Voici un prompt complet pour définir les fonctionnalités d'une plateforme de gestion de projet agile révolutionnaire :

Contexte et Vision

**Créez une plateforme de gestion de projet agile moderne et intuitive qui surpasse Jira en termes d'expérience utilisateur, de performance et de fonctionnalités. Cette solution doit couvrir l'intégralité du cycle de vie d'un projet, de l'idée initiale à la réalisation complète.**

Objectifs principaux

* **Simplicité d'utilisation** : Interface intuitive accessible à tous les membres de l'équipe
* **Performance optimale** : Temps de chargement rapides et réactivité en temps réel
* **Flexibilité** : Adaptation à différentes méthodologies agiles (Scrum, Kanban, SAFe)
* **Collaboration renforcée** : Outils de communication et de partage intégrés
* **Automatisation intelligente** : Réduction des tâches répétitives

Features core à développer

# 1. Gestion des projets et initiatives

**Features de gestion de projets agiles : Explication détaillée**

## 1. Création de projets avec templates personnalisables

**Concept principal**

**La création de projets avec templates personnalisables permet aux équipes de démarrer rapidement leurs projets en utilisant des modèles prédéfinis adaptés à leurs besoins spécifiques. Cette fonctionnalité élimine la nécessité de reconfigurer manuellement chaque nouveau projet.**

**Fonctionnalités clés**

* **Templates métier : Modèles spécialisés pour différents secteurs (développement web, marketing, événementiel, etc.)**
* **Structures prédéfinies : Workflow, statuts, types de tâches, et champs personnalisés préconfigurés**
* **Personnalisation avancée : Modification des templates existants selon les processus internes**
* **Bibliothèque de templates : Catalogue partagé au niveau de l'organisation avec versionning**
* **Templates intelligents : Suggestions automatiques basées sur l'historique des projets**

**Avantages**

* **Gain de temps : Réduction du temps de setup de 80%**
* **Standardisation : Homogenéisation des processus entre équipes**
* **Réutilisabilité : Capitalisation sur les bonnes pratiques**
* **Onboarding facilité : Nouveaux membres productifs immédiatement**

## 2. Roadmaps visuelles interactives et dynamiques

**Concept principal**

**Les roadmaps visuelles offrent une représentation graphique et interactive de la planification stratégique des projets, permettant une compréhension immédiate des objectifs, des dépendances et des jalons critiques.**

**Fonctionnalités clés**

* **Visualisation timeline : Représentation chronologique avec zoom multi-niveaux (jour, semaine, mois, trimestre)**
* **Drag & drop : Modification directe des éléments par glisser-déposer**
* **Couches d'information : Affichage sélectif des épics, features, releases**
* **Dépendances visuelles : Liens automatiques entre éléments avec détection de conflits**
* **Synchronisation temps réel : Mise à jour automatique lors des modifications**

**Avantages**

* **Alignement stratégique : Vision partagée des objectifs à long terme**
* **Identification des risques : Détection précoce des conflits et blocages**
* **Communication efficace : Présentation claire aux stakeholders**
* **Planification agile : Adaptation rapide aux changements**

## 3. Gestion des portefeuilles avec vue d'ensemble

**Concept principal**

**La gestion de portefeuilles permet de superviser simultanément plusieurs projets en offrant une vue consolidée des performances, des ressources et des priorités au niveau organisationnel.**

**Fonctionnalités clés**

* **Dashboard multi-projets : Métriques agrégées et indicateurs de santé**
* **Allocation des ressources : Répartition optimale des équipes entre projets**
* **Matrice de priorisation : Classement automatique selon valeur/effort**
* **Capacity planning : Planification des capacités avec détection de surcharge**
* **Gouvernance intégrée : Workflows d'approbation et de validation**

**Avantages**

* **Optimisation des ressources : Utilisation maximale des compétences disponibles**
* **Priorisation éclairée : Décisions basées sur des données objectives**
* **Réduction des risques : Anticipation des problèmes inter-projets**
* **ROI amélioré : Concentration sur les projets à forte valeur ajoutée**

## 4. Métriques et KPIs en temps réel

**Concept principal**

**Les métriques et KPIs en temps réel fournissent une visibilité instantanée sur la performance des projets, permettant une prise de décision rapide et des ajustements proactifs.**

**Fonctionnalités clés**

* **Tableaux de bord dynamiques : Actualisation automatique des données**
* **Métriques agiles : Velocity, burn-down/up charts, cycle time**
* **Alertes intelligentes : Notifications automatiques en cas de dérive**
* **Métriques personnalisées : Création d'indicateurs spécifiques au contexte**
* **Analyse prédictive : Projections basées sur les tendances actuelles**

**Indicateurs essentiels**

* **Vélocité d'équipe : Points de story complétés par sprint**
* **Lead time : Temps moyen de réalisation d'une tâche**
* **Taux de complétion : Pourcentage d'objectifs atteints**
* **Charge de travail : Répartition et saturation des ressources**
* **Qualité : Taux de défauts et temps de résolution**

**Avantages**

* **Transparence totale : Visibilité sur l'avancement en continu**
* **Amélioration continue : Identification des axes d'optimisation**
* **Prédictibilité : Estimations plus précises des délais**
* **Responsabilisation : Engagement renforcé des équipes**

## 5. Priorisation automatique basée sur la valeur business

**Concept principal**

**La priorisation automatique utilise des algorithmes intelligents pour classer les tâches et fonctionnalités selon leur valeur métier, optimisant ainsi l'allocation des efforts de développement.**

**Fonctionnalités clés**

* **Scoring automatique : Calcul de la valeur basé sur des critères multiples**
* **Matrice de priorisation : Impact business vs complexité technique**
* **Algorithmes d'optimisation : Techniques de recherche opérationnelle**
* **Critères configurables : Poids personnalisables selon le contexte**
* **Recommandations IA : Suggestions basées sur l'historique et les patterns**

**Critères de priorisation**

* **Valeur client : Impact sur l'expérience utilisateur et satisfaction**
* **Revenus potentiels : Contribution directe au chiffre d'affaires**
* **Risques techniques : Complexité et incertitude de réalisation**
* **Dépendances : Impact sur les autres fonctionnalités**
* **Urgence métier : Contraintes temporelles et opportunités marché**

**Avantages**

* **Objectivité : Décisions basées sur des données quantifiables**
* **Efficacité : Concentration sur les tâches à forte valeur ajoutée**
* **Agilité : Adaptation rapide aux changements de priorités**
* **Maximisation du ROI : Optimisation du retour sur investissement**

# 2. Planification et organisation

## 1. Backlog management avec drag & drop intuitif

### Fonctionnalités principales

Le backlog management avec drag & drop révolutionne la gestion des éléments de travail en permettant une manipulation directe et intuitive des tâches. Cette approche visuelle permet de réorganiser rapidement les priorités en glissant simplement les éléments d'une position à une autre.

### Conception technique

L'interface propose un système de colonnes configurables où chaque élément peut être déplacé par simple glissement. Les utilisateurs peuvent facilement créer leur backlog, puis leurs sprints et les alimenter avec les user stories rédigées. Cette fonctionnalité doit supporter différents types d'éléments : user stories, épics, bugs, et tâches techniques.

### Optimisation ergonomique

Feedback visuel immédiat : Zones de drop colorées et animations fluides

Prévisualisation en temps réel : Affichage des impacts avant validation

Raccourcis clavier : Navigation rapide pour les utilisateurs avancés

Responsive design : Adaptation au tactile sur tablettes et smartphones

### Avantages concurrentiels

Cette approche surpasse les interfaces traditionnelles en éliminant les clics multiples et les formulaires complexes. L'ordre des éléments est crucial puisque c'est lui qui indique l'ordre de réalisation et donc potentiellement l'ordre de livraison en production.

## 2. Sprint planning avec estimation automatique

### Intelligence artificielle intégrée

L'estimation automatique utilise des algorithmes d'apprentissage automatique pour analyser l'historique des sprints précédents et proposer des estimations précises basées sur la vélocité de l'équipe. Le système apprend des patterns de développement et ajuste ses prédictions en continu.

### Métriques de référence

Vélocité historique : Analyse des story points complétés par sprint

Complexité relative : Comparaison avec des tâches similaires réalisées

Facteurs contextuels : Prise en compte des congés, formations, et événements

Capacité d'équipe : Calcul automatique basé sur la disponibilité réelle

### Processus d'estimation

Le système propose automatiquement des story points en analysant la description des tâches, leur complexité technique, et les dépendances identifiées. La vélocité d'une équipe est basée sur les fonctionnalités "finies" selon un contrat clair entre l'équipe et le Product Owner.

### Validation collaborative

Malgré l'automatisation, l'équipe conserve le contrôle final avec des suggestions intelligentes plutôt que des impositions. Les estimations peuvent être ajustées en un clic avec conservation de l'historique pour améliorer l'algorithme.

## 3. Gestion des épics et user stories

Architecture hiérarchique

La gestion des épics et user stories s'organise selon une structure pyramidale : Portfolio > Epic > Feature > User Story > Tâche. Cette hiérarchie permet une vision stratégique à long terme tout en maintenant un focus opérationnel sur les détails d'implémentation.

Fonctionnalités de liaison

Le système permet de prioriser les user stories, de les ordonner dans leurs épics, et de les lier entre elles. Cette interconnexion facilite la traçabilité et assure la cohérence du développement produit.

Gestion du cycle de vie

Statuts personnalisables : Workflow adapté aux processus internes

Critères d'acceptation : Définition claire des conditions de validation

Tests intégrés : Liaison automatique avec les scénarios de test

Documentation vivante : Mise à jour automatique de la documentation

Visualisation multi-niveau

L'interface propose différentes vues selon le niveau de détail souhaité : vue épic pour la stratégie, vue user story pour l'opérationnel, et vue tâche pour l'exécution. Les transitions entre niveaux se font naturellement par zoom et drill-down.

## 4. Dépendances intelligentes avec détection automatique

Algorithmes de détection

Le système utilise des algorithmes d'analyse sémantique pour identifier automatiquement les dépendances potentielles entre tâches. L'analyse porte sur les mots-clés, les composants techniques mentionnés, et les patterns de développement historiques.

Types de dépendances

Dépendances techniques : Composants partagés, APIs, bases de données

Dépendances fonctionnelles : Prérequis métier et séquences logiques

Dépendances de ressources : Compétences spécialisées et disponibilité

Dépendances externes : Intégrations tierces et validations client

Visualisation des conflits

Le système offre une représentation graphique des dépendances avec des liens visuels automatiques entre éléments. Les conflits potentiels sont mis en évidence avec des alertes colorées et des suggestions de résolution.

Résolution proactive

Alertes en temps réel : Notification immédiate des nouveaux conflits

Suggestions d'optimisation : Réorganisation automatique du planning

Simulation d'impact : Analyse des conséquences avant modification

Résolution collaborative : Notifications aux parties prenantes concernées

## 5. Calendrier unifié multi-projets

### Vue d'ensemble consolidée

Le calendrier unifié agrège les données de plusieurs projets simultanément en offrant une vision globale des jalons, releases, et événements critiques. Cette approche évite la fragmentation et améliore la coordination inter-équipes.

### Synchronisation temps réel

Toutes les modifications apportées aux projets individuels se répercutent instantanément sur le calendrier unifié. Les conflits de ressources et les chevauchements sont détectés automatiquement avec des alertes visuelles.

### Fonctionnalités d'intégration

Import/Export iCal : Synchronisation avec les calendriers personnels

Intégrations externes : Connexion avec Outlook, Google Calendar, Slack

Notifications intelligentes : Alertes contextuelles selon les rôles

Planification des ressources : Visualisation de la charge de travail

### Personnalisation avancée

Chaque utilisateur peut customiser sa vue selon ses responsabilités : focus sur ses projets, filtrage par équipe, ou vision stratégique globale. Les couleurs, groupements, et niveaux de détail s'adaptent aux préférences individuelles.

### Modèles de planification

Le système propose des templates de planning prédéfinis pour différents types de projets (développement logiciel, marketing, événementiel) avec des jalons types et des durées optimisées basées sur l'expérience collective.

# 3. Collaboration et communication

## 1. Commentaires en temps réel avec notifications smart

### Architecture technique avancée

Les commentaires en temps réel exploitent une architecture WebSocket pour garantir une synchronisation instantanée entre tous les utilisateurs connectés. Le système utilise un event-driven architecture où chaque action de commentaire déclenche des événements propagés en temps réel à tous les clients concernés**.**

### Fonctionnalités de notifications intelligentes

* Contextualisation automatique : Les notifications incluent automatiquement le contexte complet (projet, tâche, thread de discussion)
* Priorisation dynamique : L'algorithme analyse l'urgence, la pertinence et l'historique d'interaction pour définir le niveau de priorité
* Notifications adaptatives : Réduction automatique du volume selon l'activité utilisateur pour éviter la saturation
* Digest intelligent : Regroupement des notifications similaires avec résumés contextuels

### Mécanismes de filtrage avancés

Le système propose un moteur de règles personnalisables permettant aux utilisateurs de définir leurs critères de notification :

* Filtres par mots-clés : Surveillance automatique de termes spécifiques
* Filtres par utilisateurs : Notifications préférentielles pour certains collaborateurs
* Filtres par projets : Niveau d'alerte différencié selon l'importance du projet
* Filtres temporels : Plages horaires et jours d'activation personnalisables

### Avantages concurrentiels

Cette approche surpasse les systèmes traditionnels en éliminant les délais de synchronisation et en offrant une expérience conversationnelle fluide similaire aux messageries modernes, tout en conservant la traçabilité professionnelle.

## 2. Mentions et assignations automatiques

### Intelligence artificielle contextuelle

* **Le système utilise des algorithmes de Natural Language Processing (NLP) pour analyser le contenu des commentaires et identifier automatiquement les personnes pertinentes à mentionner. L'analyse porte sur :**
* **Compétences techniques : Identification des experts selon les technologies mentionnées**
* **Historique de collaboration : Suggestions basées sur les interactions précédentes**
* **Charge de travail : Prise en compte de la disponibilité et de la capacité actuelle**
* **Rôles et permissions : Respect de la hiérarchie et des responsabilités**

### Système d'assignation intelligente

* **Auto-assignation prédictive : Suggestions automatiques basées sur l'expertise et la disponibilité**
* **Équilibrage de charge : Répartition optimale des tâches selon la capacité des équipes**
* **Escalade automatique : Redirection vers les niveaux supérieurs en cas de blocage**
* **Notifications en cascade : Alerte des parties prenantes selon un workflow prédéfini**

### Mécanismes de validation

Le système propose des suggestions d'assignation plutôt que des assignations automatiques forcées, conservant ainsi le contrôle humain tout en optimisant l'efficacité. Les suggestions incluent un score de pertinence et une justification contextuelle.

### Personnalisation des workflows

Chaque équipe peut définir ses règles d'assignation spécifiques avec des critères métier personnalisés, des seuils de charge, et des processus d'escalade adaptés à leur organisation.

## 3. Chat intégré par équipe/projet

### Architecture modulaire

Le chat intégré s'organise selon une structure hiérarchique multi-niveaux :

* **Channels globaux : Communication à l'échelle de l'organisation**
* **Channels par projet : Discussions spécifiques aux initiatives**
* **Channels par équipe : Coordination des groupes de travail**
* **Channels par feature : Focus sur des fonctionnalités spécifiques**
* **Messages privés : Communication directe entre collaborateurs**

### Fonctionnalités de productivité

* **Intégration contextuelle : Liens automatiques avec les tâches, commits, et documents**
* **Historique searchable : Recherche avancée dans l'historique avec filtres multiples**
* **Threads organisés : Réponses en fil pour maintenir la cohérence des discussions**
* **Statuts de présence : Indicateurs temps réel de disponibilité et d'activité**
* **Réactions rapides : Système d'emojis pour feedback instantané**

### Synchronisation cross-platform

Le chat fonctionne de manière transparente entre l'interface web, les applications mobiles, et les intégrations desktop, avec synchronisation instantanée des messages et notifications.

### Modération et gouvernance

* **Modération automatique : Détection de contenu inapproprié avec alertes**
* **Archivage intelligent : Conservation organisée des discussions importantes**
* **Permissions granulaires : Contrôle d'accès selon les rôles et projets**
* **Audit trail complet : Traçabilité complète des communications**

## 4. Partage de fichiers avec versioning

### Système de versioning avancé

* **Le versioning s'inspire des meilleures pratiques de Git en les adaptant au contexte de gestion de projet :**
* **Branches de versions : Création de variantes parallèles pour différents contextes**
* **Merge intelligent : Fusion automatique des modifications compatibles**
* **Rollback granulaire : Retour à des versions antérieures spécifiques**
* **Historique visuel : Timeline graphique des modifications avec diff visuels**

### Gestion des conflits

* **Détection automatique : Identification des modifications simultanées**
* **Résolution guidée : Interface intuitive pour merger les changements**
* **Notifications préventives : Alertes lors de l'édition simultanée**
* **Verrouillage temporaire : Protection contre les modifications concurrentes**

### Intégration collaborative

* **Le système permet la collaboration simultanée sur les documents avec :**
* **Édition en temps réel : Modifications visibles instantanément**
* **Commentaires inline : Annotations directement dans les documents**
* **Suggestions de modification : Workflow d'approbation intégré**
* **Suivi des contributions : Attribution automatique des modifications**

### Optimisation du stockage

* **Compression intelligente : Réduction de l'espace de stockage sans perte de qualité**
* **Déduplication : Élimination des doublons au niveau des blocs**
* **Archivage automatique : Migration des anciennes versions vers stockage froid**
* **Backup distribué : Sauvegarde automatique avec redondance géographique**

## 5. Revues de code intégrées

### Interface de revue moderne

L'interface de revue de code propose une expérience visuelle et intuitive avec :

* Diff side-by-side : Comparaison visuelle des modifications
* Syntax highlighting : Coloration syntaxique pour tous les langages
* Navigation intelligente : Saut automatique entre les modifications
* Annotations contextuelles : Commentaires liés aux lignes spécifiques

### Automatisation des contrôles

* **Analyse statique : Détection automatique des problèmes de qualité**
* **Tests automatisés : Exécution des suites de tests avant validation**
* **Métriques de qualité : Scoring automatique basé sur les bonnes pratiques**
* **Suggestions d'amélioration : Recommandations basées sur l'IA**

### Workflow de validation

Le processus de revue suit un workflow configurable avec :

* **Assignation automatique : Sélection des reviewers selon l'expertise**
* **Critères de validation : Règles personnalisables pour l'approbation**
* **Escalade temporelle : Rappels automatiques et réassignation**
* **Traçabilité complète : Historique des décisions et justifications**

### Intégration avec les outils de développement

* **Connexion Git : Synchronisation bidirectionnelle avec les repositorie**
* **Webhooks intelligents : Déclenchement automatique des revues**
* **Intégrations CI/CD : Liaison avec les pipelines de déploiement**
* **Métriques de performance : Analyse des temps de revue et de la qualité**

### Fonctionnalités d'apprentissage

* **Le système apprend des patterns de revue pour :**
* **Prédire les problèmes : Identification proactive des zones à risque**
* **Optimiser les assignations : Amélioration continue du matching reviewers**
* **Personnaliser les suggestions : Adaptation aux styles de développement**
* **Automatiser les validations : Réduction progressive des contrôles manuels**

# 4. Suivi et reporting

## 1. Dashboards personnalisables avec widgets modulaires

### Architecture modulaire avancée

Les dashboards personnalisables exploitent une architecture micro-frontend où chaque widget fonctionne comme un module autonome. Cette approche permet une composition dynamique de l'interface selon les besoins spécifiques de chaque utilisateur ou équipe.

### Système de widgets intelligents

Le système propose une bibliothèque extensive de widgets prêts à l'emploi :

* Widgets de métriques : KPIs, compteurs, jauges avec seuils personnalisables
* Widgets de visualisation : Graphiques, histogrammes, heatmaps interactives
* Widgets de listes : Tâches assignées, blocages, notifications prioritaires
* Widgets de calendrier : Événements, deadlines, disponibilités d'équipe
* Widgets de tendances : Évolution des performances, prédictions

### Personnalisation granulaire

Chaque widget offre des options de configuration poussées :

* Sources de données : Connexion flexible aux différents projets et équipes
* Filtres dynamiques : Critères personnalisables avec sauvegarde des préférences
* Styles visuels : Thèmes, couleurs, tailles adaptables au contexte
* Interactions : Drill-down, liens contextuels, actions rapides
* Rafraîchissement : Fréquences personnalisables selon la criticité

### Responsive design adaptatif

L'interface s'adapte automatiquement aux différents écrans avec :

* Grille fluide : Réorganisation automatique des widgets selon la taille d'écran
* Priorité contextuelle : Affichage préférentiel des widgets critiques sur mobile
* Gestures tactiles : Navigation intuitive sur tablettes et smartphones
* Performance optimisée : Chargement progressif et mise en cache intelligente

### Collaboration et partage

* Templates d'équipe : Dashboards partagés avec bonnes pratiques intégrées
* Snapshots automatiques : Capture périodique pour analyse historique
* Export intelligent : Génération de rapports PDF avec mise en page optimisée
* Notifications contextuelles : Alertes basées sur les seuils définis par widget

## 2. Velocity tracking et burndown charts

### Calcul de vélocité avancé

Le système de velocity tracking utilise des algorithmes sophistiqués pour calculer la vélocité réelle des équipes :

* Normalisation des story points : Compensation des variations d'estimation entre sprints
* Pondération temporelle : Prise en compte de la courbe d'apprentissage et des événements
* Facteurs correctifs : Ajustement selon les congés, formations, et indisponibilités
* Vélocité prédictive : Projection basée sur les tendances et la saisonnalité

### Burndown charts intelligents

Les burndown charts dépassent les représentations traditionnelles avec :

* Prédiction en temps réel : Projection automatique de la fin de sprint
* Analyse des déviations : Identification des causes de retard avec suggestions
* Scope creep detection : Surveillance des ajouts non planifiés
* Comparaison historique : Overlay des performances des sprints précédents

### Métriques complémentaires

* Cycle time : Temps moyen de réalisation par type de tâche
* Lead time : Délai total de la demande à la livraison
* Throughput : Nombre d'éléments complétés par unité de temps
* Work in Progress : Limitation et optimisation du flux de travail

### Visualisations interactives

Les graphiques proposent des interactions avancées :

* Zoom temporel : Focus sur des périodes spécifiques
* Couches d'information : Affichage sélectif des métriques
* Annotations contextuelles : Marquage des événements impactants
* Comparaisons multiples : Analyse parallèle de plusieurs sprints ou équipes

## 3. Rapports automatisés avec insights IA

### Génération intelligente de rapports

L'intelligence artificielle analyse automatiquement les données pour produire des rapports contextualisés :

* Analyse de tendances : Identification des patterns et anomalies
* Recommandations personnalisées : Suggestions d'amélioration basées sur l'historique
* Priorisation automatique : Mise en avant des informations critiques
* Narratif généré : Création automatique de résumés exécutifs

### Insights prédictifs

Le système utilise des modèles de machine learning pour :

* Prédiction des risques : Identification précoce des projets en danger
* Optimisation des ressources : Suggestions de réallocation basées sur les performances
* Détection d'opportunités : Identification des leviers d'amélioration
* Benchmarking intelligent : Comparaison avec des équipes similaires

### Personnalisation des rapports

* Templates adaptatifs : Formats spécifiques selon les destinataires (management, équipes, clients)
* Fréquence intelligente : Ajustement automatique selon l'activité et l'importance
* Métriques ciblées : Sélection automatique des KPIs pertinents par contexte
* Visualisations optimisées : Choix automatique du meilleur format de présentation

### Distribution intelligente

* Audience segmentée : Contenu adapté au niveau hiérarchique et aux responsabilités
* Timing optimal : Envoi au moment de plus forte attention selon les patterns utilisateur
* Canaux multiples : Email, notifications in-app, intégrations Slack/Teams
* Suivi d'engagement : Analyse de la consultation et des actions déclenchées

## 4. Time tracking simplifié

## Capture automatique du temps

Le système propose une approche non-intrusive du time tracking :

* Détection d'activité : Reconnaissance automatique du travail sur les tâches
* Intégration IDE : Plugins pour les environnements de développement
* Synchronisation calendrier : Import automatique des créneaux dédiés
* Géolocalisation intelligente : Détection du contexte de travail (bureau, télétravail)

### Interface utilisateur optimisée

* One-click tracking : Démarrage/arrêt instantané depuis n'importe quelle tâche
* Suggestions intelligentes : Propositions basées sur l'historique et le contexte
* Correction facilitée : Ajustement rapide des temps avec validation automatique
* Timesheet intelligent : Pré-remplissage basé sur les patterns récurrents

### Analyse et optimisation

Le système analyse les données temporelles pour :

* Identification des inefficacités : Détection des activités chronophages
* Optimisation des estimations : Amélioration continue basée sur les réalisations
* Patterns de productivité : Identification des créneaux de performance optimale
* Benchmarking interne : Comparaison avec des tâches similaires

### Respect de la confidentialité

* Granularité contrôlée : Niveau de détail ajustable selon les besoins
* Anonymisation intelligente : Protection des données personnelles dans les rapports
* Consent management : Contrôle utilisateur sur le niveau de tracking
* Audit trail : Traçabilité complète des accès et modifications

## 5. Analyse prédictive des délais

### Modèles probabilistes avancés

L'analyse prédictive utilise des algorithmes de Monte Carlo pour :

* Simulation de scénarios : Génération de milliers de projections possibles
* Calcul de probabilités : Estimation des chances de respect des délais
* Analyse de sensibilité : Identification des facteurs les plus impactants
* Optimisation de planning : Suggestions de réorganisation pour minimiser les risques

### Facteurs prédictifs multiples

Le système intègre de nombreuses variables :

* Historique de performance : Analyse des réalisations passées de l'équipe
* Complexité technique : Évaluation de la difficulté des tâches restantes
* Dépendances externes : Prise en compte des blocages potentiels
* Facteurs humains : Congés, formations, montée en compétence
* Événements organisationnels : Réorganisations, changements de priorités

### Alertes proactives

* Détection précoce : Identification des dérives avant qu'elles ne s'aggravent
* Recommandations d'actions : Suggestions concrètes pour rattraper les retards
* Escalade automatique : Notifications aux niveaux appropriés selon la criticité
* Scenarios de mitigation : Propositions d'alternatives pour maintenir les objectifs

### Amélioration continue

Le système s'auto-améliore en permanence :

* Feedback loop : Intégration des résultats réels pour affiner les modèles
* Apprentissage contextuel : Adaptation aux spécificités de chaque équipe
* Calibration automatique : Ajustement des paramètres selon les performances
* Validation croisée : Comparaison avec d'autres méthodes d'estimation

### Intégration stratégique

* Planification de capacité : Optimisation des ressources à moyen terme
* Gestion des risques : Identification et mitigation des menaces projet
* Communication stakeholder : Rapports de confiance avec probabilités réalistes
* Prise de décision éclairée : Support à l'arbitrage entre qualité, délais et budget

# 5. Intégrations et automatisation

* **API REST complète** pour intégrations tierces
* **Webhooks** pour automatisation
* **Intégrations natives** (GitLab, GitHub, Slack, Teams)
* **Workflows personnalisés** avec conditions
* **Notifications intelligentes** multi-canaux

Spécifications techniques

Architecture

* **Frontend** : React/TypeScript avec Next.js
* **Backend** : Node.js/TypeScript avec architecture microservices
* **Base de données** : PostgreSQL avec Redis pour le cache
* **Temps réel** : WebSockets pour la synchronisation
* **API** : GraphQL et REST

Performances

* **Temps de chargement** < 2 secondes
* **Synchronisation temps réel** < 100ms
* **Disponibilité** 99.9%
* **Scalabilité** horizontale automatique

Sécurité

* **Authentification** multi-facteurs
* **Chiffrement** end-to-end
* **Audit trail** complet
* **Conformité** RGPD et SOC 2

Différenciateurs par rapport à Jira

Expérience utilisateur

* **Interface moderne** avec design system cohérent
* **Navigation intuitive** sans courbe d'apprentissage
* **Recherche intelligente** avec filtres contextuels
* **Personnalisation poussée** de l'interface

Performance

* **Chargement instantané** des pages
* **Synchronisation temps réel** sans latence
* **Optimisation mobile** native
* **Offline support** pour les tâches critiques

Fonctionnalités avancées

* **IA intégrée** pour l'estimation et la priorisation
* **Automatisation no-code** pour les workflows
* **Analytics avancées** avec machine learning
* **Collaboration vidéo** intégrée

# Roadmap de développement

Phase 1 (MVP - 3 mois)

* Gestion basique des projets et tâches
* Interface utilisateur moderne
* Authentification et gestion des utilisateurs
* API REST de base

Phase 2 (4-6 mois)

* Fonctionnalités agiles complètes
* Intégrations principales
* Reporting et analytics
* Applications mobiles

Phase 3 (7-12 mois)

* IA et automatisation avancée
* Scalabilité enterprise
* Intégrations avancées
* Optimisations performance

# Critères de succès

* **Adoption utilisateur** : 90% des utilisateurs actifs quotidiennement
* **Performance** : Temps de réponse < 200ms
* **Satisfaction** : Score NPS > 70
* **Rétention** : Taux de churn < 5%

Cette plateforme doit révolutionner la gestion de projet agile en offrant une expérience utilisateur exceptionnelle tout en maintenant la puissance et la flexibilité nécessaires aux équipes modernes.

# BrainStorming

Matrice valeur/effort pour guider les choix

Gestion des accès et permissions granulaires

Audit et compliance (RGPD, SOC2) à approfondir

Spécifier les métriques de performance en détail (P95, P99)

Prévoir les tests de charge et stress testing

Prototyper les fonctionnalités critiques avant développement

Mettre en place des tests utilisateur itératifs

Concevoir les parcours d'onboarding intuitifs

Préparer la documentation et tutoriels

Planifier le support client multi-niveaux

Clone Copilot 365 Planner

Clone Copilot 365 Loop

…

Conçois et spécifie une application web « Jira Killer » de gestion de projet agile end-to-end.  
L’outil doit :

1. Gouvernance & Organisations
   * Créer et administrer plusieurs organisations, unités d’affaires et équipes
   * Gérer les rôles (Admin, PO, Scrum Master, Dev, Stakeholder) et les droits fins
2. Cadre Scrum complet
   * Artefacts natifs : Product Backlog, Sprint Backlog, Increment, Definition of Done
   * Évènements : Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective
   * Génération automatique des artefacts (burndown, velocity, rapports rétros)
3. Modélisation du travail
   * Hiérarchie : Initiative ▶ Epic ▶ Feature ▶ User Story ▶ Task & Bug
   * Backlog drag-&-drop (priorisation, regroupement, filtrage)
   * Détection intelligente des dépendances et suggestions de découpage
4. Sprints & Planification
   * Création de sprints, capacité calculée, auto-allocation par IA des stories
   * Estimation assistée (ML sur vélocité historique)
   * Re-forecasting continu et alertes de dérive
5. Collaboration temps-réel
   * Commentaires live, mentions, notifications smart
   * Chat par équipe / projet, partage de fichiers versionné, revue de code intégrée
   * Co-édition de documents (specs, tests, RFC)
6. Reporting & Insights
   * Dashboards modulaires, widgets personnalisables (KPI, burndown, lead-time…)
   * Rapports automatiques avec insights IA et recommandations d’action
   * Analyse prédictive des délais (Monte-Carlo, tendances vélocité)
7. Intégrations & Automatisation
   * API REST + GraphQL complètes, webhooks, intégrations GitHub/GitLab/Slack/Teams
   * Builder de workflows no-code (triggers, conditions, actions multi-canal)
8. Exigences techniques clés
   * Frontend : React + TypeScript strict (Next.js) – responsive & accessible
   * Backend : Node.js + TypeScript, micro-services gRPC / REST, CQRS + event-sourcing
   * BDD : PostgreSQL, Redis (cache), S3-compatible (fichiers)
   * Temps-réel : WebSockets (100 ms max) + OT pour co-édition
   * Sécurité : MFA, OAuth 2.1, chiffrement E2E, audit trail, RGPD & SOC 2
   * SLA : T<2 s au chargement, 99.9% uptime, scalabilité horizontale
9. Architecture de code (monorepo)  
   /apps  
   🡒 web-client/ (Next.js)  
   🡒 api-gateway/ (NestJS)  
   /services  
   🡒 auth/  (DDD , CQRS, JWT, MFA)  
   🡒 projects/ (Gestion projets, backlogs, sprints)  
   🡒 collaboration/ (chat, notifications, comments)  
   🡒 analytics/ (ETL, ML, prédictif)  
   /packages  
   🡒 ui-kit/ (Design system React + Tailwind)  
   🡒 shared-models/ (Types stricts, zod schemas)  
   🡒 hooks-utils/ (React Query, websocket hooks)  
   /infra  
   🡒 terraform/ (IaC cloud)  
   🡒 k8s-manifests/  
   /docs  
   🡒 ADR, RFC, openapi.yaml
10. Roadmap indicative  
    MVP (3 mois) : auth, organisations, projets, backlog drag-drop, sprints de base  
    V1 (6 mois) : IA d’estimation, dashboards, intégrations Git & Slack, mobile PWA  
    V2 (12 mois) : workflows no-code, analytics prédictive, mode offline, scale enterprise

Objectifs mesurables

* Adoption : > 90% utilisateurs actifs hebdo dans 3 mois après déploiement
* NPS : > 70 ; churn organisations < 5% / an
* Performance : P95 < 200 ms ; couverture de test > 80%

Livrables attendus

1. Spécification fonctionnelle détaillée (user stories, critères d’acceptation)
2. Modèle de données complet (ERD + schemas TypeScript)
3. Arborescence de code avec fichiers init (boilerplate strict mode)
4. Maquettes responsives (Figma ou Storybook)
5. Plan de migration Jira / import CSV
6. Stratégie de tests (unit, e2e, perf, sécurité)

# prisma/schema.prisma

// prisma/schema.prisma

generator client {

provider = "prisma-client-js"

previewFeatures = ["postgresqlExtensions"]

}

datasource db {

provider = "postgresql"

url = env("DATABASE\_URL")

extensions = [pg\_trgm, uuid\_ossp]

}

// ======================

// ENUMS

// ======================

enum UserRole {

ADMIN

PRODUCT\_OWNER

SCRUM\_MASTER

DEVELOPER

STAKEHOLDER

VIEWER

}

enum TaskStatus {

BACKLOG

TODO

IN\_PROGRESS

IN\_REVIEW

TESTING

DONE

BLOCKED

CANCELLED

}

enum SprintStatus {

PLANNED

ACTIVE

COMPLETED

CANCELLED

}

enum Priority {

CRITICAL

HIGH

MEDIUM

LOW

}

enum ArtifactType {

PRODUCT\_BACKLOG

SPRINT\_BACKLOG

INCREMENT

DEFINITION\_OF\_DONE

DEFINITION\_OF\_READY

RETROSPECTIVE

REVIEW

}

enum EventType {

TASK\_CREATED

TASK\_UPDATED

TASK\_ASSIGNED

SPRINT\_STARTED

SPRINT\_COMPLETED

COMMENT\_ADDED

BLOCKER\_ADDED

}

enum NotificationType {

TASK\_ASSIGNED

TASK\_COMPLETED

SPRINT\_STARTED

MENTION

COMMENT\_REPLY

DEADLINE\_REMINDER

BLOCKER\_RESOLVED

}

enum FileType {

DOCUMENT

IMAGE

VIDEO

ARCHIVE

CODE

DESIGN

OTHER

}

enum DependencyType {

BLOCKS

DEPENDS\_ON

RELATED\_TO

DUPLICATE

}

// ======================

// MODELS

// ======================

model Organization {

id String @id @default(uuid())

name String

slug String @unique

description String?

logoUrl String?

industry String?

timezone String @default("UTC")

settings Json?

isActive Boolean @default(true)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

teams Team[]

projects Project[]

templates Template[]

members OrganizationMember[]

@@index([slug])

}

model OrganizationMember {

id String @id @default(uuid())

role UserRole @default(VIEWER)

joinedAt DateTime @default(now())

lastActiveAt DateTime?

// Relations

organization Organization @relation(fields: [organizationId], references: [id], onDelete: Cascade)

organizationId String

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

@@unique([organizationId, userId])

@@index([organizationId, userId])

}

model Team {

id String @id @default(uuid())

name String

slug String

description String?

avatarUrl String?

isActive Boolean @default(true)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

organization Organization @relation(fields: [organizationId], references: [id], onDelete: Cascade)

organizationId String

members TeamMember[]

projects Project[]

channels Channel[]

@@unique([organizationId, slug])

@@index([organizationId, slug])

}

model TeamMember {

id String @id @default(uuid())

role UserRole @default(DEVELOPER)

joinedAt DateTime @default(now())

isLead Boolean @default(false)

// Relations

team Team @relation(fields: [teamId], references: [id], onDelete: Cascade)

teamId String

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

@@unique([teamId, userId])

@@index([teamId, userId])

}

model Project {

id String @id @default(uuid())

name String

slug String

description String?

startDate DateTime?

endDate DateTime?

isActive Boolean @default(true)

settings Json?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

organization Organization @relation(fields: [organizationId], references: [id], onDelete: Cascade)

organizationId String

team Team? @relation(fields: [teamId], references: [id], onDelete: SetNull)

teamId String?

members ProjectMember[]

initiatives Initiative[]

sprints Sprint[]

artifacts Artifact[]

channels Channel[]

files File[]

workflows Workflow[]

integrations ProjectIntegration[]

@@unique([organizationId, slug])

@@index([organizationId, slug])

}

model ProjectMember {

id String @id @default(uuid())

role UserRole @default(DEVELOPER)

joinedAt DateTime @default(now())

// Relations

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

@@unique([projectId, userId])

@@index([projectId, userId])

}

// ======================

// WORK ITEM HIERARCHY

// ======================

model Initiative {

id String @id @default(uuid())

name String

description String?

priority Priority @default(MEDIUM)

status String @default("active")

startDate DateTime?

endDate DateTime?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

epics Epic[]

@@index([projectId])

}

model Epic {

id String @id @default(uuid())

name String

description String?

priority Priority @default(MEDIUM)

status String @default("active")

startDate DateTime?

endDate DateTime?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

initiative Initiative @relation(fields: [initiativeId], references: [id], onDelete: Cascade)

initiativeId String

features Feature[]

@@index([initiativeId])

}

model Feature {

id String @id @default(uuid())

name String

description String?

priority Priority @default(MEDIUM)

status String @default("active")

startDate DateTime?

endDate DateTime?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

epic Epic @relation(fields: [epicId], references: [id], onDelete: Cascade)

epicId String

userStories UserStory[]

@@index([epicId])

}

model UserStory {

id String @id @default(uuid())

title String

description String?

acceptanceCriteria String?

priority Priority @default(MEDIUM)

storyPoints Int?

status TaskStatus @default(BACKLOG)

position Int @default(0)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

feature Feature @relation(fields: [featureId], references: [id], onDelete: Cascade)

featureId String

creator User @relation(fields: [creatorId], references: [id])

creatorId String

tasks Task[]

sprints Sprint[] @relation("SprintUserStories")

dependencies Dependency[] @relation("DependentUserStory")

dependents Dependency[] @relation("DependsOnUserStory")

comments Comment[]

timeEntries TimeEntry[]

labels Label[]

activityLogs ActivityLog[]

@@index([featureId])

}

model Task {

id String @id @default(uuid())

title String

description String?

priority Priority @default(MEDIUM)

status TaskStatus @default(TODO)

position Int @default(0)

estimate Int? // in hours

dueDate DateTime?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

userStory UserStory @relation(fields: [userStoryId], references: [id], onDelete: Cascade)

userStoryId String

creator User @relation(fields: [creatorId], references: [id])

creatorId String

assignees User[] @relation("TaskAssignees")

dependencies Dependency[] @relation("DependentTask")

dependents Dependency[] @relation("DependsOnTask")

comments Comment[]

timeEntries TimeEntry[]

labels Label[]

activityLogs ActivityLog[]

blockers Blocker[]

@@index([userStoryId])

}

// ======================

// SPRINTS & ARTIFACTS

// ======================

model Sprint {

id String @id @default(uuid())

name String

goal String?

startDate DateTime

endDate DateTime

status SprintStatus @default(PLANNED)

capacity Int? // in story points

velocity Int? // actual velocity

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

userStories UserStory[] @relation("SprintUserStories")

artifacts Artifact[]

timeEntries TimeEntry[]

retrospective Retrospective?

@@index([projectId])

}

model Artifact {

id String @id @default(uuid())

type ArtifactType

name String

content String // JSON or markdown

metadata Json?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

sprint Sprint? @relation(fields: [sprintId], references: [id], onDelete: Cascade)

sprintId String?

@@index([projectId, sprintId])

}

model Retrospective {

id String @id @default(uuid())

title String

notes String?

positives String[]

negatives String[]

actions Json // Array of {action: string, owner: string, dueDate: DateTime}

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

sprint Sprint @relation(fields: [sprintId], references: [id], onDelete: Cascade)

sprintId String

creator User @relation(fields: [creatorId], references: [id])

creatorId String

@@index([sprintId])

}

// ======================

// DEPENDENCIES & RELATIONS

// ======================

model Dependency {

id String @id @default(uuid())

type DependencyType

description String?

createdAt DateTime @default(now())

// Relations

dependentTask Task? @relation("DependentTask", fields: [dependentTaskId], references: [id], onDelete: Cascade)

dependentTaskId String?

dependsOnTask Task? @relation("DependsOnTask", fields: [dependsOnTaskId], references: [id], onDelete: Cascade)

dependsOnTaskId String?

dependentUserStory UserStory? @relation("DependentUserStory", fields: [dependentUserStoryId], references: [id], onDelete: Cascade)

dependentUserStoryId String?

dependsOnUserStory UserStory? @relation("DependsOnUserStory", fields: [dependsOnUserStoryId], references: [id], onDelete: Cascade)

dependsOnUserStoryId String?

@@index([dependentTaskId, dependsOnTaskId])

@@index([dependentUserStoryId, dependsOnUserStoryId])

}

model Blocker {

id String @id @default(uuid())

description String

resolved Boolean @default(false)

resolvedAt DateTime?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

task Task @relation(fields: [taskId], references: [id], onDelete: Cascade)

taskId String

reporter User @relation(fields: [reporterId], references: [id])

reporterId String

resolver User? @relation(fields: [resolverId], references: [id])

resolverId String?

@@index([taskId])

}

// ======================

// COLLABORATION

// ======================

model Comment {

id String @id @default(uuid())

content String

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

author User @relation(fields: [authorId], references: [id], onDelete: Cascade)

authorId String

task Task? @relation(fields: [taskId], references: [id], onDelete: Cascade)

taskId String?

userStory UserStory? @relation(fields: [userStoryId], references: [id], onDelete: Cascade)

userStoryId String?

parent Comment? @relation("CommentReplies", fields: [parentId], references: [id], onDelete: Cascade)

parentId String?

replies Comment[] @relation("CommentReplies")

mentions Mention[]

reactions Reaction[]

@@index([taskId, userStoryId, parentId])

}

model Mention {

id String @id @default(uuid())

createdAt DateTime @default(now())

// Relations

comment Comment @relation(fields: [commentId], references: [id], onDelete: Cascade)

commentId String

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

@@unique([commentId, userId])

}

model Reaction {

id String @id @default(uuid())

emoji String

createdAt DateTime @default(now())

// Relations

comment Comment @relation(fields: [commentId], references: [id], onDelete: Cascade)

commentId String

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

@@unique([commentId, userId, emoji])

}

model Channel {

id String @id @default(uuid())

name String

description String?

type String @default("text") // text, voice, video

isPrivate Boolean @default(false)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

project Project? @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String?

team Team? @relation(fields: [teamId], references: [id], onDelete: Cascade)

teamId String?

messages Message[]

members ChannelMember[]

@@index([projectId, teamId])

}

model ChannelMember {

id String @id @default(uuid())

joinedAt DateTime @default(now())

// Relations

channel Channel @relation(fields: [channelId], references: [id], onDelete: Cascade)

channelId String

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

@@unique([channelId, userId])

}

model Message {

id String @id @default(uuid())

content String

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

channel Channel @relation(fields: [channelId], references: [id], onDelete: Cascade)

channelId String

author User @relation(fields: [authorId], references: [id], onDelete: Cascade)

authorId String

parent Message? @relation("MessageReplies", fields: [parentId], references: [id], onDelete: Cascade)

parentId String?

replies Message[] @relation("MessageReplies")

reactions MessageReaction[]

attachments MessageAttachment[]

@@index([channelId, parentId])

}

model MessageReaction {

id String @id @default(uuid())

emoji String

createdAt DateTime @default(now())

// Relations

message Message @relation(fields: [messageId], references: [id], onDelete: Cascade)

messageId String

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

@@unique([messageId, userId, emoji])

}

model MessageAttachment {

id String @id @default(uuid())

name String

type FileType

url String

size Int

mimeType String

createdAt DateTime @default(now())

// Relations

message Message @relation(fields: [messageId], references: [id], onDelete: Cascade)

messageId String

@@index([messageId])

}

// ======================

// FILES & DOCUMENTS

// ======================

model File {

id String @id @default(uuid())

name String

type FileType

url String

size Int

mimeType String

version Int @default(1)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

uploader User @relation(fields: [uploaderId], references: [id], onDelete: Cascade)

uploaderId String

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

versions FileVersion[]

@@index([projectId])

}

model FileVersion {

id String @id @default(uuid())

version Int

url String

size Int

changes String?

createdAt DateTime @default(now())

// Relations

file File @relation(fields: [fileId], references: [id], onDelete: Cascade)

fileId String

uploader User @relation(fields: [uploaderId], references: [id])

uploaderId String

@@unique([fileId, version])

}

// ======================

// NOTIFICATIONS & ACTIVITY

// ======================

model Notification {

id String @id @default(uuid())

type NotificationType

title String

message String

data Json?

read Boolean @default(false)

createdAt DateTime @default(now())

readAt DateTime?

// Relations

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

relatedTask Task? @relation(fields: [taskId], references: [id], onDelete: Cascade)

taskId String?

relatedUserStory UserStory? @relation(fields: [userStoryId], references: [id], onDelete: Cascade)

userStoryId String?

relatedSprint Sprint? @relation(fields: [sprintId], references: [id], onDelete: Cascade)

sprintId String?

@@index([userId, read])

}

model ActivityLog {

id String @id @default(uuid())

action String

entityType String

entityId String

oldValues Json?

newValues Json?

createdAt DateTime @default(now())

// Relations

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

task Task? @relation(fields: [taskId], references: [id], onDelete: Cascade)

taskId String?

userStory UserStory? @relation(fields: [userStoryId], references: [id], onDelete: Cascade)

userStoryId String?

project Project? @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String?

@@index([userId, entityType, entityId])

@@index([taskId, userStoryId, projectId])

}

// ======================

// TIME TRACKING

// ======================

model TimeEntry {

id String @id @default(uuid())

description String?

hours Float

date DateTime

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

task Task? @relation(fields: [taskId], references: [id], onDelete: Cascade)

taskId String?

userStory UserStory? @relation(fields: [userStoryId], references: [id], onDelete: Cascade)

userStoryId String?

sprint Sprint? @relation(fields: [sprintId], references: [id], onDelete: Cascade)

sprintId String?

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

@@index([userId, projectId, sprintId])

}

// ======================

// TEMPLATES & WORKFLOWS

// ======================

model Template {

id String @id @default(uuid())

name String

description String?

type String // project, sprint, task, etc.

content Json

isPublic Boolean @default(false)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

organization Organization @relation(fields: [organizationId], references: [id], onDelete: Cascade)

organizationId String

creator User @relation(fields: [creatorId], references: [id])

creatorId String

@@index([organizationId])

}

model Workflow {

id String @id @default(uuid())

name String

description String?

steps Json // Array of {name: string, status: string, transitions: string[]}

isDefault Boolean @default(false)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

creator User @relation(fields: [creatorId], references: [id])

creatorId String

@@index([projectId])

}

// ======================

// INTEGRATIONS

// ======================

model ProjectIntegration {

id String @id @default(uuid())

type String // github, gitlab, slack, etc.

config Json

isActive Boolean @default(true)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

@@index([projectId, type])

}

// ======================

// LABELS & TAXONOMY

// ======================

model Label {

id String @id @default(uuid())

name String

color String @default("#64748b")

description String?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

userStories UserStory[]

tasks Task[]

@@unique([projectId, name])

@@index([projectId])

}

// ======================

// USER MODEL (LINKED TO BETTER AUTH)

// ======================

model User {

id String @id @default(uuid())

email String @unique

username String @unique

firstName String

lastName String

avatarUrl String?

timezone String @default("UTC")

preferences Json?

lastActiveAt DateTime?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations (all read-only from Better Auth perspective)

organizations OrganizationMember[]

teams TeamMember[]

projects ProjectMember[]

createdTasks Task[] @relation("TaskCreator")

assignedTasks Task[] @relation("TaskAssignees")

createdUserStories UserStory[]

comments Comment[]

messages Message[]

notifications Notification[]

timeEntries TimeEntry[]

activityLogs ActivityLog[]

files File[]

templates Template[]

workflows Workflow[]

blockers Blocker[]

reactions Reaction[]

messageReactions MessageReaction[]

mentions Mention[]

// Better Auth specific fields (read-only)

externalId String? @unique

isVerified Boolean @default(false)

lastLoginAt DateTime?

@@index([email, username])

}

// ======================

// JOIN TABLES

// ======================

model \_SprintUserStories {

sprintId String

userStoryId String

@@id([sprintId, userStoryId])

@@index([sprintId, userStoryId])

}

model \_TaskAssignees {

taskId String

userId String

@@id([taskId, userId])

@@index([taskId, userId])

}

model \_LabelUserStories {

labelId String

userStoryId String

@@id([labelId, userStoryId])

@@index([labelId, userStoryId])

}

model \_LabelTasks {

labelId String

taskId String

@@id([labelId, taskId])

@@index([labelId, taskId])

}

////////////////////////////////////////////

// prisma/schema.prisma

// Schéma Prisma pour l'application de gestion de projet agile "Jira Killer"

// Strictement typé pour TypeScript strict mode

generator client {

provider = "prisma-client-js"

output = "./node\_modules/.prisma/client"

}

datasource db {

provider = "postgresql"

url = env("DATABASE\_URL")

}

// =====================================

// ÉNUMÉRATIONS

// =====================================

enum UserRole {

ADMIN

PRODUCT\_OWNER

SCRUM\_MASTER

DEVELOPER

STAKEHOLDER

VIEWER

}

enum TaskStatus {

TODO

IN\_PROGRESS

CODE\_REVIEW

TESTING

DONE

BLOCKED

CANCELLED

}

enum SprintStatus {

PLANNED

ACTIVE

COMPLETED

CANCELLED

}

enum Priority {

CRITICAL

HIGH

MEDIUM

LOW

}

enum ArtifactType {

PRODUCT\_BACKLOG

SPRINT\_BACKLOG

INCREMENT

DEFINITION\_OF\_DONE

BURNDOWN\_CHART

VELOCITY\_CHART

SPRINT\_RETROSPECTIVE

}

enum EventType {

TASK\_CREATED

TASK\_UPDATED

TASK\_ASSIGNED

SPRINT\_STARTED

SPRINT\_COMPLETED

COMMENT\_ADDED

}

enum NotificationType {

TASK\_ASSIGNED

TASK\_COMPLETED

SPRINT\_STARTED

MENTION

COMMENT\_REPLY

DEADLINE\_REMINDER

}

enum FileType {

DOCUMENT

IMAGE

VIDEO

ARCHIVE

CODE

OTHER

}

// =====================================

// MODÈLES PRINCIPAUX

// =====================================

model Organization {

id String @id @default(uuid())

name String

description String?

slug String @unique

logoUrl String?

settings Json? @default("{}")

isActive Boolean @default(true)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

users User[]

teams Team[]

projects Project[]

templates Template[]

@@map("organizations")

}

model User {

id String @id @default(uuid())

email String @unique

username String @unique

firstName String

lastName String

avatarUrl String?

hashedPassword String

isActive Boolean @default(true)

lastLoginAt DateTime?

preferences Json? @default("{}")

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

organization Organization @relation(fields: [organizationId], references: [id], onDelete: Cascade)

organizationId String

teamMemberships TeamMember[]

projectMembers ProjectMember[]

assignedTasks Task[] @relation("TaskAssignees")

createdTasks Task[] @relation("TaskCreator")

comments Comment[]

notifications Notification[]

auditLogs AuditLog[]

timeEntries TimeEntry[]

fileUploads File[]

@@map("users")

}

model Team {

id String @id @default(uuid())

name String

description String?

slug String

isActive Boolean @default(true)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

organization Organization @relation(fields: [organizationId], references: [id], onDelete: Cascade)

organizationId String

members TeamMember[]

projects Project[]

channels Channel[]

@@unique([organizationId, slug])

@@map("teams")

}

model TeamMember {

id String @id @default(uuid())

role UserRole @default(DEVELOPER)

joinedAt DateTime @default(now())

// Relations

team Team @relation(fields: [teamId], references: [id], onDelete: Cascade)

teamId String

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

@@unique([teamId, userId])

@@map("team\_members")

}

model Project {

id String @id @default(uuid())

name String

description String?

slug String

startDate DateTime?

endDate DateTime?

isActive Boolean @default(true)

settings Json? @default("{}")

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

organization Organization @relation(fields: [organizationId], references: [id], onDelete: Cascade)

organizationId String

team Team? @relation(fields: [teamId], references: [id])

teamId String?

members ProjectMember[]

initiatives Initiative[]

sprints Sprint[]

artifacts Artifact[]

channels Channel[]

files File[]

@@unique([organizationId, slug])

@@map("projects")

}

model ProjectMember {

id String @id @default(uuid())

role UserRole @default(DEVELOPER)

joinedAt DateTime @default(now())

// Relations

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

@@unique([projectId, userId])

@@map("project\_members")

}

// =====================================

// GESTION DU TRAVAIL

// =====================================

model Initiative {

id String @id @default(uuid())

name String

description String?

priority Priority @default(MEDIUM)

status String @default("active")

startDate DateTime?

endDate DateTime?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

epics Epic[]

@@map("initiatives")

}

model Epic {

id String @id @default(uuid())

name String

description String?

priority Priority @default(MEDIUM)

status String @default("active")

startDate DateTime?

endDate DateTime?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

initiative Initiative @relation(fields: [initiativeId], references: [id], onDelete: Cascade)

initiativeId String

features Feature[]

@@map("epics")

}

model Feature {

id String @id @default(uuid())

name String

description String?

priority Priority @default(MEDIUM)

status String @default("active")

startDate DateTime?

endDate DateTime?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

epic Epic @relation(fields: [epicId], references: [id], onDelete: Cascade)

epicId String

userStories UserStory[]

@@map("features")

}

model UserStory {

id String @id @default(uuid())

title String

description String?

acceptanceCriteria String?

priority Priority @default(MEDIUM)

storyPoints Int?

status TaskStatus @default(TODO)

position Int @default(0)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

feature Feature @relation(fields: [featureId], references: [id], onDelete: Cascade)

featureId String

creator User @relation("TaskCreator", fields: [creatorId], references: [id])

creatorId String

tasks Task[]

sprints Sprint[] @relation("SprintUserStories")

dependencies TaskDependency[] @relation("DependentTask")

dependents TaskDependency[] @relation("DependsOnTask")

comments Comment[]

timeEntries TimeEntry[]

@@map("user\_stories")

}

model Task {

id String @id @default(uuid())

title String

description String?

priority Priority @default(MEDIUM)

status TaskStatus @default(TODO)

position Int @default(0)

estimate Int? // en heures

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

userStory UserStory @relation(fields: [userStoryId], references: [id], onDelete: Cascade)

userStoryId String

creator User @relation("TaskCreator", fields: [creatorId], references: [id])

creatorId String

assignees User[] @relation("TaskAssignees")

dependencies TaskDependency[] @relation("DependentTask")

dependents TaskDependency[] @relation("DependsOnTask")

comments Comment[]

timeEntries TimeEntry[]

@@map("tasks")

}

model TaskDependency {

id String @id @default(uuid())

type String @default("depends\_on") // depends\_on, blocks, etc.

createdAt DateTime @default(now())

// Relations

dependentTask Task? @relation("DependentTask", fields: [dependentTaskId], references: [id], onDelete: Cascade)

dependentTaskId String?

dependsOnTask Task? @relation("DependsOnTask", fields: [dependsOnTaskId], references: [id], onDelete: Cascade)

dependsOnTaskId String?

dependentUserStory UserStory? @relation("DependentTask", fields: [dependentUserStoryId], references: [id], onDelete: Cascade)

dependentUserStoryId String?

dependsOnUserStory UserStory? @relation("DependsOnTask", fields: [dependsOnUserStoryId], references: [id], onDelete: Cascade)

dependsOnUserStoryId String?

@@map("task\_dependencies")

}

// =====================================

// SPRINTS ET ARTEFACTS SCRUM

// =====================================

model Sprint {

id String @id @default(uuid())

name String

goal String?

startDate DateTime

endDate DateTime

status SprintStatus @default(PLANNED)

capacity Int? // en story points

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

userStories UserStory[] @relation("SprintUserStories")

artifacts Artifact[]

timeEntries TimeEntry[]

@@map("sprints")

}

model Artifact {

id String @id @default(uuid())

type ArtifactType

name String

content String // JSON ou markdown

metadata Json? @default("{}")

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

sprint Sprint? @relation(fields: [sprintId], references: [id], onDelete: Cascade)

sprintId String?

@@map("artifacts")

}

model Template {

id String @id @default(uuid())

name String

description String?

type String // project, sprint, task, etc.

content Json

isPublic Boolean @default(false)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

organization Organization @relation(fields: [organizationId], references: [id], onDelete: Cascade)

organizationId String

@@map("templates")

}

// =====================================

// COLLABORATION

// =====================================

model Comment {

id String @id @default(uuid())

content String

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

author User @relation(fields: [authorId], references: [id], onDelete: Cascade)

authorId String

task Task? @relation(fields: [taskId], references: [id], onDelete: Cascade)

taskId String?

userStory UserStory? @relation(fields: [userStoryId], references: [id], onDelete: Cascade)

userStoryId String?

parentComment Comment? @relation("CommentReplies", fields: [parentCommentId], references: [id], onDelete: Cascade)

parentCommentId String?

replies Comment[] @relation("CommentReplies")

@@map("comments")

}

model Channel {

id String @id @default(uuid())

name String

description String?

type String @default("text") // text, voice, etc.

isPrivate Boolean @default(false)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

project Project? @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String?

team Team? @relation(fields: [teamId], references: [id], onDelete: Cascade)

teamId String?

messages Message[]

@@map("channels")

}

model Message {

id String @id @default(uuid())

content String

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

channel Channel @relation(fields: [channelId], references: [id], onDelete: Cascade)

channelId String

author User @relation(fields: [authorId], references: [id], onDelete: Cascade)

authorId String

@@map("messages")

}

model File {

id String @id @default(uuid())

name String

type FileType

size Int

url String

mimeType String

version Int @default(1)

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

uploader User @relation(fields: [uploaderId], references: [id], onDelete: Cascade)

uploaderId String

project Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

projectId String

@@map("files")

}

model Notification {

id String @id @default(uuid())

type NotificationType

title String

message String

data Json? @default("{}")

read Boolean @default(false)

createdAt DateTime @default(now())

// Relations

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

@@map("notifications")

}

model TimeEntry {

id String @id @default(uuid())

description String?

hours Float

date DateTime

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Relations

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

task Task? @relation(fields: [taskId], references: [id], onDelete: Cascade)

taskId String?

userStory UserStory? @relation(fields: [userStoryId], references: [id], onDelete: Cascade)

userStoryId String?

sprint Sprint? @relation(fields: [sprintId], references: [id], onDelete: Cascade)

sprintId String?

@@map("time\_entries")

}

model AuditLog {

id String @id @default(uuid())

action String

entity String

entityId String

oldValues Json?

newValues Json?

createdAt DateTime @default(now())

// Relations

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

userId String

@@map("audit\_logs")

}

// =====================================

// TABLES DE RELATIONS MANY-TO-MANY

// =====================================

model \_SprintUserStories {

A String

B String

@@unique([A, B])

@@map("\_SprintUserStories")

}

model \_TaskAssignees {

A String

B String

@@unique([A, B])

@@map("\_TaskAssignees")

}

// =====================================

// INDEX ET CONTRAINTES

// =====================================

// Les index sont ajoutés automatiquement par Prisma pour les relations

// Des index supplémentaires peuvent être ajoutés selon les besoins de performance

# Scénarios

Scénario 1 : Lancement d'un nouveau projet e-commerce

**Contexte :** TechInnovate vient de décrocher un contrat pour développer une plateforme e-commerce pour un client retail.

👤 **Acteurs**

* **Sarah** (Product Owner)
* **Marc** (Scrum Master)
* **Julie** (Lead Developer)
* **Alex** (Front-end Developer)
* **Emma** (Back-end Developer)
* **Tom** (Designer UX/UI)

📋 **Étapes détaillées**

**Phase 1 : Initialisation du projet (Jour 1)**

**1. Création de l'organisation et de l'équipe**

* L'admin TechInnovate crée l'organisation "TechInnovate"
* Il invite les membres et crée l'équipe "E-commerce Squad"
* Attribution des rôles : Sarah (PO), Marc (SM), autres (Developers)

**2. Création du projet depuis un template**

* Sarah crée le projet "ShopFlow Platform" en utilisant le template "E-commerce Web App"
* Le système génère automatiquement :
  + Structure d'épics prédéfinie (Authentification, Catalogue, Panier, Paiement, Admin)
  + Workflow de développement (To Do → In Progress → Code Review → Testing → Done)
  + Definition of Done standard

**3. Définition de la vision produit**

* Sarah rédige la vision dans l'outil collaboratif intégré
* Elle crée l'initiative "Plateforme E-commerce MVP" avec objectifs business
* Tom ajoute les wireframes et maquettes dans le gestionnaire de fichiers

**Phase 2 : Planification du backlog (Jour 2-3)**

**4. Création des épics et features**

* Sarah décompose l'initiative en épics :
  + Epic 1 : "Authentification utilisateur"
  + Epic 2 : "Catalogue produits"
  + Epic 3 : "Gestion du panier"
  + Epic 4 : "Processus de commande"

**5. Détail des user stories**

* Pour l'Epic "Authentification", Sarah crée :
  + US1 : "En tant qu'utilisateur, je veux créer un compte pour accéder à la plateforme"
  + US2 : "En tant qu'utilisateur, je veux me connecter avec email/mot de passe"
  + US3 : "En tant qu'utilisateur, je veux réinitialiser mon mot de passe oublié"

**6. Priorisation automatique**

* L'IA analyse la valeur business et suggère la priorisation :
  + Authentification (Valeur: 9/10, Complexité: 6/10) → Priorité HIGH
  + Catalogue (Valeur: 8/10, Complexité: 4/10) → Priorité HIGH
  + Panier (Valeur: 7/10, Complexité: 5/10) → Priorité MEDIUM

**Phase 3 : Sprint Planning (Jour 4)**

**7. Création du premier sprint**

* Marc crée le "Sprint 1" (2 semaines, 15-28 mars)
* Capacité calculée automatiquement : 40 story points (basé sur la vélocité historique de l'équipe)

**8. Estimation collaborative**

* L'équipe utilise le planning poker intégré pour estimer :
  + US1 : 8 story points (complexité authentification OAuth)
  + US2 : 5 story points (login standard)
  + US3 : 3 story points (réinitialisation mot de passe)

**9. Sélection des stories**

* Drag & drop des user stories prioritaires vers le Sprint Backlog
* L'IA suggère automatiquement les stories en fonction de la capacité
* Détection automatique des dépendances (US2 dépend de US1)

**Phase 4 : Développement et collaboration (Jour 5-18)**

**10. Décomposition en tâches**

* Julie décompose US1 en tâches techniques :
  + "Configurer OAuth Google/Facebook"
  + "Créer API endpoint /auth/register"
  + "Implémenter validation email"
  + "Créer interface signup React"

**11. Assignation automatique**

* L'IA suggère les assignations :
  + Alex → tâches front-end (basé sur expertise React)
  + Emma → tâches back-end (basé sur expertise Node.js)
  + Tom → tâches UI/UX (basé sur rôle designer)

**12. Suivi quotidien**

* **Daily Scrum automatisé** : Dashboard live avec :
  + Burndown chart mis à jour en temps réel
  + Blocages détectés automatiquement
  + Vélocité tracking avec alertes si dérive

**13. Collaboration temps réel**

* Alex commente directement sur la tâche : "Besoin de validation des maquettes finales"
* Tom est automatiquement mentionné et reçoit une notification smart
* Partage d'écran intégré pour la revue de design

**14. Gestion des blocages**

* Emma signale un blocage : "API tierce indisponible"
* L'outil détecte automatiquement l'impact sur les dépendances
* Suggestions automatiques de réorganisation des tâches

**Phase 5 : Revue et rétrospective (Jour 19-20)**

**15. Sprint Review**

* Démonstration des fonctionnalités via l'outil de présentation intégré
* Retours stakeholders collectés directement dans l'interface
* Métriques automatiques générées : 38/40 story points complétés

**16. Sprint Retrospective**

* Template rétro "Start/Stop/Continue" automatiquement généré
* Analyse IA des patterns de performance :
  + "Temps moyen de revue code : 4h (objectif : 2h)"
  + "Taux de defects : 2% (excellent)"
* Actions d'amélioration trackées pour le prochain sprint

📊 **Résultats mesurables**

* **Vélocité équipe** : 38 story points/sprint
* **Lead time moyen** : 3.2 jours
* **Taux de satisfaction** : 4.8/5 (basé sur feedback automatique)
* **Temps setup projet** : 2h (vs 8h avec outils traditionnels)

Scénario 2 : Gestion d'une crise en production

**Contexte :** L'application mobile "FitTracker" subit un bug critique qui empêche la synchronisation des données utilisateur.

👤 **Acteurs**

* **Lisa** (Product Owner)
* **David** (Scrum Master)
* **Kevin** (Lead Developer)
* **Sofia** (DevOps Engineer)
* **Pierre** (QA Engineer)

📋 **Étapes détaillées**

**Phase 1 : Détection et escalade (Jour 1 - 9h00)**

**1. Détection automatique**

* L'intégration monitoring déclenche un webhook
* Création automatique d'un bug critique : "Sync utilisateur en échec - 85% des utilisateurs impactés"
* Alertes automatiques envoyées à l'équipe via Slack/Teams

**2. Escalade intelligente**

* L'IA analyse l'impact business et escalade automatiquement
* Lisa (PO) et David (SM) reçoivent une notification haute priorité
* Création automatique d'un "Sprint d'urgence"

**3. Mobilisation de l'équipe**

* David active le mode "Crisis Management" dans l'outil
* Dashboard spécial affiché avec métriques temps réel
* Chat d'équipe dédié créé automatiquement

**Phase 2 : Investigation et diagnostic (9h15-11h00)**

**4. Analyse des dépendances**

* L'outil affiche automatiquement les composants impactés
* Visualisation des services dépendants (API Auth, Database, Cloud Storage)
* Historique des derniers déploiements avec liens Git

**5. Assignation par expertise**

* Kevin (expert backend) assigné automatiquement sur l'investigation
* Sofia (DevOps) mobilisée pour l'analyse infrastructure
* Pierre (QA) pour reproduire le bug

**6. Investigation collaborative**

* Kevin partage son écran via l'outil intégré
* Commentaires temps réel sur les logs partagés
* Hypothèses documentées directement dans le bug report

**Phase 3 : Résolution (11h00-14h00)**

**7. Création du fix**

* Kevin crée une branche hotfix liée automatiquement au bug
* Revue de code accélérée avec reviewers automatiquement notifiés
* Pipeline CI/CD déclenché automatiquement

**8. Tests en urgence**

* Pierre crée des tests de non-régression
* Validation sur environnement de staging
* Métriques de performance comparées automatiquement

**9. Déploiement coordonné**

* Sofia déclenche le déploiement via l'interface intégrée
* Monitoring en temps réel des métriques post-déploiement
* Rollback automatique préparé en cas d'échec

**Phase 4 : Validation et communication (14h00-16h00)**

**10. Validation du fix**

* Tests automatiques validés : 100% de réussite
* Métriques utilisateurs revenus à la normale
* Notification automatique aux stakeholders

**11. Post-mortem automatique**

* Génération automatique du rapport d'incident
* Timeline reconstituée avec tous les événements
* Actions correctives suggérées par l'IA

**12. Communication publique**

* Template de communication automatiquement généré
* Historique des actions pour transparence client
* Métriques d'impact calculées automatiquement

📊 **Résultats mesurables**

* **Temps de résolution** : 5h (vs 12h objectif)
* **Temps moyen première réponse** : 15 minutes
* **Taux de disponibilité récupéré** : 99.9%
* **Satisfaction équipe** : 4.2/5 (gestion de crise)

Scénario 3 : Scaling d'une équipe pour un projet complexe

**Contexte :** L'entreprise "CloudTech" doit développer une plateforme SaaS complexe nécessitant 3 équipes et 15 développeurs.

👤 **Acteurs**

* **Marine** (Product Owner principal)
* **Lucas** (Scrum Master Team A)
* **Amélie** (Scrum Master Team B)
* **Julien** (Scrum Master Team C)
* **15 développeurs** répartis en 3 équipes

📋 **Étapes détaillées**

**Phase 1 : Architecture multi-équipes (Semaine 1)**

**1. Création de l'organisation scalable**

* Admin CloudTech crée la structure :
  + Organisation "CloudTech"
  + 3 équipes : "Core Platform", "Analytics", "Security"
  + Projet principal "SaaS Platform v2" avec sous-projets

**2. Définition des rôles et permissions**

* Marine : Product Owner principal avec vue globale
* Lucas/Amélie/Julien : Scrum Masters avec autonomie sur leurs équipes
* Développeurs : accès limité à leurs domaines respectifs

**3. Architecture du work breakdown**

* Initiative "SaaS Platform v2"
* 3 épics principales par équipe :
  + Core Platform : "API Gateway", "User Management", "Billing"
  + Analytics : "Data Pipeline", "Real-time Dashboard", "Reporting"
  + Security : "Authentication", "Authorization", "Audit Trail"

**Phase 2 : Planification coordonnée (Semaine 2)**

**4. Backlog management distribué**

* Chaque équipe maintient son backlog spécialisé
* Vue consolidée pour Marine (Product Owner)
* Priorisation automatique basée sur les dépendances inter-équipes

**5. Détection des dépendances cross-team**

* L'IA identifie automatiquement :
  + "User Management" (Core) bloque "Authentication" (Security)
  + "API Gateway" (Core) requis pour "Real-time Dashboard" (Analytics)
* Visualisation graphique des dépendances

**6. Synchronisation des sprints**

* Sprints alignés sur 2 semaines pour les 3 équipes
* Calendrier unifié avec jalons communs
* Capacity planning automatique par équipe

**Phase 3 : Développement coordonné (Semaine 3-6)**

**7. Collaboration inter-équipes**

* Channels dédiés par domaine + channel général
* Mentions automatiques entre équipes sur les dépendances
* Partage de composants via bibliothèque commune

**8. Gestion des conflits de ressources**

* Détection automatique des surcharges
* Suggestions de réallocation entre équipes
* Alerte automatique si vélocité d'une équipe impacte les autres

**9. Intégration continue coordonnée**

* Pipelines CI/CD coordonnés entre équipes
* Tests d'intégration automatiques cross-team
* Notifications automatiques en cas de breaking changes

**Phase 4 : Reporting et gouvernance (Continu)**

**10. Dashboards multi-niveaux**

* **Niveau équipe** : Métriques locales (vélocité, burndown)
* **Niveau projet** : Vue consolidée pour Marine
* **Niveau direction** : ROI, budget, timeline globale

**11. Métriques de collaboration**

* Temps de résolution des dépendances inter-équipes
* Taux de blocages cross-team
* Satisfaction de collaboration (sondages automatiques)

**12. Optimisation continue**

* Analyse prédictive des goulots d'étranglement
* Recommandations de réorganisation
* Benchmarking automatique avec projets similaires

**Phase 5 : Scaling et optimisation (Semaine 7-12)**

**13. Onboarding automatisé**

* Nouveau développeur rejoint l'équipe Analytics
* Accès automatique aux ressources pertinentes
* Parcours d'onboarding personnalisé

**14. Évolution de l'architecture**

* Création d'une 4ème équipe "Mobile"
* Réorganisation automatique des dépendances
* Migration transparente des user stories

**15. Gouvernance avancée**

* Reporting automatique aux stakeholders
* Prédiction des délais par équipe
* Alertes automatiques sur les risques projet

📊 **Résultats mesurables**

* **Vélocité globale** : 180 story points/sprint (3 équipes)
* **Temps de résolution des dépendances** : 1.5 jours moyenne
* **Satisfaction inter-équipes** : 4.6/5
* **Réduction des conflits** : 70% vs approche traditionnelle
* **Time-to-market** : 6 mois vs 9 mois estimé initialement

🎯 **Synthèse des bénéfices démontrés**

**Gains de productivité**

* **80% de réduction** du temps de setup projet
* **50% de réduction** du temps de résolution des blocages
* **40% d'amélioration** de la prédictibilité des délais

**Amélioration de la collaboration**

* **Visibilité temps réel** sur l'ensemble des activités
* **Communication contextualisée** réduisant les malentendus
* **Prise de décision éclairée** basée sur des données objectives

**Qualité et gouvernance**

* **Traçabilité complète** des décisions et changements
* **Compliance automatique** avec les processus agiles
* **Amélioration continue** guidée par l'analyse des données

Ces trois scénarios illustrent comment l'application révolutionne la gestion de projet agile en automatisant les tâches répétitives, en améliorant la collaboration et en fournissant des insights intelligents pour une prise de décision optimale.