PLAN D'ASSURANCE QUALITE LOGICIEL

TRAVAIL PRESENTE A
Mr ESSABAR Driss

SYSTEM KANBAN

PAR L'ÉQUIPE :
Mouad Nadzi
Asmaa El Bouazzaoui
Mohamed AlMahdy Baâkrim
Abderrahmane Bensalek
Aya El Feddani

LE 30 décembre 2024

Ecole Marocaine des sciences d'ingénieur

Table des matières

1 But

Le projet vise à créer un système Kanban pour la gestion de flux de travail, permettant de suivre les tâches et leur évolution à travers une interface intuitive. Ce système est conçu pour être robuste, évolutif et facile à maintenir, tout en garantissant une haute qualité tout au long de son cycle de développement.

2 Organisation

Chaque tâches est attribué à un membre dédié. BAAKRIM Mohamed ALMahdy, responsable QA, est assignée pour vérifier l'ensemble des services et assurer la cohérence globale.

2.1 Tâches

- Rédaction de la documentation
- Revues de code hebdomadaires
- Tests unitaires et d'intégration
- Validation de la visualisation graphique
- Amélioration continue du processus QA

2.2 Responsabilités

- **Responsable QA** : Validation globale et cohérence interservices
- **Chef de projet** : Suivi du planning et coordination
- **Développeurs** : Conception et développement

3 Documentation

3.1 But

Cette section vise à assurer la traçabilité du développement et à faciliter la gestion de la maintenance.

3.2 Documentation minimale exigée

Les documents suivants seront produits :

- PAQL : Plan d'assurance qualité logiciel (ce document)
- PGC : Plan de gestion de la configuration
- PGL : Plan de gestion de projet
- PVT : Plan de validation et tests
- DTL : Document de tests logiciels
- Manuel utilisateur
- Documentation des API

3.3 Autres

Cette section est non-applicable.

4 Standards, pratiques, conventions et métriques

4.1 But

Cette section a pour objectif d'assurer la cohérence du code et des pratiques d'assurance qualité.

4.2 Contenu

Normes de Codage :

- Respect des conventions de codage propres à Angular pour le développement front-end, afin d'assurer la lisibilité, la maintenabilité et la cohérence du code de l'interface utilisateur.
- Application des bonnes pratiques de Spring Boot pour le développement back-end, en mettant l'accent sur la scalabilité, la sécurité et l'optimisation des performances.

Pratiques d'Assurance Qualité :

- Tests unitaires : Utilisation de JUnit pour Spring Boot afin de valider les composants individuels et garantir leur bon fonctionnement au niveau modulaire.
- Tests fonctionnels automatisés : Mise en œuvre de WebDriver pour effectuer des tests bout-en-bout simulant les interactions réelles des utilisateurs avec le système.
- Couverture de code : Maintien d'une couverture de code minimale de 90 % pour s'assurer que les fonctionnalités essentielles sont rigoureusement testées.

Métriques de Performance :

- Temps de réponse des API : Les services backend doivent répondre en moins de 100 millisecondes afin de garantir des performances optimales.
- Taux de défauts : Limitation du nombre de défauts à un maximum de 5 par 1 000 lignes de code, afin d'assurer un niveau de qualité élevé.

Exigences des Tests Automatisés :

- Taux de réussite des tests continus : Au moins 90 % des tests automatisés doivent réussir lors des processus d'intégration continue pour garantir la stabilité du code.
- Couverture des fonctionnalités critiques : 100 % des fonctionnalités critiques du système doivent être couvertes par des tests unitaires et des tests d'intégration afin de prévenir toute défaillance dans les processus essentiels.

Seuil d'erreurs post-déploiement :

- Chaque version livrée ne doit pas comporter plus de deux bugs critiques afin de préserver la fiabilité du système et la satisfaction des utilisateurs.

5 Revues et Audits

5.1 But

Cette section vise à focaliser l'attention des développeurs sur la qualité de l'application en développement, en assurant une vérification régulière et structurée à chaque étape clé du projet.

5.2 Exigences minimales

5.2.1 Revue des exigences logicielles

Une revue complète des spécifications fonctionnelles sera réalisée avec l'ensemble de l'équipe. Elle sera dirigée par le chef de projet, qui planifie et organise la réunion.
Les exigences seront révisées à trois étapes clés :

- Lors de la présentation du brouillon initial.
- Après les retours de la version intermédiaire.
- Avant la validation finale des spécifications.

5.2.2 Revue de design préliminaire

La revue de design préliminaire porte sur l'architecture du système Kanban, y compris la structure des tâches, l'interaction entre Angular et Spring Boot, et la communication entre les différentes couches du système. Cette revue est menée par le chef de projet et implique toute l'équipe.

Les différentes options de design (structure des tâches, gestion des états dans Angular, communication via API REST) seront discutées pour sélectionner la solution la plus appropriée.

5.2.3 Revue de design critique

Avant de commencer le développement, une revue approfondie de l'architecture finale est effectuée pour évaluer chaque tâche (back-end avec Spring Boot, front-end avec Angular). L'objectif est de valider l'intégration globale du système, les algorithmes utilisés, et les performances attendues.

Chaque tâche (gestion des boards, gestion des sprints, affichage des tâches, gestion des utilisateurs) sera évaluée par des inspections techniques.

5.2.4 Revue de PVVL

Cette revue a pour but de garantir que l'ensemble des fonctionnalités du système Kanban est couvert par des tests adaptés. La revue est dirigée par le responsable de l'AQ, et tous les membres participent pour identifier les cas critiques et s'assurer que les tests sont alignés avec les objectifs du projet.
Les tests de performance (réponse rapide de l'API) et les tests limites (gestion de très nombreux utilisateurs ou tâches) seront minutieusement révisés.

5.2.5 Audit fonctionnel

L'audit fonctionnel est effectué avant chaque mise en production. Il garantit que toutes les fonctionnalités (création, mise à jour, suppression de tâches, gestion des utilisateurs) sont pleinement opérationnelles et conformes aux exigences définies.
Les audits incluent des tests de bout en bout pour valider l'interface Angular et les services back-end Spring Boot.

5.2.6 Audit physique

Avant chaque livraison, le responsable AQ vérifie que tous les livrables (code, documentation, images Docker pour les tâches Spring Boot) sont complets et correctement emballés.
Les audits physiques incluent la validation des images Docker et de la configuration pour chaque tâche

5.2.7 Audit en préparation

Aucun audit formel en préparation n'est prévu en raison de la nature agile et modulaire du projet. Cependant, des revues continues et des tests réguliers sont effectués tout au long du cycle de développement pour limiter les risques.

5.2.8 Revue de gestion

Toutes les deux semaines, une revue de gestion est conduite par le chef de projet, en présence de toute l'équipe. L'objectif est de suivre l'avancement des tâches, identifier les blocages et s'assurer que les livrables sont conformes au planning.
Les revues de gestion incluent des analyses des performances, des bugs critiques, ainsi que des retours des utilisateurs.

5.2.9 Revue finale

Après la livraison, une revue finale est réalisée pour documenter les leçons apprises et identifier les améliorations possibles pour de futurs développements.
Le responsable AQ rédigera un rapport détaillant les succès et les points à améliorer rencontrés durant les différentes phases du projet.

5.3 Inspections

Tous les livrables (code, documentation, tests) seront inspectés par au moins un membre de l'équipe.

L'inspection du code source comprendra :

- La validation du respect des conventions de codage Angular (ESLint) et des bonnes pratiques Spring Boot.
- La vérification des performances des algorithmes critiques du back-end (gestion des tâches et interactions avec la base de données).
- La cohérence de l'interface utilisateur Angular avec les spécifications initiales du projet.

6 Test

6.1 But

L'objectif de cette section est de décrire les stratégies et méthodologies de test utilisées pour garantir la qualité, la robustesse et la performance du système Kanban. Chaque tâche du projet (gestion des tâches, interaction utilisateur, et communication entre Angular et Spring Boot) sera testée de manière rigoureuse afin de prévenir les régressions, valider les fonctionnalités et assurer la stabilité de l'interface utilisateur.

6.2 Stratégie de Test

Les tests sont effectués à plusieurs niveaux pour couvrir l'ensemble des composants du projet :

- ✓ **Tests unitaires** :

Chaque tâche du projet (front-end avec Angular et back-end avec Spring Boot) est testée de manière isolée pour valider le bon fonctionnement des différentes fonctions et algorithmes.
Frameworks utilisés :

- **Spring Boot (back-end) : JUnit 5 pour tester les services RESTful, les contrôleurs et la logique métier, et même les entités.**

Objectif :

Atteindre une couverture minimale de 90% du code pour les fonctionnalités critiques (création, modification et suppression de tâches, gestion des utilisateurs).

- ✓ **Tests d'intégration** :

Vérification de l'interopérabilité entre le front-end (Angular) et le back-end (Spring Boot).

- ✓ Simulation de requêtes HTTP entre le front-end et le back-end pour tester l'intégration des API REST

et valider l'interaction avec la base de données.

- ✓ **Tests fonctionnels :**
    - Tests de bout en bout pour vérifier le bon fonctionnement de l'interface utilisateur, notamment la création de tâches, la gestion des priorités et l'affichage des informations en temps réel.
    - **Outils :**
    - **Selenium WebDriver** pour automatiser l'interaction avec l'interface utilisateur Angular, simuler des actions utilisateur telles que la création de tâches, la modification de leur statut, et la gestion des priorités.
    - **Objectif :** Détecter les anomalies dans l'interface utilisateur et valider l'expérience utilisateur complète, depuis l'ajout de tâches jusqu'à l'affichage des informations sur le tableau Kanban.
- ✓ **Tests de charge et de performance** :
    - Ces tests mesurent la capacité du système Kanban à gérer une charge élevée d'utilisateurs simultanés et de nombreuses tâches en temps réel.
    - **Outils : JMeter** pour simuler des charges importantes sur le système, tester la réactivité et la stabilité du tableau Kanban sous une forte charge.
    - **Critères :**
- ✓
    - Temps de réponse des API ≤ 100ms pour les requêtes liées à la gestion des tâches (création, mise à jour, suppression).
    - Gestion de 200 requêtes simultanées sans dégradation des performances du tableau Kanban et de l'interface utilisateur.

7 Notification des problèmes et corrections

Tous les problèmes identifiés durant le développement du système Kanban sont signalés et suivis à travers **Trello**. Lorsqu'un bug, une anomalie ou une fonctionnalité manquante est détecté, une carte est immédiatement créée dans le tableau dédié. Chaque carte comporte une description détaillée du problème, les étapes pour le reproduire, ainsi que les logs ou captures d'écran pertinents. Les problèmes sont catégorisés par priorité (Haute, Moyenne, Faible) afin de faciliter leur gestion.

Le tableau **Trello** est structuré en plusieurs colonnes reflétant l'état d'avancement des corrections :

- À faire
- En cours
- En revue
- Résolus
- Bloqués

Lorsqu'un problème est résolu, il est déplacé dans la colonne **Résolus** après validation par un autre membre de l'équipe ou après exécution réussie des tests automatisés (tests d'intégration, tests fonctionnels). Les problèmes critiques bloquant des fonctionnalités essentielles, comme la gestion des tâches ou l'interface utilisateur, sont traités en priorité. Les tâches bloquées sont signalées dans la colonne correspondante pour permettre une escalade rapide. Ce processus garantit une traçabilité complète des anomalies et une réactivité efficace, assurant ainsi la stabilité et la qualité globale du système Kanban.

8 Outils, techniques et méthodologies

Angular : Frontend, gestion dynamique des tâches

Spring Boot : Backend, gestion des API RESTful

Docker : Conteneurisation des services backend

GitHub : Contrôle de version pour la gestion des sources

GitHub Actions, Jenkins : Pipelines CI/CD pour l'intégration continue et le déploiement

Qualité : SonarQube pour l'analyse de la qualité du code, JMeter pour les tests de performance et WebDrive.

9 Contrôle du code

Le projet Kanban est organisé autour de plusieurs repositories Git distincts, correspondant aux différentes parties du système :
Backend (Spring Boot) : gestion des API, traitement des données
Frontend (Angular) : gestion de l'interface utilisateur, interactions avec l'API
Cette séparation permet de structurer clairement chaque composant, facilitant la gestion et l'évolution indépendante des tâches front-end et back-end. Chaque repository suit la convention GitFlow, avec des branches dédiées au développement, aux nouvelles fonctionnalités et aux corrections de bugs. Cette approche modulaire offre une meilleure visibilité sur l'état d'avancement de chaque partie du projet, permettant de suivre précisément la progression des tâches et d'identifier rapidement les points de blocage. Les pull requests et les revues de code sont appliquées systématiquement sur chaque repository, assurant ainsi une qualité constante à chaque étape du projet.

10 Formation

Des sessions de formation internes sont organisées régulièrement pour renforcer les compétences de l'équipe sur les technologies clés du projet Kanban :

- **Angular** pour le développement frontend
- **Spring Boot** pour la gestion backend
- **Docker** pour la conteneurisation
- **GitHub** pour la gestion de version et les bonnes pratiques Git

Ces formations couvrent à la fois les bases et les cas d'utilisation avancés, assurant que chaque membre de l'équipe maîtrise les outils nécessaires pour développer, maintenir et améliorer le système Kanban.

11 Gestion du risque

La gestion des risques est intégrée au cycle de développement de chaque tâche du projet Kanban. Une évaluation hebdomadaire est réalisée pour identifier les points critiques susceptibles de provoquer des retards ou des dysfonctionnements dans le processus de gestion des tâches, de l'interface utilisateur ou des interactions entre le front-end et le back-end. Des ajustements sont faits régulièrement pour limiter ces risques et assurer la stabilité du projet.