Plan assurance qualité

1. Objectifs du plan

Garantir la qualité du produit final (performance, accessibilité, sécurité, conformité aux besoins).

Définir les pratiques, outils et responsabilités pour chaque étape du cycle de développement.

Réduire les risques de régressions et d'anomalies en production.

2. Périmètre

Ce plan est conçu pour répondre de manière générale à un projet construit et réalisé par noesya. Il peut être précisé par un document complémentaire pour des projets spécifiques.

2.1. Stack

Le stack technique type utilisé est :

Développement back Ruby on Rails

Base de données PostgreSQL

Développement front HTML

Développement Javascript ES6 natif

Feuilles de styles CSS avec préprocesseur SASS

2.2. Environnement

Les projets s'appuient, sauf exigence contraire, sur un ensemble de 4 environnements :

- environnement de développement local

- environnement de test

- environnement de préproduction (iso prod)

- environnement de production

Les environnements de test et de préproduction sont parfois absents, en fonction des contraintes budgétaires ou des réalités fonctionnelles.

3. Référentiels et standards

noesya connait et applique les bonnes pratiques de sécurité (OWASP Top 10).

noesya connait et applique les bonnes pratiques UX / accessibilité (WCAG 2.1, RGAA si applicable).

noesya applique également des standards internes (conventions de nommage, workflows git, ...).

4. Organisation et responsabilités

Chaque projet dispose en interne chez noesya :

- d'un·e chef·fe de projet qui valide les besoins, priorise.

- d'un·e ou plusieurs développeurs·ses qui implémentent les fonctionnalités en respectant les standards.

- d'un·e référent·e qualité qui définit les tests et contrôle la conformité.

Le Client participe également aux phases de recette.

5. Processus qualité

5.1. Gestion du code

Le code est déposé sur une forge logicielle incluant des mécaniques de contrôle de version, généralement GitHub, avec des workflows définis (système de branches, fusion par pull requests, ...).

Une revue de code est obligatoire avant de fusionner des évolutions de code.

5.2. Gestion des environnements

Le travail est effectué sur un environnement de développement, puis envoyé sur un environnement de préproduction pour tests et recettage, et enfin déployé en production.

Le déploiement est automatisé.

L'environnement de production est sauvegardé tous les jours.

5.3. Vérification et validation

Les développements sont systématiquement soumis à différents tests avant d'être livrés, selon la complexité du projet, en fonction de la pertinence des tests automatisés et des budgets alloués :

- tests unitaires (couverture minimale définie).
- tests d'intégration (API, flux critiques).
- tests fonctionnels (par scénarios utilisateurs).
- tests de performance (charge, temps de réponse).
- tests d'accessibilité (audit manuel + outils).
- tests de sécurité (audit code + scan vulnérabilités).

5.4. Recette

Au besoin et pour les projets plus complexes (selon budget), un plan de recette peut être rédigé (critères d'acceptation, cas de test).

Dans tous les cas le projet se conclut par une phase de recette utilisateur qui doit être validée par le client avant mise en production.

6. Outils qualité

6.1. Linters

ESLint, Rubocop.

6.2. Tests

RSpec, Minitest, CodeCoverage.

6.3. CI/CD

GitHub Actions, CircleCI.

6.4. Monitoring

Qlty (anciennement CodeClimate), logs centralisés.

6.5. Outils accessibilité

Lighthouse.

6.6. Interception d'exception

Les éventuels bugs des projets sont via un outil (SmartBear BugSnag) qui prévient à la fois sur un canal Slack dédié et en ouvrant des GitHub Issues.

7. Révisions

1.0 Date of change: 25/08/2025 - Responsible: Technical team - Summary of Change: Initial release