**Spécification Fonctionnelle Détaillée (version draft)**

Table des matières

[Présentation Générale 2](#_Toc204215539)

[Objectif Global 2](#_Toc204215540)

[Architecture de l’application et Environnement 2](#_Toc204215541)

[Spécifications Fonctionnelles 2](#_Toc204215542)

[Inventaire des projets Git 2](#_Toc204215543)

[Vérification Git Flow 3](#_Toc204215544)

[Analyse des Merge Requests 3](#_Toc204215545)

[Vérification des standards de développement 3](#_Toc204215546)

[Détection de l’obsolescence 3](#_Toc204215547)

[Architecture Technique 4](#_Toc204215548)

[Configuration & Déploiement 5](#_Toc204215549)

[Règles Métier 5](#_Toc204215550)

# Présentation Générale

**Nom de l'application :** Dashboard-Radar

**Type :** Application batch d’analyse de projets sur Github   
**Équipe technique :** DEV  
**Statut :** rédaction de spécifications et étude faisabilité

## Objectif Global

L’application Dashboard-Radar vise à automatiser l’audit quotidien des projets Github d’une organisation. Elle collecte, analyse et classe les projets en fonction de leur conformité aux standards de leur état technique (maintenabilité, dette technique), et détecte les risques liés à l’obsolescence technologique.

Elle constitue un **radar quotidien de surveillance des projets Github**.

## Architecture de l’application et Environnement

**Composant principal :** Application Spring Boot exécutée en batch

**Déclenchement** : CronJob OpenShift (fréquence : 1/jour : cron nocturne)

**Image Docker** : Image Spring Boot packagée (JDK 17, Maven)

**Technologies principales :** Spring Boot 3.x, OpenShift (cron job), Github API , PostgreSQL Java 17

**Entrées :** Github API (projets, branches, MR, fichiers, métadonnées, etc.)

**Sorties :** Données structurées enregistrées dans une base de données relationnelle

# Spécifications Fonctionnelles

## Inventaire des projets Git

Pour chaque projet :

* Nom du projet
* ID Github
* Nom du groupe / sous-groupe parent
* Nombres des branches
* Langages de programmation utilisés.
* Frameworks identifiés (via analyse de pom.xml, build.gradle, package.json, etc.) **les règles de identifications doivent être définis par l’équipe lignes des dev** pour les ligne de dev standard supportées par la banque et des règles générales

## Vérification Git Flow

Pour chaque projet :

* Vérification de la présence des branches develop, main/master, feature/\*, hotfix/\* conformément aux normes adoptées par l’équipe exa.
* Détection les nombre des branches en écart du modèle
* Nombre total de lignes de code toutes branche confondus
* la vérification de l’existence de binaire dans git.

## Analyse des Merge Requests

Pour chaque projet :

* Nombre total de MRs ouvertes
* Nombre de MRs ouvertes depuis plus de 7 jours sans activité (updated\_at)
* MRs non assignées ou sans reviewers

## Vérification des standards de développement

Pour chaque projet :

* Présence de fichier Jenkinsfile et conformité avec pipeline standard
* Présence et structure du Dockerfile s’il existe
* Structure des répertoires du projet (par exemple pour la ligne java microservices /src/main/java, /resources, src/test/java, etc.) . **l’agent codex** doit définir la structure préconisée pour chaque ligne de développement.(java, angular)
* Langages/frameworks utilisés (ex : Java 17 + Spring Boot 3.x autorisé, Angular < 10 interdit, etc**.) l’agent codex va définir critère d’obsolescence à la base de bon pratique de la commaunité et les date de l’absolesence et de fin de support comminique par les fournisseur**

## Détection de l’obsolescence

Pour chaque projet :

* Vérification des versions de framework/langage dans les fichiers de build
* Comparaison avec la matrice interne d’**obsolescence** (fournie sous forme de config YAML/JSON locale) **à définir par codex.**
* Identification des composants en fin de vie (ex : Spring Boot < 2.7, Java 8, etc.) **à à définir par codex.**
* Génération d’un rapport projet obsolète avec niveau de sévérité (à **à définir par codex**) :
* mineur : version en fin de support
* majeur : version interdite ou critique

# Architecture Technique

* Dashboard-Radar : application Spring Boot avec les modules suivants (**à**) :
  + GithubScannerService (appel API Github)
  + MetadataAnalyzerService (analyse des fichiers et structures projet)
  + ComplianceCheckerService (vérification standards internes)
  + ObsolescenceDetectorService (analyse versions / dates EOL)
  + PersistenceService (stockage des **revoir par codex** résultats en base)
* L’application Auditor pour l'affichage
* Structure de Base de Données postgreSql : **revoir par codexù**

| **Table** | **Champs principaux** |
| --- | --- |
| project | id, name, group, subgroup, is\_archived, date\_derniere\_maj |
| branch | id, name, last\_commit\_date, is\_main, is\_feature |
| merge\_request | id, title, created\_at, updated\_at, author, reviewers |
| file\_check | id, project\_id, jenkins\_ok, dockerfile\_ok, has\_ci\_file |
| tech\_stack | project\_id, language, framework, version |
| obsolescence | project\_id, component, current\_version, status |

# Configuration & Déploiement

* **JDK :** 17+
* **Spring Boot :** 3.x (avec Spring Web, Spring Data JPA, Spring Batch)
* **Base de données :** PostgreSQL 14+
* **Authentification Github :** Token personnel ou OAuth2 (via Authorization: Bearer <token>) ayant le droit en lecture sur les toutes les liste des api github.
* **Déploiement :** Container Docker sur OpenShift, déclenché par CronJob YAML
* **Mécanisme de retry :** En cas de timeouts sur Github

# Règles Métier

À extraire d’un référentiel JSON/YAML configurable par **par codex**