Faculté des Sciences et Techniques
Cycle d'Ingénieur – Génie Informatique
Projet de Fin d'Année (PFA)

Cahier des Charges

Plateforme d'Intelligence Opérationnelle et d'Optimisation de Code par IA

Version : 1.0
Date : 09 Juillet 2025

Auteur : RAIDI Oualid

Encadrant : ENABIH Abdessamad

Table des matières

1 Introduction

1.1 But du Document

Le présent document a pour but de spécifier en détail les exigences fonctionnelles et non fonctionnelles du projet *IntelliDevFlow*, une plateforme web d'intelligence opérationnelle et d'optimisation de code par IA. Il servira de référence principale pour toutes les phases de conception, de développement, de test et de validation du projet.

1.2 Portée du Projet

IntelliDevFlow est une application web visant à centraliser et optimiser le suivi de projet et l'analyse de la qualité du code au sein des équipes de développement logiciel. Elle intègre des fonctionnalités d'IA pour aller au-delà du simple suivi de tâches, en fournissant des insights sur les commits et des suggestions d'optimisation de code.

1.3 Public Cible et Utilisateurs

La plateforme s'adresse à trois catégories principales d'utilisateurs :

— **Directeurs/Responsables** : Besoin d'une vue macro stratégique, de KPIs de performance projet et équipe.

— **Managers/Chefs de Projet** : Suivi détaillé des tâches, gestion des ressources, identification des risques, insights sur l'efficacité des équipes.

— **Développeurs** : Feedback direct sur le code, suggestions d'optimisation, suivi de leurs contributions et de l'évolution de la qualité de leur code.

1.4 Définitions, Acronymes et Abréviations

— **PFA** : Projet de Fin d'Année

— **IA** : Intelligence Artificielle

— **UI** : User Interface (Interface Utilisateur)

— **UX** : User Experience (Expérience Utilisateur)

— **SPA** : Single Page Application

— **API** : Application Programming Interface

— **MVP** : Minimum Viable Product (Produit Minimum Viable)

— **KPI** : Key Performance Indicator (Indicateur Clé de Performance)

— **2FA** : Two-Factor Authentication (Authentification à Deux Facteurs)

— **Code Smells** : Indicateurs de problèmes de qualité dans le code

— **Commit** : Enregistrement des modifications apportées au code source dans un système de contrôle de version comme Git

1.5 Références

— Lien vers Maquettes Figma du projet

— Lien vers Dépôt GitHub du projet (quand créé)

— Normes IEEE 830 (pour la rédaction des SRS) – Référence académique, non obligatoire de suivre à la lettre pour un PFA mais utile pour la structure

2 Description Générale

2.1 Perspectives du Produit

IntelliDevFlow est une application web autonome qui peut être intégrée via API avec des services tiers comme GitHub. Elle vise à être la plateforme centrale pour les équipes de développement souhaitant optimiser leurs processus et la qualité de leur produit logiciel via des données intelligentes.

2.2 Fonctions du Produit

Les fonctions principales du produit incluent :

— Gestion et authentification des utilisateurs (rôles différenciés)

— Création, gestion et suivi de projets et de tâches

— Intégration avec des dépôts Git (notamment GitHub) pour l'extraction de commits et de code

— Analyse intelligente des commits et du code source (détection de code smells, complexité, duplication)

— Génération de suggestions d'optimisation de code basées sur l'analyse

— Visualisation graphique de la progression des tâches, de la performance des équipes et des tendances de qualité de code

— Génération de rapports personnalisés pour différents niveaux de gestion

2.3 Caractéristiques des Utilisateurs

— **Directeurs/Responsables** : Maîtrise des outils de gestion, besoin de rapports synthétiques

— **Managers/Chefs de Projet** : Habitués aux plateformes de gestion de projet, souci du détail et de l'efficacité d'équipe

— **Développeurs** : Familiarisés avec Git, les concepts de qualité de code, et ouverts aux outils d'amélioration continue

Tous les utilisateurs sont des professionnels recherchant une interface intuitive et des données fiables.

2.4 Contraintes Générales

— **Technologiques** : Utilisation de Laravel (Backend), Vue.js (Frontend), MySQL (Base de Données), Git/GitHub (Contrôle de Version)

— **Sécurité** : Authentification robuste, protection des données de projet et de code

— **Performance** : Temps de réponse rapide pour les tableaux de bord et l'exécution des analyses

— **Intégration** : Dépendance de l'API GitHub pour l'extraction des données de commits

2.5 Hypothèses et Dépendances

— Les utilisateurs ont une connexion internet stable pour accéder à la plateforme

— Les dépôts de code à analyser sont hébergés sur GitHub (pour le MVP)

— La configuration de l'environnement de développement (Laragon, VS Code, XDebug) est fonctionnelle

3 Exigences Fonctionnelles Détaillées

3.1 Module d'Authentification et Gestion des Utilisateurs

3.1.1 F.1.1 : Inscription des utilisateurs

— Le système **DOIT** permettre aux nouveaux utilisateurs de s'inscrire avec une adresse e-mail et un mot de passe sécurisé.

— Le système **DOIT** envoyer un e-mail de vérification après l'inscription.

3.1.2 F.1.2 : Connexion des utilisateurs

— Le système **DOIT** permettre aux utilisateurs de se connecter via e-mail/mot de passe.

— Le système **DOIT** permettre la connexion via des fournisseurs tiers (GitHub, Google).

— Le système **DOIT** supporter l'authentification à deux facteurs (2FA) pour une sécurité renforcée.

3.1.3 F.1.3 : Réinitialisation de mot de passe

— Le système **DOIT** permettre aux utilisateurs de réinitialiser leur mot de passe via e-mail.

3.1.4 F.1.4 : Gestion de profil utilisateur

— Le système **DOIT** permettre aux utilisateurs de consulter et modifier leurs informations de profil (nom, e-mail, mot de passe).

— Le système **DOIT** permettre l'activation/désactivation du 2FA.

3.1.5 F.1.5 : Gestion des rôles et permissions

— Le système **DOIT** distinguer au moins trois rôles : Développeur, Manager, Directeur, chacun avec des permissions spécifiques d'accès aux fonctionnalités et aux données.

3.2 Module de Gestion de Projet et de Tâches

3.2.1 F.2.1 : Création et gestion de projets

— Le système **DOIT** permettre aux Managers/Directeurs de créer de nouveaux projets (nom, description, date de début/fin, équipe associée).

— Le système **DOIT** permettre de modifier et supprimer les projets existants.

3.2.2 F.2.2 : Gestion des tâches au sein d'un projet

— Le système **DOIT** permettre de créer, attribuer, modifier (statut, priorité, échéance) et supprimer des tâches.

— Les développeurs **DOIVENT** pouvoir mettre à jour le statut de leurs tâches.

3.2.3 F.2.3 : Suivi de la progression des tâches

— Le système **DOIT** afficher la progression individuelle des tâches et la progression globale du projet.

3.2.4 F.2.4 : Gestion des membres d'équipe

— Le système **DOIT** permettre aux Managers/Directeurs d'ajouter et de gérer les membres d'une équipe pour chaque projet.

3.3 Module d'Intégration et d'Analyse de Code/Commits par IA

3.3.1 F.3.1 : Connexion aux dépôts Git (GitHub)

— Le système **DOIT** permettre aux Managers de connecter un projet à un dépôt GitHub spécifique.

— Le système **DOIT** pouvoir récupérer l'historique des commits du dépôt connecté.

3.3.2 F.3.2 : Analyse des messages de commits

— Le système **DOIT** analyser la longueur, la clarté et la pertinence des messages de commit.

— Le système **DOIT** identifier les patterns de commits (fréquence, régularité, taille moyenne).

3.3.3 F.3.3 : Analyse de la qualité du code (Statique)

— Le système **DOIT** détecter les *code smells* courants (code dupliqué, fonctions trop complexes, violations de conventions).

— Le système **DOIT** calculer des métriques de complexité (ex : complexité cyclomatique) pour les fonctions et classes.

— Le système **DOIT** pouvoir appliquer des règles d'analyse statique spécifiques au langage (PHP, JavaScript).

3.3.4 F.3.4 : Génération de suggestions d'optimisation

— Le système **DOIT** générer des suggestions d'amélioration concrètes et actionnables basées sur l'analyse de code (ex : "Refactoriser cette fonction", "Supprimer ce code dupliqué", "Ajouter des commentaires").

— Les suggestions **DOIVENT** être associées aux commits ou aux lignes de code concernées.

3.3.5 F.3.5 : Suivi des tendances de qualité de code

— Le système **DOIT** enregistrer et afficher l'évolution des métriques de qualité de code au fil du temps pour un projet ou par développeur.

3.4 Module de Visualisation et Rapports

3.4.1 F.4.1 : Tableau de bord général (Directeur/Manager)

— Le système **DOIT** afficher un tableau de bord synthétique avec les KPIs clés (progression globale des projets, état des risques, tendances de qualité de code).

3.4.2 F.4.2 : Tableau de bord de projet détaillé (Manager/Développeur)

— Le système **DOIT** présenter un tableau de bord spécifique à chaque projet avec la progression des tâches, la performance de l'équipe, et les résumés d'analyse de code.

3.4.3 F.4.3 : Tableau de bord personnel du développeur

— Le système **DOIT** fournir à chaque développeur une vue de ses contributions, des commits analysés et des suggestions d'optimisation le concernant.

3.4.4 F.4.4 : Visualisations interactives

— Toutes les données **DOIVENT** être présentées via des graphiques interactifs (barres, lignes, radars, etc.) avec des filtres et des drill-downs.

3.4.5 F.4.5 : Rapports exportables

— Le système **DEVRAIT** permettre l'exportation de certains rapports (ex : PDF, CSV).

3.5 Module de Notifications et Alertes

3.5.1 F.5.1 : Notifications internes

— Le système **DOIT** envoyer des notifications aux utilisateurs pour les changements de statut de tâche, les nouvelles attributions, ou les suggestions d'optimisation importantes.

3.5.2 F.5.2 : Alertes intelligentes

— Le système **DOIT** générer des alertes pour les managers en cas de détection de problèmes majeurs dans l'analyse de code ou de retards significatifs.

4 Exigences Non Fonctionnelles

4.1 Performance

— **P.4.1.1** : Temps de réponse des tableaux de bord : Les tableaux de bord principaux **DOIVENT** se charger en moins de 3 secondes pour un volume de données standard (ex : 10 projets, 500 commits).

— **P.4.1.2** : Temps d'analyse de code : L'analyse d'un dépôt avec X commits **DOIT** être complétée en Y minutes (à définir précisément après quelques benchmarks).

4.2 Sécurité

— **S.4.2.1** : Authentification robuste : Utilisation de l'authentification par jeton (ex : JWT ou Laravel Sanctum) pour les communications API.

— **S.4.2.2** : Protection des données : Toutes les données sensibles (informations utilisateur, données de code) **DOIVENT** être stockées de manière sécurisée (hashage des mots de passe, chiffrement si nécessaire).

— **S.4.2.3** : Gestion des autorisations : Accès aux fonctionnalités et données basé sur les rôles et permissions définis.

— **S.4.2.4** : Protection contre les attaques courantes : Le système **DOIT** être protégé contre les attaques OWASP Top 10 (Injection SQL, XSS, CSRF, etc.).

— **S.4.2.5** : Sécurité des intégrations : L'intégration avec GitHub **DOIT** utiliser OAuth 2.0 avec les permissions minimales requises.

4.3 Utilisabilité (UX/UI)

— **U.4.3.1** : Interface intuitive : L'interface utilisateur **DOIT** être simple à comprendre et à naviguer pour tous les rôles.

— **U.4.3.2** : Cohérence visuelle : Tous les écrans **DOIVENT** respecter le Design System et la charte graphique définie dans Figma.

— **U.4.3.3** : Responsivité : L'interface **DOIT** être entièrement responsive et utilisable sur les navigateurs web de bureau et mobiles (bien que l'accent soit mis sur le bureau pour un PFA).

— **U.4.3.4** : Accessibilité : Les éléments d'interface **DOIVENT** respecter les normes d'accessibilité de base (ex : contrastes suffisants, labels pour les champs).

4.4 Fiabilité

— **F.4.4.1** : Taux de disponibilité : Le système **DEVRAIT** viser un taux de disponibilité de 99%.

— **F.4.4.2** : Gestion des erreurs : Le système **DOIT** afficher des messages d'erreur clairs et utiles aux utilisateurs en cas de problème.

4.5 Maintenabilité

— **M.4.5.1** : Code propre : Le code **DOIT** être propre, bien commenté et suivre les conventions de codage (PSR pour PHP, ESLint pour JS).

— **M.4.5.2** : Documentation technique : Une documentation claire de l'API, de l'architecture et des composants clés **DOIT** être fournie.

4.6 Scalabilité

— **S.4.6.1** : Gestion de la croissance des données : La base de données **DOIT** être conçue pour gérer un volume croissant de projets, tâches et données d'analyse de code.

— **S.4.6.2** : Charges concurrentes : Le système **DEVRAIT** être capable de supporter un nombre X d'utilisateurs concurrents (à estimer pour le PFA).

4.7 Conformité et Réglementations

— **C.4.7.1** : GDPR/RGPD (simplifié pour PFA) : Le système **DOIT** respecter les principes de base de protection des données personnelles (consentement, droit à l'oubli simplifié, etc.).