Mini-projet SupDeVinci Travel Hub
Intégration de bases NoSQL (Redis, MongoDB, Neo4j)

SupDeVinci Travel Hub (STH) est une plate-forme B2C qui agrège vols, hébergements et activités touristiques afin de construire des itinéraires personnalisés quasi en temps réel.
Pour absorber la charge et offrir une latence 200 ms, l'entreprise adopte une architecture *poly-glotte*:

Besoin métier	Modèle de données dominant	Base NoSQL utilisée
Cache, sessions, notifications	Clé–valeur, TTL, Pub/Sub	Redis
Catalogue d'offres	Documents JSON semi-structurés	MongoDB
Recommandations & graphe destinations	Relations orientées nœuds-arêtes	Neo4j

Micro-service à produire

Vous exposez *une unique API HTTP/JSON* (FastAPI, Express, Spring Boot ou équivalent). Toutes les routes doivent gérer les erreurs (HTTP 40x/50x) de façon lisible et consigner leur durée d'exécution.

1. Recherche d'offres `/offers`

- **Méthode** : `GET /offers?from=PAR&to=TYO&limit=10`
- **Flux interne** :
  1. Clé cache Redis `offers:PAR:TYO` (TTL 60 s). Si *hit* → retour immédiat.
  2. Sinon, requête MongoDB sur la collection `offers` *(champ from, champ to, tri par prix ascendant)*.
  3. Stockage JSON compressé dans Redis `SET EX 60`.
- **Réponse** : tableau d'offres (`limit`), chaque offre contenant : `id, provider, price, currency, legs[],`

2. Recommandations `/reco`

- **Méthode** : `GET /reco?city=PAR&k=3`
- **Cypher demandé (exemple)** :

```
MATCH (c:City {code:$city})-[:NEAR]->(n:City)
RETURN n.code AS city ORDER BY n.weight DESC LIMIT $k
```

- **Réponse** : tableau des `k` codes-ville recommandés avec un score.

3. Authentification `/login`

- **Méthode** : `POST /login { "userId": "u42" }`
- **Flux interne** : génération UUID v4 → `SET session:<uuid> u42 EX 900`.
- **Réponse** : `{ "token": "<uuid>", "expires_in": 900 }`.

4. Détails d'une offre `/offers/{id}`

- **Méthode** : `GET /offers/{id}`
- Lecture directe MongoDB (*find by _id*) + cache Redis (`offers:id` TTL 300 s).
- La réponse inclut les champs complets et un tableau `relatedOffers` (3 ID obtenus via Neo4j : villes proches + mêmes dates).

5. Notification temps réel (canal Redis Pub/Sub)

- Lorsque l'API insère une nouvelle offre dans MongoDB (opération hors MVP mais simple), elle publie sur le canal `offers:new` un message JSON :

  `{ "offerId": "abc123", "from": "PAR", "to": "TYO" }`.

- Un client de test (`redis-cli SUBSCRIBE offers:new`) doit voir passer ce message.

6. Structure minimale des données

MongoDB – collection `offers`

```
{
  _id: ObjectId,
  from: "PAR",
  to:   "TYO",
  departDate: ISODate,
  returnDate: ISODate,
  provider: "AirZen",
  price: 750.00,
  currency: "EUR",
  legs: [{ flightNum, dep, arr, duration }],
  hotel: { name, nights, price } | null,
  activity: { title, price } | null
}
```

Index attendus : `{ from:1, to:1, price:1 }` (+ texte sur `provider`).

Neo4j – graphe

- Nœuds : `(c:City {code:"PAR", name:"Paris", country:"FR"})`.
- Relations : `(c1)-[:NEAR {weight:0.8}]->(c2)`.

Redis – espaces de clés

- `session:<uuid>` → `userId` (EX 900)
- `offers:<from>:<to>` → liste d'offres JSON gzip (EX 60)
- `offers:<id>` → JSON détaillé (EX 300)
- Canal Pub/Sub : `offers:new`

7. Contraintes non fonctionnelles

- Latence moyenne d'une route `/offers` **200 ms** (cache *hit*) ; tolérance **700 ms** (cache *miss* + accès Mongo).
- Toutes les réponses sont `application/json; charset=utf-8`.
- Conteneur unique `docker-compose up` pour démarrer la stack.

8. Extensions (facultatives)

- *Recherche textuelle* : implémenter un paramètre `q=hotel` en s'appuyant sur l'index texte MongoDB.
- *Analytics* : route `/stats/top-destinations` (agrégation Mongo + cache Redis).
- *Monitoring* : exposer `/metrics` au format Prometheus (temps moyen, taux HIT cache).