Projet DON5 – Architecture & Bases de Données

Réseau social orienté rencontres physiques (IRL)

HE2B – Développement d'application

OBJECTIF

Un projet centré sur l'architecture

Système multi-SGBD

Concevoir une architecture intégrant plusieurs systèmes de gestion de bases de données pour répondre à des besoins variés

Choix technologiques

Mettre en évidence des décisions architecturales pertinentes et justifiées par les cas d'usage

Modélisation avancée

Priorité absolue à l'architecture et à la modélisation plutôt qu'à l'interface utilisateur

Interface volontairement minimale pour se concentrer sur la qualité de l'architecture

Problématique : la complexité des données

Défis techniques

Le projet fait face à des **données hétérogènes** nécessitant des approches différenciées pour chaque type de besoin.

Intégrité

Garantir la cohérence des données critiques et métier

Recherche

Recherche floue et performante sur plusieurs champs

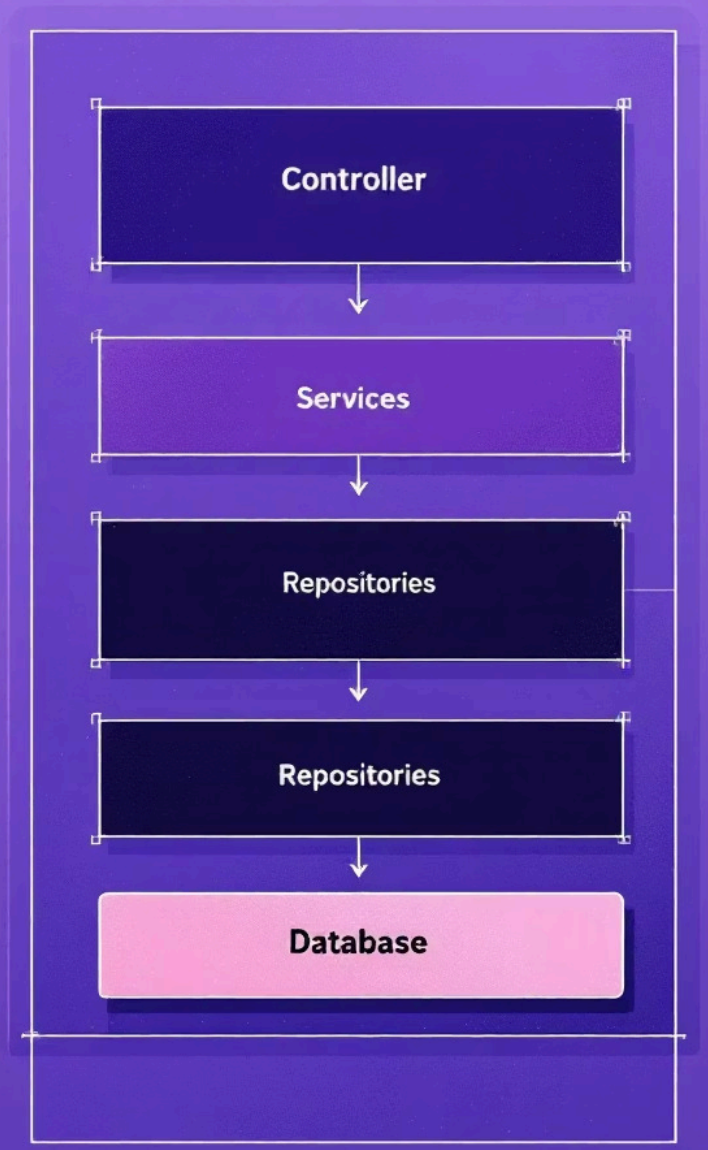
Relations sociales

Modéliser des connexions complexes entre utilisateurs

Performance

Optimiser les temps de réponse et la scalabilité

Une seule base de données ne peut répondre à tous ces besoins



Architecture générale du système

API REST

Spring Boot expose les endpoints pour les clients

Couche Services

Logique métier centralisée et orchestration des données

Repositories

Abstraction de l'accès aux différentes bases de données

Isolation des données

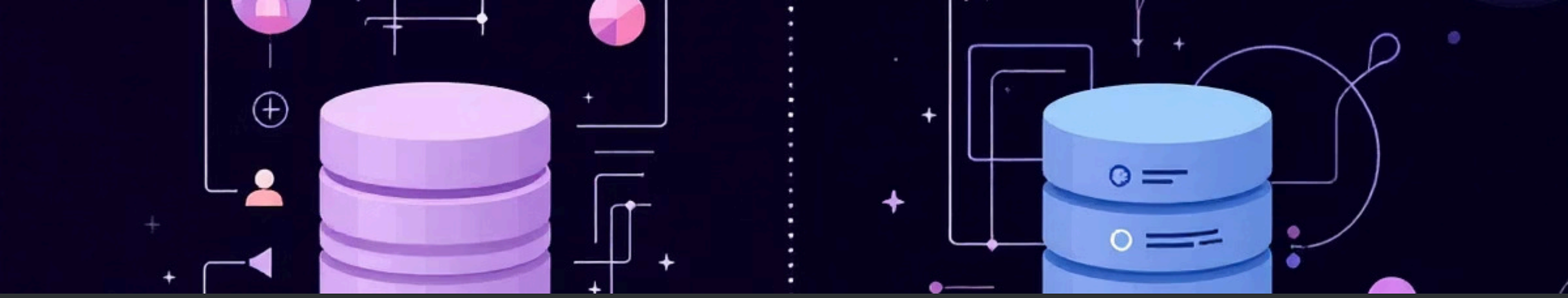
Aucune base n'est accédée directement par les services

Cette architecture en couches garantit la **séparation des responsabilités** et facilite la maintenance du système.

Répartition stratégique des bases de données

Chaque technologie a été sélectionnée pour ses forces spécifiques dans le traitement de types de données particuliers.

Technologie	Rôle dans l'architecture
MySQL	Données structurées critiques servant de source de vérité du système (utilisateurs, rencontres, points)
MongoDB	Profils utilisateurs et centres d'intérêt avec structure flexible
Elasticsearch	Index de recherche plein texte dérivé des données métiers (pas une source de vérité)
Neo4j	Graphe social pour modéliser les relations entre utilisateurs
Redis	Cache haute performance et gestion de la sécurité



MySQL & MongoDB : données structurées et flexibles

MySQL

Garantir la **cohérence structurelle et métier** des données critiques du système.

- Gestion des **utilisateurs** avec contraintes d'unicité
- Enregistrement des **rencontres** avec dates et lieux
- Système de points lié aux rencontres
- Source de vérité du système

Garantir la cohérence des données critiques et métier

MongoDB

Base NoSQL offrant une **flexibilité structurelle** pour les données évolutives :

- **Profils utilisateurs** avec attributs personnalisables
- **Centres d'intérêt** sous forme de documents imbriqués
- Structure adaptable sans migration de schéma

Permet d'ajouter facilement de nouveaux champs sans impact.

MySQL

```
mysql> DESCRIBE users;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| id       | bigint       | NO   | PRI | NULL    | auto_increment |
| username | varchar(100) | NO   | UNI | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> DESCRIBE meetings;
+------------+--------------+------+-----+-------------------+-------------------+
| Field      | Type         | Null | Key | Default           | Extra             |
+------------+--------------+------+-----+-------------------+-------------------+
| id         | bigint       | NO   | PRI | NULL              | auto_increment    |
| user_a     | bigint       | NO   | MUL | NULL              |                   |
| user_b     | bigint       | NO   | MUL | NULL              |                   |
| interest   | varchar(100) | YES  |     | NULL              |                   |
| created_at | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+------------+--------------+------+-----+-------------------+-------------------+
5 rows in set (0.00 sec)

mysql> DESCRIBE points;
+------------+----------+------+-----+-------------------+-------------------+
| Field      | Type     | Null | Key | Default           | Extra             |
+------------+----------+------+-----+-------------------+-------------------+
| id         | bigint   | NO   | PRI | NULL              | auto_increment    |
| user_id    | bigint   | NO   | MUL | NULL              |                   |
| meeting_id | bigint   | NO   | MUL | NULL              |                   |
| amount     | int      | NO   |     | NULL              |                   |
| created_at | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+------------+----------+------+-----+-------------------+-------------------+
```

MySQL est idéal pour les données structurées et critiques. Il garantit la cohérence des données avec des contraintes d'unicité pour les utilisateurs et des enregistrements précis des rencontres, servant de source de vérité pour le système.

Exemple Requete :

```
String sql = """
    INSERT INTO meetings(user_a, user_b, interest)
    VALUES (?, ?, ?)
""";
```

Table Meetings :

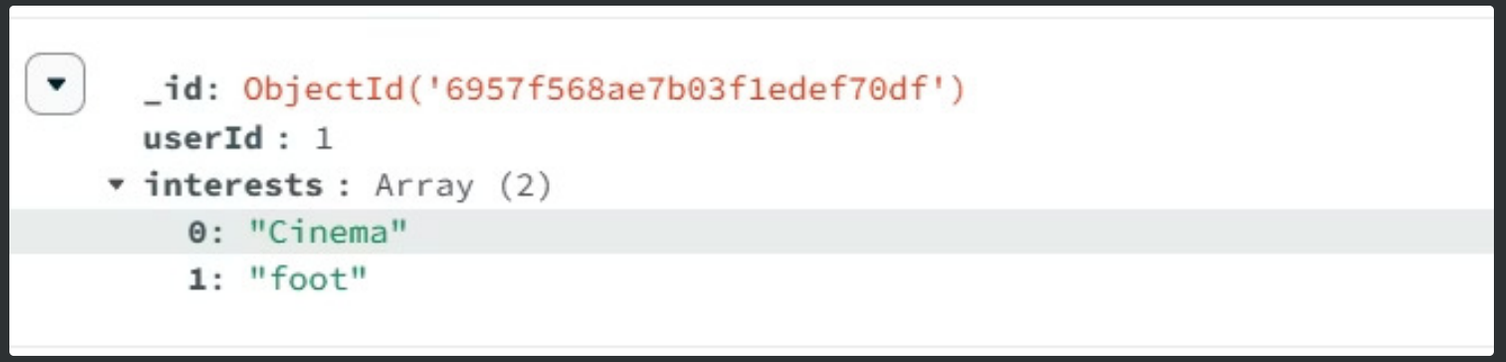
```
CREATE TABLE IF NOT EXISTS meetings (
  id BIGINT AUTO_INCREMENT PRIMARY KEY,
  user_a BIGINT NOT NULL,
  user_b BIGINT NOT NULL,
  interest VARCHAR(100) NOT NULL,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,

  CONSTRAINT fk_meeting_user_a
    FOREIGN KEY (user_a) REFERENCES users(id)
    ON DELETE CASCADE,

  CONSTRAINT fk_meeting_user_b
    FOREIGN KEY (user_b) REFERENCES users(id)
    ON DELETE CASCADE
);

CREATE INDEX idx_meetings_user_a ON meetings(user_a);
CREATE INDEX idx_meetings_user_b ON meetings(user_b);
CREATE INDEX idx_meetings_users ON meetings(user_a, user_b);
```

Table Points :

```
CREATE TABLE IF NOT EXISTS points (
  id BIGINT AUTO_INCREMENT PRIMARY KEY,
  user_id BIGINT NOT NULL,
  meeting_id BIGINT NOT NULL,
  amount INT NOT NULL,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,

  CONSTRAINT fk_points_user
    FOREIGN KEY (user_id) REFERENCES users(id)
    ON DELETE CASCADE,

  CONSTRAINT fk_points_meeting
    FOREIGN KEY (meeting_id) REFERENCES meetings(id)
    ON DELETE CASCADE
);

CREATE INDEX idx_points_user ON points(user_id);
CREATE INDEX idx_points_meeting ON points(meeting_id);
```

```
SELECT SUM(points) FROM points WHERE user_id = ?
```

MongoDB



MongoDB, une base NoSQL, offre une flexibilité structurelle pour les données évolutives. Il permet des profils utilisateurs personnalisables et des centres d'intérêt imbriqués, facilitant l'ajout de nouveaux champs sans migration de schéma.

Exemple Requete :

```
col.updateOne(
    Filters.eq("userId", userId),
    Updates.addToSet("interests",interest),
    new UpdateOptions().upsert(true)
);
```

Elasticsearch & Redis : recherche et performance

Elasticsearch

Moteur de recherche plein texte optimisé pour les requêtes complexes :

- **Recherche floue** tolérant les fautes de frappe
- Recherche **multi-champs** simultanée
- Indexation rapide et résultats pertinents

Redis

Améliorer les performances et protéger l'API via des mécanismes temporaires en mémoire :

- Cache des points utilisateurs pour accès rapide
- **Rate limiting** pour protéger l'API
- Stockage clé-valeur ultra-rapide

Ces deux technologies améliorent considérablement l'**expérience utilisateur** en réduisant les temps de réponse.

Elasticsearch

Moteur de recherche plein texte performant

- Index optimisé pour la recherche plein texte
- Tolérance aux fautes (fuzzy search)
- Recherche multi-champs (username, interests,)
- Scoring automatique des résultats


```
"hits": [
  {
    "_index": "users_search",
    "_id": "1",
    "_score": 1,
    "_source": {
      "userId": 1,
      "username": "abdel",
      "interests": [
        "cinema"
      ]
    }
  }
]
```


```
SearchResponse<UserSearchDocument> response =
        client.search(s -> s
.index(INDEX)
.query(q -> q                                .multiMatch(m -> m
.fields("username", "interests")
.query(query)
.fuzziness("AUTO")
                                    )),
                        UserSearchDocument.class
);
```

Redis

Cache haute performance et protection de l'API

- Cache des points utilisateurs
- Rate limiting pour protéger l'API
- Stockage clé-valeur ultra-rapide
- TTL pour expiration automatique

Exemple de requete :

```
Integer cached = cache.getCachedTotal(userId);
if (cached != null) {
    return cached;
}

int total = repo.getTotalPoints(userId);
cache.cacheTotal(userId, total);
return total;
```

```
public Integer getCachedTotal(long userId) {
        try (Jedis jedis = pool.getResource()) {
String value = jedis.get(key(userId));
return value != null Integer.parseInt(value) : null;
}
}
```

```
public void cacheTotal(long userId, int total) {
    try (Jedis jedis = pool.getResource()) {
        jedis.setex(key(userId), TTL_SECONDS,
String.valueOf(total));
    }
}
```

Ces deux technologies améliorent considérablement l'expérience utilisateur en réduisant les temps de réponse et en sécurisant l'API.

Neo4j : le pouvoir des graphes sociaux

Neo4j modélise naturellement les **relations complexes** entre utilisateurs, permettant des requêtes sociales sophistiquées.

Nœuds utilisateurs

Chaque utilisateur est représenté comme un nœud dans le graphe

Relations de rencontres

Les rencontres physiques créent des arêtes entre les nœuds

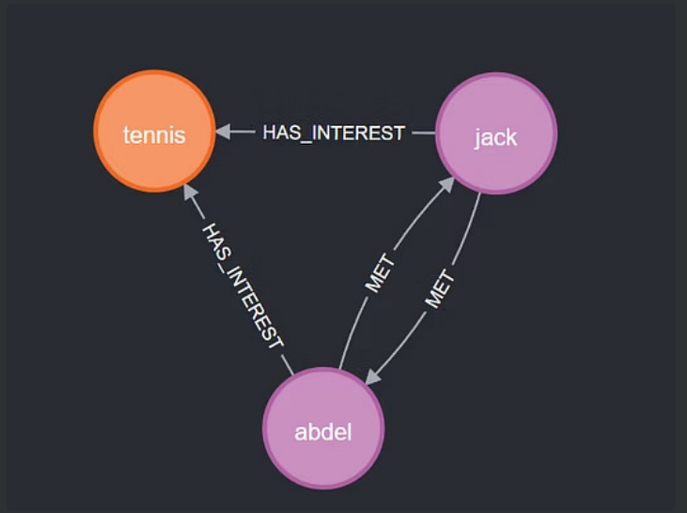
Recommandations intelligentes

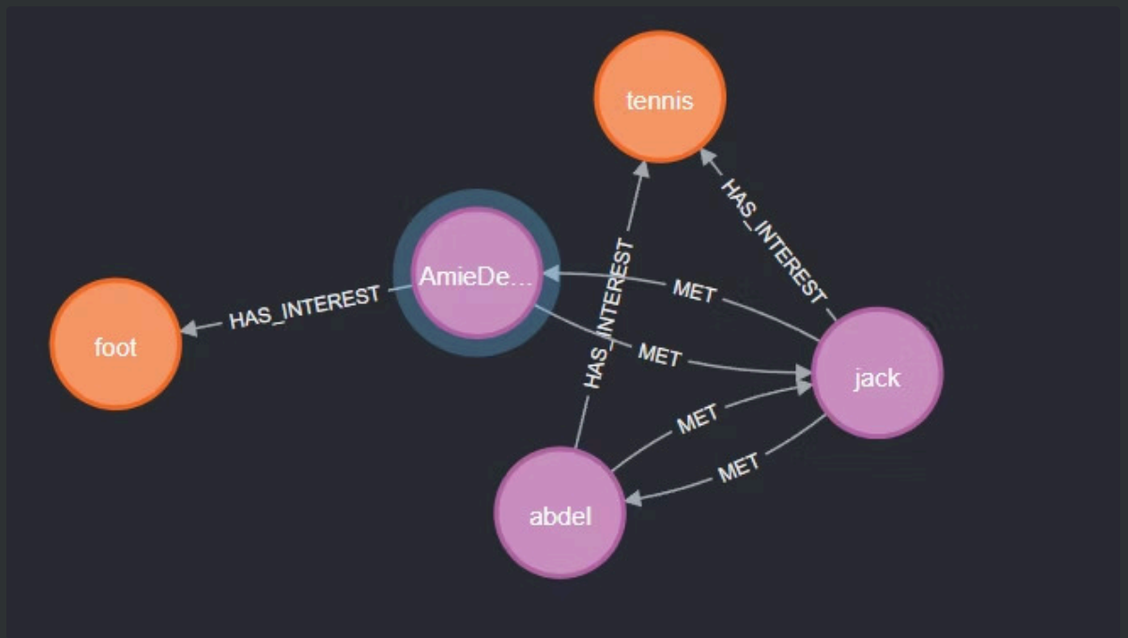
Suggestions basées sur les connexions et affinités communes

Centres d'intérêt

Intégrés au graphe pour enrichir les recommandations

Les algorithmes de parcours de graphe permettent de trouver des connexions pertinentes en quelques millisecondes





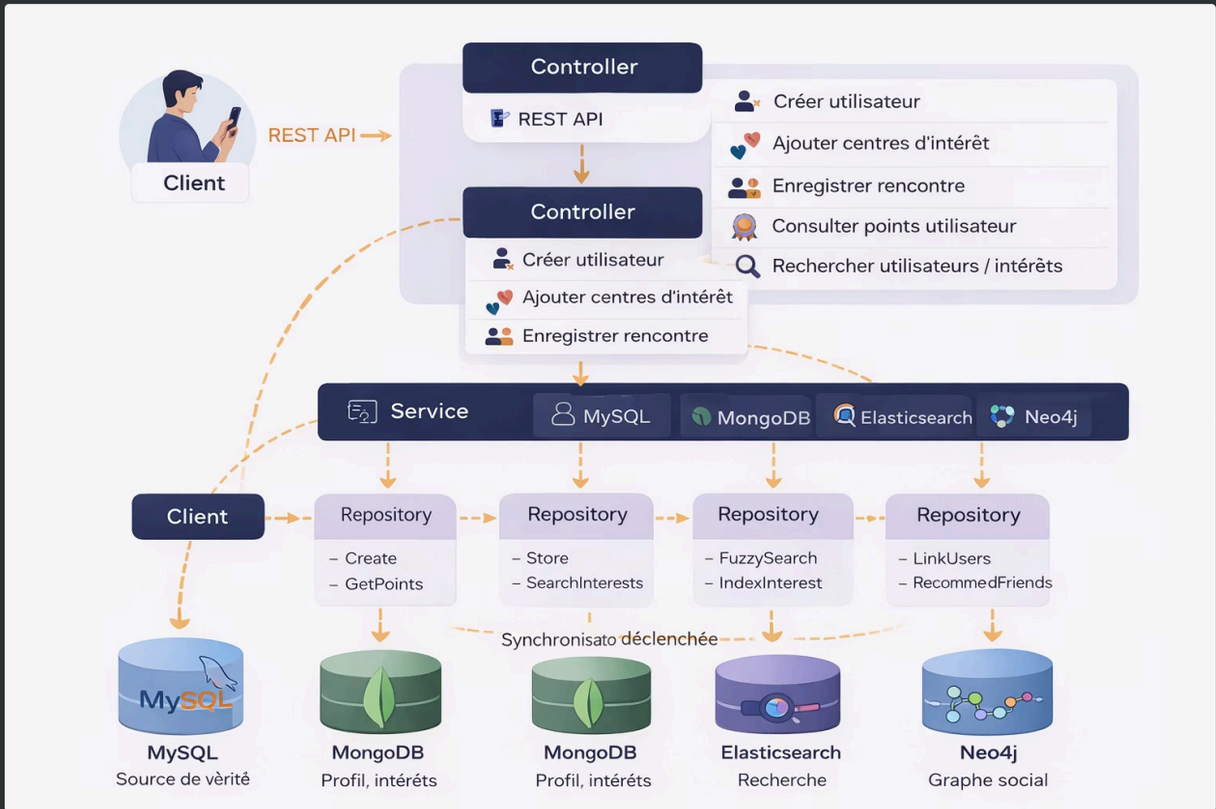
Creation du lien user :

```
MERGE (a:User {id: $a}) SET a.name = $nameA
MERGE (b:User {id: $b}) SET b.name = $nameB
CREATE (a)-[:MET {interest: $t, ts: datetime()}]->(b)
CREATE (b)-[:MET {interest: $t, ts: datetime()}]->(a)
```

Recommandation Ami de Ami :

```
MATCH (me:User {id: $id})-[:MET]-  (friend)-[:MET]-(fof:User)
WHERE NOT (me)-[:MET]-(fof) AND me   <>fof
RETURN DISTINCT fof.name AS suggestName
```

Flux entre services et systèmes de données.



Il n'y a pas de réplication automatique entre bases. La synchronisation est volontairement gérée par la couche service, afin de garder le contrôle et éviter une complexité inutile.

Synchronisation et cohérence des données

Synchronisation applicative

MySQL est la source de vérité

Les autres bases contiennent des données dérivées

Les mises à jour sont déclenchées lors des événements métier

Conclusion : une architecture réfléchie

Architecture cohérente

Chaque composant a un rôle clair et contribue à l'ensemble du système

Technologies complémentaires

Les forces de chaque base compensent les limites des autres

Système évolutif

L'architecture permet d'ajouter facilement de nouvelles fonctionnalités

Ce projet démontre qu'une architecture multi-bases bien conçue peut répondre efficacement à des besoins complexes et hétérogènes.

Scénario d'utilisation global

Parcours complet d'un utilisateur dans le système, de l'inscription aux recommandations sociales.

1

Création d'un utilisateur

MySQL (utilisateur) + MongoDB (profil) + Neo4j (nœud social)

2

Ajout de centres d'intérêt

MongoDB (stockage flexible) + Elasticsearch (indexation)

3

Recherche floue d'utilisateurs ou d'intérêts

Elasticsearch (fuzzy search multi-champs)

4

Simulation d'une rencontre physique

MySQL (meeting + points) + Neo4j (relation sociale)

5

Consultation des points

Redis (cache) → MySQL si nécessaire

6

Recommandations sociales

- par intérêts communs (Elasticsearch)
- par graphe social (Neo4j)