## ****Phase 1 — Structuration du processus de développement****

### **1.1. Processus d’assurance qualité logicielle**

Afin de garantir la qualité et la pérennité du développement de la nouvelle plateforme **petitemaisondelepouvante.com**, un processus qualité s’appuie sur les principes de la **norme ISO 25010**.  
Cette norme définit huit caractéristiques de qualité logicielle ; dans le cadre de ce projet, quatre d’entre elles ont été retenues comme prioritaires :

1. **Fiabilité** : assurer la stabilité de la plateforme, même en cas de charge importante (pics lors du festival ou de ventes exclusives).
2. **Performance / efficacité** : garantir des temps de réponse rapides pour une expérience fluide.
3. **Sécurité** : protéger les données des utilisateurs et les transactions commerciales.
4. **Maintenabilité** : faciliter les évolutions futures (v2, v3) et éviter la dette technique.

L’ensemble du cycle de développement sera aligné sur une démarche **DevSecOps**, intégrant la qualité et la sécurité à chaque étape du pipeline CI/CD.

**Indicateurs et métriques de qualité**

| **Exigence ISO 25010** | **Indicateur / métrique** | **Objectif** | **Outil / suivi** | **Lien avec la dette technique** |
| --- | --- | --- | --- | --- |
| **Performance / efficacité** | Temps moyen de réponse (p95) | Maintenir un temps de réponse < 300 ms pour les appels API principaux | JMeter / Grafana / Prometheus | Permet de détecter les régressions de performance avant qu’elles ne dégradent l’expérience utilisateur. |
| **Sécurité** | % de dépendances sans vulnérabilités connues | Maintenir > 95 % de dépendances saines | OWASP Dependency Check / Snyk | Réduit les risques d’exploitation de failles et évite l’accumulation de dettes de sécurité. |
| **Maintenabilité** | Taux de couverture de tests automatisés | Objectif ≥ 80 % de couverture | Jest / Pytest / SonarQube | Garantit la robustesse du code et limite les régressions lors des futures versions. |
| **Fiabilité** | Taux d’erreurs applicatives (logs niveau “error”) | Maintenir < 2 % d’erreurs sur 1000 requêtes | Stack observabilité : Loki / Grafana | Permet de corriger rapidement les anomalies et d’éviter l’érosion de la qualité du code. |

Ces métriques seront intégrées dans le pipeline CI/CD et feront l’objet d’un **suivi hebdomadaire** à l’aide de **tableaux de bord** automatisés (Grafana, SonarQube, etc.).  
Elles permettent d’anticiper les dégradations et d’éviter la **dette technique** en imposant des contrôles continus.

**Détail étape par étape : pipeline CI/CD DevSecOps**

**Étape 1 : Planification**

* Définir les besoins, les user stories et les exigences de sécurité.
* Exemple : définir des politiques de code (branch policies, revues obligatoires).

🛠️ *Outils* : Jira, Azure Boards, GitHub Projects.

**Étape 2 : Code (Développement)**

* Les développeurs écrivent le code et les tests unitaires.
* Respect des standards de sécurité et de qualité.
* Gestion du code dans un **repository Git**.

🛠️ *Outils* : VS Code, IntelliJ, Git, GitHub/GitLab, pre-commit hooks.

**Étape 3 : Build (Intégration continue)**

* Compilation du code.
* Téléchargement des dépendances.
* Création d’artefacts (jar, war, container Docker…).

🛠️ *Outils* : Maven, Gradle, npm, Docker, Jenkins, GitLab CI.

**Étape 4 : Tests automatisés**

* **Tests unitaires** → valident les fonctions isolées.
* **Tests d’intégration** → valident la communication entre modules.
* **Tests de sécurité automatisés** (linting, dépendances).

🛠️ *Outils* : JUnit, pytest, Jest, OWASP Dependency Check, Snyk.

**Étape 5 : Analyse de la qualité et sécurité (SonarQube & Co)**

* **Analyse statique du code** → SonarQube.
* **Analyse des dépendances** → OWASP Dependency Check, Snyk.
* **Scan de conteneurs** → Trivy, Anchore.

🧠 Objectif : détecter bugs, vulnérabilités et dette technique **avant le déploiement**.

🛠️ *Outils* : SonarQube, Snyk, Trivy, Checkmarx, Bandit.

**Étape 6 : Packaging & Déploiement (CD)**

* Création d’images Docker ou paquets.
* Déploiement automatique sur un environnement de test, staging, puis production.
* Validation par tests de non-régression et smoke tests.

🛠️ *Outils* : Docker, Kubernetes, Jenkins, GitLab CD, ArgoCD, Helm.

**Étape 7 : Tests de sécurité dynamiques (DAST)**

* Analyse du comportement d’une application en exécution.
* Simulation d’attaques (ex. : injections SQL, XSS).

🛠️ *Outils* : OWASP ZAP, Burp Suite, Nikto.

**Étape 8 : Monitoring & Feedback**

* Surveillance de la performance, des erreurs, et des incidents de sécurité.
* Alertes automatiques, tableaux de bord et feedback aux développeurs.

🛠️ *Outils* : Prometheus, Grafana, ELK (Elasticsearch, Logstash, Kibana), Splunk.

On aura 2 pipelines:

* 1 pipeline qui execute tous les etapes quand y a un nouveau push/merge request
* 1 pipeline de scan qui execute seulement l’etape 5 , déclenchée par un cron job :

Même si ton code source est identique :

* Tes **bibliothèques** et **packages** (ex. npm, pip, Maven) peuvent être **mis à jour**.
* Ces nouvelles versions peuvent introduire :
  + des **failles de sécurité**,
  + des **changements de compatibilité**,
  + ou des **comportements différents**.

### **Compétences et formation**

| **Rôle** | **Responsabilités principales** | **Compétences clés** |
| --- | --- | --- |
| **Lead Developer (toi)** | Supervision technique, CI/CD, sécurité | DevSecOps, orchestration cloud, tests automatisés |
| **Développeur junior 1** | Backend (API, bases de données) | Node.js / Express, SQL / NoSQL |
| **Développeur junior 2** | Frontend (interface, accessibilité) | React.js, intégration RGAA |
| **Architecte logiciel** | Schéma d’architecture, intégration sécurité | Design système, Kubernetes, observabilité |

#### **Plan de montée en compétence**

* **Formation ciblée “Sécurité et DevSecOps pour développeurs”** (2 jours) : intégration de la sécurité dans le pipeline.
* **Atelier pratique interne “Docker & CI/CD avec GitHub Actions”** : automatisation et mise en place du pipeline.
* **Session de partage hebdomadaire** : suivi des métriques, bonnes pratiques de code et retours d’expérience.