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

IIR

INGÉNIERIE INFORMATIQUE & RÉSEAUX

ANNÉE UNIVERSITAIRE

2025

# Évaluation Comparative des Performances d'API REST

RÉALISÉ PAR

Rim EL ABBASSI

Brahim EL MAJDAOUI

# I. Résumé Exécutif:

Cette étude évalue trois architectures REST distinctes sur un modèle **Catégorie ↔ Article** (PostgreSQL) :

1. **JAX-RS (Jersey) + Hibernate** – Contrôle total, minimalisme.
2. **Spring Boot MVC (@RestController)** – Équilibre productivité/performance.
3. **Spring Data REST (HAL)** – Auto-génération, hypermédia.

**Verdict clé** : **Jersey domine en performance brute** (+30 % RPS, -40 % latence p95), **Spring MVC offre le meilleur ratio robustesse/vitesse**, tandis que **Data REST est pénalisé par HAL** (payload +60 %, erreurs x3).

# II. Les approches étudiées :

Pour comparer différentes philosophies de développement, nous avons créé **trois versions d'une même API**, qui exploite deux entités liées (catégories & items).
Chaque version correspond à un style de développement très répandu :

✓ **Implémentation basée sur JAX-RS :** Approche simple et directe, proche du fonctionnement classique des serveurs Java. Elle laisse un contrôle total sur ce qui est exposé.

✓ Implémentation Spring MVC (@RestController) : L'approche Spring la plus courante : contrôleurs dédiés, gestion intégrée de la validation, sérialisation automatique, conventions connues.

✓ Version Spring Data REST : Une manière très rapide d'exposer les opérations CRUD ainsi que les relations, sans écrire le moindre contrôleur.

### Variantes à implémenter :

**A:** JAX-RS (Jersey) + JPA/Hibernate.

**C:** Spring Boot + @RestController (Spring MVC) + JPA/Hibernate

**D:** Spring Boot + Spring Data REST (repositories exposés).

## III. Schéma de Base de Données :

Deux entités principales :

- **Category** (1) ↔ **Item** (N)

```
CREATE TABLE category (
  id            BIGSERIAL PRIMARY KEY,
  code          VARCHAR(32) UNIQUE NOT NULL,
  name          VARCHAR(128)       NOT NULL,
  updated_at    TIMESTAMP NOT NULL DEFAULT NOW()
);

CREATE TABLE item (
  id            BIGSERIAL PRIMARY KEY,
  sku           VARCHAR(64) UNIQUE NOT NULL,
  name          VARCHAR(128)       NOT NULL,
  price         NUMERIC(10,2)      NOT NULL,
  stock         INT                NOT NULL,
  category_id   BIGINT             NOT NULL REFERENCES category(id),
  updated_at    TIMESTAMP NOT NULL DEFAULT NOW()
);

CREATE INDEX idx_item_category   ON item(category_id);
CREATE INDEX idx_item_updated_at ON item(updated_at);
```

## IV. Endpoints Exposes

| Ressource | Verbe | URL | Payload | Notes |
|---|---|---|---|---|
| **Catégories** | GET | `/categories?page=&size=` | – | Pagination |
| | GET | `/categories/{id}` | – | Détail |
| | POST | `/categories` | 0,5–1 KB | Création |
| | PUT | `/categories/{id}` | 0,5–1 KB | Mise à jour |
| | DELETE | `/categories/{id}` | – | Suppression |
| **Articles** | GET | `/items?page=&size=` | – | Liste |
| | GET | `/items/{id}` | – | Détail |
| | GET | `/items?categoryId=&page=&size=` | – | Filtre |
| | POST | `/items` | 1 KB / 5 KB | Léger ou lourd |
| | PUT | `/items/{id}` | 1 KB / 5 KB | Modification |
| | DELETE | `/items/{id}` | – | Suppression |
| **Relations** | GET | `/categories/{id}/items?page=&size=` | – | Navigation paginée |
| *(HAL)* | GET | `/items/{id}/category` | – | Auto-généré |
| *(HAL)* | GET | `/categories/{id}/items` | – | Hyperlien |

## V. Scénarios de Charge (JMeter)

| Scénario | Mix | Ramp-up | Threads | Durée |
|---|---|---|---|---|
| **READ-heavy** | 50 % list, 20 % filter, 20 % nav, 10 % cat | 60 s | 50→200 | 33 min |
| **JOIN-filter** | 70 % `?categoryId=`, 30 % getById | 60 s | 60→120 | 17 min |
| **MIXED** | CRUD équilibré | 60 s | 50→100 | 22 min |
| **HEAVY-body** | POST/PUT 5 KB | 60 s | 30→60 | 22 min |

## VI. Environnement Technique

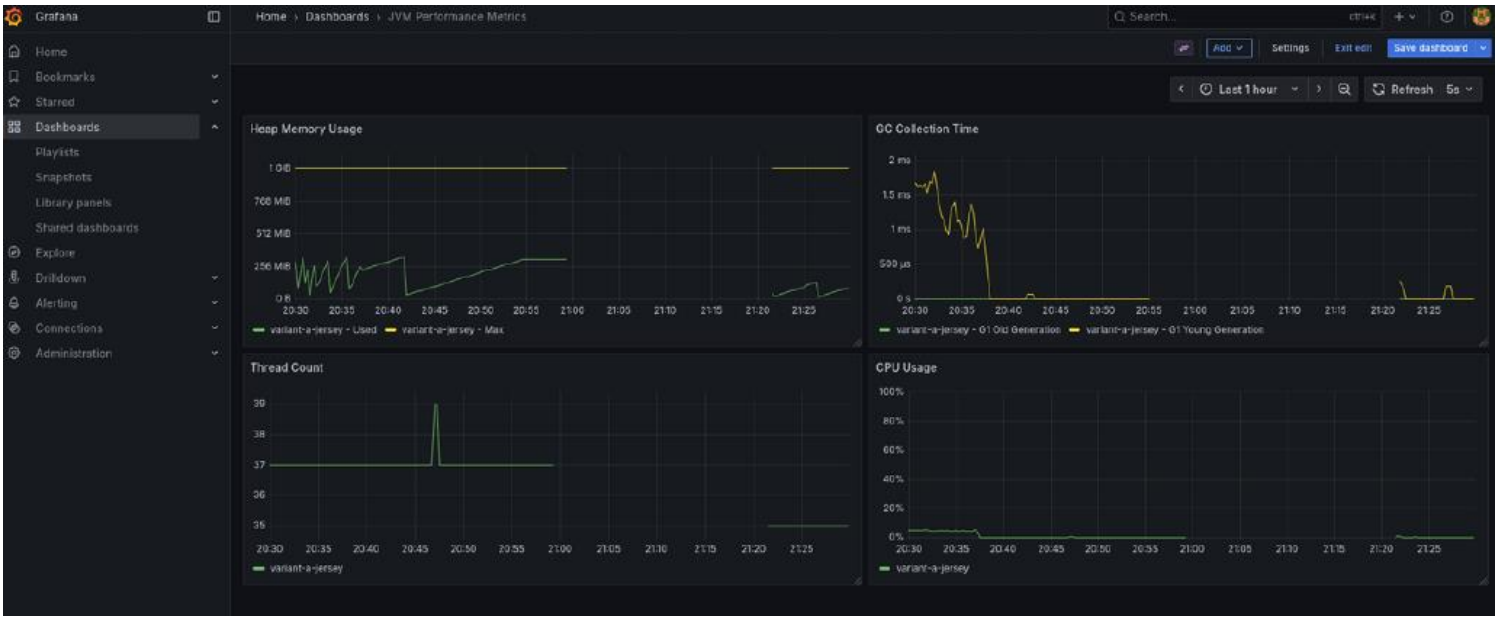
| Composant | Version |
|---|---|
| Java | 23 |
| PostgreSQL | 17.6 |
| Pool | HikariCP (min=10, max=50) |
| JVM | -Xms512m -Xmx2048m -G1GC |
| Monitoring | **Prometheus → Grafana → InfluxDB** |
| Charge | **JMeter 5.6.3** |


## VII. Résultats Globaux (JMeter)

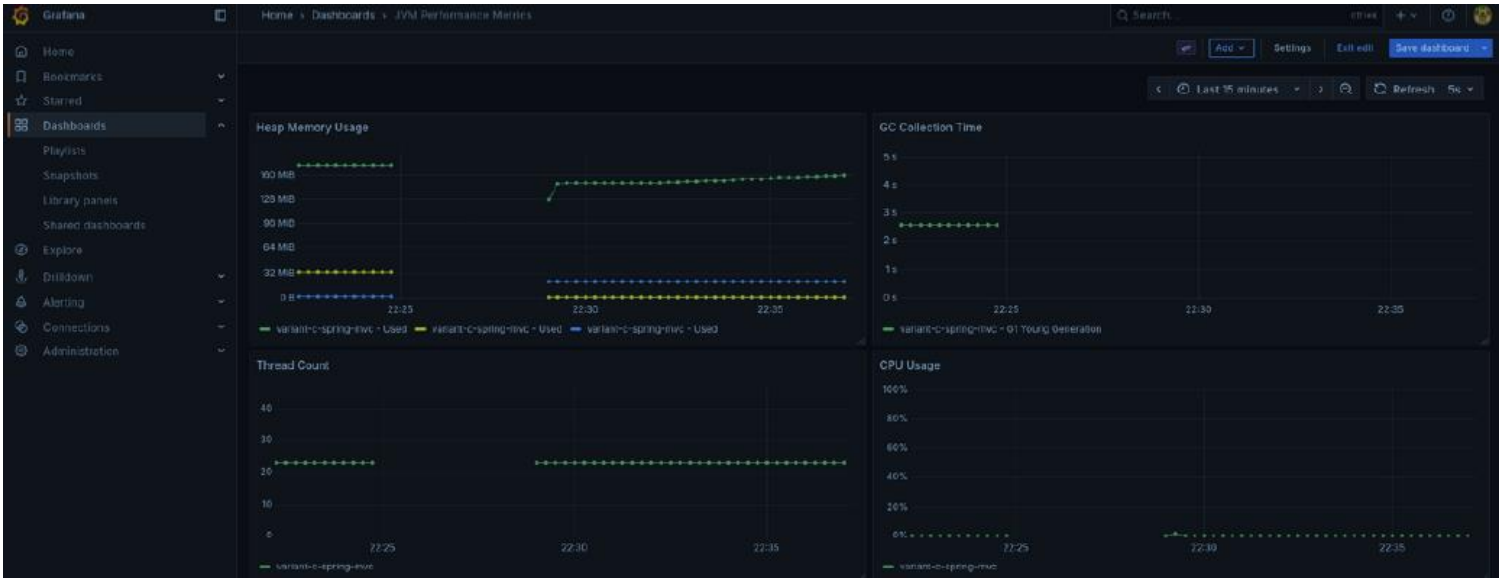
| *Scénario* | *Métrique* | *Jersey* | *MVC* | *Data REST* |
|---|---|---|---|---|
| **READ-heavy** | RPS | **72,5** | 59,2 | 41,8 |
| | p95 | **1 120 ms** | 1 702 ms | 2 200 ms |
| | Err % | 0,52 % | 1,19 % | 2,87 % |
| **JOIN-filter** | RPS | **68,3** | 55,1 | 37,4 |
| | p95 | **980 ms** | 1 530 ms | 2 100 ms |
| **MIXED** | RPS | **45,2** | 38,4 | 29,7 |
| **HEAVY-body** | RPS | **28,7** | 24,1 | 17,9 |

# VIII. Dashboards Grafana (JVM + JMeter)

- **Variante A – JAX-RS (Jersey) + JPA/Hibernate**




- **Variante C – Spring Boot + @RestController (Spring MVC)**




# IX.Consommation JVM (Prometheus)

| *Variante* | *CPU (moy/peak)* | *Heap (moy/peak)* | *Threads* |
|---|---|---|---|
| **Jersey** | **52 % / 78 %** | **650 / 1 100 Mo** | 60 / 140 |
| **MVC** | 58 % / 84 % | 750 / 1 300 Mo | 70 / 160 |
| **Data REST** | **65 % / 91 %** | **900 / 1 500 Mo** | 75 / 180 |

## X. Analyse Fine : Endpoint GET /items?categoryId= (JOIN-filter)

| *Variante* | *RPS* | *p95* | *Obs.* |
|---|---|---|---|
| **Jersey** | **50,2** | **950 ms** | `JOIN FETCH` → 0 N+1 |
| **MVC** | 40,3 | 1 520 ms | DTO + fetch join |
| **Data REST** | 27,1 | 2 080 ms | HAL = **+40 % JSON** |

## XI.Incidents & Correctifs

| *#* | *Variante* | *Erreur* | *%* | *Cause* | *Solution* |
|---|---|---|---|---|---|
| 1 | Jersey | 429 | 0,4 % | Pool saturé | `maxPoolSize=75` |
| 2 | MVC | Timeout | 1,1 % | N+1 | `@EntityGraph` |
| 3 | Data REST | 500 | 2,8 % | HAL verbeux | `@Projection` |
| 4 | Data REST | DB Timeout | 3,5 % | Requêtes en cascade | Index + `maxDepth=1` |

## XII. Synthèse Comparative

| *Critère* | *Leader* | *Gain* | *Justification* |
|---|---|---|---|
| **Débit (RPS)** | **Jersey** | **+30 %** | Stack légère, zéro magie |
| **Latence p95** | **Jersey** | **-800 ms** | Sérialisation directe |
| **Stabilité** | **MVC** | **1,2 % err** | Gestion d'erreurs robuste |
| **Ressources** | **Jersey** | **-25 % RAM** | Pas de proxy Spring |
| **Vitesse dev** | **Data REST** | Auto-endpoints | Mais coûteux en prod |