ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

HONORIS UNITED UNIVERSITIES

Rapport Final

4ème année IIR 18

Ingénierie Informatique et Réseaux

PROJET CRIME ANALYTICS DASHBOARD

Réalisé par :

CHARRO AMINE

Table des matières

1. Introduction

a. Contexte du projet

Dans le cadre du module "Bases de Données NoSQL", nous avons développé une plateforme de renseignement criminel. Les enquêtes modernes nécessitent de croiser des données hétérogènes (fichiers de police, relevés téléphoniques, réseaux d'influence) qui ne peuvent pas être gérées efficacement par une seule technologie.

b. Problématique métier choisie

Comment centraliser et visualiser des informations dispersées pour identifier rapidement les chefs de réseaux criminels et détecter des activités suspectes en temps réel ? Les bases relationnelles classiques peinent à modéliser des réseaux complexes ou à gérer des documents non structurés.

c. Objectifs du système

1. Centraliser les fiches suspectes (Documents).
2. Analyser les relations et hiérarchies criminelles (Graphe).
3. Surveiller des lignes téléphoniques en temps réel (Clé-Valeur).
4. Assurer l'intégrité de l'état civil (Relationnel).

d. Justification de l'architecture polyglotte

Nous avons choisi la persistance polyglotte pour utiliser "le bon outil pour le bon usage" :

- MongoDB pour la flexibilité des dossiers d'enquête.
- Neo4j pour la puissance d'analyse des réseaux (algorithmes de graphe).
- Redis pour la rapidité du temps réel (alertes).
- MySQL pour la rigueur des données administratives.

2. Analyse des besoins

a. Description fonctionnelle

Le système est un Dashboard Web unifié permettant :

- La recherche textuelle de suspects.
- La visualisation interactive du réseau criminel.
- La simulation d'écoutes téléphoniques.
- La géolocalisation des opérations.

b. Acteurs et cas d’usage

L'Enquêteur :

- Cherche un suspect par nom ou mot-clé.
- Examine ses relations pour trouver son supérieur.

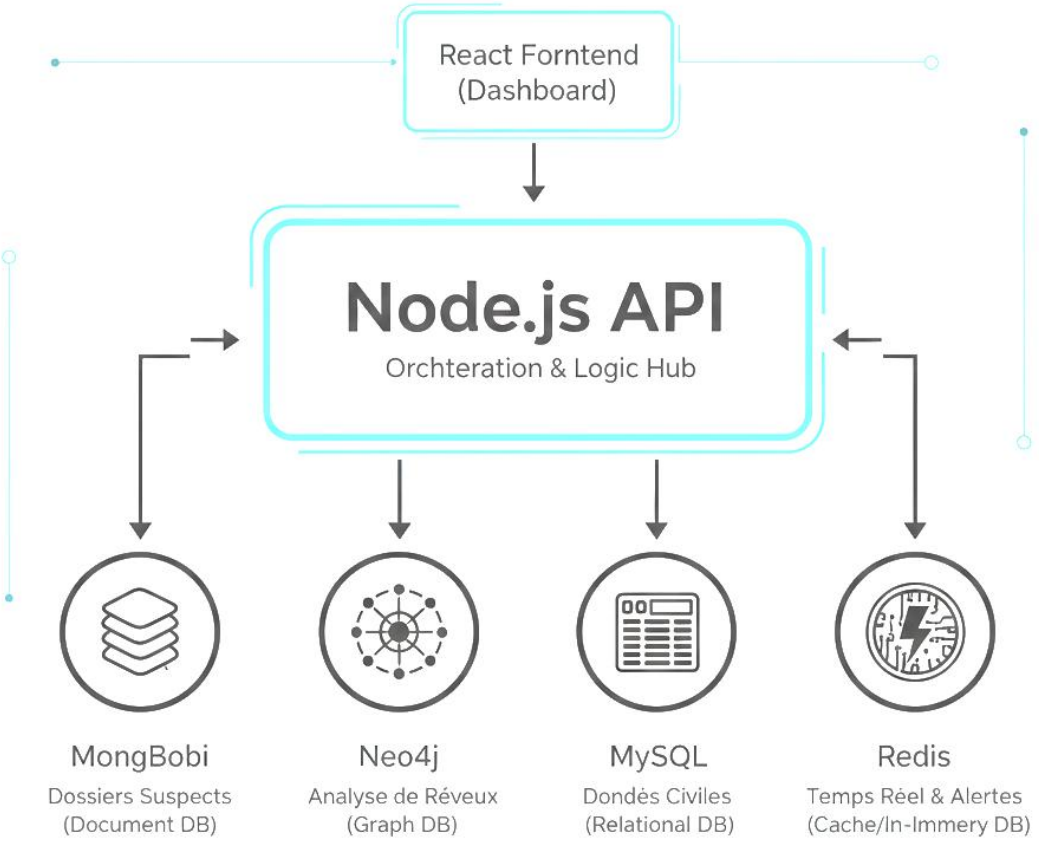
- Met un numéro sur écoute et reçoit des alertes.

c. Contraintes techniques

- **Performance :** Les calculs de graphe (PageRank) et le temps réel doivent être rapides.
- **Cohérence :** Un suspect "Amine" dans MongoDB doit correspondre au nœud "Amine" dans Neo4j.
- **Disponibilité :** L'architecture Docker assure une isolation et un redémarrage rapide.

3. Architecture globale du système

a. Schéma d'architecture



b. Description des composants

- Frontend (React) : Interface utilisateur responsive avec visualisations avancées (Leaflet pour les cartes, React-Force-Graph pour les réseaux).
- Backend (Node.js) : Orchestrateur qui interroge la bonne base selon la requête API.
- Bases de données :

  MySQL : Stocke l'état civil (`Personne`), les `Telephones` et les `Appels` (Logs).

  MongoDB : Stocke les fiches riches (`biographie`, `signalement`, `crimes`).

  Neo4j : Modélise les relations (`CONNAIT`, `DIRIGE`) pour l'analyse.

  Redis : Compte les appels entrants sur une courte fenêtre de temps (TTL).

c. Flux de données

1. Dossier : `GET /api/suspect/:nom` -> Interroge MongoDB.
2. Réseau : `GET /api/reseau/chemin` -> Interroge Neo4j (ShortestPath).
3. Alerte : `POST /api/alerte/appel` -> Écrit dans Redis (INCR + EXPIRE).

4. Modélisation des données SQL (MySQL)

Respect de la structure relationnelle normalisée (3NF) :

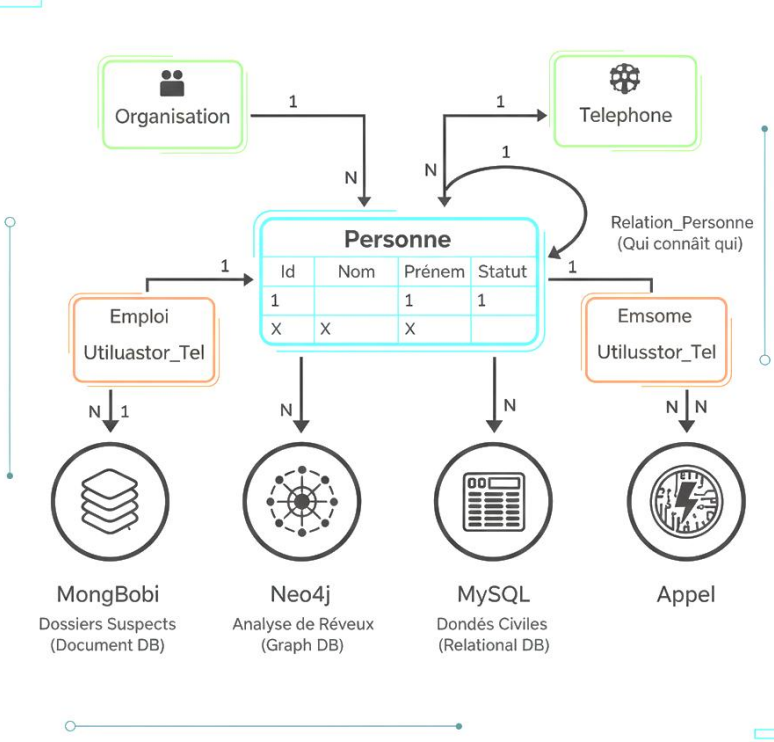
`Personne` (id, nom, prenom, statut)

`Organisation` (id, nom_org)

`Telephone` (id, numero, operateur)

`Relation_Personne` (Table de jointure réflexive : qui connaît qui ?)

`Appel` (Log historique : appelant, appelé, durée, date)



5. Modélisation des données NoSQL

a) Base orientée documents (MongoDB)

Collection : `suspects`

Schéma JSON : Utilisation de documents embarqués pour les crimes (indivisibles du suspect).

```json
{
  "nom": "CHARRO",
  "prenom": "Amine",
  "adresse": "Villa 14, Anfa Supérieur, Casablanca",
  "biographie": "Chef présumé du Cartel...",
  "signalement": "Cicatrice joue gauche",
  "crimes": ["Trafic international", "Blanchiment"] // Embedded
}
```

b) Base graphe (Neo4j)

Nœuds (Labels) : `:Personne`, `:Telephone`

Relations (Types) :

`(:Personne)-[:DIRIGE]->(:Personne)`

`(:Personne)-[:POSSEDE]->(:Telephone)`

`(:Telephone)-[:A_APPELE]->(:Telephone)`

Exemple Cypher :

```cypher
MATCH (chef:Personne)-[:DIRIGE]->(subalterne:Personne) RETURN chef, subalterne
```

c) Base clé–valeur (Redis)

Clé : `alerte:{numero_telephone}` (String)

Valeur : Compteur (Entier)

Cas d'usage : Détection de pics d'activité. Si compteur > 5 en 60s -> ALERTE.

6. Implémentation

a. Technologies

- Langage : JavaScript (Node.js/React).
- Conteneurisation : Docker & Docker Compose.
- Drivers : `mysql2/promise`, `mongodb`, `neo4j-driver`, `redis`.

b. Connexion & Gestion d'erreurs

Chaque base a sa propre connexion initialisée au démarrage du serveur (`connectDBs()`).

Nous avons implémenté le Caching pour limiter la charge sur Neo4j : le résultat de l'algorithme PageRank est mis en cache dans Redis.

7. Scénarios d'utilisation

Scénario : Investigation Complète

1. Lecture : L'enquêteur tape "CHARRO" -> MongoDB renvoie la fiche.
2. Analyse : Il clique sur "Voir Réseau" -> Neo4j affiche ses lieutenants (Hassan).
3. Action : Il note le numéro de Hassan et lance une écoute ("Simuler Appel").
4. Alerte : Au 6ème appel simulé, Redis déclenche l'alerte rouge sur le Dashboard.

8. Analyse critique

a. Avantages

- Optimisation : Chaque requête est ultra-rapide car elle utilise le moteur de BDD adapté (Graph traversal instantané vs jointures SQL lentes).
- Modularité : On peut changer la structure des dossiers (Mongo) sans casser les logs d'appels (MySQL).

b. Difficultés

Cohérence des données : Maintenir le même nom "Amine CHARRO" dans Mongo, MySQL et Neo4j est complexe. Nous avons dû créer des scripts d'initialisation synchronisés.

c. Comparaison Mono-base

Une architecture 100% SQL aurait rendu la détection de chemins criminels (A connait B qui connait C...) très lente et complexe à écrire (Jointures récursives).

9. Conclusion

a. Apprentissages

Ce projet a permis de maîtriser l'orchestration de conteneurs Docker et de comprendre concrètement les forces de chaque famille NoSQL.

b. Perspectives

Pour aller plus loin, nous pourrions ajouter un bus d'événements (Kafka) pour synchroniser les écritures en temps réel entre toutes les bases, garantissant une cohérence parfaite automatiquement.