

Institut des Arts et des Technologies Avancées

InATA



Mémoire de Fin d'Etudes en vue de l'obtention du diplôme de

LICENCE

« Conception et Développement d'une Application de Gestion de Projet Agile : Un Outil Innovant pour l'Optimisation des Processus Collaboratifs »

Présenté par : NIRINA MAMPIONONA FELANIAINA

Devant le jury composé de :

Président : Professeur RAVELOMANANTSOA RAMANAMBE Nicole
Encadreur : Monsieur RANDRIAMAHARO ANDRIAMALALA Mamy
Examineurs : RANDRIAMIHAJARISON Jimmy

REMERCIEMENTS

- Présidente de l'Institut des Arts et des Technologies Avancées - InATA
(Professeur RAVELOMANANTSOA RAMANAMBE Nicole)
- Directeur de l'Institut des Arts et des Technologies Avancées - InATA
(Monsieur RANDRIAMAHARO ANDRIAMALALA Mamy)
- Encadreur (s) pédagogique : Monsieur RANDRIAMAHARO
ANDRIAMALALA Mamy
- Membres du Jury
- Personnel Enseignant de l'InATA
- Mes parents, ma famille, mes amis, ainsi qu'à tous ceux qui, de près ou de loin,
ont contribué à la réussite de ce mémoire.

SOMMAIRE

INTRODUCTION GENERALE	1
PARTIE I : PRESENTATION DU PROJET	3
Chapitre 1 : Présentation de la Problématique	3
Chapitre 2 : Analyse des Solutions Existantes	3
Chapitre 3 : Justification de la Recherche et la proposition de la solution.....	4
Chapitre 4 : Objectifs du Projet	5
Chapitre 5 : Planification du Projet	5
Chapitre 6 : Résultats Attendus	7
PARTIE II : REVUE DE LA LITTERATURE	8
Chapitre 1 : Définition des Concepts Fondamentaux	8
Chapitre 2 : La Gestion de Projet Agile	10
Chapitre 3 : Outils et Méthodes Existants	15
PARTIE III : CONCEPTION ET MISE EN ŒUVRE DE L'APPLICATION	16
Chapitre 1 : Analyse des Besoins	16
Chapitre 2 : Choix technologiques et justification	18
Chapitre 3 : Conception de l'Application	28
Chapitre 4 : Implémentation	38
Chapitre 5 : Tests de l'application :	43
Chapitre 6 : Présentation de l'application :	45
PARTIE IV : CONCLUSION GENERALE.....	51

GLOSSAIRE

API (1) : est une interface applicative de programmation qui permet d'établir des connexions entre plusieurs logiciels pour échanger des données. Plus techniquement, il s'agit d'un ensemble de fonctions qui vont permettre à un développeur d'utiliser simplement une application dans son programme.

AXIOS (2) : est une bibliothèque JavaScript permettant de réaliser des requêtes HTTP depuis le navigateur ou Node.js, offrant une syntaxe simple et expressive pour effectuer des appels vers des API.

BACKEND (ou développement côté serveur) fait référence à la partie de l'application qui gère la logique, le stockage des données, et les opérations en arrière-plan. Il inclut le serveur, la base de données, et l'application serveur qui communique avec le frontend pour fournir des fonctionnalités dynamiques et des données.

BACKLOG (3) : Le backlog est une liste de tâches priorisées définissant les caractéristiques d'un produit. Il est un des éléments fondamentaux de la méthodologie Scrum.

BASES DE DONNEES (4) : est une collection organisée d'informations structurées, généralement stockées électroniquement dans un système informatique. Elle permet de gérer et de manipuler ces données de manière efficace grâce à des systèmes de gestion de base de données (SGBD)

FRAMEWORK (5) : Un framework est un ensemble d'outils et de composants logiciels organisés conformément à un plan d'architecture et des patterns, l'ensemble formant ou promouvant un « squelette » de programme.

FRONTEND (ou développement côté client) fait référence à la partie visible d'une application web ou d'un site internet. Il englobe tout ce que les utilisateurs voient et avec quoi ils interagissent directement dans leur navigateur. Cela inclut l'interface utilisateur (UI), le design, la mise en page, et l'interaction.

HTTP (Hypertext Transfer Protocol) est un protocole de communication basé sur le modèle client-serveur, qui permet de transmettre des données telles que des pages web, des images, des vidéos, et d'autres ressources entre un serveur et un client via le Web.

LE PRODUCT BACKLOG (6) est un élément fondamental issu de la méthode agile SCRUM, qui repose sur la fragmentation d'un projet en cycles de travail itératifs. Il s'agit d'une liste de fonctionnalités d'un produit à développer, classées grâce à un travail de priorisation. On les appelle aussi items ou user stories (ou scénarios d'utilisation).

SPRINT : Un Sprint est un intervalle de temps fixe, au cours duquel une équipe Scrum se concentre sur la réalisation d'un ensemble prédéterminé de tâches ou d'objectifs issus du Product Backlog.

ACRONYMES (liste des abréviations)

API : Application Programming Interface

HTTP : Hypertext Transfer Protocol

IDE : Integrated Development Environment

InATA : Institut des Arts et des Technologies Avancées

JWT : JSON Web Token

NPM : Node Package Manager

UML : Unified Modeling Language

LISTE DES FIGURES

Figure 1: Sprint 1 du diagramme de Gantt du projet.....	6
Figure 2: Sprint 2 du diagramme de Gantt du projet.....	6
Figure 3: Sprint 3 du diagramme de Gantt du projet.....	7
Figure 4: Sprint 4 du diagramme de Gantt du projet.....	7
Figure 5: Sprint 5 du diagramme de Gantt du projet.....	7
Figure 6: Triangle d'or en gestion de projet.....	9
Figure 7: Frise Chronologique de l'Évolution de l'Agilité en Gestion de Projet	11
Figure 8: Schéma comparatif de la méthode Traditionnelle et méthode agile	13
Figure 9: Vue d'Ensemble du Processus Scrum en Gestion de Projet Agile.....	14
Figure 10: Illustration d'un tableau kanban	15
Figure 11 : Illustration de l'architecture 3-tiers	30
Figure 12: Diagramme de cas d'utilisation global du projet	30
Figure 13 : Diagramme des classes de l'application de la gestion de projet.....	32
Figure 14: Diagramme de séquence de l'authentification	33
Figure 15 : Diagramme de séquence de Création d'un projet	33
Figure 16: Diagramme de séquence de rédaction du backlog.....	34
Figure 17: Diagramme de séquence de création de sprint	34
Figure 18 : Diagramme de séquence d'invitation de responsable scrum master.....	34
Figure 19 : Diagramme de séquence de la gestion des collaborateurs	35
Figure 20 : Diagramme de séquence d'ajout des taches	35
Figure 21 : Diagramme de séquence de la visualisation de tableau à bord.....	35
Figure 22 : Diagramme de séquence d'invitations des membres.....	36
Figure 23 : Diagramme de séquence du passage au sprint suivant.....	36
Figure 24 : Diagramme de séquence de l'assignation des tâches.....	36
Figure 25 : Diagramme de séquence de la visualisation des listes des projets	36
Figure 26 : Diagramme de séquence pour la visualisation de suivi des tâches	37
Figure 27 : Diagramme de séquence pour la le suivi des taches personnelles.....	37
Figure 28 : Diagramme de séquence pour l'envoi des messages	37
Figure 29: commande pour créer une application react	39
Figure 30 : Commande utilisée pour l'installation des "package" nécessaires.....	39
Figure 31: Commande pour la création d'un projet nodeJS	39
Figure 32 : Commande utilisée pour l'installation des "package" pour le développement backend.....	40
Figure 33 : Structure du projet côté serveur.....	41
Figure 34 : Structure du projet React côté client	42
Figure 35 : Page d'authentification	46
Figure 36 : Interface pour la création d'un nouveau projet et les projets déjà existants.....	47
Figure 37 : Rédaction du backlog du projet	47
Figure 38 : Page de la planification de sprint.....	48
Figure 39 : Page de la gestion des équipes	48
Figure 40 : Tableau de bord de l'application.....	49
Figure 41 : Page de suivi des taches.....	50
Figure 42 : Création d'une tâche	50
Figure 43 : Extrait du code de l'authentification de l'application	XII
Figure 44: Extrait du code pour l'envoi de mail d'invitation	XIII
Figure 45:Extrait du code qui gère l'importation des fichiers.....	XIII

LISTE DES TABLEAUX

Tableau 1 : Tableau d'analyse des outils de gestion de projet existants	4
Tableau 2 : Tableau comparatif des deux méthodologies du gestion de projet.....	12
Tableau 3: Tableau comparatif des framework front-end possible pour le développement.....	19
Tableau 4 : Tableau comparatif des technologies utilisées pour le développement backEnd	21
Tableau 5 : Tableau Comparatif des bases de données.....	23
Tableau 6 : Technologies d'authentification et autorisation	24
Tableau 7 : Tableau comparatif des technologies de communication en temps réel	25

LISTE DES ANNEXES

Annexe 1 : Extrait du code de l'authentification de l'application

Annexe 2 : Extrait du code pour l'envoi de mail d'invitation

Annexe 3 : Extrait du code qui gère l'importation des fichiers

INTRODUCTION GENERALE

Dans un monde où l'innovation technologique est au cœur de la compétitivité des entreprises, la gestion de projets doit être non seulement flexible et réactive, mais aussi hautement efficiente. La méthodologie agile, reconnue pour sa capacité à s'adapter rapidement aux changements et à favoriser une collaboration étroite entre les équipes, s'est imposée comme un standard incontournable dans de nombreux secteurs. Cependant, l'efficacité de cette méthodologie dépend largement de l'utilisation d'outils spécialement conçus pour faciliter la gestion agile des projets.

C'est dans ce contexte que s'inscrit ce mémoire, avec pour objectif de concevoir et développer une application de gestion de projet agile qui soit à la fois intuitive, adaptable, et spécifiquement orientée vers les besoins des équipes de projet modernes. Bien que de nombreux outils performants soient disponibles sur le marché, il existe encore des opportunités pour développer des solutions mieux adaptées aux besoins spécifiques des petites et moyennes entreprises, notamment en matière de flexibilité et d'intégration dans des environnements de travail diversifiés.

La problématique fondamentale à laquelle ce mémoire se focalise est la suivante : comment concevoir une application de gestion de projet agile qui soit à la fois intuitive et adaptable, répondant de manière optimale aux besoins variés des équipes de projet ? Cette interrogation soulève des défis techniques et organisationnels complexes, nécessitant une réflexion approfondie sur l'architecture logicielle, la conception de l'interface utilisateur, et la gestion des données.

L'objectif principal de ce travail est de développer une solution qui non seulement simplifie la gestion des tâches, mais aussi améliore la communication et la coordination au sein des équipes, tout en restant fidèle aux principes agiles. En adoptant une approche itérative et incrémentale, ce projet vise à créer un outil capable de transformer les pratiques de gestion de projet en milieu professionnel.

Le mémoire se structure en trois grandes parties : la première partie présente une vue d'ensemble du projet, en exposant le contexte, les enjeux et les objectifs de la recherche, ainsi que les résultats attendus. La deuxième partie propose une analyse approfondie de la littérature existante, en explorant les principes et les outils de la gestion de projet agile. Enfin, la troisième partie se concentre sur l'étude pratique,

détaillant le processus de conception et de développement de l'application, ainsi que les étapes de validation pour assurer son efficacité.

PARTIE I : PRESENTATION DU PROJET

Cette première partie du mémoire est consacrée à l'exposé détaillé du projet de développement d'une application de gestion de projet agile. Elle fournit un cadre complet pour comprendre les fondements du projet, sa justification, ses objectifs, ainsi que la planification nécessaire pour sa réalisation. En abordant les différents aspects du projet, cette partie permet de cerner les enjeux et les attentes, tout en établissant une base solide pour les développements ultérieurs.

Chapitre 1 : Présentation de la Problématique

À l'ère de la transformation numérique, les entreprises doivent naviguer dans un environnement en perpétuelle mutation, où la gestion efficace des projets est un facteur clé de succès. La méthodologie agile, en raison de sa flexibilité et de sa capacité à s'adapter rapidement aux changements, s'est imposée comme une norme dans la gestion de projet. Elle favorise une collaboration étroite au sein des équipes et permet une réponse rapide aux évolutions des exigences.

Cependant, bien que le marché propose une variété d'outils de gestion de projet agile, ces solutions ne répondent pas toujours de manière optimale aux besoins spécifiques des entreprises. Les outils disponibles peuvent présenter des limitations en termes de personnalisation, d'adaptabilité aux processus spécifiques des organisations, et d'intégration dans des environnements de travail diversifiés. Ces lacunes peuvent entraver la gestion efficace des projets et limiter les capacités des équipes à tirer pleinement parti des principes agiles.

Pour répondre à ces défis, le développement d'une application de gestion de projet agile, spécialement conçue pour combler ces lacunes, représente une solution pertinente. Cette approche vise à offrir un outil qui non seulement simplifie la gestion des tâches mais améliore également la communication et la coordination au sein des équipes.

Chapitre 2 : Analyse des Solutions Existantes

Ce chapitre offre une évaluation des solutions populaires de gestion de projet agile (Trello et Jira), afin de mettre en lumière leurs avantages et leurs limitations.

Tableau 1 : Tableau d'analyse des outils de gestion de projet existants

	Avantages	Inconvénient
Trello	<ul style="list-style-type: none"> • Interface simple, visualisation claire des tâches • Facilité d'utilisation pour les petites équipes • Intégrations variées 	<ul style="list-style-type: none"> • Manque de personnalisation pour les projets complexes • Limites dans le suivi des performances et l'analyse détaillée des tâches.
Jira	<ul style="list-style-type: none"> • Conçu pour les projets complexes avec des fonctionnalités avancées • Intégration avec les méthodologies agiles (Scrum, Kanban) • Outils de reporting et de suivi performants 	<ul style="list-style-type: none"> • Interface complexe, peu intuitive pour les débutants • Peut-être coûteux pour les petites équipes

Malgré leurs avantages, Trello et Jira présentent des limites en termes de personnalisation et d'adaptabilité aux besoins spécifiques des petites et moyennes entreprises. Ces lacunes motivent l'exploration de solutions alternatives mieux adaptées aux contraintes et processus de certaines organisations.

Chapitre 3 : Justification de la Recherche et la proposition de la solution

La recherche est motivée par l'identification de lacunes significatives dans les outils de gestion de projet agile actuellement disponibles, tels que Trello et Jira. Bien qu'efficaces, ces solutions présentent des limitations en termes de flexibilité, d'intégration et de personnalisation, notamment pour les petites et moyennes

entreprises. Ces lacunes peuvent nuire à l'adaptation aux besoins spécifiques des équipes et compromettre l'efficacité des processus de gestion de projet.

Pour répondre à ces défis, ce projet propose le développement d'une application de gestion de projet agile sur mesure. Cette application vise à offrir une interface intuitive et conviviale, des options de personnalisation avancées, ainsi qu'une intégration fluide dans divers environnements de travail. L'accent sera mis sur l'amélioration de la collaboration et de la communication au sein des équipes, ainsi que sur une gestion des tâches optimisée, permettant ainsi de mieux répondre aux exigences uniques des utilisateurs.

Chapitre 4 : Objectifs du Projet

Les principaux objectifs de ce projet sont :

- **Développer une application de gestion de projet agile** qui soit à la fois intuitive et adaptable aux divers besoins des équipes de projet.
- **Améliorer la communication et la coordination** au sein des équipes en fournissant des outils et des fonctionnalités qui facilitent la collaboration.
- **Aligner l'application avec les principes agiles**, afin de garantir que les pratiques de gestion de projet sont respectées et optimisées.
- **Offrir une solution qui répond aux besoins spécifiques des PME**, en tenant compte des contraintes et des exigences particulières de ces organisations.

Chapitre 5 : Planification du Projet

Cette section détaille la planification du projet, en abordant les aspects financiers, temporels, et les outils de gestion de projet.

1 . Coûts du Projet

Le budget prévisionnel du projet est établi pour inclure les coûts associés au développement, aux tests, et au déploiement de l'application. Cette section présente une répartition des dépenses, couvrant les coûts des ressources humaines, des technologies utilisées, et des éventuels frais opérationnels. L'objectif est de garantir une gestion financière rigoureuse et de s'assurer que le projet reste dans les limites budgétaires prévues.

2 . Délais du Projet

Un calendrier détaillé est élaboré pour planifier les principales phases du projet, depuis la conception jusqu'au déploiement. Cette section présente une estimation des délais pour chaque étape clé, en définissant les jalons importants et les échéances associées. La planification temporelle permet de suivre l'avancement du projet et de s'assurer que les objectifs sont atteints dans les délais impartis.

3 . Diagramme de Gantt

Le diagramme de Gantt est un outil visuel permettant de représenter la chronologie d'un projet. Il montre les différentes phases, les principaux jalons et les échéances, facilitant ainsi la visualisation de l'avancement du projet et la coordination des activités. Les figures suivantes présentent le diagramme de Gantt du projet, offrant une vue d'ensemble des étapes suivies.

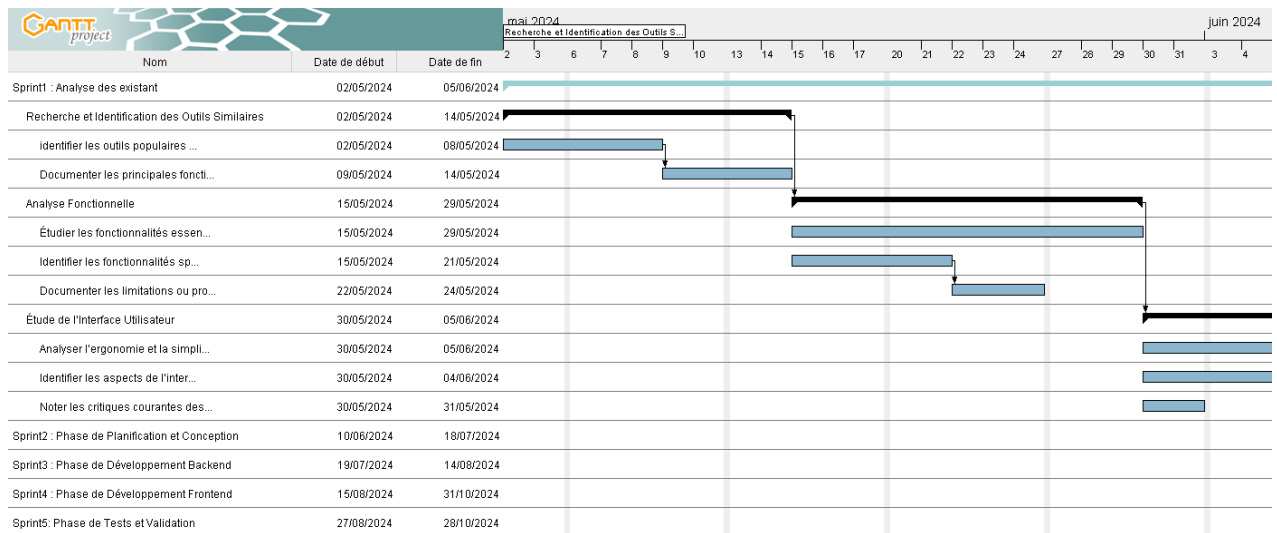


Figure 1: Sprint 1 du diagramme de Gantt du projet

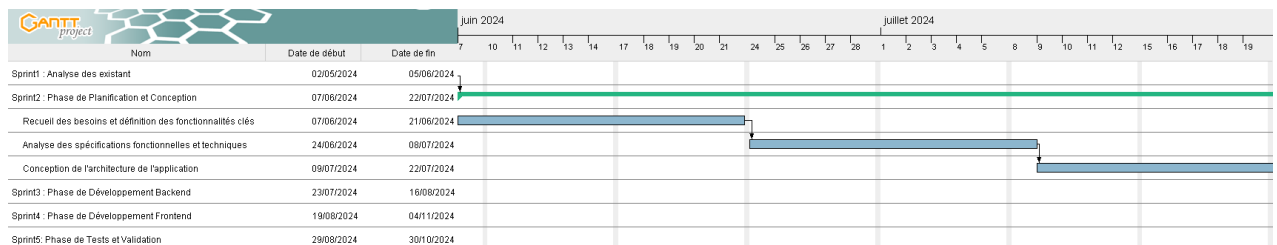


Figure 2: Sprint 2 du diagramme de Gantt du projet

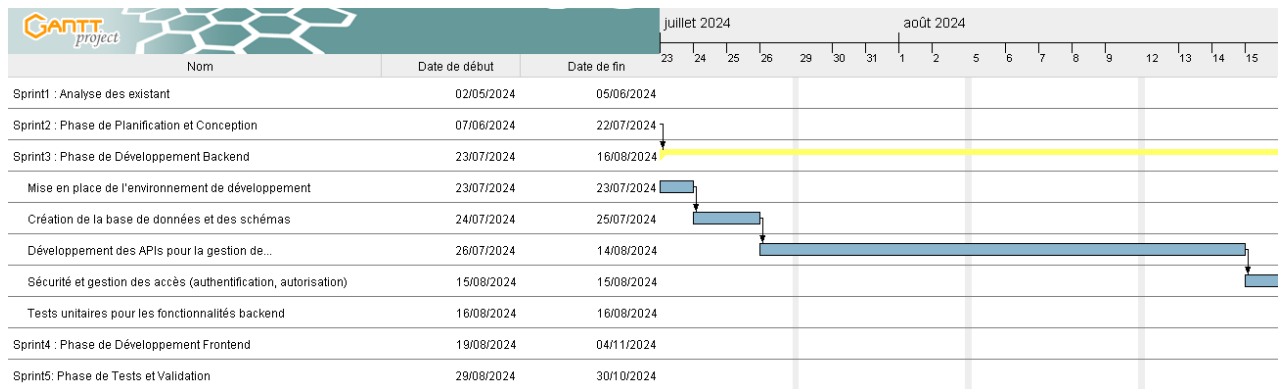


Figure 3: Sprint 3 du diagramme de Gantt du projet

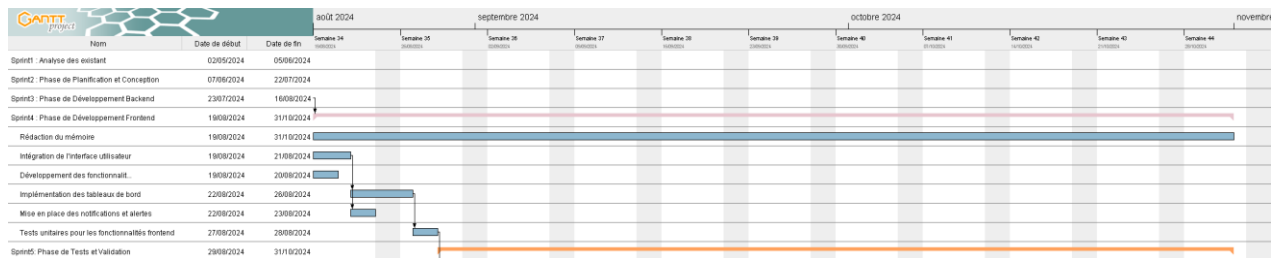


Figure 4: Sprint 4 du diagramme de Gantt du projet

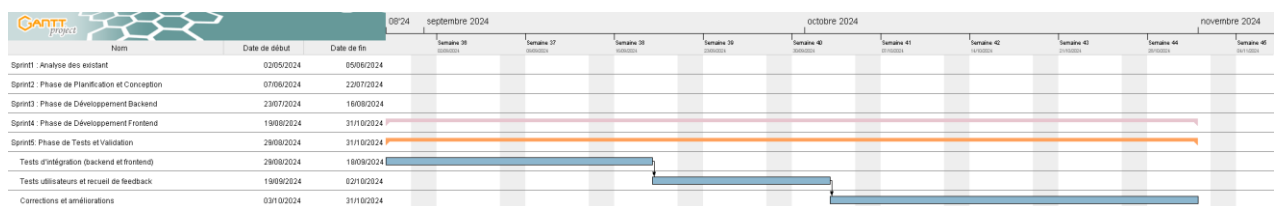


Figure 5: Sprint 5 du diagramme de Gantt du projet

Chapitre 6 : Résultats Attendus

Les résultats attendus de ce projet incluent :

- **Une application de gestion de projet agile opérationnelle**, qui répond aux besoins des équipes de projet et est conforme aux objectifs définis.
- **Une amélioration de la gestion des tâches**, avec une interface utilisateur intuitive et des fonctionnalités qui favorisent une collaboration efficace.
- **Une validation des principes agiles**, avec des pratiques de gestion de projet alignées avec les standards de l'industrie.
- **Des bénéfices tangibles pour les entreprises**, en offrant une solution plus flexible et adaptée à leurs besoins spécifiques.

PARTIE II : REVUE DE LA LITTERATURE

La Revue de la Littérature constitue une analyse approfondie des concepts, outils, et méthodologies associées à la gestion de projet agile. Cette partie vise à établir une compréhension claire des fondements théoriques et pratiques sur lesquels repose la gestion de projet agile, en fournissant un cadre pour évaluer et contextualiser l'application développée dans le projet. Nous aborderons les définitions clés, les outils existants, les pratiques agiles, et les comparaisons entre différentes méthodologies.

Chapitre 1 : Définition des Concepts Fondamentaux

1 . Un Projet

Un projet (7) est un ensemble de tâches à réaliser afin d'atteindre un objectif défini, dans un contexte précis, dans les délais impartis et selon le niveau de qualité souhaité. Il est mené et géré par un groupe de personnes dont la taille peut évoluer de quelques collaborateurs à plusieurs centaines en fonction de sa complexité.

C'est un processus unique et temporaire, entrepris pour atteindre un objectif spécifique et conforme à des exigences prédéfinies. Il se caractérise par un ensemble d'activités coordonnées et maîtrisées, avec une date de début et de fin clairement définies, visant à produire un résultat unique. Les projets sont réalisés pour répondre à une demande précise, dans le but de satisfaire les besoins des bénéficiaires tout en respectant des contraintes de délais, de coûts et de ressources. En somme, un projet est une démarche structurée qui permet de transformer une idée en réalité, en suivant une méthode progressive et organisée pour produire des résultats tangibles.

2 . La gestion de projet :

La gestion de projet (8) c'est l'ensemble des outils et méthodes de gestion de projet déployés pour s'assurer que l'équipe projet puisse atteindre les objectifs fixés.

Elle est l'application méthodique de méthodes, d'outils et de techniques pour planifier, exécuter, contrôler et clore toutes les phases d'un projet, de sa planification à son achèvement. Elle a pour objectif de réaliser les résultats attendus tout en respectant les contraintes de temps, de budget et de ressources. Ce processus implique l'organisation et la coordination des activités du projet afin de satisfaire les exigences des parties prenantes, en équilibrant les priorités et en garantissant la qualité des livrables. La gestion de projet vise à optimiser l'utilisation des ressources

disponibles pour produire des résultats conformes aux attentes, tout en adaptant les stratégies en fonction des évolutions et des défis rencontrés.

Le triangle d'or, ou **triangle de la triple contrainte**, est un modèle clé qui illustre l'équilibre entre trois facteurs majeurs : **le coût**, **le délai** et **la qualité**. Tout ajustement de l'un de ces éléments impacte les deux autres, influençant ainsi le succès global du projet.

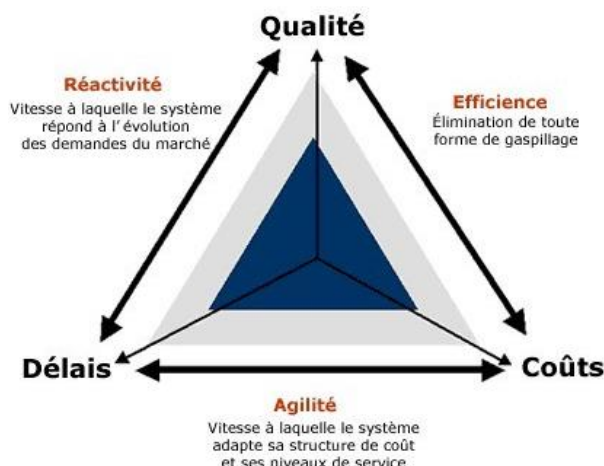


Figure 6: Triangle d'or en gestion de projet

3 . Un Jalon :

Un jalon (9) est un point d'arrêt dans le processus permettant le suivi du projet. C'est l'occasion pour l'équipe de faire un bilan intermédiaire, de valider une étape, des documents ou d'autres livrables, puis de reprendre le déroulement des travaux.

Il est un point de référence important dans le déroulement d'un projet, servant à marquer des étapes clés ou des transitions significatives dans le processus. Il permet de suivre l'avancement du projet en fournissant des repères pour évaluer les progrès réalisés. À ce stade, le retour en arrière n'est pas disponible : une fois le jalon franchi, l'équipe poursuit le projet sans revenir sur les étapes précédentes qui ont été validées. Les jalons sont essentiels pour la planification et le contrôle, car ils permettent de mesurer l'achèvement de phases spécifiques et de garantir que les objectifs sont atteints selon le calendrier établi.

Chapitre 2 : La Gestion de Projet Agile

La gestion de projet agile découpe un projet en différentes itérations ou sous-projets indépendants, appelés sprints ou itérations, qui sont répétées jusqu'à atteindre le résultat souhaité. Cette méthode se concentre sur la flexibilité, la collaboration, et l'amélioration continue. Elle permet de tester le produit au fur et à mesure, de réduire le délai entre la formulation d'un besoin et sa concrétisation, et d'adapter le produit en fonction des retours réguliers des utilisateurs. Les changements sont considérés comme des opportunités plutôt que des obstacles, et la communication claire et régulière est essentielle.

1 . Historique de la Gestion de Projet Agile

La gestion de projet agile (10) a émergé comme une réponse aux limitations des méthodologies de gestion de projet traditionnelles, telles que le modèle en cascade (waterfall). Voici un aperçu des évolutions clés :

- Origines et Évolution : À la fin des années 1990 et au début des années 2000, la nécessité d'une approche plus flexible et réactive à la gestion de projet est devenue évidente. Les méthodologies traditionnelles, souvent linéaires et rigides, ont montré leurs limites, notamment en termes de réactivité aux changements et de capacité à répondre aux besoins des utilisateurs en évolution.
- Publication du Manifeste Agile : En 2001, un groupe de professionnels du développement logiciel a rédigé le Manifeste Agile, un document fondateur qui a défini les valeurs et principes de l'agilité. Ce manifeste a marqué un tournant significatif en promouvant une approche itérative, collaborative, et centrée sur le client. Les quatre valeurs fondamentales du Manifeste Agile sont :
 - Les individus et leurs interactions plutôt que les processus et les outils.
 - Un logiciel fonctionnel plutôt qu'une documentation exhaustive.
 - La collaboration avec le client plutôt que la négociation contractuelle.
 - L'adaptation au changement plutôt que le suivi d'un plan.
- Méthodologies Agiles : Depuis la publication du Manifeste Agile, plusieurs méthodologies spécifiques ont été développées, chacune offrant des approches distinctes pour mettre en œuvre les principes agiles :

- Scrum : Développé pour organiser le travail en cycles de développement courts appelés sprints, Scrum met l'accent sur les rôles définis, les cérémonies régulières, et la livraison incrémentielle de produit.
- Kanban : Introduit pour améliorer la gestion visuelle du flux de travail, Kanban utilise un tableau pour suivre l'avancement des tâches et limiter le travail en cours.
- Lean : Inspiré des principes de fabrication Lean, cette approche se concentre sur l'élimination des gaspillages et l'amélioration continue.

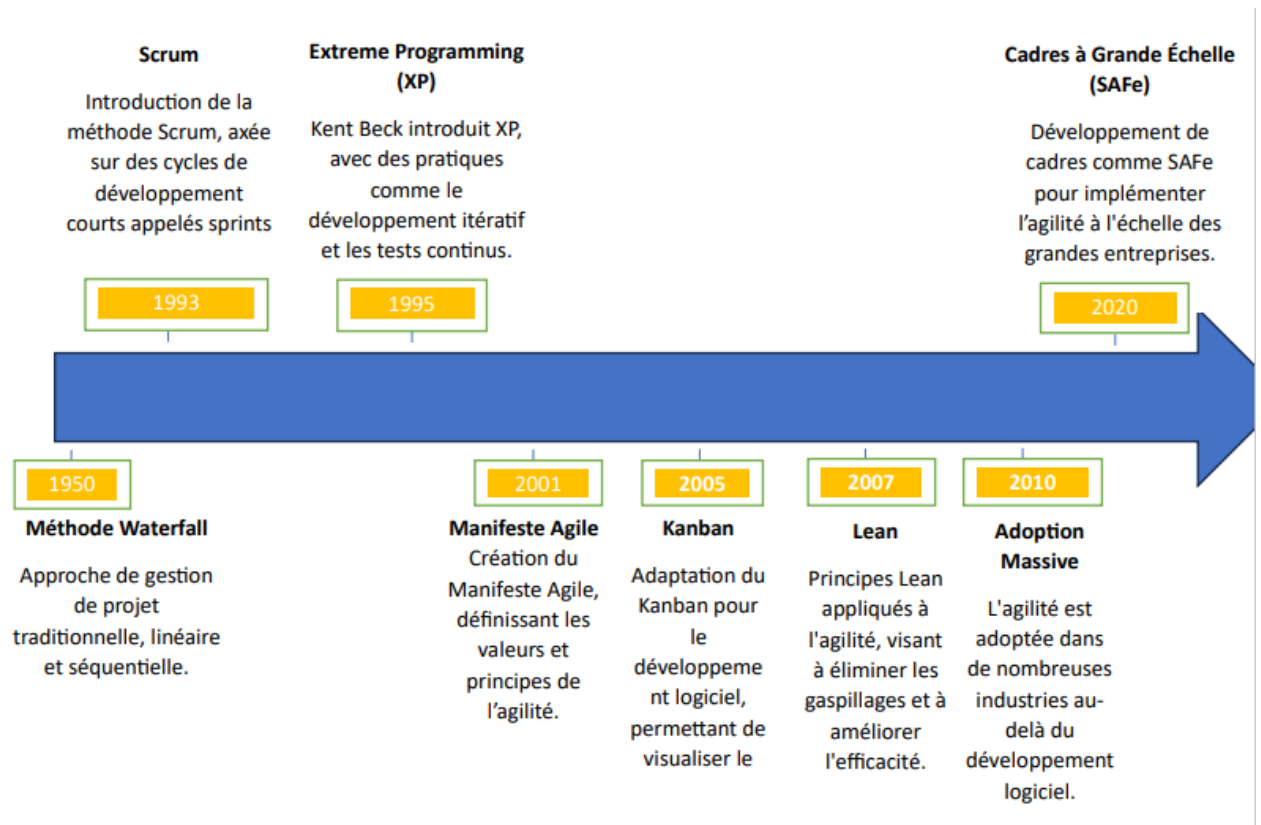


Figure 7: Frise Chronologique de l'Évolution de l'Agilité en Gestion de Projet

2 . Comparaison avec la Gestion de Projet Traditionnelle

La gestion de projet agile s'oppose aux méthodologies traditionnelles telles que le modèle en cascade (waterfall). Voici une comparaison des deux approches :

Tableau 2 : Tableau comparatif des deux méthodologies du gestion de projet

Critère	Méthodologie traditionnelle	Méthodologie Agile
Processus	Linéaire et prédictif (ex : modèle en cascade)	Itératif et adaptatif
Planification	Planification détaillée au début du projet.	Planification continue et ajustable à chaque itération
Exécution	Phases exécutées de manière séquentielle	Livraisons régulières de versions fonctionnelles
Gestion des changements	Changements difficiles à intégrer	Adaptation rapide aux changements
Risque	Risque de déviation par rapport aux attentes des utilisateurs	Réduction du risque grâce aux ajustements réguliers
Collaboration	Collaboration limitée entre les membres de l'équipe et les parties prenantes	Collaboration constante avec toutes les parties prenantes
Livrables	Produit livré à la fin du projet	Versions fonctionnelles livrées régulièrement
Gestion du budget et des délais	Dépassements possibles en cas de changement des besoins	Meilleure maîtrise grâce à l'adaptabilité

Ci-dessous une figure expliquant la différence entre ces deux méthodes :

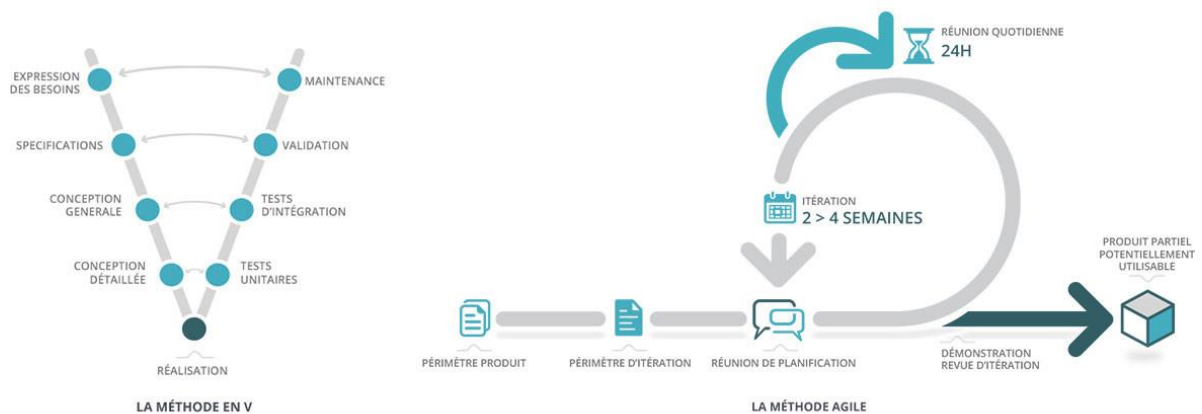


Figure 8: Schéma comparatif de la méthode Traditionnelle et méthode agile

3 . Composants et Structures des Méthodes Agiles

Les méthodologies agiles se composent de diverses structures et pratiques qui orientent la gestion des projets de manière efficace et collaborative. Voici un aperçu des deux cadres agiles les plus connus, leurs composants clés et la manière dont ils guident la gestion des projets :

a . Scrum :

Scrum est un cadre agile qui organise le travail en cycles de développement appelés sprints, généralement de deux à quatre semaines. Il se compose des éléments suivants :

- Rôles :
 - Scrum Master : Responsable de la facilitation du processus Scrum, de la suppression des obstacles et de la protection de l'équipe contre les interruptions externes.
 - Product Owner : Représente les parties prenantes et les clients, définit et priorise les exigences du produit dans le backlog, et assure que l'équipe travaille sur les tâches les plus importantes.

- Équipe de Développement : Composée de professionnels qui travaillent ensemble pour créer le produit selon les exigences définies. L'équipe est auto-organisée et multidisciplinaire.
- Cérémonies :
 - Daily Stand-up : c'est une réunion quotidienne qui permet à l'équipe de synchroniser ses efforts, d'identifier les obstacles, et de planifier les actions pour la journée.
 - Sprint Review : Revue de sprint permettant de démontrer le travail accompli pendant le sprint aux parties prenantes et de recueillir des retours.

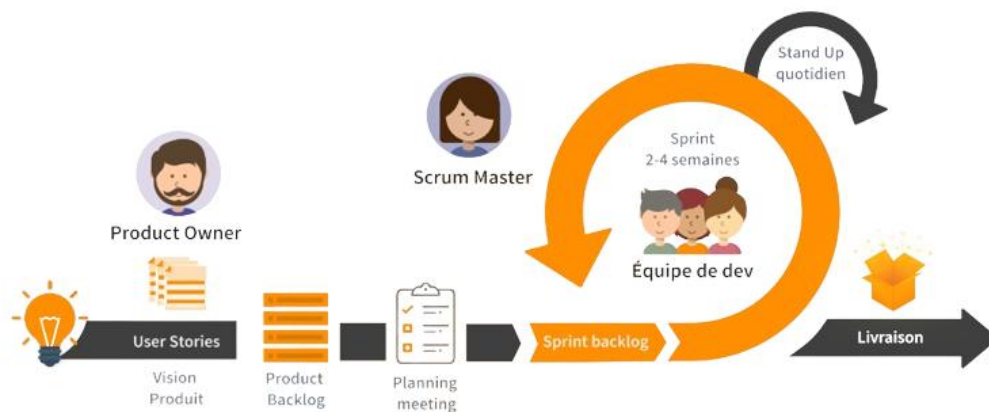


Figure 9: Vue d'Ensemble du Processus Scrum en Gestion de Projet Agile

b . Kanban :

Kanban est une méthode de gestion visuelle du flux de travail qui utilise un tableau Kanban pour suivre l'avancement des tâches. Ses composants incluent :

- Tableau Kanban : Un outil visuel qui divise le travail en différentes colonnes représentant les étapes du processus (par exemple, À faire, En cours, Terminé). Les tâches sont représentées par des cartes qui se déplacent à travers les colonnes en fonction de leur statut.
- Limitation du Travail en Cours (WIP) : Fixe des limites sur le nombre de tâches pouvant être en cours dans chaque colonne pour éviter la surcharge et améliorer le flux de travail.

- Réunions de Coordination : Bien que Kanban ne prescrive pas des cérémonies spécifiques comme Scrum, des réunions régulières peuvent être organisées pour discuter de l'état des tâches et des améliorations possibles.



Figure 10: Illustration d'un tableau kanban

Ces méthodologies offrent des approches distinctes mais complémentaires pour la gestion des projets agiles, chacune avec ses propres outils et pratiques pour améliorer l'efficacité et la collaboration au sein des équipes.

Chapitre 3 : Outils et Méthodes Existants

De nombreux outils et méthodes sont disponibles pour soutenir la gestion de projet agile. Les plus courants incluent :

- Jira : Un outil puissant pour le suivi des tâches et la gestion des projets agiles, offrant des fonctionnalités telles que la gestion des sprints, des tableaux de bord personnalisables, et des rapports détaillés.
- Trello : Utilisé pour la gestion visuelle des tâches à l'aide de cartes et de colonnes, Trello est particulièrement apprécié pour sa simplicité et son interface intuitive.
- Asana : Un outil de gestion de projet qui facilite la planification des tâches, le suivi de la progression, et la collaboration entre les membres de l'équipe.

Ces outils offrent des fonctionnalités variées qui répondent aux besoins des équipes agiles, mais présentent également des limites en termes de flexibilité et d'adaptation aux besoins spécifiques des entreprises.

PARTIE III : CONCEPTION ET MISE EN ŒUVRE DE L'APPLICATION

Cette troisième partie vise à détailler le processus de réalisation de l'application, depuis l'analyse initiale des besoins jusqu'à la validation finale. Elle couvre l'ensemble des étapes critiques du développement, permettant ainsi de mettre en lumière les décisions prises, les défis rencontrés, et les solutions apportées pour atteindre les objectifs du projet. Une attention particulière sera portée à la justification des choix technologiques, qui jouent un rôle central dans la conception et la réussite de l'application.

Chapitre 1 : Analyse des Besoins

L'analyse des besoins et des spécifications constitue une étape cruciale pour la réussite de tout projet de développement. Elle permet de cerner précisément les attentes des utilisateurs et les fonctionnalités indispensables au bon fonctionnement de l'application. Cette section décrit les besoins fonctionnels et non fonctionnels, ainsi que les spécifications techniques de l'application.

1 . Identification des Parties Prenantes

Dans le cadre du développement de cette application de gestion de projet Agile, les parties prenantes clés ont été identifiées comme suit :

- **Product Owner** : Le Product Owner est responsable de la définition des exigences fonctionnelles de l'application et de la priorisation des fonctionnalités dans le backlog. Il agit comme l'interface principale entre l'équipe de développement et les utilisateurs finaux, en s'assurant que le produit final correspond aux besoins du client.
- **Scrum Master** : Le Scrum Master est chargé de faciliter les cérémonies Scrum, telles que les daily stand-ups, les rétrospectives et les revues de sprint. Il s'assure que l'équipe suit les pratiques Agile, aide à surmonter les obstacles et assure le bon déroulement du processus de développement.

- **Développeurs** : Les développeurs sont responsables de l'implémentation des fonctionnalités définies dans le backlog. Ils travaillent en étroite collaboration avec le Product Owner pour s'assurer que les exigences sont bien comprises et que le produit est développé conformément aux spécifications.
- **Testeurs** : Les testeurs sont chargés de vérifier que les fonctionnalités développées répondent aux exigences spécifiées et qu'elles fonctionnent correctement sans bugs. Ils jouent un rôle crucial dans la validation du produit avant sa mise en production.

2 . Besoins Fonctionnels

Les besoins fonctionnels de l'application se déclinent en plusieurs modules principaux, chacun correspondant à une fonctionnalité clé :

- **Gestion des Backlogs :**
 - Ajout, modification et suppression de tâches dans le backlog.
 - Priorisation des tâches selon leur importance et leur urgence.
 - Association des tâches à des utilisateurs et à des sprints spécifiques.
- **Gestion des Projets :**
 - Création, modification et suppression de projets.
 - Définition des objectifs du projet et des critères de succès.
 - Suivi de l'état d'avancement du projet à travers différents sprints.
- **Gestion des Utilisateurs :**
 - Ajout, modification et suppression d'utilisateurs.
 - Attribution de rôles (chef de projet, développeur, client, etc.) et gestion des permissions associées.
 - Suivi des activités des utilisateurs au sein des projets.
- **Définition et Gestion des Sprints :**
 - Création de sprints avec des dates de début et de fin.

- Allocation des tâches aux sprints selon leur priorité et le planning.
- Suivi de la progression des sprints et ajustements en cours de route si nécessaire.
- **Suivi du temps ainsi que notifications et rappels pour chaque tâche :**
L'application doit envoyer des notifications pour rappeler aux utilisateurs de suivre le temps ou d'alerter lorsque le temps alloué pour une tâche est sur le point d'être dépassé.
- **Module de Chat :** Communication en temps réel entre les membres de l'équipe

3 . Besoins Non-Fonctionnels

En plus des besoins fonctionnels, l'application doit répondre à des exigences non-fonctionnelles essentielles, telles que :

- **Performance :** Réponse rapide et fluidité d'utilisation, même avec un grand nombre d'utilisateurs simultanés.
- **Sécurité :** Les sessions utilisateur expirent automatiquement après une période d'inactivité pour prévenir tout accès non autorisé. Ces mesures assurent la protection des données utilisateurs et projets, garantissant une expérience sécurisée.

Chapitre 2 : Choix technologiques et justification

Dans le cadre du développement de cette application, il est essentiel de choisir des technologies adaptées aux besoins du projet, en tenant compte des exigences fonctionnelles, des contraintes de performance, de la sécurité, et de la maintenabilité. Les choix technologiques effectués se basent sur une analyse rigoureuse des besoins, ainsi que sur une évaluation des outils disponibles sur le marché. Cette section détaille les technologies sélectionnées et justifie leur adoption en fonction de leur pertinence et de leur efficacité pour atteindre les objectifs fixés.

1 . Utilisation de TypeScript

TypeScript a été retenu comme langage principal pour le développement de cette application, à la fois pour le frontend et le backend. Ce choix repose sur plusieurs facteurs déterminants :


- **Typage statique** : TypeScript introduit un typage statique à JavaScript, permettant de détecter les erreurs lors de la compilation plutôt qu'à l'exécution. Cela contribue à la sécurité du code et réduit le nombre de bugs en production.
- **Meilleure gestion des erreurs** : Grâce au typage statique et à une meilleure IDE, TypeScript permet de repérer et corriger les erreurs plus facilement, ce qui améliore l'efficacité du développement.
- **Interopérabilité avec JavaScript** : TypeScript est une extension de JavaScript, ce qui permet une adoption progressive dans les projets existants. Il offre également une compatibilité totale avec les bibliothèques et frameworks JavaScript, facilitant ainsi le développement et l'intégration avec des outils existants.




2 . Comparaison entre les Framework web pour le développement de l'application :

Le choix de la technologie frontend est essentiel pour garantir une interface utilisateur réactive, fluide et facile à maintenir. Voici les différentes technologies envisagées et la justification du choix final.

a . Comparaison entre Framework web pour le frontEnd



Tableau 3: Tableau comparatif des framework front-end possible pour le développement

Framework	Avantages	Faiblesses
React 	<ul style="list-style-type: none"> - Grande communauté : Large écosystème et support - Flexibilité : Grande liberté pour choisir les outils et architectures - Réutilisation des composants : Composants modulaires et réutilisables - Performance : DOM virtuel pour une manipulation efficace 	<ul style="list-style-type: none"> - Courbe d'apprentissage : Concepts avancés comme les hooks peuvent être complexes - Écosystème fragmenté : Nécessite des intégrations pour des fonctionnalités comme la gestion d'état et le routage



<p>Angular</p> 	<ul style="list-style-type: none"> - Solution complète : Inclut des fonctionnalités comme le routage, la gestion d'état, et plus encore. - Réactivité : Utilisation de RxJS pour une gestion efficace des flux de données - Performances : Compilation Ahead-of-Time (AOT) améliore les temps de chargement 	<ul style="list-style-type: none"> - Complexité : Courbe d'apprentissage raide en raison de la richesse du framework - Moins de flexibilité : Structure et conventions strictes peuvent limiter les choix - Taille : Framework relativement volumineux qui peut impacter les temps de chargement
<p>Vue.js</p> 	<ul style="list-style-type: none"> - Facilité d'apprentissage : Courbe d'apprentissage douce avec une documentation claire - Flexibilité : Options pour utiliser avec des configurations variées, de simples à complexes - Réactivité : Système réactif simple et efficace pour le suivi des changements de données - Performance : Taille légère et optimisation efficace 	<ul style="list-style-type: none"> - Adoption limitée : Moins utilisé par les grandes entreprises comparé à React et Angular - Écosystème moins riche : Moins de bibliothèques et outils intégrés - Maturité : Moins mature pour des projets très complexes
<p>Svelte</p> 	<ul style="list-style-type: none"> - Performance : Compilé en code JavaScript natif, ce qui réduit la taille du bundle et améliore la vitesse - Simplicité : Moins de code à écrire et une syntaxe claire - Réactivité : Système réactif intégré sans gestion complexe des états - Facilité d'apprentissage : Syntaxe simple et intuitive 	<ul style="list-style-type: none"> - Communauté plus petite : Moins de support et d'écosystème comparé aux autres frameworks - Moins d'outils : Moins de bibliothèques et outils disponibles - Moins éprouvé : Moins utilisé dans des environnements de production à grande échelle

b . Comparaison entre Framework web pour le BackEnd :

Tableau 4 : Tableau comparatif des technologies utilisées pour le développement backEnd





Framework	Avantages	Faiblesses
<p>Node.js avec Express</p> 	<ul style="list-style-type: none"> - Flexibilité : Grande liberté pour structurer les applications et choisir les outils - Performance : Asynchrone et événementiel, adapté aux applications à fort trafic - Écosystème : Richesse de modules via npm ¹. - JavaScript : Utilisation d'un seul langage pour le front-end et le back-end 	<ul style="list-style-type: none"> - Manque de structure : Nécessite des décisions architecturales importantes - Gestion de l'état : Peut devenir complexe pour les applications très grandes - Code Standard : Peut nécessiter plus de code pour des fonctionnalités de base
<p>Django</p> 	<ul style="list-style-type: none"> - Solution complète : Inclut des fonctionnalités intégrées comme l'authentification et l'ORM - Sécurité : Bonne gestion des aspects de sécurité comme la protection contre les attaques CSRF - Productivité : Développement rapide avec des conventions strictes et une forte intégration - Python : Utilisation d'un langage puissant et lisible 	<ul style="list-style-type: none"> - Monolithique : Peut être plus rigide avec une structure moins flexible - Performance : Peut être plus lourd comparé à des solutions plus légères - Courbe d'apprentissage : Peut être difficile à appréhender pour les débutants
Ruby on Rails	<ul style="list-style-type: none"> - Convention over Configuration : Favorise le 	<ul style="list-style-type: none"> - Performance : Moins performant pour des

¹ NPM : Node Package Manager

	<p>développement rapide grâce à des conventions strictes</p> <ul style="list-style-type: none"> - Productivité : Offre une grande rapidité de développement avec une syntaxe élégante - Communauté : Forte communauté avec de nombreuses gemmes (bibliothèques) - DRY (Don't Repeat Yourself) : Encourage la réutilisation du code 	<p>applications très évolutives</p> <ul style="list-style-type: none"> - Complexité : Peut devenir complexe avec la croissance de l'application - Consommation de ressources : Peut consommer plus de mémoire comparé à des frameworks plus légers
<p>Spring Boot</p> 	<ul style="list-style-type: none"> - Solution complète : Offre une configuration simplifiée et une intégration facile avec Spring Framework - Scalabilité : Adapté aux grandes entreprises et applications complexes - Sécurité : Intégration avec Spring Security pour des fonctionnalités de sécurité robustes - Écosystème Java : Bénéficie de l'écosystème Java et de ses outils 	<ul style="list-style-type: none"> - Complexité : Peut être complexe à configurer et à personnaliser - Courbe d'apprentissage : Peut être difficile à maîtriser pour les débutants - Lourdeur : Peut être plus lourd et consommateur de ressources par rapport à des solutions plus légères

c . Comparaison entre les Base de Données :

Tableau 5 : Tableau Comparatif des bases de données.



Base de données	Avantages	Inconvénients
<p>MongoDB</p> 	<ul style="list-style-type: none"> - Base de données NoSQL, flexible pour les données non structurées. - Scalabilité horizontale facile. - Gestion native des documents JSON. 	<ul style="list-style-type: none"> - Transactions complexes moins performantes - Pas de schéma strict, pouvant entraîner des incohérences de données
<p>PostgreSQL</p> 	<ul style="list-style-type: none"> - Système relationnel avec support ACID (transactions fiables) - Très robuste et extensible (types de données personnalisés) - Compatible avec les requêtes complexes et les procédures stockées 	<ul style="list-style-type: none"> - Plus lourd à configurer et à maintenir que MySQL - Moins rapide pour les lectures simples sur de grandes bases de données
<p>MySQL</p> 	<ul style="list-style-type: none"> - Très rapide pour les lectures et écritures simples - Communauté vaste et active - Facile à configurer et à utiliser 	<ul style="list-style-type: none"> - Moins adapté aux requêtes complexes - Moins de support pour les fonctionnalités avancées comparé à PostgreSQL
<p>Firebase</p> 	<ul style="list-style-type: none"> - Base de données en temps réel pour applications web et mobiles - Intégration native avec l'écosystème Google 	<p>Base de données NoSQL en temps réel proposée par Google, conçue pour la synchronisation de données instantanée et le</p>


	- Synchronisation en temps réel des données entre les utilisateurs	développement mobile.
--	--	-----------------------

d . Authentification et Autorisation :

Sécuriser l'application est une priorité. Plusieurs technologies peuvent être utilisées pour l'authentification et l'autorisation. Voici les options.

Tableau 6 : Technologies d'authentification et autorisation


Technologies	Avantages	Inconvénients
JSON Web Token (JWT) 	<ul style="list-style-type: none"> - Stateless (pas de stockage côté serveur) - Facile à utiliser pour des API RESTful - Peut contenir des informations supplémentaires (payload) dans le token 	<ul style="list-style-type: none"> - La taille du token peut devenir importante, ce qui peut affecter les performances - Pas de mécanisme intégré pour révoquer un token une fois émis
Open Authorization (OAuth) 	<ul style="list-style-type: none"> - Permet l'autorisation déléguée entre applications - Très utilisé pour les connexions via des services tiers (Google, Facebook, etc.) - Séparation claire entre l'authentification et l'autorisation 	<ul style="list-style-type: none"> - Complexe à implémenter (nécessite plusieurs étapes d'autorisation) - Peut poser des problèmes de sécurité si mal configuré
Session-based Auth	<ul style="list-style-type: none"> - Approche simple et efficace pour la gestion des utilisateurs connectés - Stockage sécurisé des 	<ul style="list-style-type: none"> - Nécessite du stockage côté serveur (peut devenir lourd avec de nombreux utilisateurs)

	sessions côté serveur - Les sessions peuvent être invalidées facilement	- Moins adapté pour des applications distribuées et des API REST
---	--	--

e . Communication en Temps Réel :

La communication en temps réel est essentielle pour certaines fonctionnalités de l'application comme le chat et les notifications. Voici les technologies possibles :

Tableau 7 : Tableau comparatif des technologies de communication en temps réel

Technologies	Avantages	Faiblesses
Socket.io 	- Basé sur WebSocket, permet une communication bidirectionnelle en temps réel - Prend en charge la détection de déconnexions et les reconnections automatiques - Compatible avec de nombreux langages backend	- Nécessite une infrastructure serveur dédiée - Peut être plus complexe à mettre en place pour des applications à grande échelle
Firebase Realtime Database 	- Synchronisation en temps réel des données entre les utilisateurs - Scalabilité facile via l'infrastructure de Google - Facile à configurer avec des SDKs pour mobile et web	- Dépendance à Google (vendor lock-in) - Moins de contrôle sur la logique côté serveur - Moins flexible pour des cas d'utilisation complexes
Pusher 	- Facile à intégrer avec des bibliothèques front-end et back-end - Très performant pour les notifications push et les mises à jour	- Modèle de tarification basé sur l'utilisation (peut devenir coûteux avec de grandes quantités)

	en temps réel - Support des événements et canaux privés	d'événements) - Dépendance à un service tiers
--	--	--

3 . Choix final des technologies ainsi que leurs liaisons :

Pour le développement de cette application, j'ai opté pour un stack technologique moderne et performante, composée de **React** pour le front-end, **Node.js** pour le back-end, **MongoDB** pour la base de données, **JWT (JSON Web Token)** pour l'authentification et l'autorisation, et **Socket.io** pour la communication en temps réel. Ces choix s'articulent autour de la nécessité de créer une application web robuste, réactive, évolutive, et sécurisée.

a . React (Front-End)

React est une bibliothèque JavaScript populaire pour la création d'interfaces utilisateur dynamiques et interactives. Il permet de construire des composants réutilisables et de gérer efficacement l'état de l'application avec une architecture orientée vers la performance. React s'intègre parfaitement avec les API du back-end, en particulier celles basées sur Node.js, facilitant ainsi la création d'applications à page unique (SPA) offrant une expérience fluide et rapide pour les utilisateurs.

b . Node.js (Back-End)

Node.js est un environnement d'exécution JavaScript orienté serveur. Il est particulièrement performant pour les applications nécessitant une gestion intensive d'entrées/sorties, telles que les applications en temps réel. Grâce à sa nature non-bloquante et asynchrone, Node.js peut traiter un grand nombre de requêtes simultanément, ce qui le rend idéal pour gérer l'interaction entre le front-end et la base de données MongoDB, ainsi que pour intégrer des fonctionnalités comme l'authentification avec JWT et la communication en temps réel avec Socket.io.

c . MongoDB (Base de Données)

MongoDB est une base de données NoSQL qui stocke les données sous forme de documents JSON. Elle est particulièrement adaptée pour les applications modernes qui nécessitent de la flexibilité dans la structure des données. MongoDB s'intègre facilement avec Node.js grâce à Mongoose, un ODM (Object Data Modeling) qui simplifie les opérations CRUD (Create, Read, Update, Delete) et offre une gestion efficace des collections. De plus, MongoDB permet une scalabilité horizontale, ce qui est essentiel pour les applications qui évoluent rapidement en volume de données.

d . JWT (Authentication et Autorisation)

Pour garantir la sécurité des échanges de données, j'ai choisi d'utiliser JWT pour l'authentification et l'autorisation. Les tokens JWT permettent de gérer les sessions de manière stateless (sans stockage côté serveur), ce qui réduit la charge sur le serveur et améliore les performances. Lorsqu'un utilisateur se connecte, un token est généré et stocké côté client, permettant à chaque requête subséquente d'inclure ce token pour vérifier l'identité de l'utilisateur et ses droits d'accès.

e . Socket.io (Communication en Temps Réel)

Pour gérer la communication en temps réel, Socket.io est une solution puissante et facile à intégrer avec Node.js. Socket.io permet d'établir des connexions bidirectionnelles entre le client et le serveur, ce qui est crucial pour les fonctionnalités telles que les notifications en temps réel, les mises à jour instantanées de données ou les chats. Il offre une gestion efficace des événements, avec une faible latence, tout en assurant une gestion automatique des connexions et reconnections.

f . Liaison des Technologies

Ces technologies s'intègrent de manière fluide pour créer une architecture complète et cohérente :

- **React** communique avec le **back-end Node.js** via des API REST et des WebSockets pour les interactions en temps réel. Les données sont récupérées et envoyées selon les besoins de l'interface utilisateur.

- **Node.js** traite les requêtes entrantes, gère la logique métier et interagit avec la base de données **MongoDB** pour stocker ou récupérer les informations nécessaires.
- L'authentification et l'autorisation des utilisateurs sont gérées par **JWT**, garantissant la sécurité des échanges entre le front-end et le back-end.
- **Socket.io** assure la communication en temps réel pour les fonctionnalités nécessitant des mises à jour instantanées, comme le chat, les notifications, ou la collaboration en temps réel.

Ce choix technologique permet ainsi de créer une application moderne, rapide, évolutive et sécurisée, adaptée aux besoins des utilisateurs en termes d'interactivité et de performance.

Chapitre 3 : Conception de l'Application

La conception de l'application est une étape primordiale dans le développement de tout projet logiciel. Elle vise à définir l'architecture, les composants principaux, et les interactions entre ces composants pour répondre aux besoins identifiés dans l'analyse des besoins. Dans ce chapitre, nous détaillerons les choix architecturaux, la modélisation des données, ainsi que l'organisation des différentes couches de l'application.

1 . Architecture Globale de l'Application

L'application adopte une **architecture 3-tiers**, qui sépare distinctement les trois couches principales : l'interface utilisateur (frontend), la logique métier (backend), et la couche de données. La communication entre l'interface utilisateur et la logique métier est assuré par **une API RESTful**, facilitant un échange fluide des données. La couche de données, quant à elle, gère le stockage et la récupération des informations et communique avec la logique métier pour assurer la persistance des données.

a . Client (Frontend) :

- Technologies utilisées : React, Typescript.
- Description : Le frontend est responsable de la présentation des données et de l'interaction utilisateur. Les actions réalisées par l'utilisateur, telles que la création

de projets, la gestion des tâches, et l'assignation des sprints, sont envoyées au serveur via des requêtes API.

b . API RESTful :

- **Technologies utilisées** : Express.js pour la création des routes et endpoints.
- **Description** : L'API est le point d'entrée principal pour la communication entre le frontend et le backend. Elle reçoit les requêtes du client, exécute les actions appropriées sur le serveur (récupérer des données, les modifier ou les supprimer) et renvoie les réponses sous forme de données JSON.
- **Méthodes API** : Les méthodes HTTP sont utilisées pour gérer les différentes opérations :
 - **GET** : Récupérer des données (par exemple, la liste des tâches ou projets).
 - **POST** : Ajouter des données (comme une nouvelle tâche).
 - **PUT** : Mettre à jour des données existantes (modifier une tâche).
 - **DELETE** : Supprimer des données (supprimer un projet ou une tâche).
- **Sécurité** : L'API est protégée par des tokens JWT (JSON Web Token) pour sécuriser les accès aux endpoints sensibles.

c . Serveur (Backend) :

- **Technologies utilisées** : Node.js, Express.js.
- **Description** : Le backend gère la logique métier, traite les requêtes reçues via l'API, et interagit avec la base de données pour stocker ou récupérer les informations nécessaires.

d . Base de Données :

- Technologie utilisée : MongoDB.
- Description : La base de données NoSQL stocke toutes les informations relatives aux projets, sprints, tâches, utilisateurs, etc. Le backend interagit avec la base de données pour récupérer ou modifier les données en fonction des requêtes reçues via l'API.

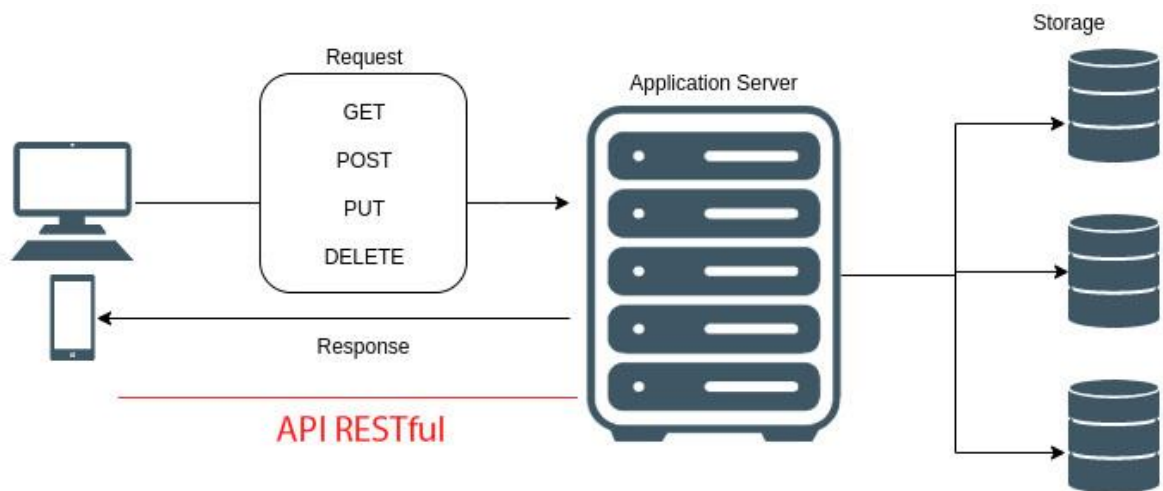


Figure 11 : Illustration de l'architecture 3-tiers

2 . Modélisation globale du diagramme de cas d'utilisation :

Le diagramme de cas d'utilisation illustre les interactions entre les **acteurs** (product owner, scrum master, membre.) et le système. C'est une vue d'ensemble des fonctionnalités principales de l'application et des utilisateurs impliqués.

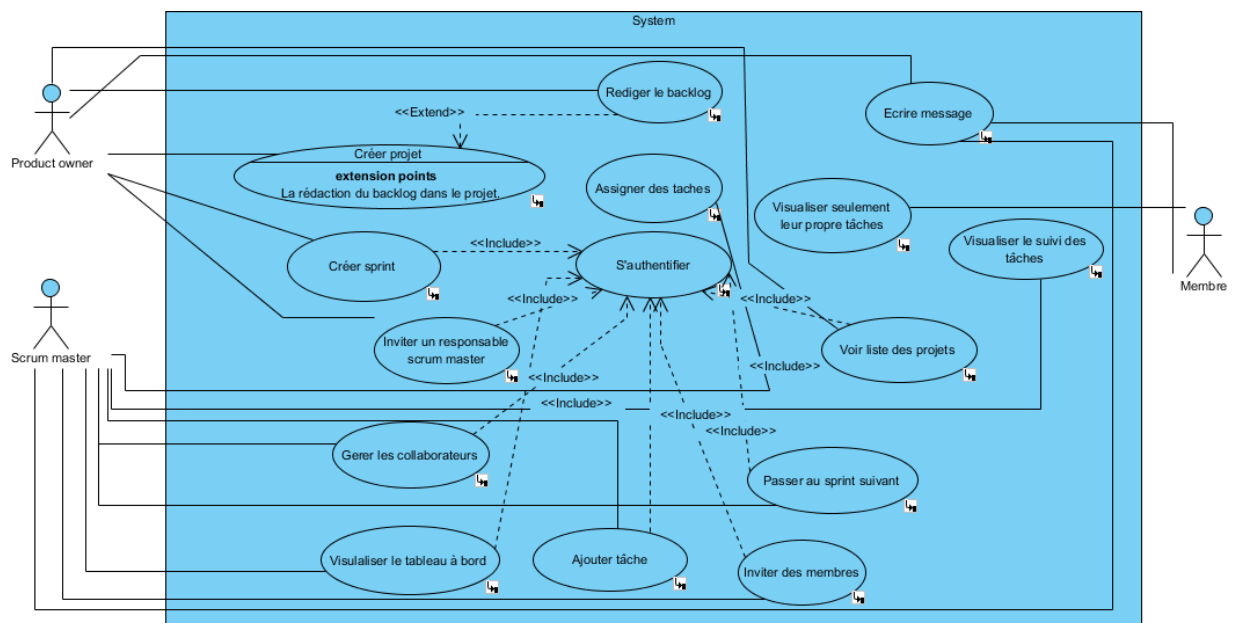


Figure 12: Diagramme de cas d'utilisation global du projet

3 . Modélisation des Données

La modélisation des données est essentielle pour structurer et organiser les informations de manière efficace. Dans cette application, MongoDB est utilisé pour stocker les données sous forme de documents.

a . Les principales collections de la base de données :

Projets : Cette collection stocke les informations relatives aux projets.

Attributs principaux : id, name, description, startDate, endDate.

Utilisateurs : Cette collection gère les informations des utilisateurs, y compris leurs rôles.

Attributs principaux : id, username, firstname, lastname, email, image, role, password, idProject.

Rôles : Cette collection définit les rôles disponibles dans l'application.

Attributs principaux : id, name.

Notifications : Les notifications permettent de notifier les utilisateurs des événements importants, tels que l'approche des dates limites.

Attributs principaux : id, idProject, message, read.

Sprints : Les sprints sont des périodes de temps spécifiques durant lesquelles certaines tâches doivent être accomplies. Cette collection contient des informations sur chaque sprint, son association avec un projet.

Attributs principaux : id, idProject, name, startDate, endDate, column[].

Columns : Les columns gèrent les tâches à réaliser dans un sprint. Elles sont fondées par quatre colonnes par défaut : A faire, En cours, Terminé.

Attributs principaux : id, name, cards[].

Cards : Les cartes représentent les actions ou les tâches à réaliser dans un sprint. Elles sont associées à des utilisateurs et sont suivies dans le cadre du suivi du temps.

Attributs principaux : id, title, description, attachment, assignee, dueDate, progress

Backlogs : Le backlog contient les listes tâches assignées à un sprint spécifique.

Attributs principaux: id, epic, priority, task, userStory, sprint, idProject.

Messages : Cette collection gère les messages échangés entre les utilisateurs.

Attributs principaux: id, username, room, text.

Invitations : Les invitations sont utilisées pour inviter des utilisateurs à rejoindre le projet.

Attributs principaux: id, mail, idProject, role.

b . Diagramme UML des classes :

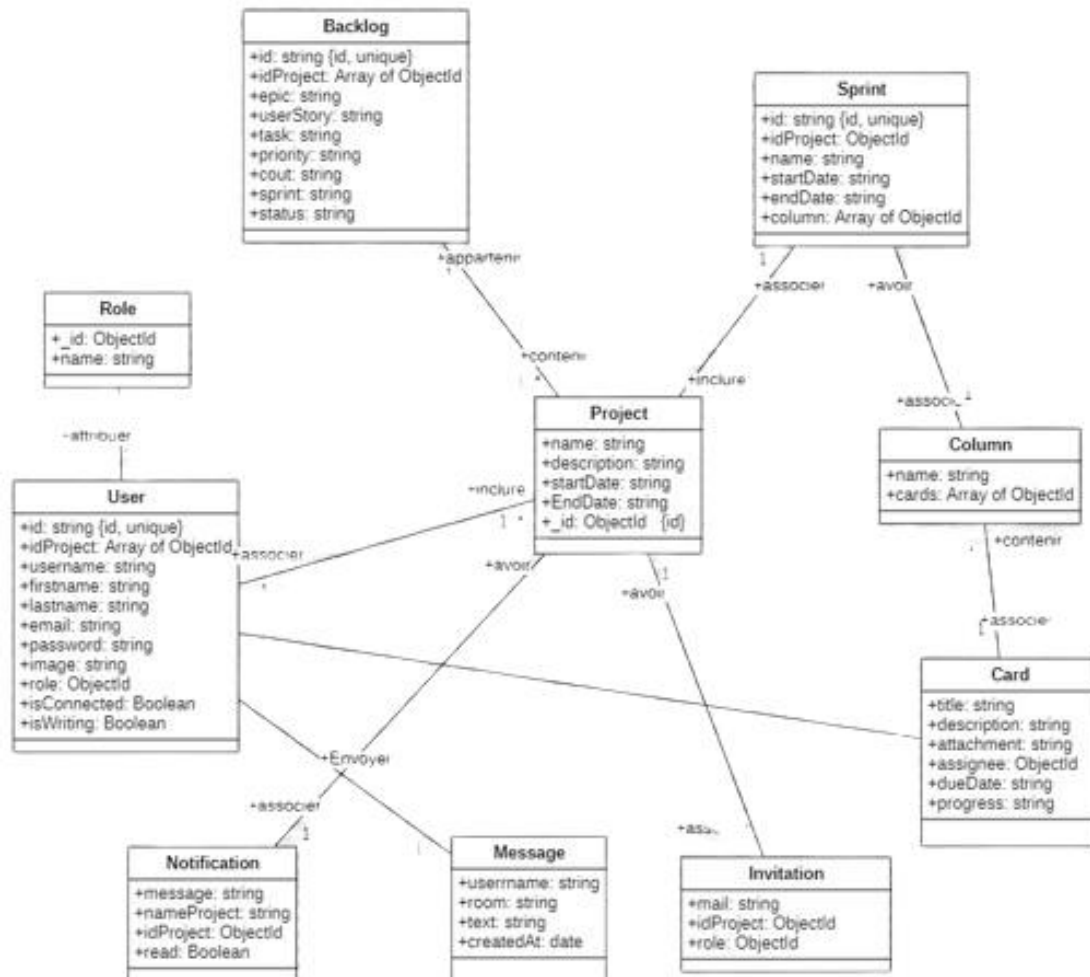


Figure 13 : Diagramme des classes de l'application de la gestion de projet

c . Diagramme UML de séquence

Ce diagramme montre **comment les objets interagissent entre eux au fil du temps**. Il est utile pour visualiser le déroulement de chaque processus spécifique.

- S'authentifier

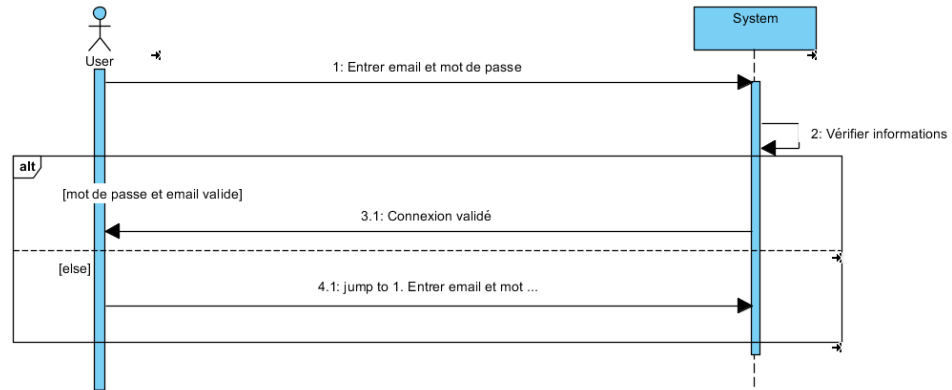


Figure 14: Diagramme de séquence de l'authentification

- « Créer projet »

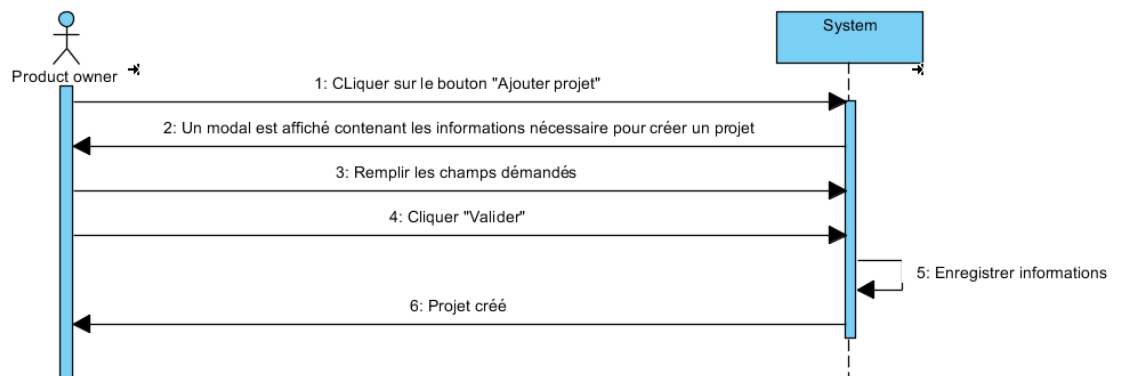


Figure 15 : Diagramme de séquence de Création d'un projet

- « Rediger backlog »

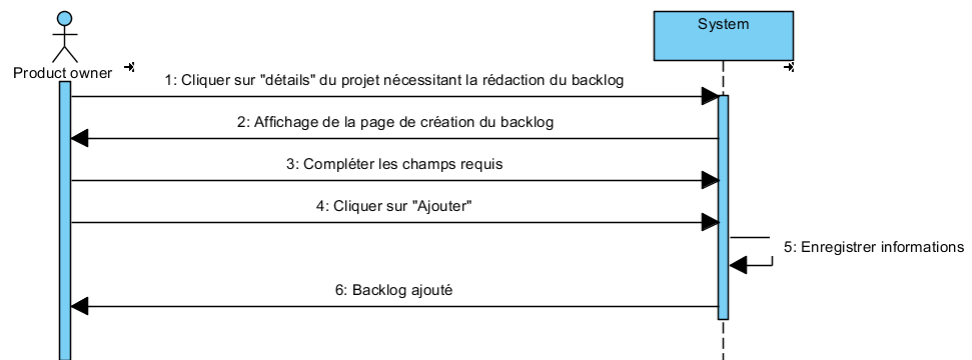


Figure 16: Diagramme de séquence de rédaction du backlog

- « Créer sprint »

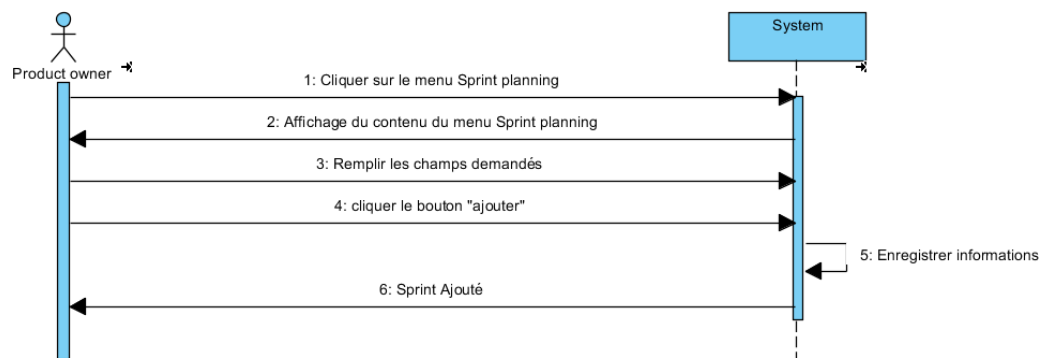


Figure 17: Diagramme de séquence de création de sprint

- « Inviter un responsable scrum master »

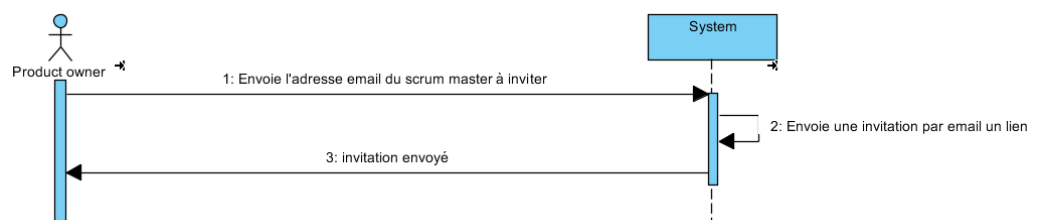


Figure 18 : Diagramme de séquence d'invitation de responsable scrum master

- « Gerer les collaborateurs »

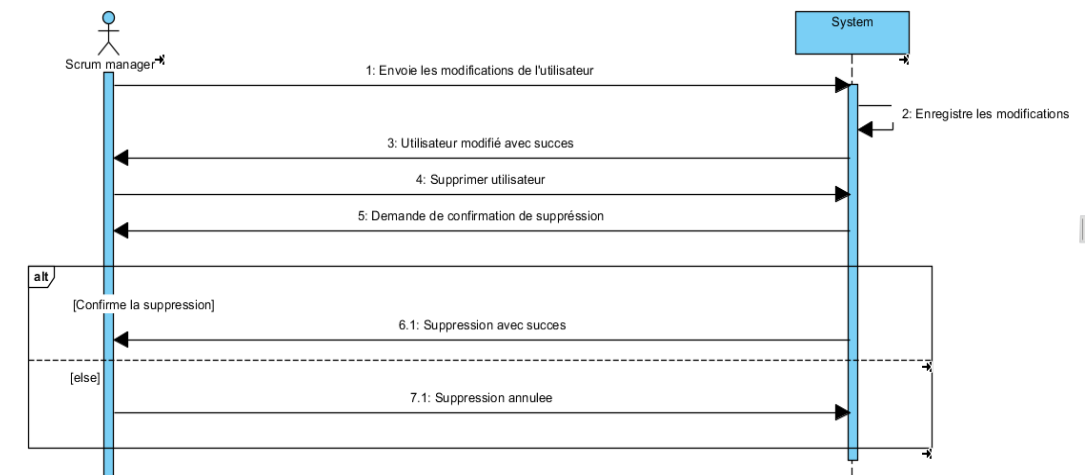


Figure 19 : Diagramme de séquence de la gestion des collaborateurs

- « Ajouter tache »

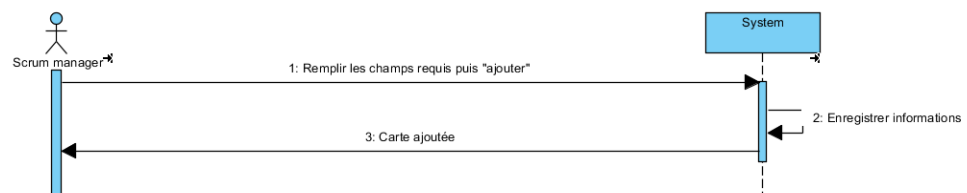


Figure 20 : Diagramme de séquence d'ajout des taches

- « Visualiser le tableau à bord »

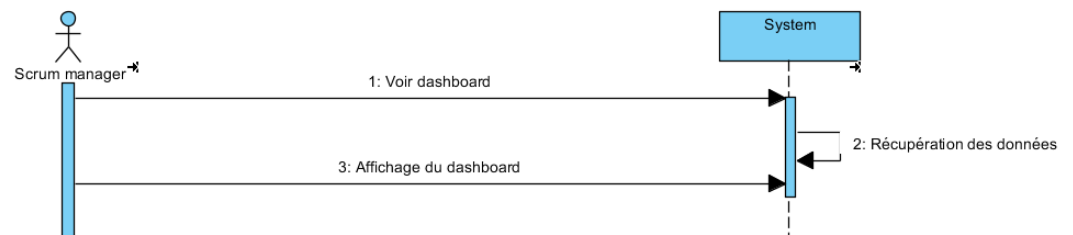


Figure 21 : Diagramme de séquence de la visualisation de tableau à bord

- « Inviter des membres »

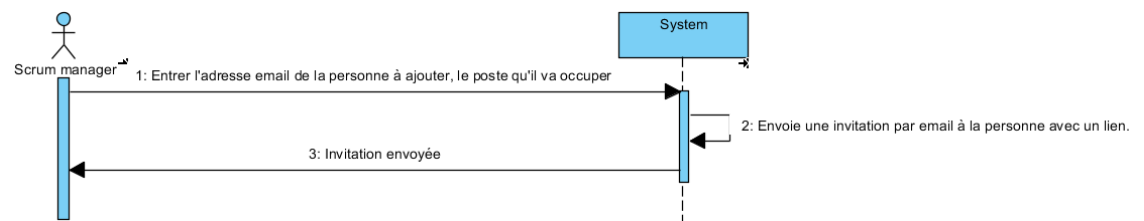


Figure 22 : Diagramme de séquence d'invitations des membres

- « Passer au sprint suivant »

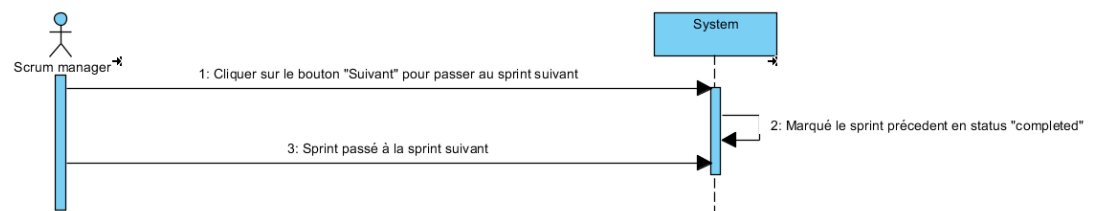


Figure 23 : Diagramme de séquence du passage au sprint suivant

- « Assigner des taches »

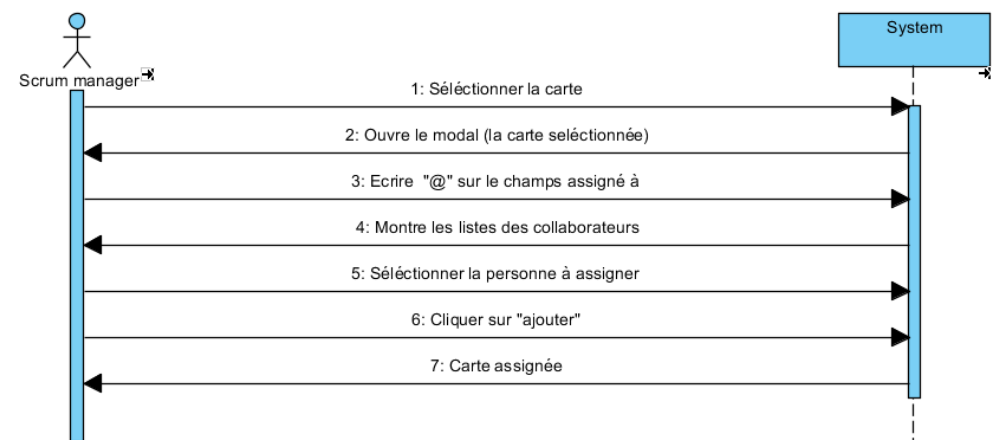


Figure 24 : Diagramme de séquence de l'assignation des tâches

- « Voir listes des projets »

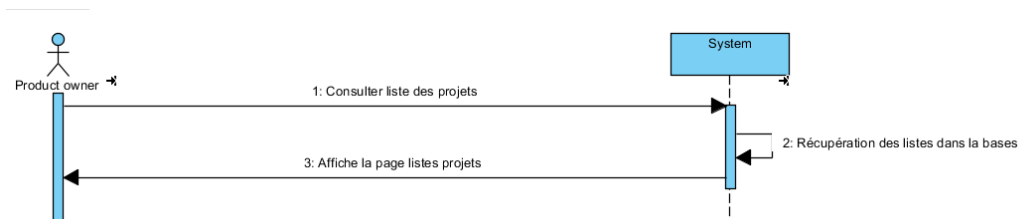


Figure 25 : Diagramme de séquence de la visualisation des listes des projets

- « Visualiser le suivi des tâches »

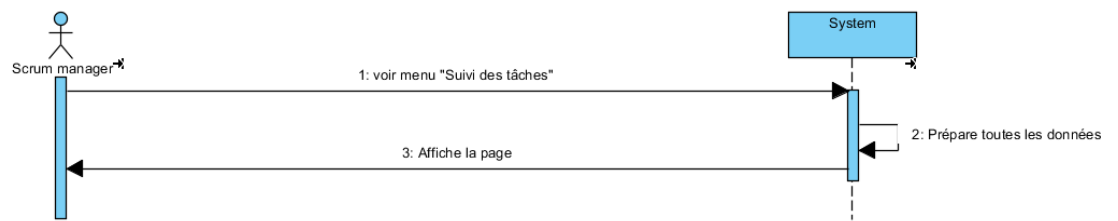


Figure 26 : Diagramme de séquence pour la visualisation de suivi des tâches

- « Visualiser seulement leur propre tâches »

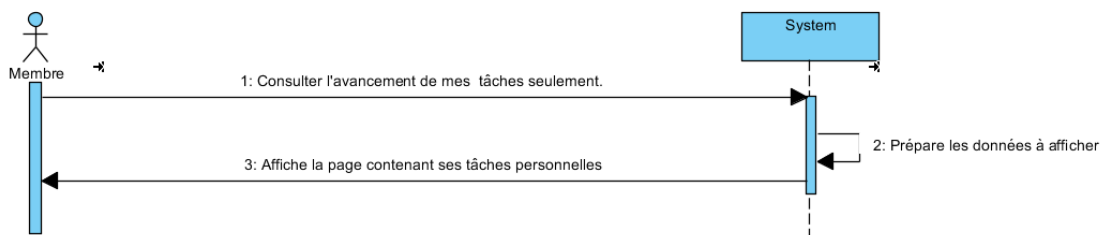


Figure 27 : Diagramme de séquence pour la le suivi des taches personnelles

- « Envoyer message »

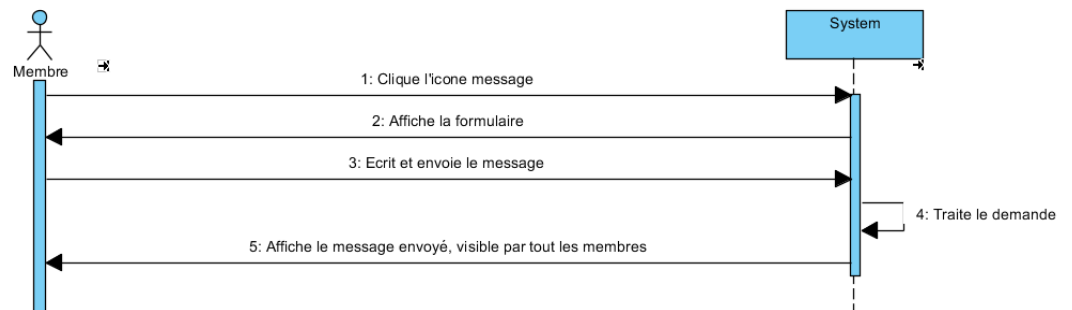


Figure 28 : Diagramme de séquence pour l'envoi des messages

4 . Interaction entre les Composants

Les interactions entre les différentes couches de l'application suivent un flux bien défini :

1. **Interaction Utilisateur-Frontend** : L'utilisateur interagit avec l'application via une interface réactive et intuitive. Chaque action de l'utilisateur déclenche des appels API pour récupérer ou manipuler des données.

2. **Communication Frontend-Backend** : Les appels API RESTful sont utilisés pour envoyer des requêtes au serveur. Les requêtes incluent des actions telles que la création de projets, l'ajout de tâches au backlog, l'assignation de tâches à un sprint, etc.
3. **Traitement des Requêtes par le Backend** : Le backend, via Express.js, traite les requêtes, effectue les opérations nécessaires sur la base de données, et renvoie une réponse au frontend.
4. **Gestion des Sessions et de la Sécurité** : Lors de chaque requête nécessitant une authentification, le backend vérifie les tokens JWT pour s'assurer que l'utilisateur est autorisé à effectuer l'action demandée.

5 . Sécurité de l'Application

La sécurité est un aspect critique, particulièrement dans la gestion des données sensibles des utilisateurs et des projets. L'application implémente plusieurs mécanismes de sécurité :

1. **Authentification par JWT** : Chaque utilisateur se connecte à l'application en utilisant un jeton JWT qui authentifie ses actions tout au long de la session.
2. **Expiration des Sessions** : Pour prévenir les accès non autorisés, les sessions utilisateur expirent automatiquement après une période d'inactivité. Les utilisateurs devront se reconnecter pour continuer à utiliser l'application.
3. **Contrôles d'Accès** : Les utilisateurs n'ont accès qu'aux fonctionnalités qui correspondent à leur rôle (Product Owner, Scrum Master, Développeur, Testeur). Par exemple, un développeur ne pourra pas accéder aux sections réservées aux Product Owners.

Chapitre 4 : Implémentation

1 . Configuration et Installation

Pour mettre en place l'environnement de développement, les étapes suivantes ont été suivies :

a . Installation des outils de développement :

- **Node.js** a été installé pour exécuter JavaScript côté serveur et gérer les dépendances du projet avec npm².
- **npm** est utilisé pour installer et gérer les « package » nécessaires au projet.

b . Configuration du projet front-end avec React :

- Un nouveau projet React a été créé en utilisant la commande suivante :

```
npx create-react-app projectManagement
```

Figure 29: commande pour créer une application react

- Une fois dans le répertoire du projet, les « package » nécessaires ont été installés :

```
npm install @mui/material @emotion/react @emotion/styled zustand axios
```

Figure 30 : Commande utilisée pour l'installation des "package" nécessaires

- Ces « package » incluent **Material-UI** pour les composants visuels, **Zustand** pour la gestion des états, et **axios** pour les requêtes HTTP³.

c . Configuration du projet back-end avec Node.js et Express :

- Le projet back-end a été initialisé avec npm :

```
npm init -y
```

Figure 31: Commande pour la création d'un projet nodeJS

- Les packages nécessaires pour le back-end ont été installés

² NPM : Node Package Manager

³ HTTP : Hypertext Transfer Protocol

```
npm install express mongoose cors body-parser
```

Figure 32 : Commande utilisée pour l'installation des "package" pour le développement backend

- Ces packages incluent **Express** pour le framework serveur, **Mongoose** pour l'interaction avec MongoDB, **cors** pour la gestion des requêtes cross-origin, et **body-parser** pour le traitement des corps de requêtes HTTP⁴.

2 . Développement du Back-end :

Pour le développement du back-end de l'application, **Node.js** avec le framework **Express** a été utilisé. Cette combinaison permet une gestion efficace des requêtes HTTP et une intégration fluide avec MongoDB.

- **Création des API** ⁵: Des API RESTful ont été conçues pour permettre la communication entre le front-end et le back-end. Ces API⁶ incluent des endpoints pour gérer les projets, les utilisateurs, les sprints, les colonnes, les cartes, les backlogs, les rôles, les notifications, les messages, et les invitations. Les routes sont sécurisées et permettent des opérations CRUD (Créer, Lire, Mettre à jour, Supprimer) pour chaque entité.
- **Gestion des données avec MongoDB** : MongoDB est utilisé pour stocker les données sous forme de documents JSON. Nous avons défini les modèles Mongoose pour chaque collection (Projets, Utilisateurs, Sprints, etc.) afin de structurer et valider les données. Les relations entre les collections sont gérées via des références, permettant de lier les utilisateurs aux projets et les cartes aux sprints.
- **Contrôleurs** : Les contrôleurs sont responsables de la logique métier de l'application. Ils traitent les requêtes entrantes, interagissent avec la base de données, et renvoient les réponses appropriées au front-end. Par exemple, le NotificationController gère la création et l'envoi de notifications, tandis que le MessageController s'occupe de la gestion des messages entre utilisateurs.

⁴ HTTP : Hypertext Transfer Protocol

⁵ API : Application Programming Interface

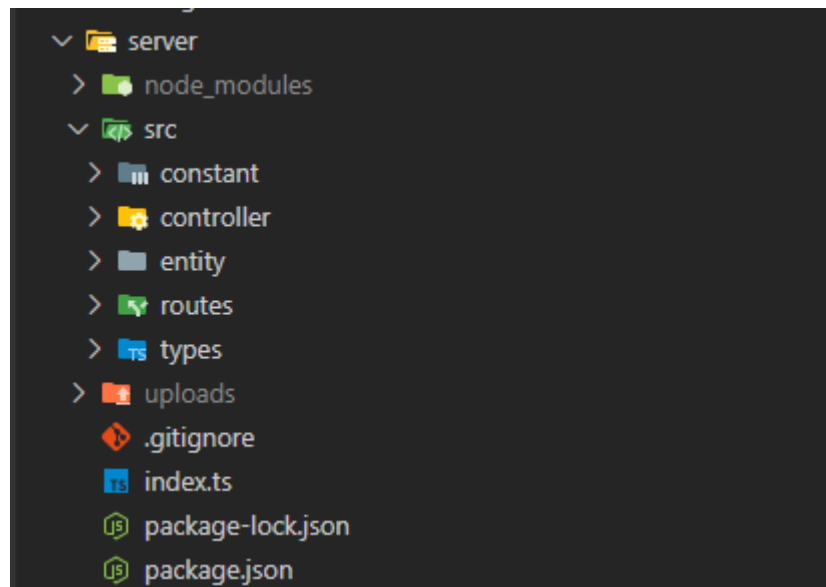


Figure 33 : Structure du projet côté serveur

3 . Développement du Front-end

Pour le développement du front-end, **React** a été utilisé pour construire l'interface utilisateur, et **Material-UI** a été choisi pour les composants visuels. **Zustand** a été intégré pour la gestion des états.

- **Composants React** : Les principaux composants React incluent les vues pour la gestion des projets, des sprints, des backlogs, suivis de tâches, et des utilisateurs. Chaque composant est responsable de l'affichage des données et de la gestion des interactions utilisateur. Les composants sont organisés de manière modulaire pour faciliter la maintenance et l'évolution de l'application.
- **Gestion des états avec Zustand** : **Zustand** a été utilisé pour la gestion des états globaux de l'application. **Zustand** permet une gestion centralisée des états avec une API⁷ simple et intuitive. Les magasins Zustand sont utilisés pour gérer des états comme les informations sur l'utilisateur connecté, les données des projets, et l'état des notifications. Cela simplifie le partage des états entre différents composants et améliore la performance en évitant des rendus inutiles.
- **Intégration avec les API** : Le front-end communique avec le back-end via des appels API REST. Nous avons utilisé axios pour effectuer les requêtes HTTP⁸ et

⁷ API : Application Programming Interface

⁸ HTTP : Hypertext Transfer Protocol

gérer les réponses. Les données récupérées sont stockées dans les états React et utilisées pour mettre à jour l'interface utilisateur en temps réel.

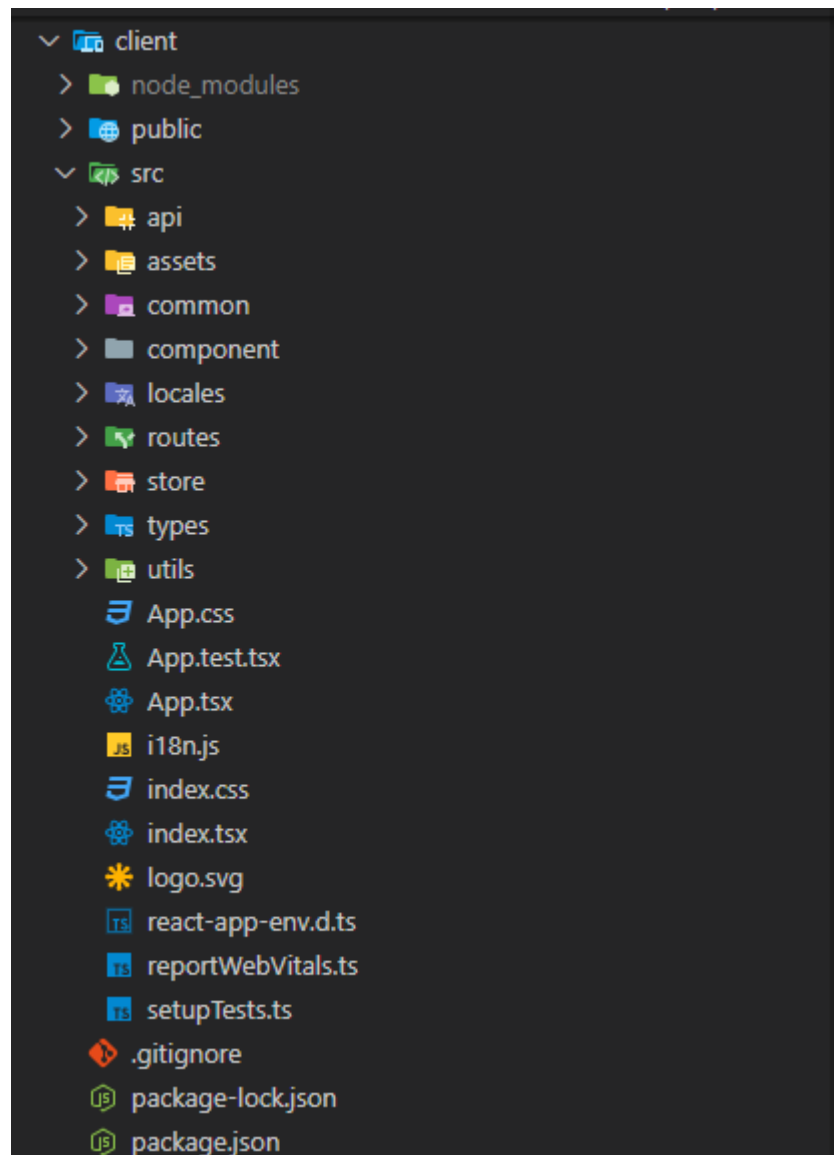


Figure 34 : Structure du projet React côté client

4 . Gestion des versions avec Git

Git a été utilisé tout au long du développement pour assurer une gestion efficace des versions de l'application. Cet outil de contrôle de version a permis de structurer le projet, de suivre les modifications, et de garantir la stabilité du code tout au long du processus de développement.

a . Utilisation de Git dans le projet

Dès le début, un dépôt Git a été initialisé pour centraliser le code source. Voici les pratiques adoptées :

- Ajout et suivi des fichiers : Toutes les modifications ont été ajoutées au suivi avec la commande git add --all, garantissant que chaque changement pertinent était inclus dans l'historique du projet.
- Commits descriptifs : Chaque ensemble de modifications a été enregistré avec des messages clairs. Ces messages ont permis de garder un historique détaillé et compréhensible des changements.
- Push vers le dépôt distant : Une fois les commits effectués, les modifications ont été poussées sur un dépôt distant à l'aide de la commande git push, garantissant une sauvegarde sécurisée et accessible.

b . Avantages observés

L'utilisation de Git a offert plusieurs bénéfices :

- Suivi précis des versions, permettant de revenir à un état antérieur en cas de problème.
- Organisation et clarté grâce à l'historique des commits.
- Sauvegarde et partage du code via un dépôt distant, assurant une continuité du développement.

Chapitre 5 : Tests de l'application :

Le chapitre sur les tests et validation vise à garantir que l'application développée répond aux exigences fonctionnelles et non fonctionnelles définies. Cette étape cruciale assure non seulement le bon fonctionnement des différentes fonctionnalités, mais aussi leur intégration harmonieuse dans l'ensemble du système. Les tests ont été conçus pour identifier et corriger les bugs, vérifier la performance de l'application, et s'assurer que toutes les spécifications sont respectées.

1 . Tests Unitaires

Les tests unitaires ont été réalisés pour vérifier individuellement chaque fonctionnalité clé de l'application. Ces tests ont couvert :

- **Ajout des Backlogs, Projets, et Utilisateurs** : Chaque fonctionnalité a été testée isolément pour s'assurer qu'elle fonctionne correctement.
- **Gestion des Sprints** : Les tests ont vérifié la création, la modification, et la suppression des sprints.
- **Système de Chat** : Testé pour garantir l'envoi et la réception des messages sans erreurs.
- **Suivi du Temps des Tâches** : Validé pour s'assurer que les temps sont correctement enregistrés et affichés.
- **Notifications** : Testé pour vérifier que les utilisateurs reçoivent bien les notifications à chaque événement important, comme l'approche de la date de fin d'un sprint ou un nouveau message.

2 . Tests d'Intégration

Les tests d'intégration ont été effectués pour s'assurer que l'ensemble des fonctionnalités fonctionne harmonieusement ensemble. Cela a inclus :

- **Interaction entre le Frontend et le Backend** : Vérification que les données circulent correctement entre l'interface utilisateur et les services backend.
- **Gestion des Sessions** : Testé pour s'assurer que les sessions d'utilisateur expirent correctement pour des raisons de sécurité.
- **Intégration du Système de Notification et de Messagerie** : Vérification que les notifications sont envoyées et reçues en temps réel, et que le système de messagerie fonctionne de manière fluide.

Chapitre 6 : Présentation de l'application :

Dans cette section, nous allons présenter la fonctionnalité principale et quelques captures d'écrans des interfaces de l'application.

1 . Fonctionnalités principales :

- **Création de projet** : Seul le **Product Owner** peut créer un projet, en définissant les informations essentielles telles que le nom du projet et ses objectifs.
- **Gestion du Backlog** : Après la création du projet, le **Product Owner** rédige le backlog, qui constitue la liste des tâches et fonctionnalités à développer.
- **Rédaction des Sprints** : Le **Product Owner** planifie et rédige les sprints, en définissant les tâches à réaliser pour chaque sprint.
- **Gestion des rôles** : Le **Product Owner** invite un **Scrum Master** à gérer la suite du projet.
- **Suivi des tâches** : Le **Scrum Master** prend en charge le suivi des tâches, en veillant à leur progression et à leur attribution aux membres des équipes.
- **Gestion des équipes** : Le **Scrum Master** invite des membres à rejoindre les équipes, en les affectant aux tâches et en suivant leur avancement.
- **Changement de sprint** : Le **Scrum Master** gère le passage d'un sprint à l'autre. Un PDF récapitulatif est généré pour chaque sprint clôturé, fournissant un aperçu des tâches effectuées et des résultats obtenus.
- **Envoi de messages** : Tous les utilisateurs (Product Owner, Scrum Master, équipes) peuvent envoyer des messages pour faciliter la communication entre les membres du projet.
- **Notifications** : Des notifications automatiques sont envoyées pour informer les utilisateurs des actions importantes, telles que la création de nouvelles tâches, l'avancement des sprints, ou l'ajout de membres à une équipe.

2 . Ecran de l'application

La figure 35 montre la page d'authentification de l'application.



Figure 35 : Page d'authentification

La figure ci-dessous présente la liste des projets existants ainsi que la possibilité de créer un nouveau projet. Cette interface est visible uniquement par le Product Owner, qui est le seul autorisé à ajouter un nouveau projet.

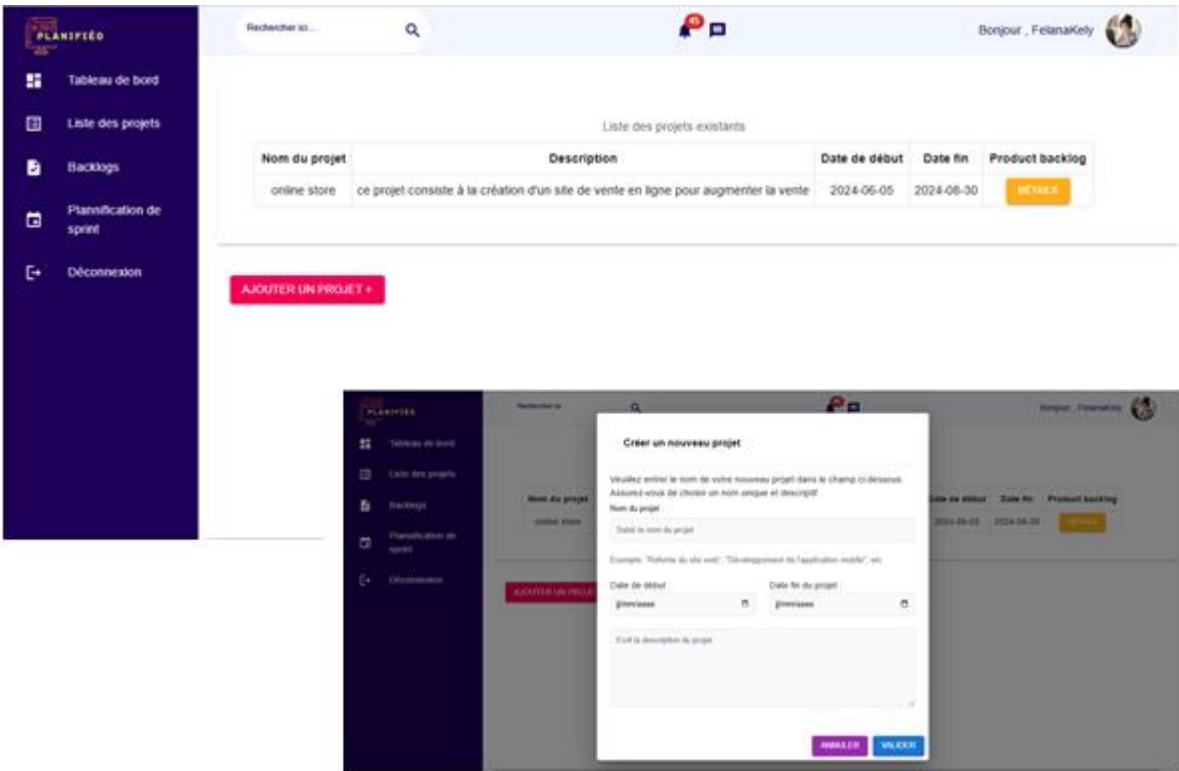


Figure 36 : Interface pour la création d'un nouveau projet et les projets déjà existants

La figure 37 suivante montre la page de rédaction du backlog, accessible uniquement au product owner.

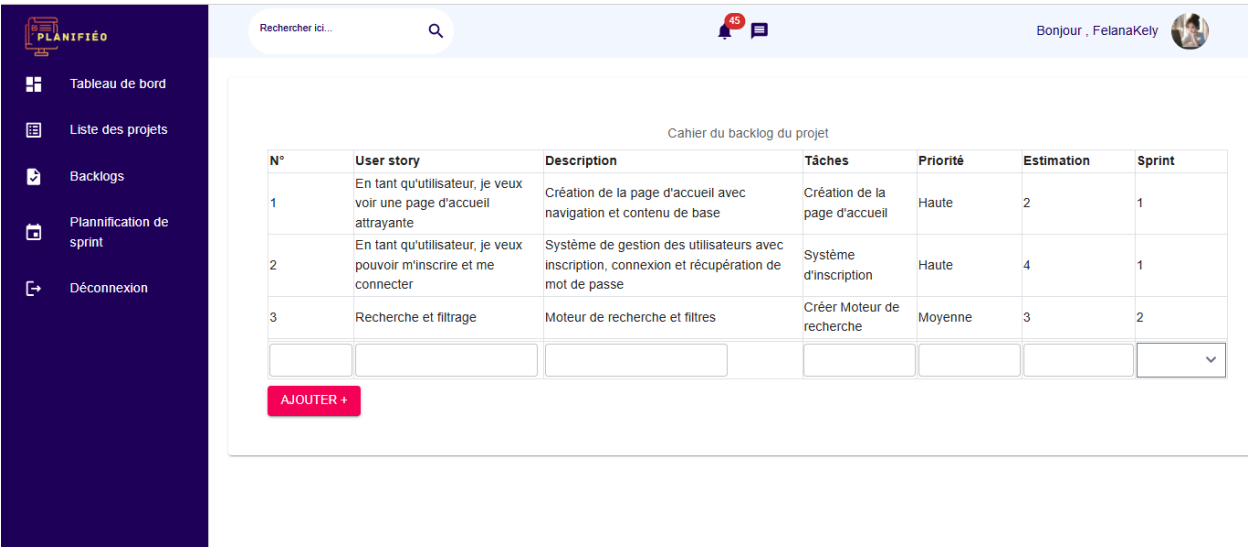


Figure 37 : Rédaction du backlog du projet

La figure 38 montre l'interface de planification des sprints. Seul le product owner est autorisé à ajouter, modifier ou supprimer un sprint, tandis que le scrum master peut uniquement consulter les sprints sans possibilité de les modifier ou de les supprimer.

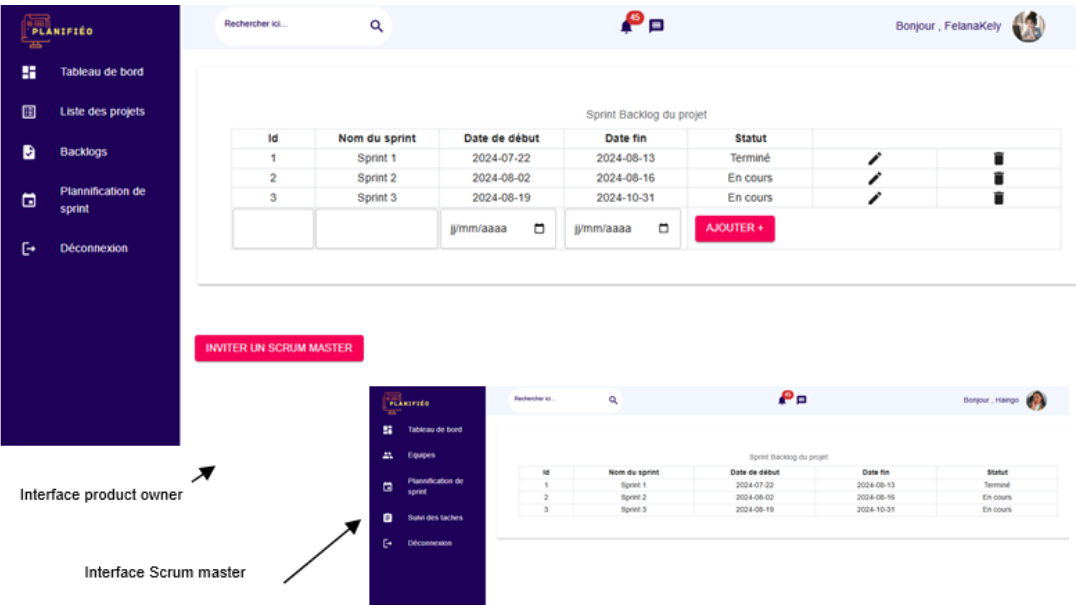


Figure 38 : Page de la planification de sprint

La figure 39 montre l'interface de gestion des équipes, qui figure uniquement dans le menu du Scrum master, car c'est son rôle de gérer les équipes.

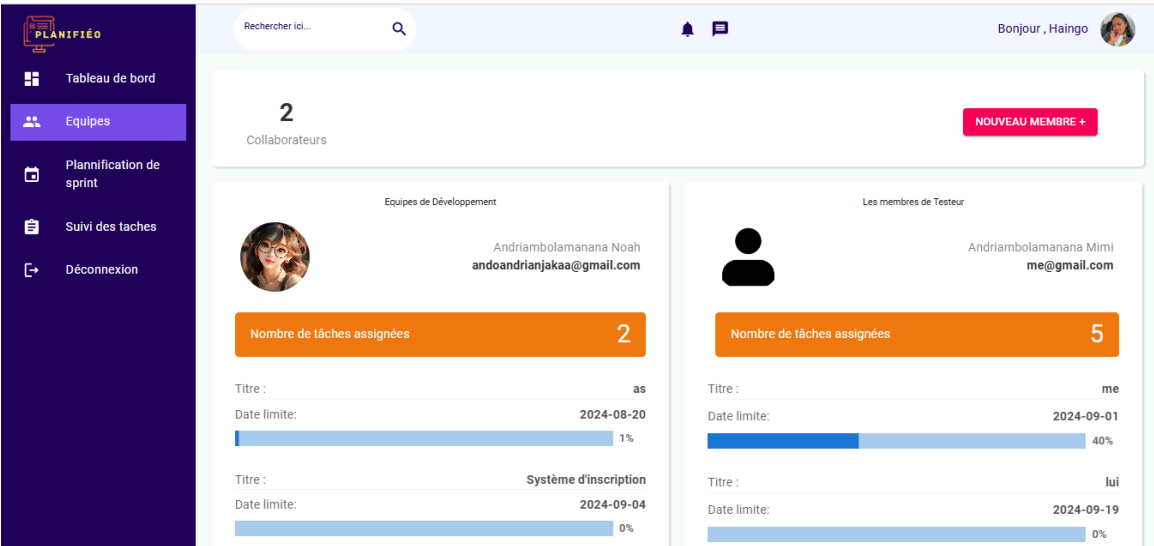


Figure 39 : Page de la gestion des équipes

La figure 40 montre le tableau de bord de l'application, qui se divise en deux versions distinctes selon le rôle de l'utilisateur. Le product owner se concentre sur le suivi global du projet, incluant la gestion des ressources humaines, des tâches accomplies et des heures de travail. En revanche, le scrum master se focalise principalement sur le suivi des sprints.

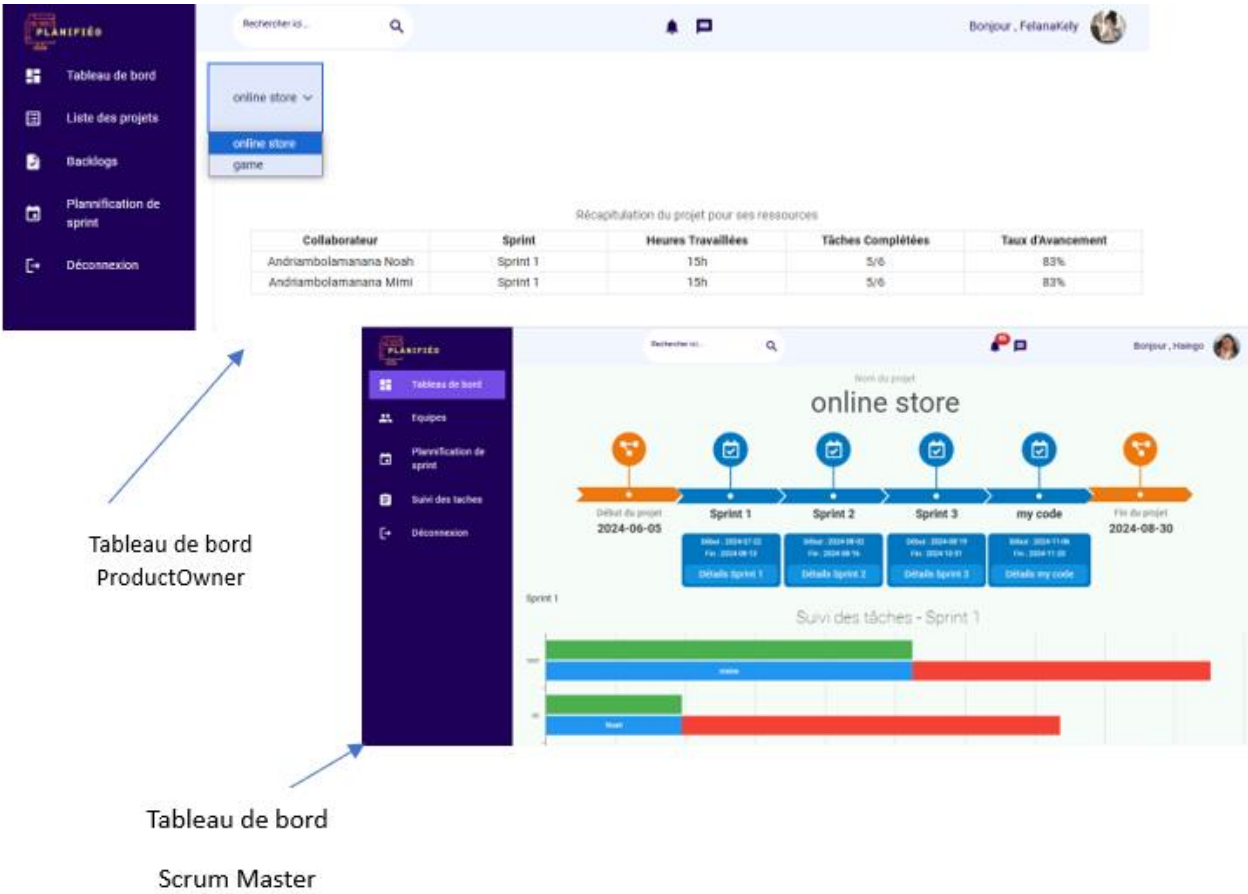


Figure 40 : Tableau de bord de l'application

La figure 41 ci-dessous présente la page de suivi des tâches de chaque sprint.

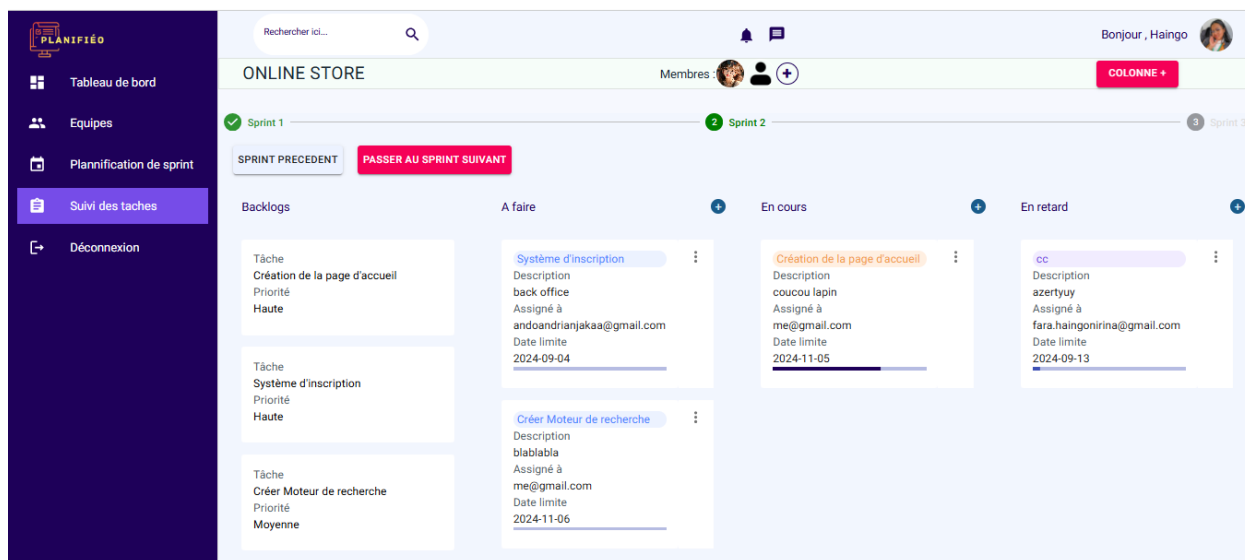


Figure 41 : Page de suivi des tâches

La figure suivante présente l'interface permettant de créer une tâche et de l'assigner à un membre.

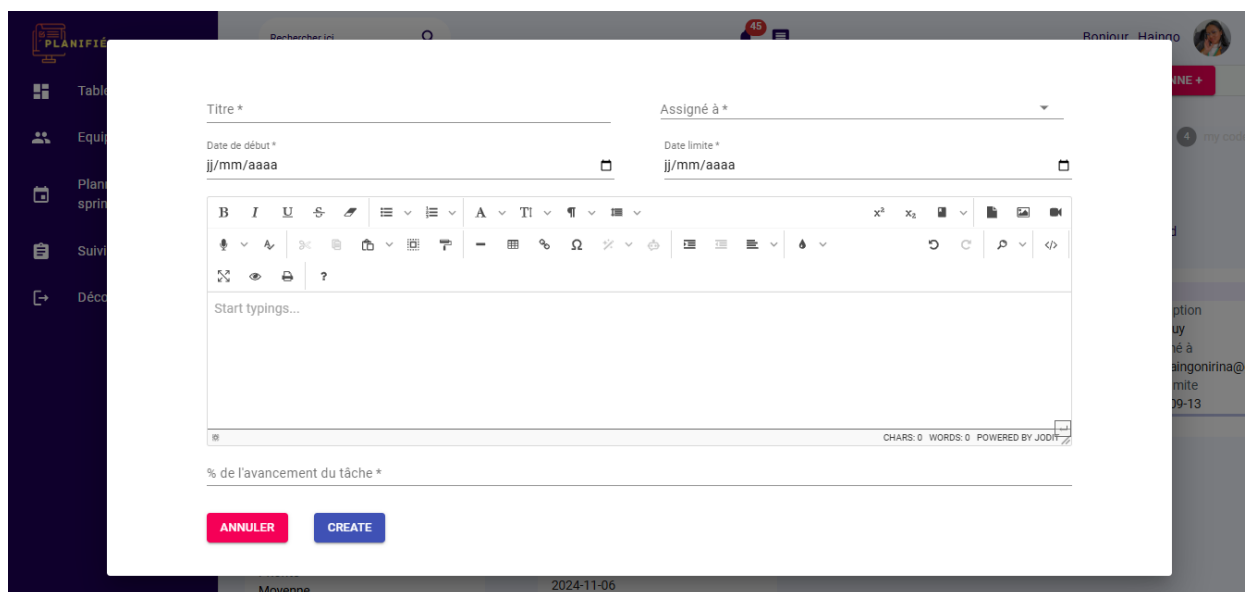


Figure 42 : Création d'une tâche

PARTIE IV : CONCLUSION GENERALE

La réalisation du projet a abouti à une application de gestion de projet agile qui répond avec précision aux besoins définis. Le choix de React pour le front-end a permis de créer une interface utilisateur réactive et intuitive. Le processus de développement a été marqué par une configuration rigoureuse, une intégration harmonieuse des fonctionnalités, et une validation exhaustive par des tests unitaires et d'intégration, assurant ainsi la performance et la sécurité de l'application. En facilitant la visibilité sur l'avancement des projets et en améliorant la coordination entre les équipes, l'application se positionne comme une solution efficace pour la gestion de projet agile. Les développements futurs se concentreront sur l'enrichissement des fonctionnalités et l'adaptation continue aux besoins changeants, consolidant ainsi la pertinence et la robustesse de la solution dans un environnement dynamique. Le projet a posé des bases solides pour une gestion de projet agile optimisée, tout en offrant des perspectives prometteuses pour des améliorations continues.

REFERENCES WEBOGRAPHIQUES

- (1). <https://www.cnil.fr/fr/definition/interface-de-programmation-dapplication-api>, consulté le 15 août 2024.
- (2). <https://www.shecodes.io/athena/172357-what-is-axios-a-javascript-library-for-making-http-requests> consulté le 15 août 2024.
- (3). <https://www.hubvisory.com/fr/blog/qu-est-qu-un-backlog-comment-le-construire-et-le-gerer> consulté le 15 août 2024.
- (4). <https://datascientest.com/base-de-donnees-definition> consulté le 15 août 2024.
- (5). <https://fr.wikipedia.org/wiki/Framework> consulté le 15 août 2024.
- (6). <https://blog.harvestr.io/fr/product-backlog> consulté le 17 août 2024.
- (7). <https://www.reussirsesprojets.com/projet-definition-et-exemples/>, consulté le 25 Septembre 2024.
- (8). <https://www.reussirsesprojets.com/projet-definition-et-exemples/>, consulté le 25 Septembre 2024.
- (9). <https://www.manager-go.com/gestion-de-projet/glossaire/jalon>, consulté le 27 Septembre 2024
- (10). <https://www.wrike.com/fr/project-management-guide/les-fondements-de-la-methodologie-agile/>, consulté le 27 Septembre 2024

ANNEXES

Découvrez dans cette annexe des extraits pertinents de code source issus du projet **Planifiéo**, offrant un aperçu concis des choix de programmation et des fonctionnalités clés implémentées.

L'extrait de code ci-dessous concerne l'authentification, la gestion des sessions et le cryptage des mots de passe.

```
// Login Controller
login = async (req: Request, res: Response) => {
  try {
    const { email, password } = req.body;
    const checkUser = await User.findOne({ email: email });
    if (!checkUser) {
      return res.status(404).send("User not found !");
    }

    // Vérifie si le mot de passe fourni correspond au mot de passe haché stocké pour l'utilisateur.
    const checkPassword = await bcrypt.compareSync(
      password,
      checkUser.password,
      10
    );

    if (!checkPassword) {
      return res.status(404).send("Password not match !");
    }

    let redirectPath = "";
    let role = await Role.findById(checkUser.role);
    let idProject = checkUser.idProject;
    switch (role.name) {
      case "PRODUCT OWNER":
        redirectPath = "/productOwnerDashboard";
        break;
      case "SCRUM MANAGER":
        redirectPath = "/dashboardScrum/${idProject}";
        break;
      case "DEVELOPPEUR":
        redirectPath = "/accueil/${idProject}";
        break;
      case "TESTEUR":
        redirectPath = "/accueil/${idProject}";
        break;
      default:
        redirectPath = "/";
        break;
    }

    // Génère un token JWT pour l'utilisateur avec un identifiant unique,
    // et expire après 3600 secondes (1 heure).
    const token = jwt.sign(
      {
        id: checkUser._id,
      },
      SECRET_JWT_CODE,
      { expiresIn: 3600 }
    );

    //met à jour le status de l'utilisateur en connecté
    await User.updateOne(
      { email: email },
      {
        isConnected: true,
      }
    )
  }
}
```

Figure 43 : Extrait du code de l'authentification de l'application

```

server > src > controller > Invitation.ts ...
9 //Contrôleur qui gère l'envoi de mail pour inviter un membre à intégrer le projet
10
11 export default class InvitationController {
12   sendInvitation = async (req: Request, res: Response) => {
13     console.log(req.body);
14     const { idProject, nameProject, role, mail } = req.body;
15     const userRole = await Role.findById(role);
16     let mailOption = {
17       from: "nirina.felananiaina@gmail.com",
18       to: mail,
19       subject: "test",
20       html: `<!DOCTYPE html>
21         <html lang="en">
22           <head>
23             <meta charset="UTF-8">
24             <meta name="viewport" content="width=device-width, initial-scale=1.0">
25             <title>Document</title>
26           </head>
27           <body>
28             <h3>nirina.felananiaina@gmail.com vous a invité(e) à rejoindre son projet ${nameProject} en occupant le rôle de ${userRole?.name}</h3>
29             <p>Veuillez cliquer le bouton ci-dessous pour y rejoindre</p>
30             <button><a href="http://localhost:3000/createUsers">Rejoindre</a></button>
31           </body>
32         </html>`,
33     };
34     try {
35       const sendingInvitation = await Invitation.create({ ...req.body });
36       transporter.sendMail(mailOption, (error: any, info: any) => {
37         if (error) {
38           return console.log("error sendMail :::", error.message);
39         }
40         console.log("success");
41       });
42       res.status(200).send("success");
43     } catch (error) {
44       res.status(500).send("internal server error");
45     }
46   }
47 }

```

Figure 44: Extrait du code pour l'envoi de mail d'invitation

```

server > src > controller > UploadFileController.ts > createMulterStorage > filename
1 import { Response, Request } from "express";
2 import path from "path";
3
4 const multer = require("multer");
5 const upload = multer({ dest: "uploads/" });
6
7 const createMulterStorage = (fileDirectory: string) => {
8   return multer.diskStorage({
9     destination: `./uploads/${fileDirectory}`,
10    filename: function (req: Request, file: any, cb: any) {
11      const name = file.originalname.trim();
12      const splitName = name.split(".");
13      cb(
14        null,
15        splitName[0] +
16          "-" +
17          new Date().getTime() +
18          path.extname(file.originalname)
19      );
20    },
21  });
22 };
23 interface MulterRequest extends Request {
24   file: any;
25 }
26
27 export default class UploadFileController {
28   uploadFile = (req: Request, res: Response) => {
29     try {
30       const storage = createMulterStorage(req.params.path);
31       const uploadStorage = multer({ storage: storage });
32
33       uploadStorage.single("file")(req, res, () => {
34         return res.status(200).send((req as MulterRequest).file);
35       });
36     } catch (e: any) {
37       console.log("error:::", e);
38     }
39   }
40 }

```

Figure 45: Extrait du code qui gère l'importation des fichiers

TABLE DES MATIÈRES

INTRODUCTION GENERALE	1
PARTIE I : PRESENTATION DU PROJET	3
Chapitre 1 : Présentation de la Problématique	3
Chapitre 2 : Analyse des Solutions Existantes	3
Chapitre 3 : Justification de la Recherche et la proposition de la solution.....	4
Chapitre 4 : Objectifs du Projet	5
Chapitre 5 : Planification du Projet	5
1 . Coûts du Projet	5
2 . Délais du Projet	6
3 . Diagramme de Gantt.....	6
Chapitre 6 : Résultats Attendus	7
PARTIE II : REVUE DE LA LITTERATURE	8
Chapitre 1 : Définition des Concepts Fondamentaux	8
1 . Un Projet.....	8
2 . La gestion de projet :	8
3 . Un Jalon :	9
Chapitre 2 : La Gestion de Projet Agile	10
1 . Historique de la Gestion de Projet Agile	10
2 . Comparaison avec la Gestion de Projet Traditionnelle	12
3 . Composants et Structures des Méthodes Agiles.....	13
a . Scrum :	13
b . Kanban :	14
Chapitre 3 : Outils et Méthodes Existants	15
PARTIE III : CONCEPTION ET MISE EN ŒUVRE DE L'APPLICATION	16
Chapitre 1 : Analyse des Besoins	16
1 . Identification des Parties Prenantes	16
2 . Besoins Fonctionnels	17

3 . Besoins Non-Fonctionnels.....	18
Chapitre 2 : Choix technologiques et justification	18
1 . Utilisation de TypeScript.....	18
2 . Comparaison entre les Framework web pour le développement de l’application :.....	19
a . Comparaison entre Framework web pour le frontEnd.....	19
b . Comparaison entre Framework web pour le BackEnd :.....	21
c . Comparaison entre les Base de Données :	23
d . Authentification et Autorisation :	24
e . Communication en Temps Réel :	25
3 . Choix final des technologies ainsi que leurs liaisons :	26
a . React (Front-End)	26
b . Node.js (Back-End)	26
c . MongoDB (Base de Données)	27
d . JWT (Authentification et Autorisation).....	27
e . Socket.io (Communication en Temps Réel)	27
f . Liaison des Technologies	27
Chapitre 3 : Conception de l'Application	28
1 . Architecture Globale de l'Application	28
a . Client (Frontend) :	28
b . API RESTful :	29
c . Serveur (Backend) :	29
d . Base de Données :	29
2 . Modélisation globale du diagramme de cas d’utilisation :	30
3 . Modélisation des Données.....	30
a . Les principales collections de la base de données :	31
b . Diagramme UML des classes :	32
c . Diagramme UML de séquence	33
4 . Interaction entre les Composants.....	37

5 . Sécurité de l'Application.....	38
Chapitre 4 : Implémentation	38
1 . Configuration et Installation	38
a . Installation des outils de développement :	39
b . Configuration du projet front-end avec React :	39
c . Configuration du projet back-end avec Node.js et Express :	39
2 . Développement du Back-end :	40
3 . Développement du Front-end	41
4 . Gestion des versions avec Git.....	42
a . Utilisation de Git dans le projet	42
b . Avantages observés	43
Chapitre 5 : Tests de l'application :	43
1 . Tests Unitaires	43
2 . Tests d'Intégration	44
Chapitre 6 : Présentation de l'application :	45
1 . Fonctionnalités principales :	45
2 . Ecran de l'application.....	46
PARTIE IV : CONCLUSION GENERALE.....	51



AUTEUR : NIRINA MAMPIONONA FELANIAINA

TITRE : « Conception et Développement d'une Application de Gestion de Projet Agile : Un Outil Innovant pour l'Optimisation des Processus Collaboratifs ».

NOMBRE DE PAGES : 67

NOMBRE DE FIGURES : 45

NOMBRE DE TABLEAUX : 7

FAMINTINANA

Ity tatitra ity dia mampiseho ny fampandrosoana ny fampiharana fitantanana tetikasa agile izay novolavolaina tamin'ny rafitra 3-tiers, mampiasa React, Node.js, ary MongoDB. Izy io dia mandinika ny foto-kevitra momba ny fitantanana tetikasa agile toy ny Scrum sy Kanban, ary ahitana fitsapana sy fanamarinana mba hiantohana ny fahamarinan'ny fampiharana.

TENY MANANDANJA: Fitantanana tetikasa, Agile, Rafitra 3-tiers, Scrum, Kanban, Fampandrosoana fampiharana, React, Node.js, MongoDB, Fitsapana, Fanamarinana

RESUME

Ce mémoire présente le développement d'une application de gestion de projet agile sur une architecture 3-tiers, en utilisant React, Node.js et MongoDB. Il explore les concepts de gestion de projet agile tels que Scrum et Kanban, et inclut des tests et une validation pour assurer la fiabilité de l'application.

MOTS CLÉS: Gestion de projet, Agile, Architecture 3-tiers, Scrum, Kanban, Développement d'application, React, Node.js, MongoDB, Tests, Validation.

ABSTRACT

This thesis presents the development of an agile project management application built on a 3-tier architecture, using React, Node.js, and MongoDB. It explores agile project management concepts such as Scrum and Kanban, and includes testing and validation to ensure the application's reliability.

KEY WORDS: Project Management, Agile, 3-Tier Architecture, Scrum, Kanban, Application Development, React, Node.js, MongoDB, Testing, Validation.

ENCADREUR: Monsieur RANDRIAMAHARO ANDRIAMALALA Mamy

CO- ENCADREUR :

ADRESSE:

CONTACT : tel /mail

