**Dossier de Spécifications Techniques - Application de Veille des Marchés Publics**

**Objectif du projet**

Développer une application web permettant aux entreprises de:

* S'inscrire et définir leurs domaines d'intervention (fournitures, travaux, prestations intellectuelles)
* Recevoir automatiquement les publications pertinentes des marchés publics (Quotidien n°XXXX)
* Être alertées en cas de publication d'appels d'offres ou de résultats les concernant

**Fonctionnalités principales**

**Gestion des entreprises**

* Inscription/Connexion sécurisée avec validation par email
* Interface de définition des domaines d'intervention et secteurs d'activité
* Dashboard pour consulter les appels d'offres et résultats pertinents

**Extraction automatique**

* Scraping quotidien du site [www.dgcmef.gov.bf](http://www.dgcmef.gov.bf)
* Téléchargement automatique des fichiers PDF (Quotidien)
* Historisation et indexation des publications

**Analyse intelligente**

* Extraction par OCR/PyMuPDF du contenu structuré des PDF
* Identification des sections (résultats provisoires, appels d'offres)
* Reconnaissance des lots, montants, dates, ministères et entreprises attributaires

**Notification et alertes**

* Alertes par email pour appels d'offres correspondant aux domaines d'activité
* Notifications spéciales pour les entreprises mentionnées dans les résultats
* Rappels pour dates limites de dépôt approchantes

**Interface administrateur**

* Gestion des utilisateurs et entreprises
* Validation manuelle des extractions ambiguës
* Tableaux de bord statistiques par ministère, région, type de marché

**Architecture technique proposée**

**Frontend**

* Framework: React.js avec Redux
* UI: Material UI (thème personnalisable)ou tailwind css
* Responsive: Adapté à tous les appareils

**Backend**

* Framework: Django REST Framework
* Scraper: Scrapy avec Splash pour le JavaScript
* Tâches asynchrones: Celery avec Redis

**Traitement des documents**

* Extraction de texte: PyMuPDF/pdfplumber
* OCR si nécessaire: Tesseract
* NLP: spaCy avec modèle français personnalisé
* Expressions régulières spécifiques aux formats du Quotidien

**Base de données**

* PostgreSQL pour les données structurées (relations, utilisateurs)
* MongoDB pour le stockage des documents et extractions textuelles
* Elasticsearch pour la recherche avancée

**Infrastructure**

* Conteneurisation: Docker-compose
* CI/CD: GitLab CI
* Hébergement: VPS dédié ou AWS

**Modèle de données hybride**

**PostgreSQL (données structurées)**

Une image contenant texte, diagramme, Plan, Police

Le contenu généré par l’IA peut être incorrect.

**MongoDB (données documentaires)**

* PDF originaux
* Texte extrait
* Résultats d'analyse
* Logs d'extraction

•documentsRaw (publicationId, nomFichier, contenuFichier, tailleFichier, typeMime, dateTelechargement, sourceUrl, referenceQuotidien, hashMd5)

• contenusExtraits (publicationId, documentRawId, texteComplet, dateExtraction, statutTraitement, qualiteExtraction, sections, entites, domaines, motsCles, classification)

• lots (contenuExtraitId, marcheId, numeroLot, description, montantEstime, montantAttribue, entrepriseAttributaire, page, contexte)

• logsExtraction (publicationId, horodatage, niveau, message, composant, details, traceErreur)

• fileAlertes (entrepriseId, publicationId, contenuExtraitId, typeAlerte, priorite, statut, dateCreation, dateTraitement, contenu)

• indexRecherche (publicationId, contenuExtraitId, texteIndexe, datePublication, motsCles, domaines, ministeres, regions, typeMarche, typeProcedure, derniereMiseAJour)

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

**Intégration entre les Bases de Données SQL et NoSQL**

L’intégration entre PostgreSQL (base relationnelle SQL) et MongoDB (base orientée documents NoSQL) répond aux besoins hybrides du système, en conciliant données structurées (utilisateurs, entreprises, marchés) et données non structurées (PDFs, texte extrait). Cette double approche permet d’exploiter les forces complémentaires des deux technologies pour un système de veille performant, évolutif et cohérent. Voici les points clés de cette intégration :

**1. Liaison entre les données**

* Le champ publicationId dans les collections MongoDB référence la clé primaire de la table Publication dans PostgreSQL.
* Cette liaison crée un pont entre les données textuelles issues des documents et les entités relationnelles structurées.

**2. Processus de flux de données**

* **Étape 1** : Les documents PDF sont récupérés et stockés dans la collection documentsRaw de MongoDB.
* **Étape 2** : Un processus d’extraction analyse les fichiers et enregistre le texte dans contenusExtraits.
* **Étape 3** : Les informations structurées (marchés, lots, résultats) sont extraites de ce contenu et insérées dans les tables PostgreSQL.
* **Étape 4** : Des références croisées entre les deux bases assurent l’intégrité des données et permettent des correspondances fiables.

**3. Génération d’alertes**

* MongoDB analyse le contenu textuel et stocke les alertes temporaires dans la collection fileAlertes.
* Une fois validées, ces alertes sont insérées dans la table structurée Alerte de PostgreSQL.
* Cela permet un traitement textuel souple dans MongoDB avant une structuration rigoureuse dans le système relationnel.

**4. Fonction de recherche**

* MongoDB utilise la collection indexRecherche pour proposer une recherche plein texte rapide et efficace.
* Les résultats obtenus via cette recherche sont liés aux données relationnelles stockées dans PostgreSQL, permettant une navigation fluide entre contenu documentaire et entités métiers.

**5. Avantages de la double base**

* **PostgreSQL** assure la gestion des relations complexes, la cohérence transactionnelle (ACID) et la logique métier.
* **MongoDB** excelle dans le traitement massif de documents, l’analyse de texte et l’adaptabilité du schéma.
* Cette répartition optimise les performances globales : MongoDB prend en charge les traitements lourds de texte, tandis que PostgreSQL gère les opérations structurées et critiques.

**6. Considérations de mise en œuvre**

* La synchronisation entre les bases nécessite la mise en place de processus ETL (Extract, Transform, Load) ou d’une architecture événementielle (ex. : message broker, triggers applicatifs).
* Les transactions inter-bases n’étant pas supportées nativement, il faut gérer la cohérence au niveau applicatif.
* Les requêtes doivent être orientées selon le type de besoin (texte ou relation), et les jointures entre les données des deux bases doivent être effectuées dans la couche backend.

**7. Exemple pratique d’un traitement complet**

* Lorsqu’un nouvel avis de marché est détecté, le scraper télécharge le PDF et le stocke dans documentsRaw (MongoDB).
* Le texte est extrait et sauvegardé dans contenusExtraits.
* Les données pertinentes (entité publique, montant, lots) sont extraites et insérées dans les tables PostgreSQL.
* Si des mentions d’entreprises sont détectées, une alerte est générée dans fileAlertes, puis validée et transférée dans la table Alerte.
* L’index de recherche MongoDB est mis à jour pour permettre des recherches futures efficaces.

Cette architecture assure ainsi un équilibre entre performance, flexibilité et intégrité, parfaitement alignée avec les besoins de traitement documentaire, d’analyse textuelle et de gestion relationnelle du système de veille des marchés publics.

Une image contenant texte, capture d’écran, Police, Parallèle

Le contenu généré par l’IA peut être incorrect.

**Exigences non fonctionnelles**

* Haute disponibilité: 99.9% de temps de fonctionnement
* Performance: extraction d'un nouveau PDF en moins de 5 minutes
* Sécurité: chiffrement des données, authentification forte
* Conformité RGPD: consentement, droit à l'oubli

**Tests et validation**

* Tests unitaires pour chaque extracteur (lots, montants, dates)
* Tests d'intégration du pipeline complet
* Validation manuelle sur échantillon représentatif de documents