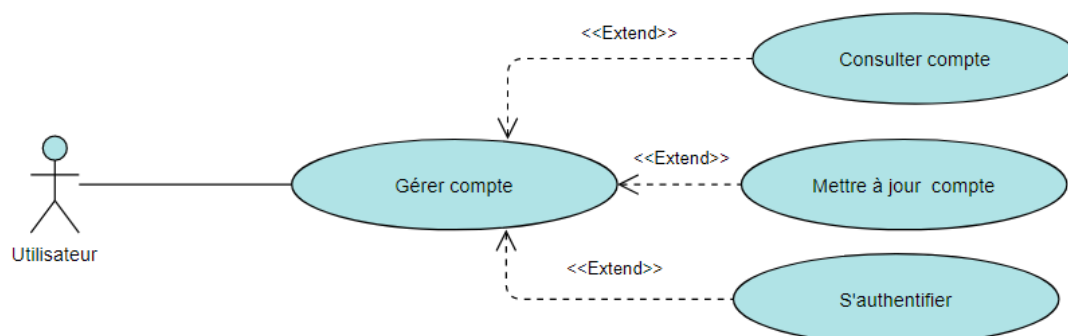


Figure 29 : Diagramme de cas d'utilisation de gestion de compte pour chaque utilisateur de la plateforme

2.2.2. Raffinement des cas d'utilisation

Dans cette section, nous choisissons quelques cas d'utilisation à détailler par une description textuelle afin de mieux expliquer leurs scénarios.

- Cas d'utilisation « Ajouter un ressource »

Tableau 9 : Description textuelle de cas d'utilisation "Ajouter une ressource"

Objectif	Ce cas d'utilisation permet à l'administrateur de la plateforme de créer une nouvelle ressource.
Acteur	Administrateur
Précondition	Utilisateur authentifié et rôle = administrateur
Postcondition	Une ressource ajoutée
Scénario nominal	<ol style="list-style-type: none"> 1. L'administrateur clique sur le bouton « New Resource » 2. Le système affiche un formulaire qui contient 2 parties « Personal info » et « Technical info » 3. Le chef de projet saisit les champs demandés 4. Le système vérifie les champs et ajoute la ressource 5. Le système redirige l'administrateur à la liste des ressources créées
Exceptions	<ul style="list-style-type: none"> - Champs vides ou invalides - Si l'administrateur choisit que le rôle de la ressource qu'il vient de créer est un administrateur alors le système cache les champs de formulaire : « Department » (Team), « Skills » et « Profile » car une ressource de type administrateur sera directement affectée au département « Administrators » (IT Department in Business & Decision)

- Cas d'utilisation « Mettre à jour un compte »

Tableau 10 : Description textuelle du cas d'utilisation "Mettre à jour un compte"

Objectif	Ce cas d'utilisation permet l'utilisateur de la plateforme de mettre à jour son compte.
Acteur	Administrateur Chef de projet Membre.
Précondition	Utilisateur authentifié et rôle = Administrateur Chef de projet Membre.
Postcondition	Compte mise à jour.

Scénario nominal	<ol style="list-style-type: none"> 1. L' utilisateur clique sur le dropdown du navbar de la plateforme et selectionne « Account ». 2. Le système affiche le compte de l'utilisateur. 3. L'utilisateur met à jour son compte. 4. Le système rafraiche les mises à jour effectués.
Exceptions	<ul style="list-style-type: none"> - Si l'administrateur choisit que le rôle de la ressource qu'il vient de créer est un administrateur alors le système cache les champs de formulaire : « Departement » (Team), « Skills » et « Profile » car une ressource de type administrateur sera directement affectée au département « Administraors » (IT Departement in Business & Decision)

2.3. Conception du sprint 2.1

Au niveau de cette phase de conception, nous allons présenter la vue statique de notre système au moyen de diagramme de classes ainsi que la vue dynamique au moyen des diagrammes de séquence.

2.3.1. Diagramme de classe du sprint 2.1

La figure 27 illustre le diagramme de classe du sprint 2.1 :

Nous avons une classe « Resource » qui désigne la ressource du projet, une entité « Role » désignant les rôles qui seront inscrits dans notre plateforme, une classe « Skill » designant une compétence qui sera attribuée à 1 ou plusieurs ressources et finalement la classe « Department » représentant les pôles ou les unités business dans une entreprise (tenons par exemple Digital Business Unit, CRM Business Unit, BI Business Unit).

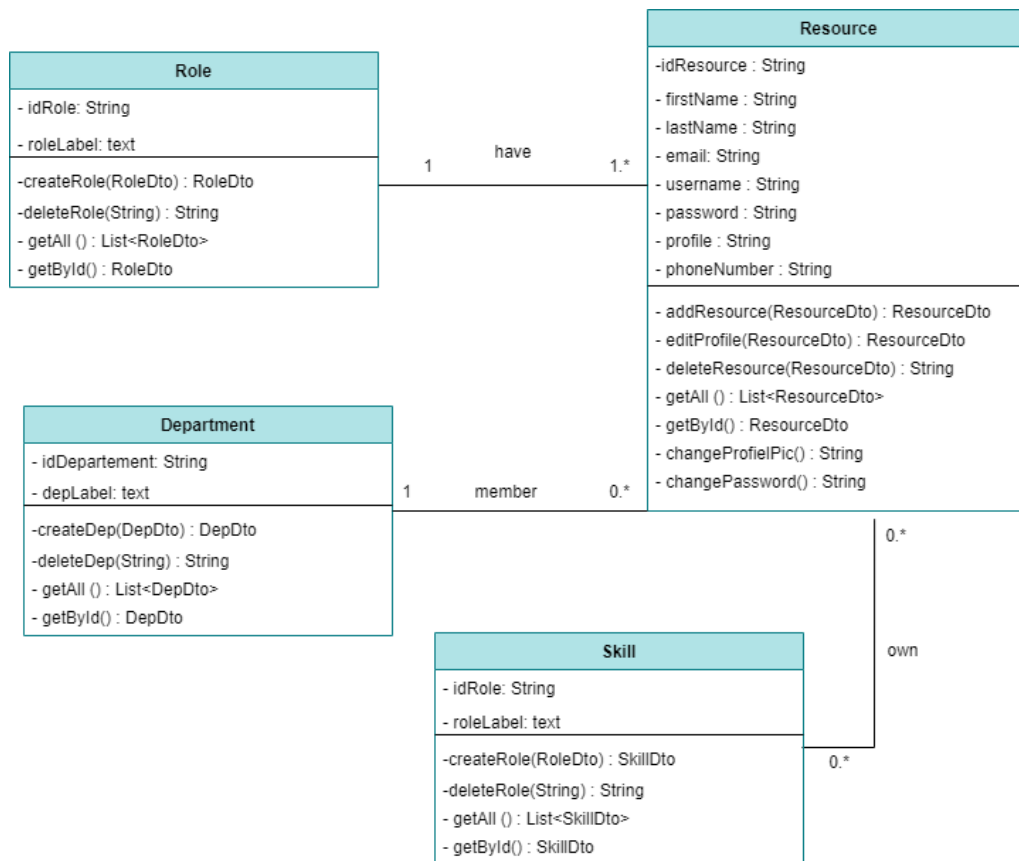


Figure 30 : Diagramme de classes du sprint 2.1

2.3.2. Diagrammes de séquence

Après le diagramme de classes, nous passons maintenant à voir l'aspect dynamique de notre système en détaillant le cas d'utilisation « Ajouter une ressource » schématisé par la figure 31.

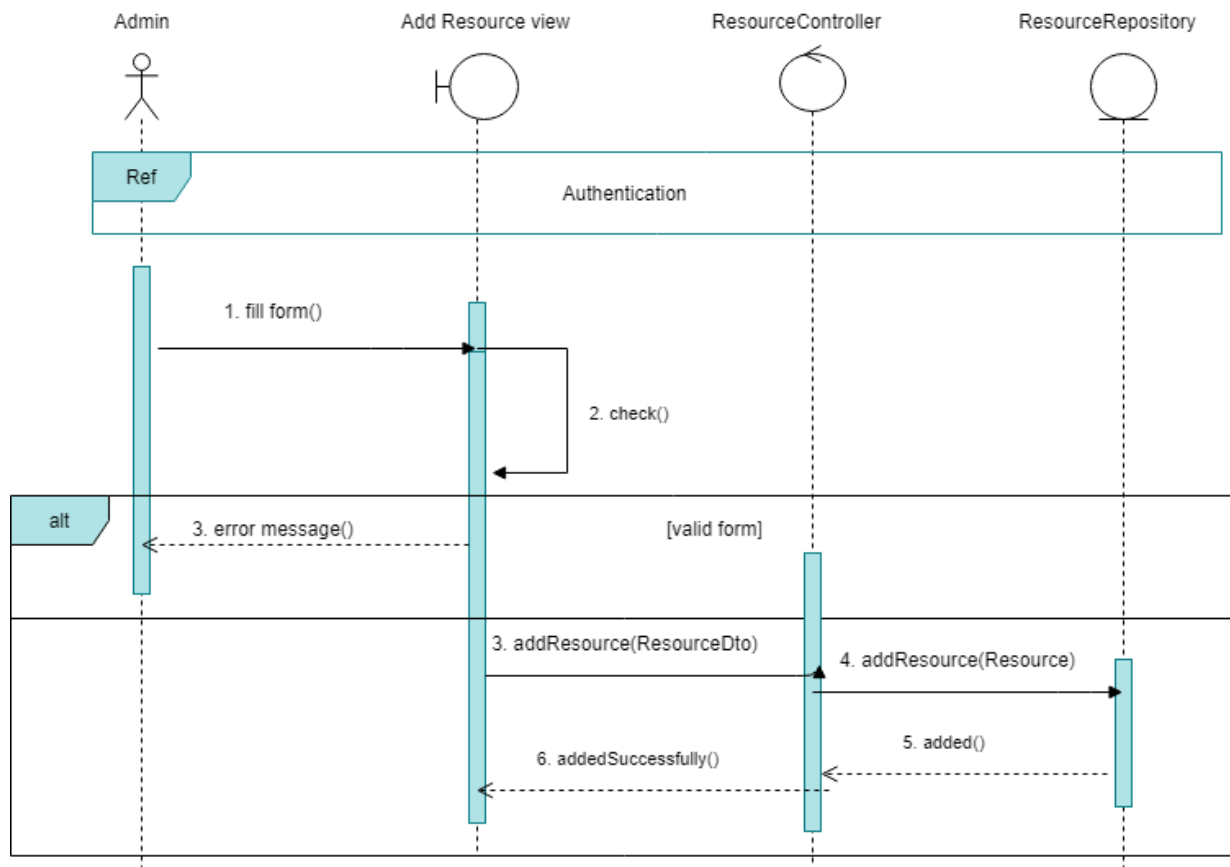


Figure 31 : Diagramme de séquence de cas d'utilisation "Ajouter une ressource"

2.4. Réalisation du sprint 2.1

Tout utilisateur veillant accéder à notre plateforme doit se connecter en introduisant son identifiant et son mot de passe. La figure 32 présente l'interface d'authentification.

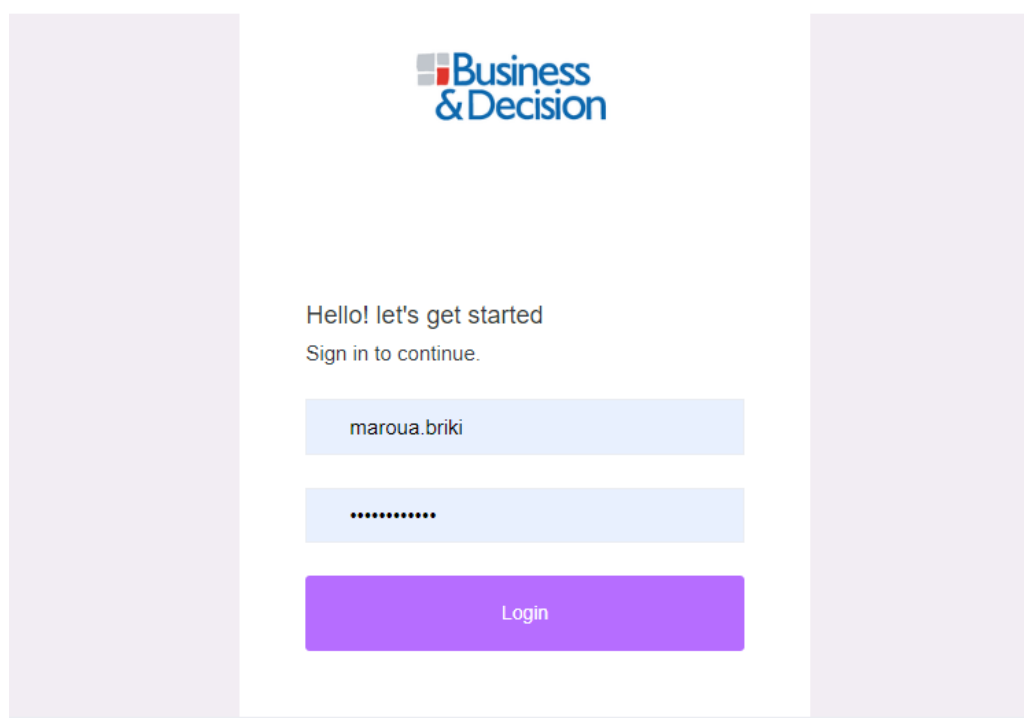


Figure 32 : Interface de login

Afin d'ajouter une nouvelle ressource l'administrateur doit remplir le formulaire illustré par la figure 33.

Figure 33 : Interface d'ajout d'une ressource

Chaque utilisateur de la plateforme peut consulter son compte et le mettre à jour via l'interface illustrée par la figure 34.

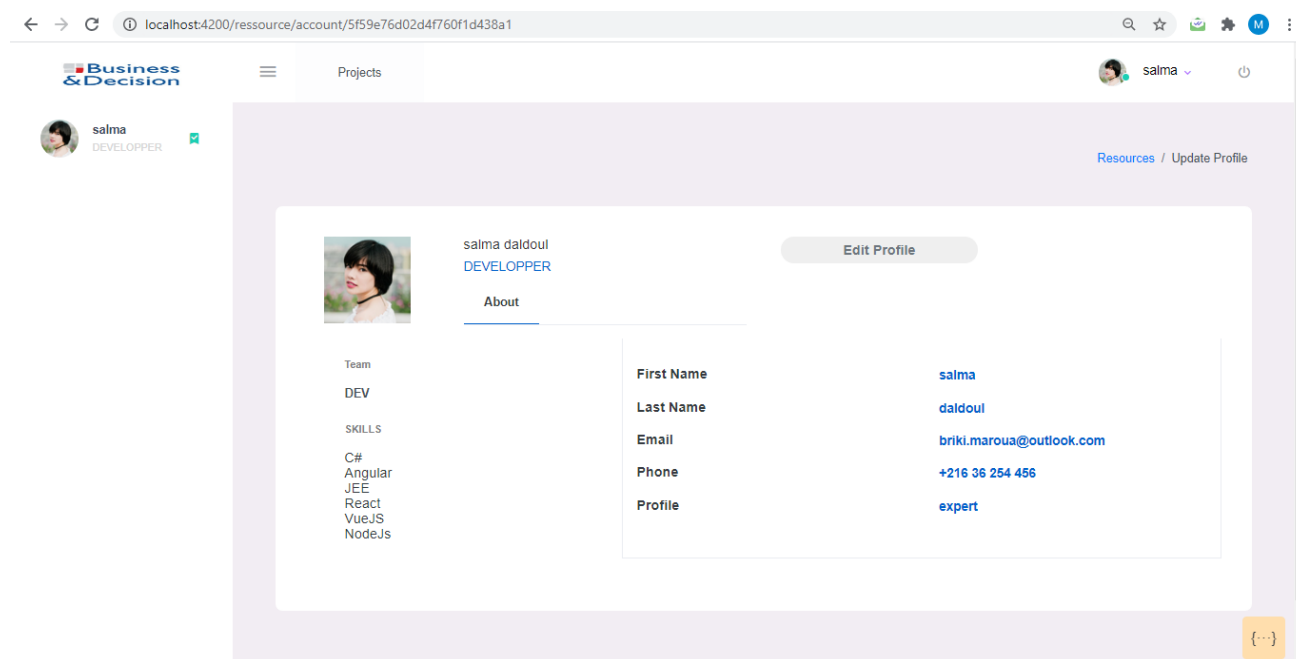


Figure 34 : Interface de mise à jour d'un compte

3. Sprint 2.2 Création du projet et construction de son équipe

Le sprint 2.2 a pour objectif de d'abord développer une interface pour la gestion d'un projet (consulter, ajouter, modifier et supprimer) ainsi que les API backend nécessaires, et ensuite construire une équipe projet. Le seul acteur de ce sprint est le chef de projet.

En premier lieu, nous préparerons le backlog du sprint et nous exposerons les besoins fonctionnels au moyen des diagrammes de cas d'utilisation. Par la suite, nous allons décrire la phase de conception via le diagramme de classes et ceux de séquence. Enfin, nous allons exposer quelques interfaces spécifiques à ce sprint.

3.1. Backlog de sprint 2.2

Avant de se lancer dans ce sprint, nous présentons les « uses stories » et les tâches à développer avec les estimations proposées. Le tableau 11 décrit le backlog du deuxième sprint du release 2.

Tableau 11 : Backlog du sprint 2.2

User Story	Tâches	Estimation
En tant que chef de projet, je peux créer, mettre à jour, supprimer ou filtrer un projet.	<p>[Backend] Implémentation du service d'ajout d'un projet.</p> <p>[Backend] Implémentation du service de modification d'un projet.</p> <p>[Backend] Implémentation du service de modification d'un projet.</p> <p>[Backend] Implémentation du service de suppression d'un projet</p> <p>[Frontend] Préparation de l'interface d'ajout et mise à jour de projet et intégration avec la partie métier.</p> <p>[Frontend] Préparation de l'interface d'affichage de tous les projets, le détail par projet et intégration avec la partie métier.</p>	5
En tant que chef de projet, je peux construire mon équipe projet.	<p>[Backend] Implémentation du service de filtrage les ressource par département.</p> <p>[Backend] Implémentation du service de filtrage des ressources par compétence.</p> <p>[Backend] Implémentation du service de vérification de disponibilité d'une ressource.</p> <p>[Backend] Implémentation du service d'invitation d'une ressource pour rejoindre un projet.</p> <p>[Backend] Implémentation du service de suppression d'un membre de projet.</p> <p>[Frontend] Réalisation d'interface d'invitation des membres pour le projet intégration avec la partie métier.</p> <p>[Frontend] Réalisation d'interface d'affichage de tous les membre d'un projet.</p>	15

3.2. Analyse des besoins du sprint 2.2

Dans cette partie, nous allons présenter le diagramme de cas d'utilisation et la description textuelle de quelques cas en basant sur les fonctionnalités décrites dans le backlog du tableau 11.

3.2.1. Diagramme de cas d'utilisation du sprint 2.2

La figure 35 présente le diagramme de cas d'utilisation global pour le deuxième sprint de notre deuxième release.

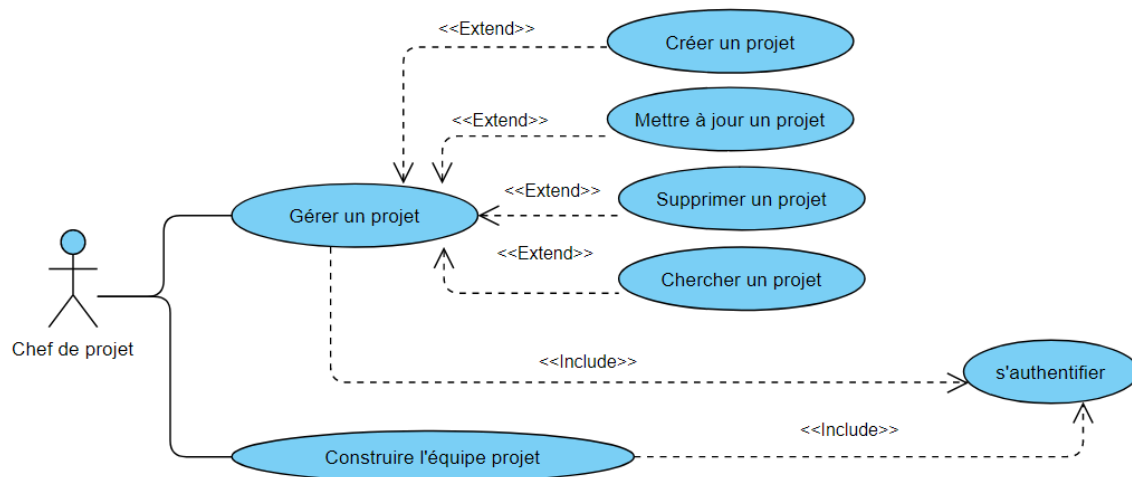


Figure 35 : Diagramme de cas d'utilisation du sprint 2.2

3.2.2. Raffinement des cas d'utilisation

Dans cette section, nous choisissons quelques cas d'utilisation à détailler par une description textuelle afin de mieux expliquer leurs scénarios.

▪ Cas d'utilisation « Créer un projet »

Le tableau 12 présente la description textuelle du cas d'utilisation « Créer un projet ».

Tableau 12 : Description textuelle du cas utilisation "Créer un projet"

Objectif	Ce cas d'utilisation permet au chef de projet de créer un projet.
Acteur	Chef de projet
Précondition	Utilisateur authentifié et rôle = chef de projet
Postcondition	Un projet créé
Scénario nominal	<ol style="list-style-type: none"> 5. Le chef de projet clique sur le bouton « New Project » 6. Le système affiche un formulaire 7. Le chef de projet saisit les champs demandés 8. Le système vérifie les champs et ajoute le projet 9. Le système redirige le chef de projet à sa liste de projet

Exceptions	Champs vides ou invalides
-------------------	---------------------------

▪ **Cas d'utilisation « Construire l'équipe projet »**

La figure 36 présente le raffinement de cas d'utilisation « Construire l'équipe projet ».

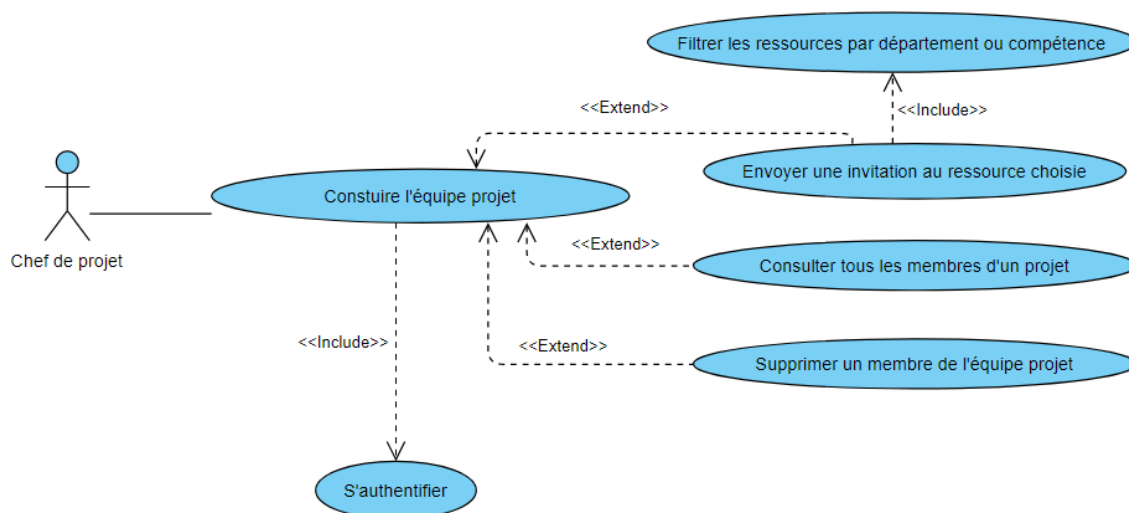


Figure 36 : Raffinement de cas d'utilisation "Construire l'équipe projet"

Le tableau 13 fournit les détails du cas d'utilisation "Construire l'équipe projet".

Tableau 13 : Description textuelle du cas d'utilisation "Construire l'équipe projet"

Objectif	Ce cas d'utilisation permet au chef de projet d'ajouter un nouveau membre à son équipe
Acteur	Chef de projet
Précondition	Utilisateur authentifié et rôle = chef de projet
Postcondition	Membre ajouté
Scénario nominal	<ol style="list-style-type: none"> 1. Le chef de projet filtre les ressources de la plateforme par département ou par compétence. 2. Le système affiche une liste des ressources filtrées avec la possibilité de filtrer encore par profil 3. Le chef de projet clique sur le bouton « inviter » 4. Le système vérifie si la ressource est disponible et renvoie un message de succès d'envoi de mail sinon il affiche un message décrivant l'état de la ressource.

	5. Le système redirige le chef de projet à la liste de l'équipe projet.
Exceptions	- Ressource n'est pas disponible c'est-à-dire déjà affectée sur un autre projet.

3.3. Conception du sprint 2.2

Au niveau de cette phase de conception, nous allons présenter la vue statique de notre système au moyen de diagramme de classes ainsi que la vue dynamique au moyen des diagrammes de séquence.

3.3.1. Diagramme de classe du sprint 2.2

Dans la figure 37, nous présentons le diagramme de classe associé à ce sprint.

La classe Projet a un identifiant de projet « idProject », un intitulé « topic », une date de début et date de fin « startDate, endDate ».

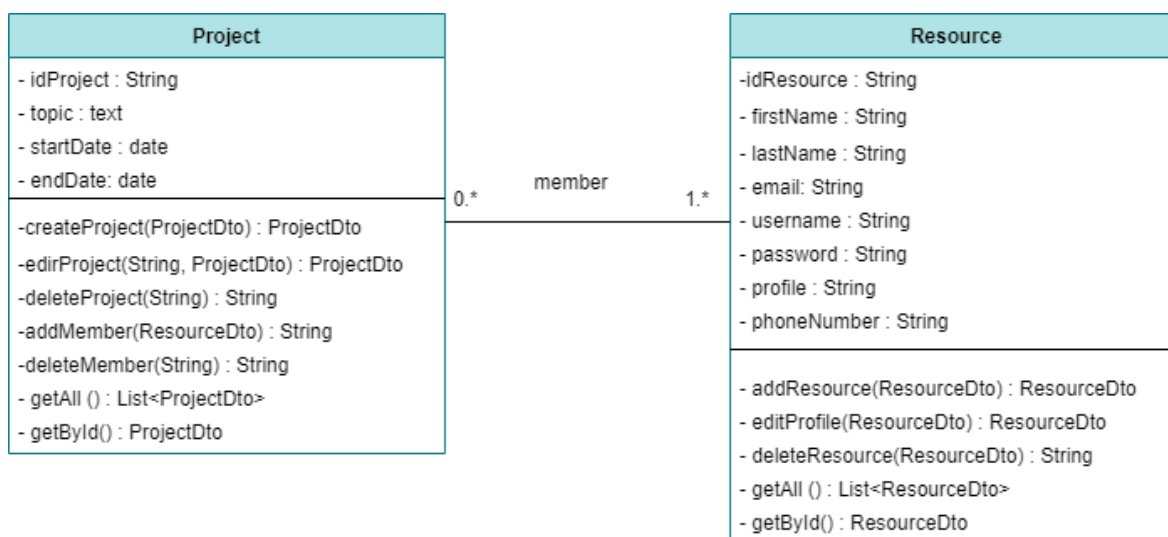


Figure 37: diagramme de classe de sprint 2.2

3.3.2. Diagrammes de séquence

Pour présenter la nature des interactions entre les différentes entités de notre système on a choisi de détailler deux diagrammes de séquence correspondants aux deux cas d'utilisation « supprimer un projet » et « inviter une ressource à rejoindre l'équipe projet » (Figure 38).

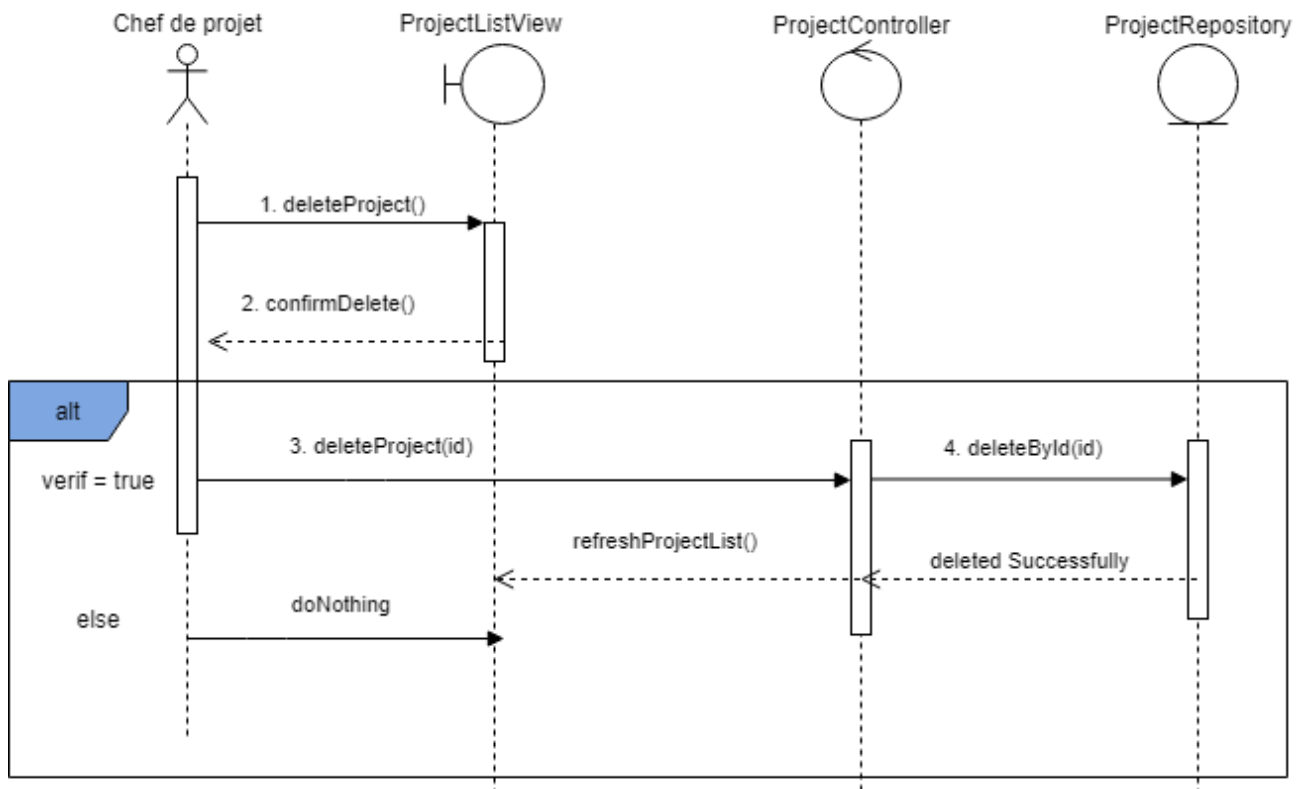


Figure 38 : Diagramme de séquence Supprimer un projet

Le cas d'utilisation « Inviter une ressource à rejoindre l'équipe projet » fait intervenir les trois microservices Resource-MS, ProjectManagement-MS et Contact-MS. Si le chef veut ajouter une nouvelle ressource à son projet, il va tout d'abord filtrer les ressources via le microservice Resource-MS selon le critère qu'il a besoin (« skill » qui présente une compétence ou bien par département par exemple son projet inclut des experts dans le département de l'apprentissage automatique "ML department" et des développeurs appartenant au pôle de développement "DEV department"). Puis le système vérifie dans le microservice ProjectManagement-MS si la ressource est réservée pendant la période du présent projet alors il affiche que la ressource n'est pas disponible, sinon une invitation par mail serait envoyée via le microservice Contact-MS à la ressource concernée. La figure 39 détaille ce cas d'utilisation.

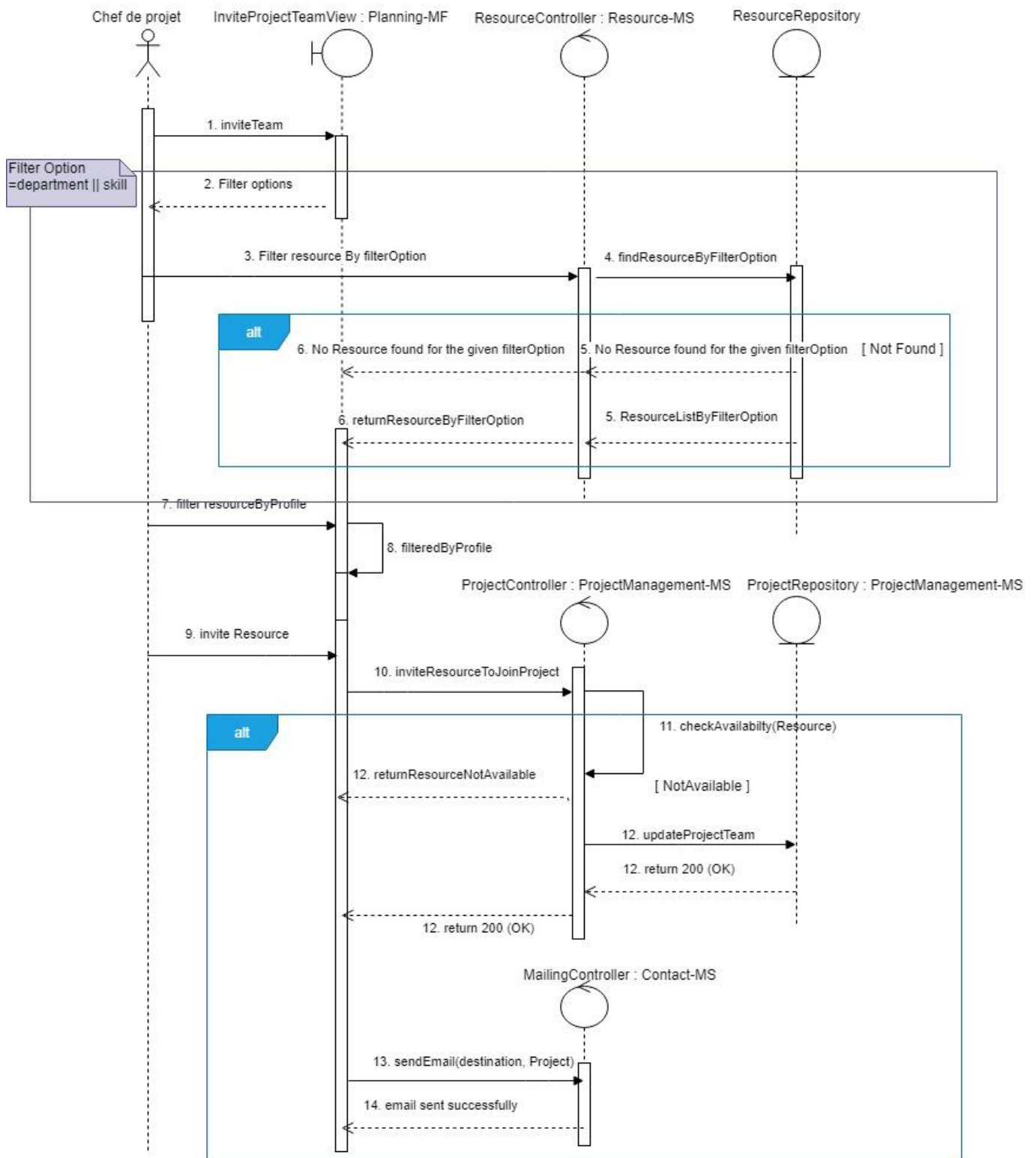


Figure 39 : Diagramme de séquence "Inviter un nouveau membre pour rejoindre l'équipe projet"

3.4. Réalisation du sprint 2.2

La figure 41 et la figure 42 présentent respectivement l'interface de création d'un projet et l'interface d'affichage de tous les projet appartenant à la ressource connectée dans notre cas c'est un chef de

projet qui est connecté donc le système lui affiche les projet créés par lui. Si la ressource connectée est un membre de projet il va trouver que les projets qu'il est y invité avec restriction des droits d'accès.

Business & Decision

Projects

anas
MANAGER

Projects / Create - Project

Create New Project

Fill The description below

Project Topic *

Mise en place d'une solution RH pour calculer les KPIs des collaborateurs

Start Date *

26/09/2020

End Date *

18/03/2021

Cancel Create

Figure 40 : Interface de création d'un projet

Business & Decision

Projects

anas
MANAGER

Planning / Projects

Projects

+ New Project

Filter By Project Title

Filter

Topic	Start Date	End Date	Actions
Mise en place d'une solution RH pour calculer les KPIs des collaborateurs	2020-09-26	2021-03-18	
Système de recommandation d'articles scientifiques	2021-01-02	2022-01-02	

Items per page: 5 0 of 0 |< < > >|

Figure 41 : Interface d'affichage de la liste des projets

Les figures 42 et 43 présente les étapes pour ajouter un nouveau membre à l'équipe projet.

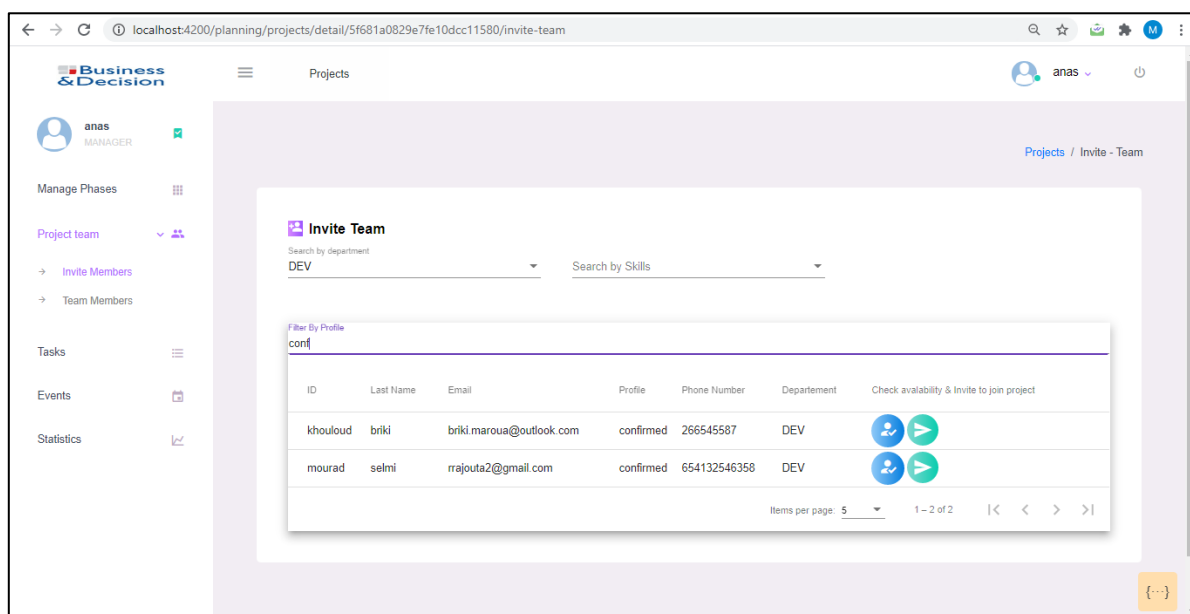


Figure 42 : Interface du filtre des ressources par département du microservice Resource-MS suivi d'un filtre par profil

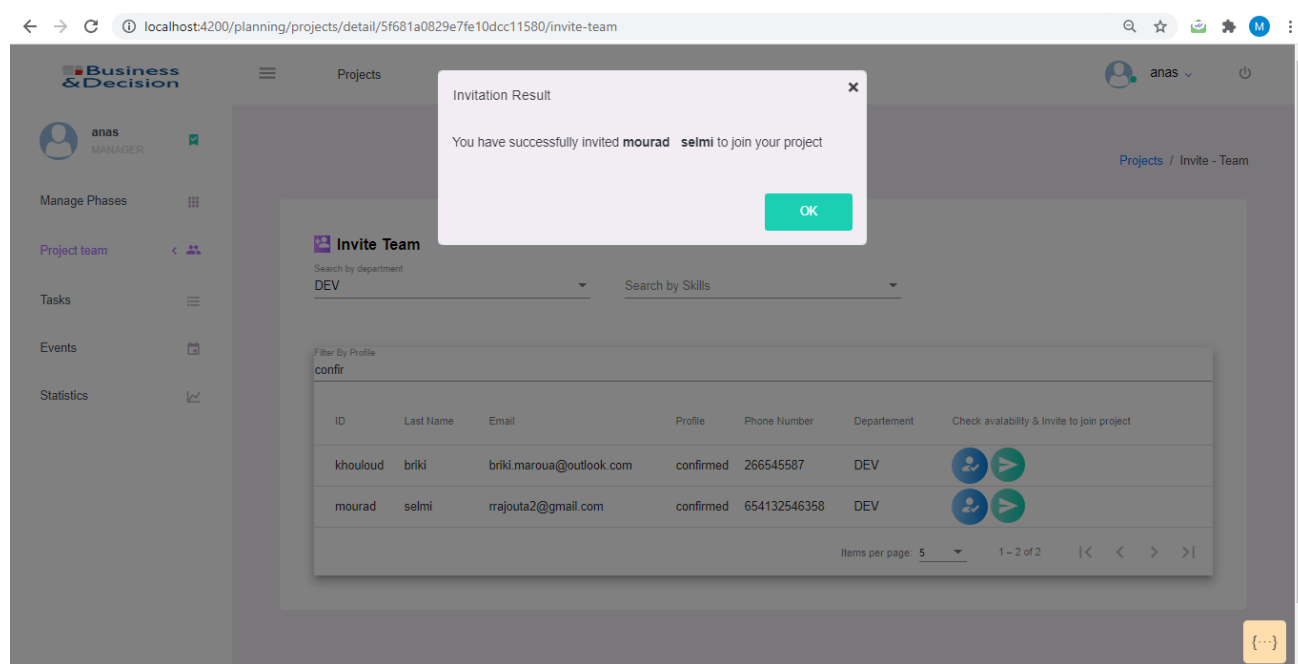


Figure 43 : Invitation envoyée

La figure 44 regroupe les membres inscrits dans un projet avec la possibilité de supprimer un membre et cette fonctionnalité concerne le chef de projet.

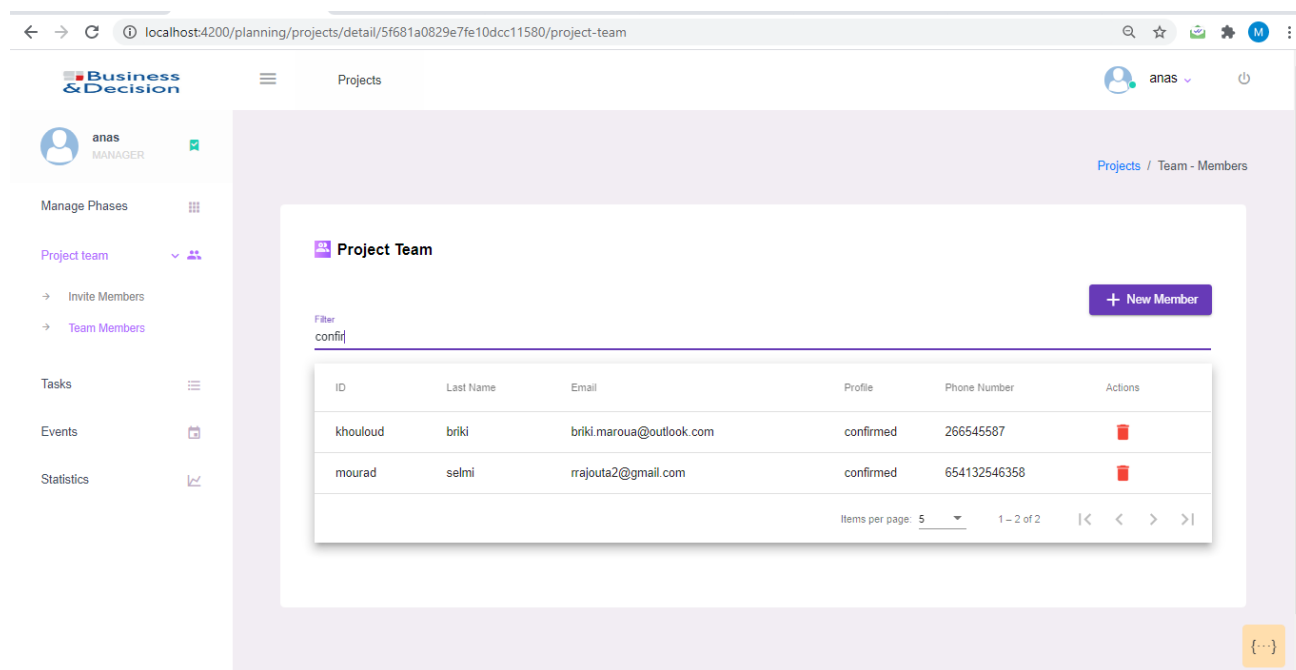


Figure 44 : Equipe projet

Conclusion

Au cours de ce chapitre, nous avons réussi à produire le premier livrable du notre projet qui comporte les sprints les plus prioritaires telles que « la gestion des ressources » et « la configuration d'un projet ». Dans le chapitre qui suit, notre effort sera consacré pour produire un nouveau livrable couvrant les sprints du troisième release.

Planification et suivi de projet

Introduction

Dans ce présent chapitre, nous exposons le travail réalisé lors du troisième release. Nous commençons par une phase de planification dans laquelle nous soulignons les différents sprints à réaliser. Ensuite, pour chaque sprint, nous procédons à la présentation du backlog produit, l'analyse des besoins, la conception et la réalisation.

1. Planification du release 3

La release 3 regroupe 3 sprints comme il est illustré dans le tableau 14.

Tableau 14 : Planification du release 3

Release 3		
Sprint 3.1	Sprint 3.2	Sprint 3.3
Gestion des phases De à	Task board et gestion des événements De à	Imputation et statistiques De à

2. Sprint 3.1 Gestion des phases

Le présent sprint sera consacré à la gestion des phases. Une liste de tâche sera attribuée à chaque phase. Pour amener bien le travail chaque tâche va être affectée à un membre de l'équipe projet.

2.1. Backlog de sprint 3.1

Nous représenterons les fonctionnalités à développer par le backlog de sprint illustré par le tableau 15.

Tableau 15 : Backlog de sprint 3.1

User Story	Tâches	Estimation
En tant que chef de projet, je peux gérer les phases	<p>[Backend] Implémentation du service d'ajout d'une phase.</p> <p>[Backend] Implémentation du service de modification d'une phase.</p> <p>[Backend] Implémentation du service de suppression d'une phase.</p> <p>[Frontend] Préparation de l'interface d'ajout et mise à jour d'une phase et intégration avec la partie métier.</p> <p>[Frontend] Préparation de l'interface d'affichage de toutes les phases et intégration avec la partie métier.</p>	5
En tant que chef de projet, je peux gérer les tâches	<p>[Backend] Implémentation du service d'ajout d'une tâche.</p> <p>[Backend] Implémentation du service de modification d'une tâche.</p> <p>[Backend] Implémentation du service de suppression d'une tâche.</p> <p>[Frontend] Préparation de l'interface d'ajout et mise à jour d'une tâche et intégration avec la partie métier.</p> <p>[Frontend] Préparation de l'interface d'affichage de toutes les tâches, de les exporter en pdf en cas de besoin et intégration avec la partie métier.</p>	10
En tant que chef de projet, je peux affecter les tâches à un membre de l'équipe projet	<p>[Backend] Implémentation du service d'affectation d'une tâche.</p> <p>[Frontend] Préparation de l'interface d'affectation des tâches et intégration avec la partie métier.</p>	5
En tant que membre de l'équipe je peux modifier l'état d'une tâche qui m'a été affecté	<p>[Backend] Implémentation du service de modification de l'état d'une tâche.</p> <p>[Frontend] Préparation de l'interface de drag & drop des tâches.</p>	4

2.2. Analyse des besoins du sprint 3.1

Pour détailler les fonctionnalités du sprint 3.1 nous avons recours au diagramme de cas d'utilisation détaillé ainsi que des descriptions textuelles détaillées pour quelques cas d'utilisation.

2.2.1. Diagramme de cas d'utilisation du sprint 2.2

La figure 45 présente le diagramme de cas d'utilisation global pour le deuxième sprint de notre deuxième release.

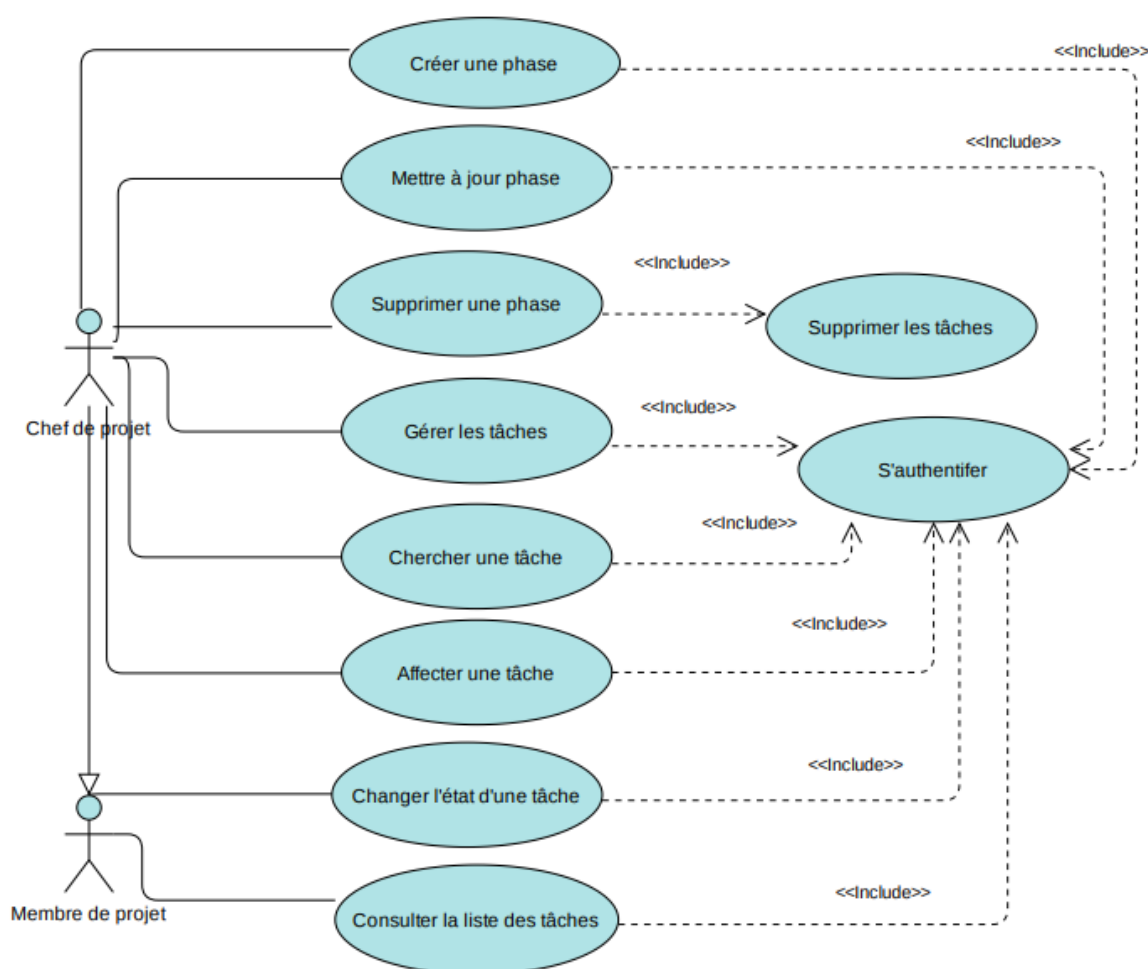


Figure 45 : Diagramme de cas d'utilisation du sprint 3.1

2.2.2. Raffinement des cas d'utilisation

Dans ce qui suit nous procédons au raffinement de quelques cas d'utilisation.

▪ **Cas d'utilisation « Mettre à jour une phase »**

Le tableau 16 présente la description textuelle du cas d'utilisation « Mettre à jour une phase ».

Tableau 16 : Description textuelle du cas utilisation " Mettre à jour une phase "

Acteur	Chef de projet
Précondition	Utilisateur authentifié et rôle = chef de projet
Postcondition	Une phase mise à jour
Scénario nominal	<ol style="list-style-type: none"> 1. Le chef de projet clique sur le bouton de mise à jour dans le tableau qui affiche la liste des phases 2. Le système récupère les données de la phase et les affiche dans le formulaire 3. Le chef de projet met à jour les champs qu'il veut changer 4. Le système vérifie les champs et met à jour la phase 5. Le système redirige le chef de projet à sa liste de des phases
Exceptions	Champs vides ou invalides

▪ **Cas d'utilisation « Supprimer une phase »**

Le tableau 17 présente nous montre les détails du cas d'utilisation « Supprimer une phase ».

Tableau 17 : Description textuelle du cas d'utilisation "Supprimer une phase"

Objectif	Ce cas d'utilisation permet au chef de projet de supprimer une phase d'un projet
Acteur	Chef de projet
Précondition	Utilisateur authentifié et rôle = chef de projet
Postcondition	Phase supprimée
Scénario nominal	<ol style="list-style-type: none"> 1. Le chef de projet clique sur le bouton de suppression dans le tableau qui affiche la liste des phases 2. Le système indique au chef de projet que la suppression de cette phase va engendrer la perte de toutes les tâches qu'y appartiennent. 3. Le chef de projet confirme sa demande de suppression ou l'annule

	<ol style="list-style-type: none"> 4. Si le système reçoit une confirmation de suppression il supprime la phase ainsi que toutes ses tâches 5. Le système rafraîchit la liste des phases en retirant la phase supprimée.
--	--

▪ **Cas d'utilisation « Modifier l'état d'une tâche »**

Le tableau 18 détaille la description du cas d'utilisation « Modifier l'état d'une tâche ».

Tableau 18 : Description textuelle du cas utilisation "Modifier l'état d'une tâche"

Objectif	Ce cas d'utilisation permet au membre de projet de modifier l'état d'une tâche qui lui a été affectée
Acteur	Membre de projet
Précondition	Utilisateur authentifié et rôle = Membre de projet
Postcondition	Etat d'une tâche modifié
Scénario nominal	<ol style="list-style-type: none"> 1. Le membre de projet clique sur le bouton « TaskBoard » 2. Le système affiche le task board du membre de projet qui contient les tâches déjà affecté classées selon leur ancien état « TO DO, IN PROGRESS, DONE » 3. Le membre de projet glisse une tâche d'un état à un autre 4. Le système change l'état de la phase.

2.3. Conception du sprint 3.1

Au niveau de cette phase de conception, nous allons présenter la vue statique de notre système au moyen de diagramme de classes ainsi que la vue dynamique au moyen des diagrammes de séquence.

2.3.1. Diagramme de classe du sprint 3.1

La figure 46 décrit le diagramme de classe du sprint 3.1. Nous avons une classe Phase désignée par "Phase", Tâche désignée par "Task ", Chef de projet désigné par "ProjectManager" et le membre de

projet designé par "ProjectMember ". Une phase est composée par un ensemble de tâches et chaque tâche est affectée à un membre de projet.

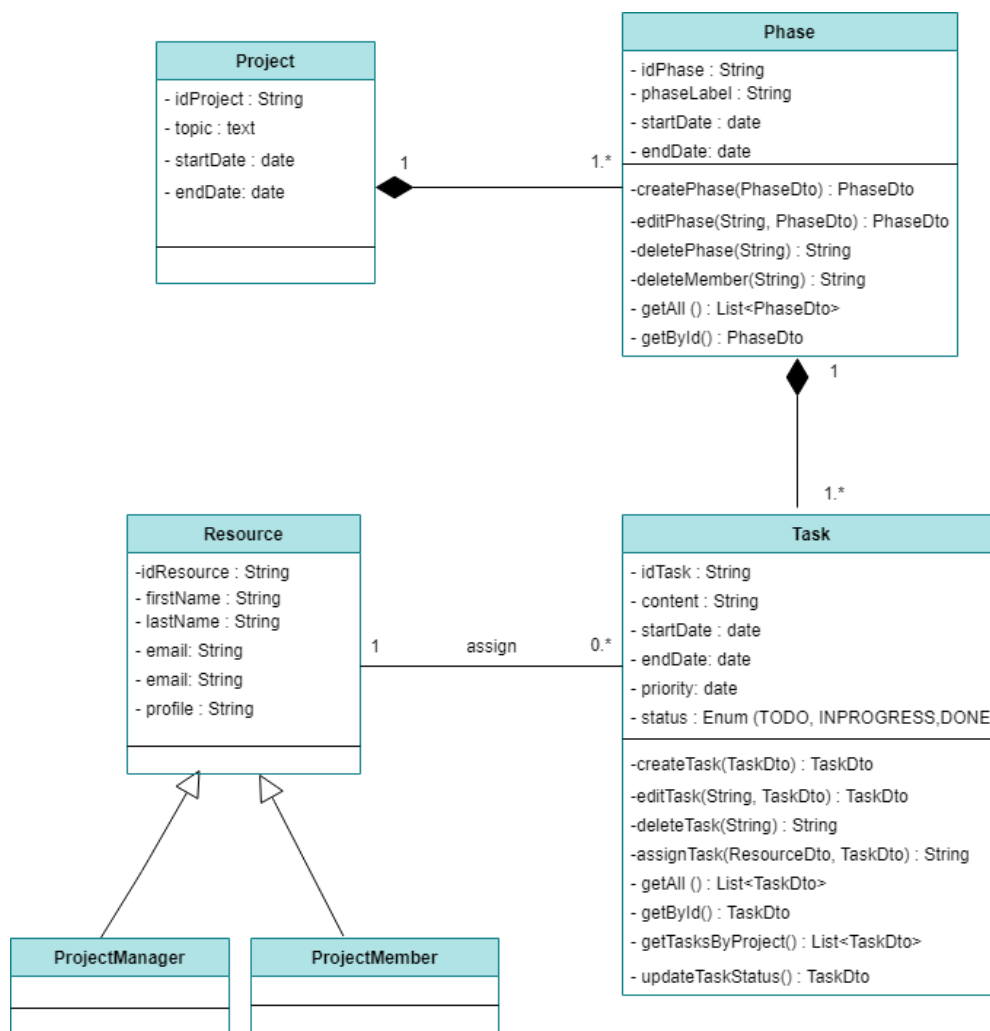


Figure 46 : Diagramme de classe de sprint 3.1

2.3.2. Diagramme de séquence

Le diagramme de séquence présenté par la figure 44 décrit l'interaction entre le chef de projet et le système lors de l'affectation d'une tâche à un membre de l'équipe projet.

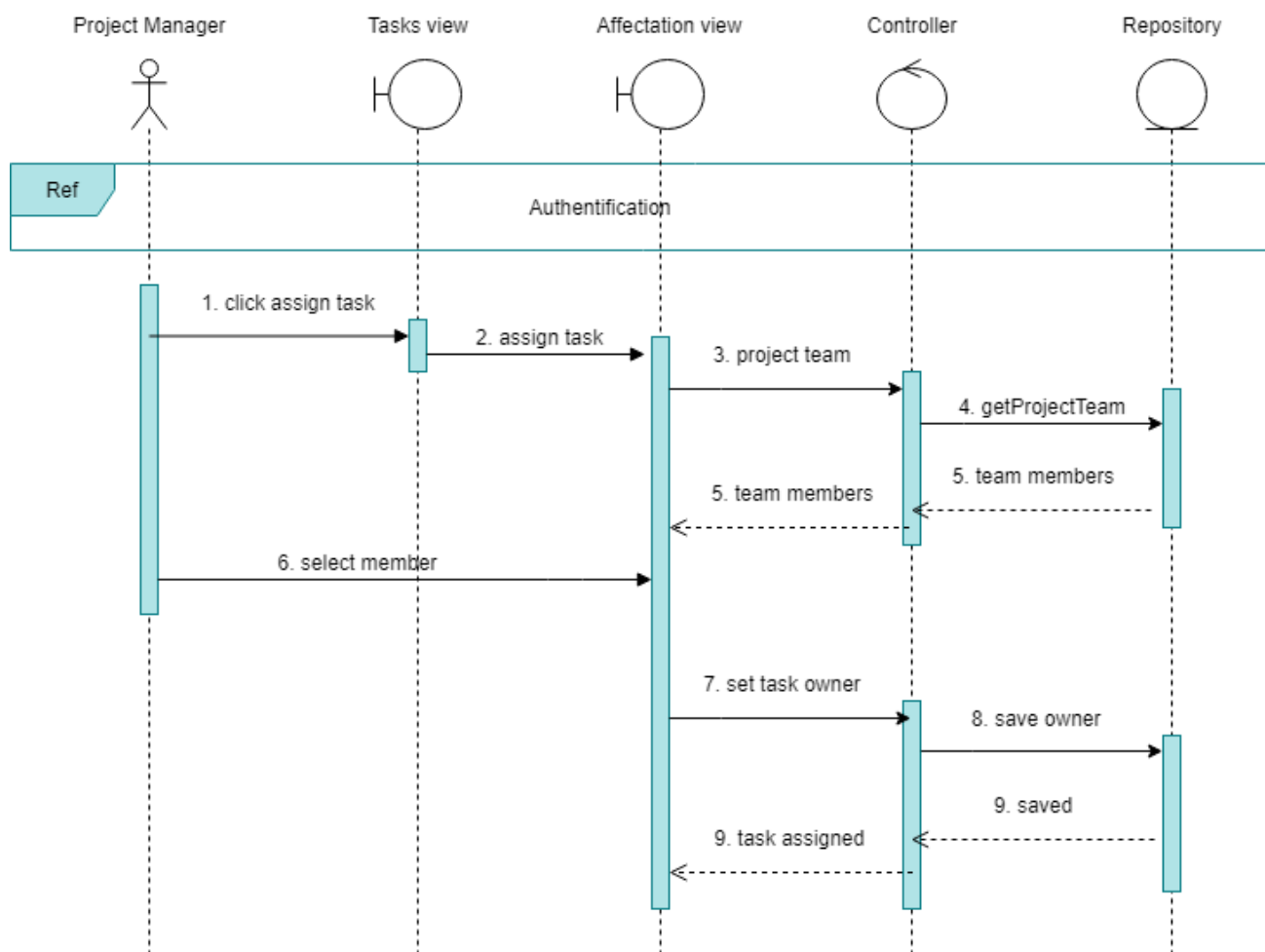


Figure 47 : Diagramme de séquence "Affecter une tâche"

2.4. Réalisation du sprint 3.2

Pour créer une nouvelle phase le chef de projet remplit les champs du formulaire fournit dans la figure 48.

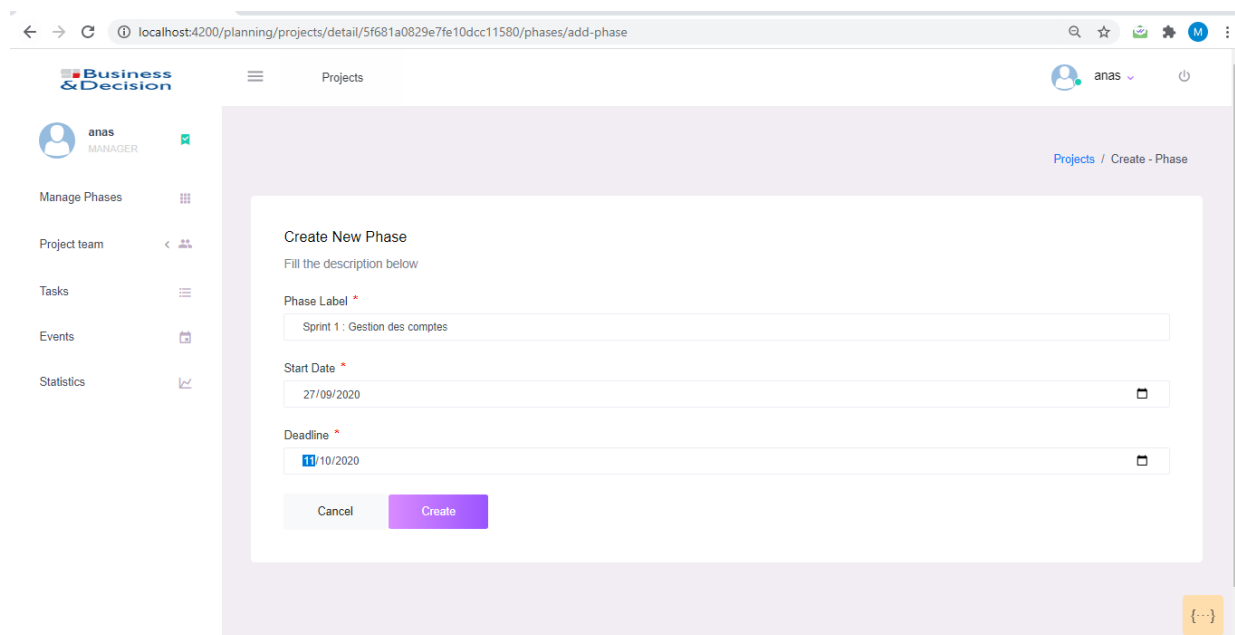


Figure 48 : Interface d'ajout d'une phase

Le chef de projet consulte les phases créées dans son projet et il peut ajouter des tâches pour chaque phase. Une tâche peut être modifiée, supprimée ou affectée à un membre de projet comme il est illustré dans la figure 49.

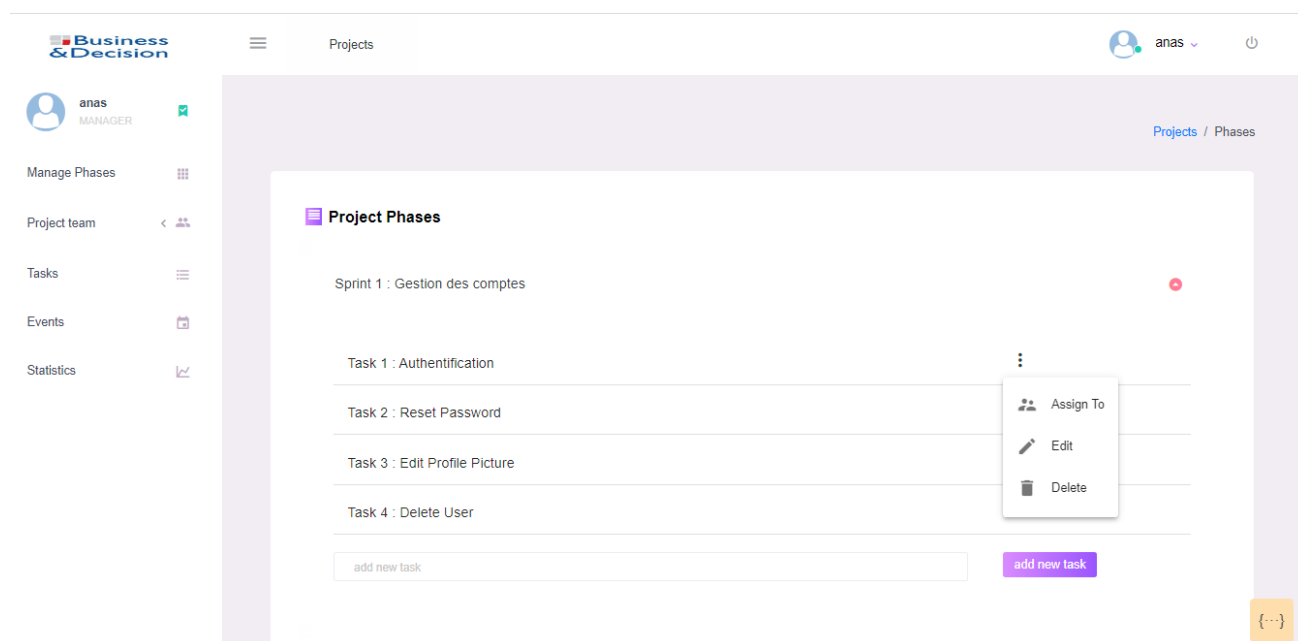


Figure 49 : Interface d'affichage des phases avec leurs tâches

Lorsque le chef de projet procède pour supprimer une phase le système lui renvoie une alerte car cette action va engendrer la suppression de toutes les tâches qui y sont incluses ceci est tangible via la figure 50.

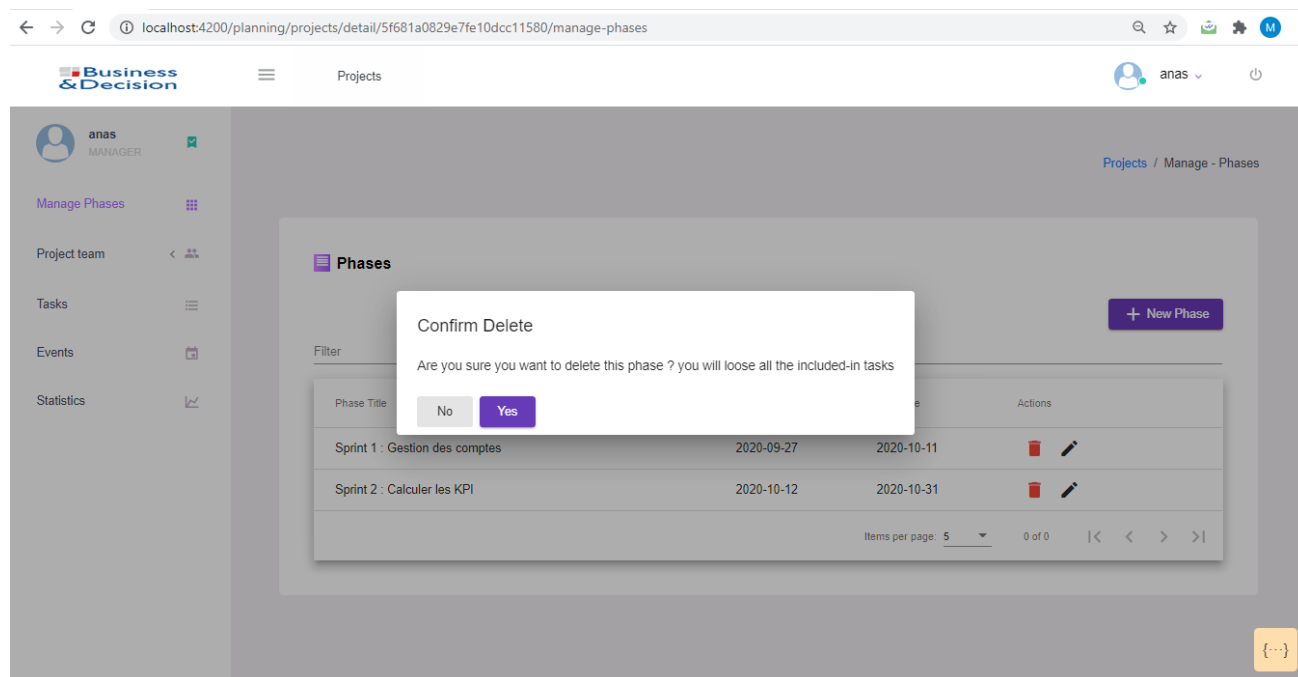


Figure 50 : Interface de suppression d'une phase

En premier lieu lors de la création d'une nouvelle tâche, le chef de projet indique le contenu de cette tâche, en deuxième lieu il peut la modifier en saisissant les autres champs comme la date de début et fin, la priorité et le statut (Figure 51).

Figure 51 : Interface de mise à jour d'une phase

Pour affecter une tâche le chef de projet est redirigé à une interface qui contient la liste des membres de projet (Figure 52).

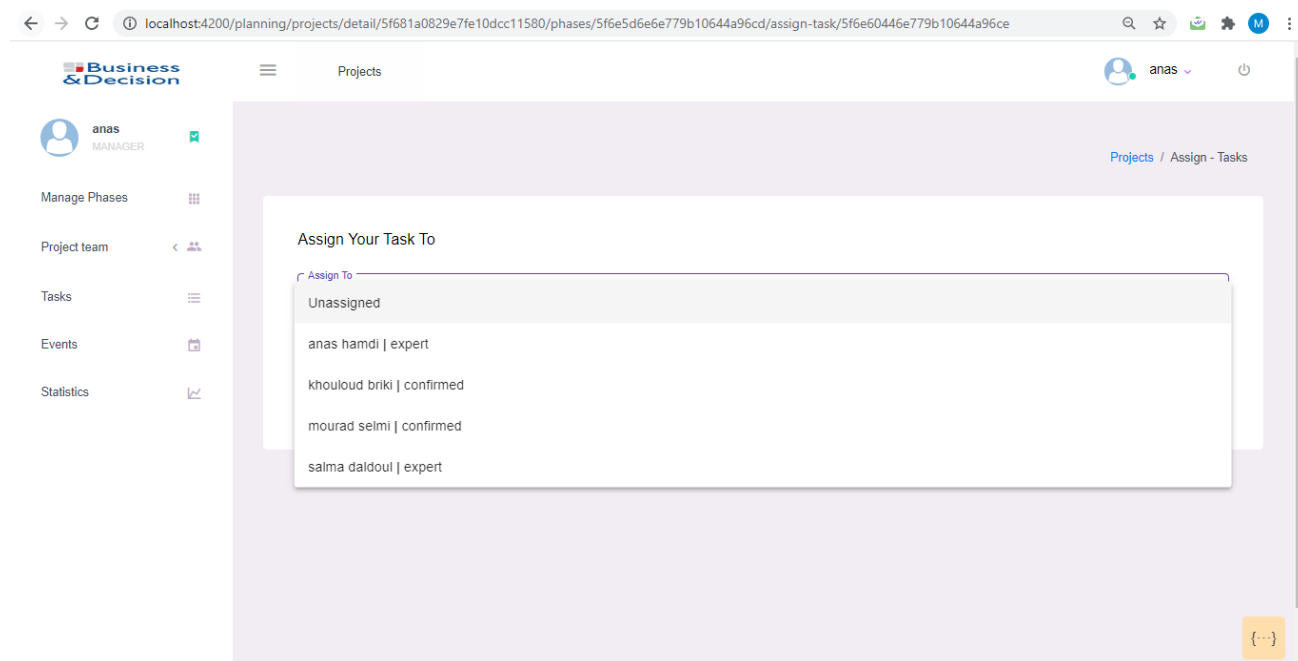


Figure 52 : Interface d'affectation d'une tâche à un membre de l'équipe

Chaque membre connecté, ayant des tâches, peut accéder à son « task-board » pour changer l'état d'une tâche en la glissant d'une liste à une autre (Figure 53).

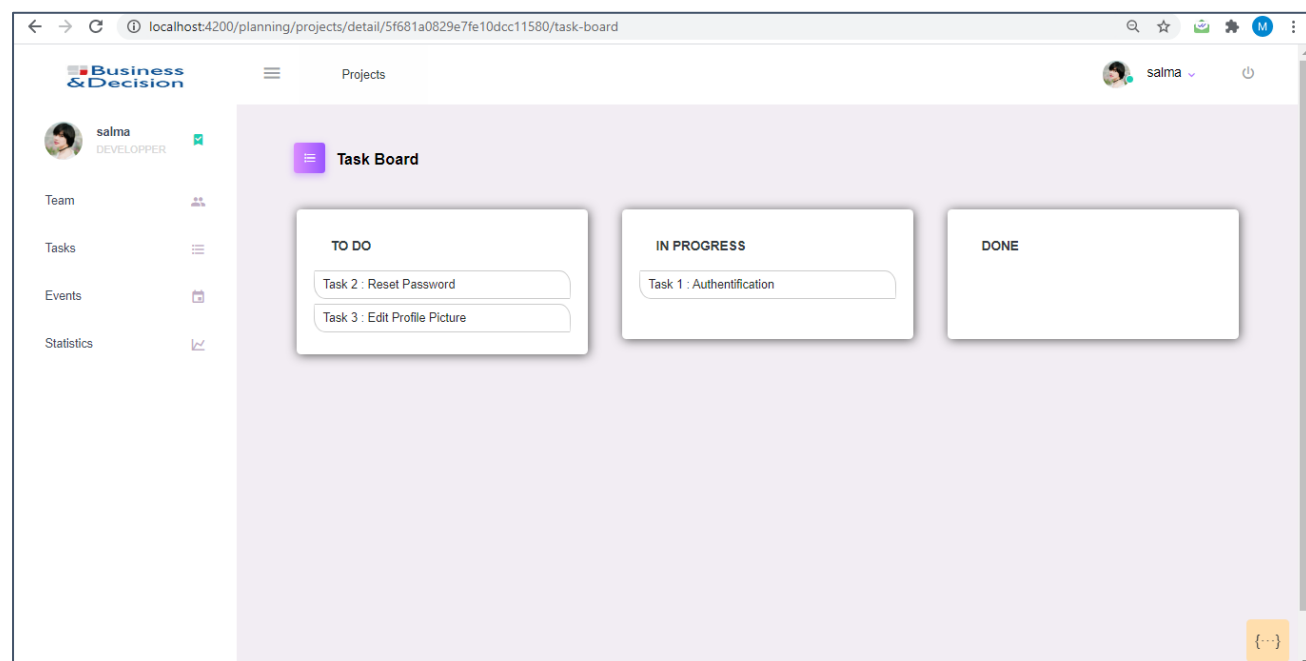


Figure 53 : Task Board pour un membre de projet

3. Sprint 3.2 Gestion des tickets et des événements

Le sprint 3.2 se focalise sur la gestion des tickets et des événements par projet.

Un ticket constitue un bug ou un incident identifié par un membre lors de la réalisation d'une tâche.

Un événement représente une réunion ou une livraison survenue au cours d'un projet.

3.1. Backlog de sprint 3.2

Dans le tableau 19, nous présentons le backlog de produit de sprint 3.2.

Tableau 19: Backlog de produit de sprint 3.2

User Story	Tâches	Estimation
En tant que membre de l'équipe projet je peux créer un ticket et inviter le chef de projet à intervenir.	[Backend] Implémentation du service d'ajout d'un ticket. [Backend] Implémentation du service d'invitation du membre de projet pour intervenir sur le ticket. [Frontend] Préparation de l'interface d'ajout d'une ticket et intégration avec la partie métier.	5
En tant que chef de projet, je peux gérer les événements	[Backend] Implémentation du service d'ajout d'un événement. [Backend] Implémentation du service de modification d'un événement. [Frontend] Préparation de l'interface d'ajout et mise à jour d'un événement et intégration avec la partie métier.	10
En tant que chef de projet ou un membre, je peux consulter un calendrier représentant les événements	[Backend] Implémentation du service récupération des événements par projet [Frontend] Préparation du calendrier et affichage des événements.	5

3.2. Analyse des besoins du sprint 3.2

Dans cette phase, nous décrivons les fonctionnalités de notre sprint d'une manière formelle grâce au diagramme de cas d'utilisation. Ensuite, nous expliquons ces derniers de manière textuelle.

3.2.1. Diagramme de cas d'utilisation du sprint 3.2.

Le diagramme de cas d'utilisation, illustré par la figure 50, représente les fonctionnalités à concevoir et à développer dans ce sprint.

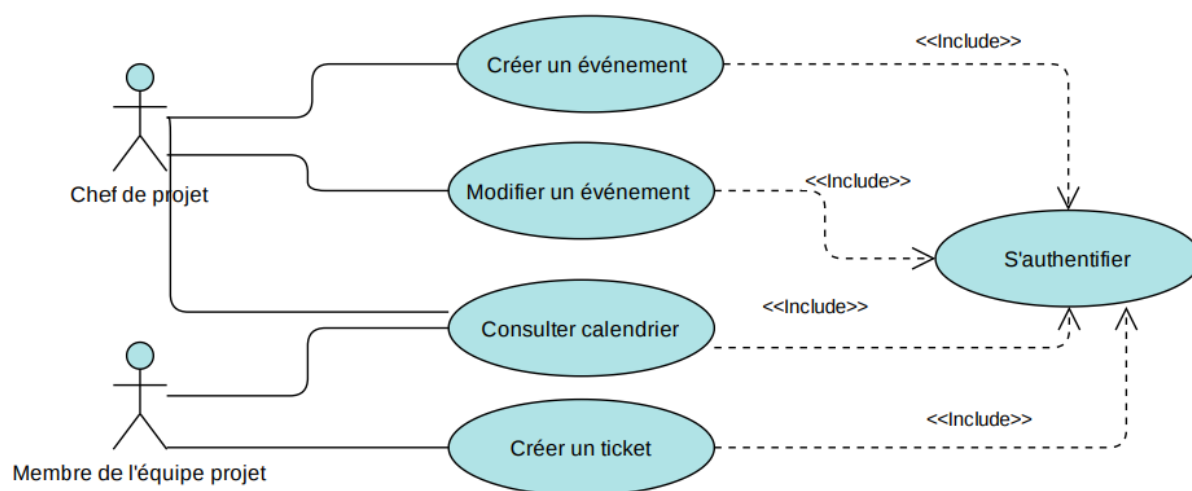


Figure 54: Diagramme de classe de sprint 3.2

3.2.2. Raffinement des cas d'utilisation

Dans ce qui suit nous procédons au raffinement de quelques cas d'utilisation.

- **Cas d'utilisation « Créer un événement »**

Le tableau 20 présente la description textuelle du cas d'utilisation « Mettre à jour une phase ».

Tableau 20 : Description textuelle du cas d'utilisation "Créer évènement"

Objectif	Ce cas d'utilisation permet au chef de projet de créer un nouvel événement
Acteur	Chef de projet
Précondition	Utilisateur authentifié et rôle = chef de projet
Postcondition	Un événement créé
Scénario nominal	6. Le chef de projet clique sur le bouton de « New event » 7. Le système affiche le formulaire de création d'un événement 8. Le chef de projet remplit les champs demandés 9. Le système redirige le chef de projet à la liste des événements
Exceptions	Evènement ayant la même date

▪ **Cas d'utilisation « Créer un ticket »**

Le tableau 21 présente décrit le cas d'utilisation « Créer un ticket ».

Tableau 21 : Description textuelle du cas d'utilisation " Créer un ticket "

Objectif	Ce cas d'utilisation permet au membre de projet de créer un ticket
Acteur	Membre de projet
Précondition	Utilisateur authetifié et rôle = membre de projet
Postcondition	Ticket créée
Scénario nominal	<ol style="list-style-type: none"> 1. Le membre de projet clique sur le bouton « add ticket » 2. Le système affiche un formulaire à remplir 3. Le membre de projet remplit le formulaire et clique sur send 4. Le système envoie une invitation au chef de projet pour qu'il intervient sur le ticket.

3.3. Diagramme de classe de sprint 3.2

Le diagramme de classe du sprint 3.2 illustré par la figure 55 contient les deux classes qui feront l'objet de travail de ce sprint (la classe « Event » désignant un événement et la classe « Ticket » qui sera communiquée au responsable de projet afin qu'il résolve la problématique).

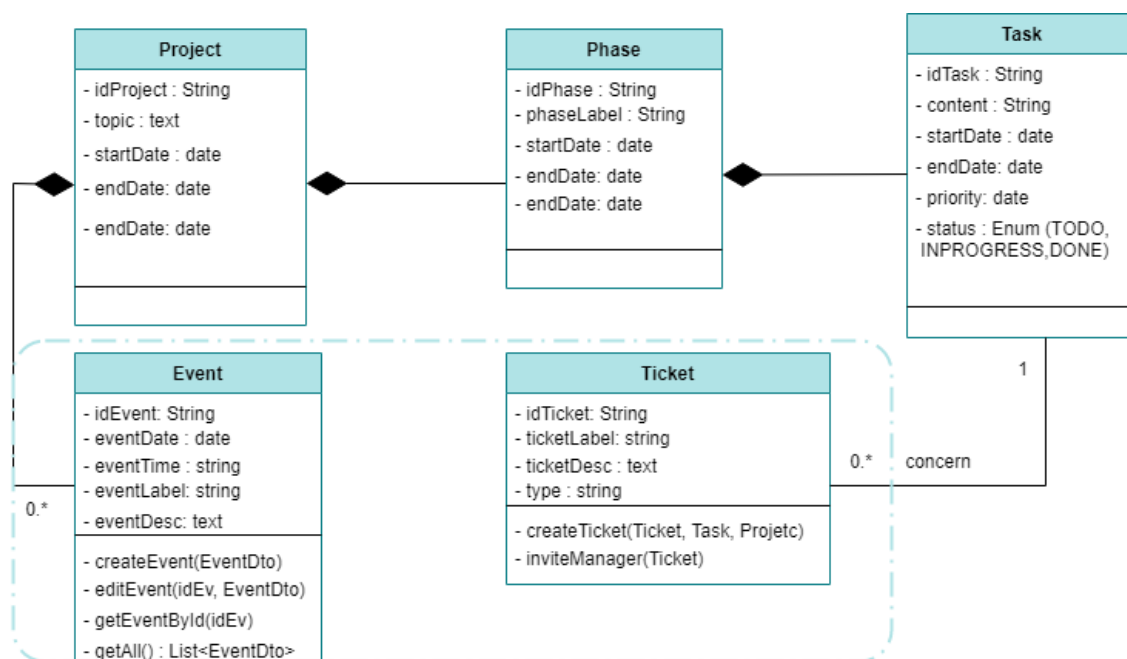


Figure 55 : Diagramme de classe de sprint 3.2

3.4. Réalisation du sprint 3.2

La figure 56 présente le calendrier des événements qui est accessible à chaque membre de projet pour qu'il puisse suivre les différentes réunions survenues au cours de son projet.

September 2020						
Previous	Today	Next				
Month	Week	Day				
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3

Figure 56 : Calendrier des évènements

4. Sprint 3.3 : Imputation et statistiques

4.1. Backlog de sprint 3.3

User Story	Tâches	Estimation
En tant que membre de l'équipe ou le chef le projet je peux consulter le dashboard de statistiques	[Backend] Implémentation du service de récupération de l'équipe projet. [Frontend] Implémentation des fonctions de classifications des tâches [Frontend]	4
En tant chef de projets ou membre de l'équipe je peux exporter les tâches ou les ressources en format pdf	[Frontend] Implémentation de la fonctionnalité export pdf pour les ressources et les tâches	2
En tant que membre de l'équipe je peux saisir les imputations effectuées pour mes tâches	[Backend] Implémentation de la communication inter-microservice via openFeign [Backend] Implémentation des apis de saisie de l'imputation	7

4.2. Analyse des besoins de sprint 3.3

4.2.1. Diagramme de cas d'utilisation de sprint 3.3

4.2.2. Raffinement des cas d'utilisation de sprint 3.3

4.3. Réalisation de sprint 3.3

Chaque membre du projet peut accéder à l'interface des statistiques (Figure 51) où il peut savoir la composition en profil de son équipe (1), le nombre de phases dans le projet ainsi que la phase en cours (2), le nombre total des tâches accompagné du nombre de tâches déjà validées (3) et la classification des tâches selon l'état (4) [le pourcentage d'un état est égal au nombre de tâches possédant cet état multiplié par 100 et divisé par le nombre total des tâches].

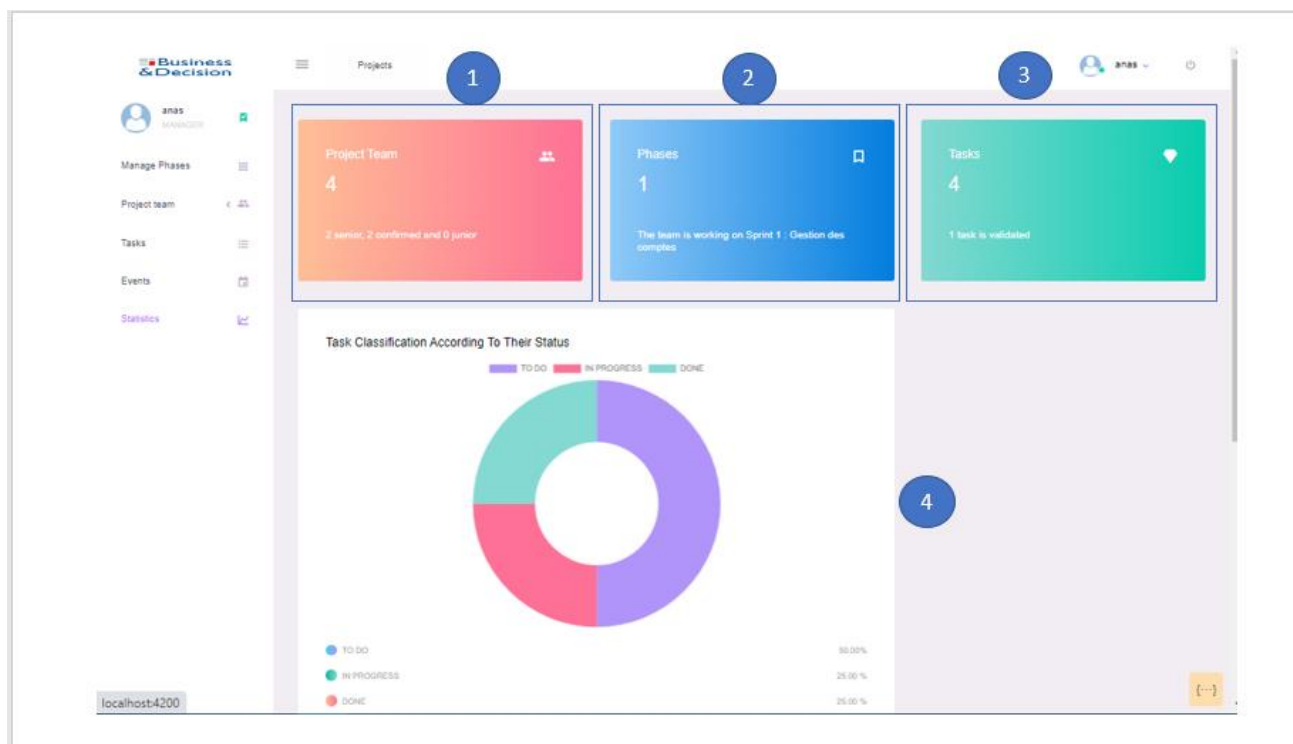


Figure 57 : Dashboard de statistiques

Le membre de projet peut également exporter toutes les tâches d'un projet en format PDF en cas de besoin (Figure58).

Project Tasks					PDF
Assignee	Content	Priority	Status	End Date	
salma daidoul	Task 1 : Authentification	HIGH	INPROGRESS	2020-10-13	
UNASSIGNED	Task 2 : Reset Password	MEDIUM	TODO		
UNASSIGNED	Task 3 : Edit Profile Picture	LOW	TODO		
UNASSIGNED	Task 4 : Delete User	LOW	DONE		

Figure 58 : Export PDF des tâches

Conclusion

Au cours de cette dernière release, nous avons détaillé les trois sprints 3.1, 3.2 et 3.3, en les expliquant par un backlog de sprints ainsi qu'une conception détaillée. Ensuite, nous avons exposé les différentes interfaces développées. Dans ce qui suit, nous présenterons une conclusion générale qui récapitule la démarche que nous avons effectuée tout au long de notre stage.

Conclusion Générale

Notre projet de fin d'études, présenté dans ce rapport, avait pour objet de réaliser une plateforme de gestion de projets qui permet à Business & Decision Tunisie de suivre la planification de ses projets.

Pour atteindre nos objectifs, nous avons commencé par mettre le projet dans son contexte général. Ensuite, nous sommes passés à une étude et critique de l'existant pour mieux cerner nos objectifs. Dans l'étape suivante, nous avons entamé la partie analyse des besoins et la planification des releases. Suivie par la réalisation de ces derniers qui s'est étalée sur les 3 derniers chapitres dont chacun contient une étude et analyse des besoins, conception et réalisation.

Ce stage nous a été très constructif, nous avons eu la chance de travailler sur des nouveaux styles architecturaux qui présentent le futur de développement logiciel qui suivent le principe « diviser pour régner » ce sont les architectures microservices et microfrontends. De même, nous avons pu mettre en pratique nos connaissances théoriques acquises tout au long de notre formation à l'École Nationale Supérieure d'Ingénieurs de Tunis et approfondir davantage nos compétences techniques en suivant les bonnes pratiques proposées par nos encadrants au sein de Business & Decision ainsi qu'en communication et la prise de l'initiative. Notre projet est donc une source d'enrichissement technique, culturel, personnel et humain.

Néanmoins, nous tenons à présent à souligner quelques extensions ou perspectives intéressantes de notre projet. Grâce à son caractère extensible et sa modularité, notre application pourra être améliorée en ajoutant d'autres fonctionnalités citant par exemple, l'implémentation d'un espace collaboratif disposant d'une boîte de messagerie interne à chaque équipe projet et un espace de partage de documents et connaissances, l'intégration du système des notifications en temps réel afin de notifier le chef de projet s'il y a un dépassement des coûts, ajout des outils de test automatique et finalement nous avons prototyper comment transformer nos microservices à des images docker donc notre étape suivante sera de procéder pour déployer notre plateforme.

Webographie

- [1] Business & Decision Group <https://www.businessdecision.fr/> [consulté le (14/02/2020)]
- [2] Architecture microservices vs monolithique, accessible sur <https://www.lemagit.fr/conseil/Architecture-monolithique-vs-microservices-avantages-et-inconvenients> [consulté le (14/02/2020)]
- [3] Architecture microservices vs monolithique, accessible sur <https://medium.com/@raycad.seedotech/monolithic-vs-microservice-architecture-e74bd951fc14> [consulté le (20/04/2020)]
- [5] Documentation officielle microfrontends, accessible sur <https://micro-frontends.org/> [consulté le (22/04/2020)]
- [6] Outil de gestion de projet, accessible sur <https://www.appvizer.fr/magazine/operations/gestion-de-projet/gestion-de-projet-gratuit> [consulté le (23/04/2020)]
- [7] Manifeste Agile, accessible sur <https://www.ideematic.com/actualites/2015/01/methodes-agiles-definition/#:~:text=La%20m%C3%A9thode%20agile%20nomm%C3%A9e%20Manifeste,plut%C3%B4t%20que%20la%20n%C3%A9gociation%20contractuelle.> [consulté le (03/03/2020)]
- [9] IntelliJ, accessible sur <https://www.jetbrains.com/fr-fr/idea/> [consulté le (14/09/2020)]
- [10] Visual studio code, accessible sur <https://code.visualstudio.com/Docs> [consulté le (14/09/2020)]
- [11] Postman, accessible sur <https://riptutorial.com/Download/postman-fr.pdf> [consulté le (20/09/2020)]
- [12] GitLab, accessible sur <https://blog.axopen.com/2017/02/gitlab-cest-quoi/> [consulté le (20/09/2020)]
- [13] Mongo compass, accessible sur <https://docs.mongodb.com/compass/master/> [consulté le (20/09/2020)]
- [14] Draw.io, accessible sur <https://drawio-app.com/> [consulté le (20/09/2020)]
- [15] Spring Cloud, accessible sur <https://spring.io/projects/spring-cloud> [consulté le (20/09/2020)]

- [16] Spring boot, accessible sur https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm [consulté le (20/09/2020)]
- [17] JWT, accessible sur <https://jwt.io/introduction/> [consulté le (20/09/2020)]
- [18] MapStruct, accessible sur <https://mapstruct.org/> [consulté le (20/09/2020)]
- [19] Single-spa, accessible sur <https://single-spa.js.org/docs/getting-started-overview> [consulté le (20/09/2020)]
- [20] Angular, accessible sur <https://angular.io/docs> [consulté le (20/09/2020)]
- [21] Spring data mongoDB, accessible sur <https://spring.io/projects/spring-data-mongodb> [consulté le (20/09/2020)]
- [24] Documentation officielle de spring cloud, accessible sur <https://spring.io/cloud> [consulté le (20/05/2020)]
- [25] Différence entre un proxy et un reverse-proxy, accessible sur <https://www.ionos.fr/digitalguide/serveur/know-how/quest-ce-quun-reverse-proxy-le-serveur-reverse-proxy/> [consulté le (22/05/2020)]
- [26] Différence entre un proxy et un reverse-proxy, accessible sur <https://smartproxy.com/blog/the-difference-between-a-reverse-proxy-and-a-forward-proxy> [consulté le (22/05/2020)]
- [27] <https://docs.spring.io/spring/docs/current/javadoc-api/org.springframework.web.client.RestTemplate.html> [consulté le (13/06/2020)]
- [28] Comparaison kafka, ActiveMQ et RabbitMQ <https://stackshare.io/stackups/activemq-vs-kafka-vs-rabbitmq#:~:text=RabbitMQ%2C%20Kafka%2C%20and%20ActiveMQ%20all,RabbitMQ%20is%20written%20in%20Erlang> [consulté le (14/09/2020)]

Bibliographie

- [5] Geers, M. (2020). *Micro Frontends in Action*. Manning Publications.
- [8] Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum* (Vol. 1). Upper Saddle River: Prentice Hall.
- [22] Fowler M (2014) *Microservices prerequisites*.
- [23] Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

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 Business & Decision.

Le travail consiste à concevoir et implémenter une plateforme de gestion de projet basée sur des Microservices et des Microfrontends.

Cette solution permet d'une part aux équipes de développement logiciel de planifier et suivre leur projet et d'autre part elle résout les problèmes de l'architecture monolithique.

Mots clés : Agile Scrum, Gestion de projet, Microservices, Microfrontends, Spring Boot, Spring Cloud, Single SPA, Angular

Abstract

Keywords :

المخلص

الكلمات المفتاحية :