Plan de Qualité pour le Projet "Buildinguerie"

1. Objectifs du Plan de Qualité

Le plan de qualité du projet "Buildinguerie" vise à définir les standards, processus et critères nécessaires pour garantir que le produit final répond aux attentes des utilisateurs et aux exigences fonctionnelles et techniques spécifiées. Les principaux objectifs sont :

- Assurer la conformité avec les exigences du cahier des charges.
- Garantir la fiabilité, la performance et la sécurité de l'application.
- Fournir une expérience utilisateur fluide et intuitive.
- Faciliter la maintenance et les mises à jour futures.

2. Responsabilités et Équipe Qualité

Le responsable qualité et le développeur principal sont chargés de la mise en œuvre et du suivi du plan de qualité. Ils travaillent en étroite collaboration pour s'assurer que les objectifs de qualité sont atteints.

3. Processus de Développement

3.1. Méthodologie

- **Méthodologie Agile** : Adoption d'une méthodologie Agile pour permettre une adaptation rapide aux changements et une amélioration continue.
- **Sprints** : Les différents sprints ont déjà été définis sur Jira. L'équipe se réunira à la fin de chaque sprint pour débriefer ce dernier et éventuellement planifier les tâches qui n'ont pas pu être finis à temps.

3.2. Documentation

- **Cahier des Charges** : Document initial détaillant les exigences fonctionnelles et techniques.
- **Spécifications Techniques** : Documentation des technologies utilisées, des architectures et des interfaces.
- **Plan de Test** : Documentation des stratégies et des cas de test pour valider chaque fonctionnalité.

4. Normes et Standards

- **Conformité au RGPD** : Garantir la protection des données personnelles des utilisateurs conformément au Règlement Général sur la Protection des Données.
- **Sécurité des Transactions** : Utilisation de protocoles HTTPS et conformité avec les normes PCI-DSS pour les paiements en ligne.
- **Code de Conduite** : Respect des bonnes pratiques de codage, de la documentation et des revues de code régulières.

5. Stratégies de Test

5.1. Types de Tests

- **Tests Unitaires** : Chaque composant sera testé individuellement pour vérifier son bon fonctionnement.
- **Tests d'Intégration** : Validation de l'interaction entre différents modules et composants.
- **Tests Fonctionnels** : Vérification de la conformité avec les exigences fonctionnelles spécifiées.
- **Tests de Performance** : Évaluation de la performance sous différentes charges pour garantir la réactivité de l'application.
- **Tests de Sécurité** : Identification et correction des vulnérabilités potentielles.
- **Tests d'Acceptation Utilisateur** : Validation par un groupe d'utilisateurs finaux pour assurer que l'application répond à leurs besoins et attentes.

5.2. Outils de Test

Les outils utilisés pour les tests sont laissés au choix de l'équipe de test. Cette approche permet à l'équipe de sélectionner les outils les plus appropriés et efficaces en fonction de leur expertise et des besoins spécifiques du projet. Qu'il s'agisse de frameworks de tests unitaires, d'outils de tests automatisés pour l'interface utilisateur, ou de solutions pour les tests de performance et de sécurité, l'équipe de test a la liberté d'adopter les technologies qui garantissent la meilleure qualité et la couverture la plus complète pour l'application "Buildinguerie".

6. Gestion des Défaillances et Améliorations

- **Suivi des Bugs** : Utilisation d'un système de suivi des bugs (Jira) pour enregistrer, suivre et gérer les problèmes.

- **Revues de Code** : Réalisation de revues de code régulières pour identifier et corriger les problèmes de qualité du code.
- **Feedback Utilisateur** : Collecte continue des retours des utilisateurs pour identifier les améliorations possibles et les problèmes non détectés.

7. Mises à Jour et Maintenance

- **Plan de Mises à Jour** : Planification des mises à jour régulières pour intégrer de nouvelles fonctionnalités, corriger les bugs et améliorer la performance.
- **Documentation de Maintenance** : Fourniture d'une documentation complète pour faciliter la maintenance future de l'application.

8. Suivi et Reporting

- **Rapports de Qualité** : Génération de rapports réguliers sur l'état de la qualité, incluant les résultats des tests, les défaillances détectées et les actions correctives prises.
- **Réunions de Revue** : Tenue de réunions de revue de qualité à la fin de chaque sprint pour évaluer les progrès et ajuster les stratégies si nécessaire.

9. Formation et Support

- **Formation Continue** : Sessions de formation régulières pour s'assurer que les meilleures pratiques et outils de qualité sont utilisés.
- **Support Utilisateur** : Mise en place d'un système de support utilisateur pour aider les utilisateurs à résoudre les problèmes et répondre à leurs questions.

10. Utilisation de Branches GitHub pour la Gestion des Environnements

10.1. Branches de Développement

- **Branch "main"** : Utilisée pour la version stable de l'application, cette branche représente la version de production déployée et accessible aux utilisateurs finaux.
- **Branch "develop"** : Branche principale de développement où toutes les nouvelles fonctionnalités et correctifs sont fusionnés avant de passer à la phase de test en pré-production.

10.2. Branches Fonctionnelles

- **Branches de Fonctionnalité (feature branches)** : Créées à partir de la branche "develop" pour le développement de nouvelles fonctionnalités. Nommées selon la convention `feature/nom_fonctionnalité`.
- **Branches de Correction de Bugs (bugfix branches)** : Créées à partir de la branche "develop" pour la correction de bugs identifiés. Nommées selon la convention `bugfix/nom_bug`.

10.3. Branches de Pré-Production et Production

- **Branch "pre-prod"** : Utilisée pour la phase de test en pré-production. Avant chaque déploiement en production, les nouvelles fonctionnalités et correctifs sont fusionnés dans cette branche pour des tests finaux.
- **Branch "release"** : Pour les versions de préparation à la production, cette branche permet d'effectuer les derniers ajustements avant de fusionner avec "main".

10.4. Normes de Commits

- **Messages de Commit** : Les messages doivent être clairs et concis, décrivant précisément le changement effectué. La convention de message de commit est :
  - **Ajout de fonctionnalité** : `feat: description de la fonctionnalité`
  - **Correction de bug** : `fix: description du bug corrigé`
  - **Mise à jour de documentation** : `docs: description de la mise à jour`
  - **Amélioration de performance** : `perf: description de l'amélioration`
  - **Refactorisation** : `refactor: description du refactoring`
- **Commits Atomiques** : Chaque commit doit représenter un changement unique et logique, facilitant le suivi des modifications et la gestion des versions.

11. Gestion des Mots de Passe

11.1. Complexité des Mots de Passe

- **Exigences de Complexité** : Les mots de passe des utilisateurs doivent comporter au moins 8 caractères, incluant des majuscules, des minuscules, des chiffres et des caractères spéciaux.

- **Expiration des Mots de Passe** : Les mots de passe doivent être changés tous les 90 jours pour renforcer la sécurité.

11.2. Gestionnaire de Mots de Passe

- **Intégration d'un Gestionnaire de Mots de Passe** : L'application propose une intégration avec des gestionnaires de mots de passe pour faciliter la gestion et la sécurité des mots de passe.
- **Stockage Sécurisé** : Les mots de passe sont stockés de manière sécurisée en utilisant des techniques de hachage et de salage robustes.

Conclusion

Le plan de qualité pour le projet "Buildinguerie" est conçu pour garantir que l'application est développée et maintenue selon les plus hauts standards de qualité. En mettant l'accent sur des processus rigoureux, des tests approfondis, une gestion structurée des branches GitHub, des normes de commit claires et une gestion sécurisée des mots de passe, nous visons à offrir une plateforme fiable, sécurisée et performante qui répond aux besoins de nos utilisateurs.