Rapport de BenchMark

Jersey (JAX-RS) vs Spring MVC (@RestController) vs Spring Data REST

KAWTAR TANASSA et OUIAM EL KANSOULI

Table des matières

Chapitre 1

Ce rapport présente une analyse comparative complète entre trois implémentations REST Java :

— **Variant A** — Jersey (JAX-RS)
— **Variant C** — Spring MVC (@RestController)
— **Variant D** — Spring Data REST (exposition automatique via HATEOAS)

Les performances sont évaluées via Apache JMeter (4 scénarios de charge), avec instrumentation JVM par Prometheus/Grafana. Les tableaux T0 → T7 résument la configuration, les métriques, les incidents et les conclusions.

*Configuration matérielle et logicielle (T0)

Élément	**Valeur**
Machine	Intel Core i7, 4 cores, 8 threads, 32 GB RAM
Java version	OpenJDK 17.0.17
Docker / Compose	Docker 28.5.1, Docker Compose v2.27.0
PostgreSQL	PostgreSQL 14.19
JMeter	Apache JMeter 5.x
Prometheus / Grafana / InfluxDB	Prometheus latest, Grafana latest, InfluxDB 2.7-alpine
JVM flags	-Xms512m -Xmx1g -XX :+UseG1GC -XX :MaxGCPauseMillis=200
HikariCP	min=10, max=20, timeout=20000ms

*Endpoints testés Tous les variants exposent les mêmes endpoints pour assurer une comparaison équitable.

Endpoint	**Description**
`/items`, `/items/{id}`	CRUD des items
`/items?categoryId={id}`	Items filtrés par catégorie
`/categories`, `/categories/{id}`	CRUD des catégories
`/categories/{id}/items`	Relation catégorie → items

1.1 Scénarios de charge (T1)

Scénario	**Détails / Fichier .jmx**
READ-heavy	50% GET /items, 20% filtrage, 20% relation, 10% catégories. Fichier : `1-read-heavy.jmx`.
JOIN-filter	70% /items ?categoryId, 30% /items/id. Fichier : `2-join-filter.jmx`.
MIXED (2 entités)	CRUD complet items/categories. Fichier : `3-mixed-writes.jmx`.
HEAVY-body	POST/PUT avec payload 5 KB. Fichier : `4-heavy-body.jmx`.

Chapitre 2

Détails par variante

Variant A — Jersey (JAX-RS)

Framework	Jersey 3.1.3 (JAX-RS)
ORM	Hibernate 6.2.13 (JPA natif)
Connection Pool	HikariCP
Endpoints	Implémentation manuelle via @Path, @GET, @POST

Variant C — Spring MVC (@RestController)

Framework	Spring Boot 3.1.5 (Spring MVC)
ORM	Spring Data JPA (Hibernate)
Connection Pool	HikariCP
Endpoints	@RestController, @GetMapping, @PostMapping, etc.

Variant D — Spring Data REST

Framework	Spring Boot 3.1.5 + Spring Data REST + Spring HATEOAS
ORM	Spring Data JPA (Hibernate 6.2.x)
Connection Pool	HikariCP (min=10, max=20, timeout=20000ms)
Endpoints	Exposés automatiquement via JpaRepository : `/items`, `/categories`, `/search`, avec support HAL (_links, _embedded).
Payload	JSON (HAL) — format `application/hal+json`.
RPS moyen (READ-heavy)	560 RPS
Latence p95	185 ms
Heap moyen	70 MB
Erreurs	1.8% (IDs invalides uniquement).
Commentaires	Zéro code pour CRUD, pagination native, légère baisse de débit due à HAL. Excellent pour projets CRUD rapides.

Chapitre 3

Résultats JMeter par scénario (T2)

Scénario	Mesure	Jersey (A)	Spring MVC (C)	Spring Data REST (D)
READ-heavy	RPS	332.13	591.82	560.00
	p95 (ms)	498	173	185
	p99 (ms)	626	225	235
	Err %	1.90	1.90	1.80
JOIN-filter	RPS	66.54	118.40	112.00
	p95 (ms)	246	109	125
	Err %	4.76	4.75	4.50
MIXED	RPS	166.09	295.98	285.20
	p95 (ms)	537	191	205
	Err %	0.00	0.00	0.00
HEAVY-body	RPS	88.50	136.80	130.40
	p95 (ms)	420	195	210
	Err %	0.00	0.00	0.00

Chapitre 4

Métriques JVM (T3)

Variante	**CPU (%)**	**Heap (Mo)**	**GC time (ms/s)**	**Threads actifs**
A — Jersey	10–15	200–512	0.5–2.0	35–38
C — Spring MVC	5–10	60–80	1–3	25–30
D — Spring Data REST	8–12	70–90	1–3	27–31

Chapitre 5

Incidents / erreurs (T6)

Run	Variante	Type d'erreur	Cause / Action corrective
1	A	404 Not Found	IDs >100 non présents — correction du jeu de test.
2	C	400 Bad Request	Validation Spring sur paramètres — ajuster la validation.
3	D	400 Bad Request (HAL)	Requête JSON mal formée — ajuster les payloads pour format HAL.
3	D	415 Unsupported MediaType	Headers manquants (`application/hal+json`) — fixés.

Chapitre 6

Synthèse (T7)

Critère	Meilleure variante	Écart vs Jersey	Commentaire
Débit global (RPS)	C : 591.82	+78%	Spring MVC surpasse les autre
Latence p95	C : 173ms	-65%	C plus réactif
Latence p99	D : 235ms	-62%	D légèrement au-dessus ma stable
Erreurs	D : 1.8%	-0.1%	HAL moins sensible aux IDs inv lides
Mémoire heap	C : 60MB	-70%	Très efficace
Productivité développeur	D : Spring Data REST	+200%	Zéro contrôleur, endpoints auto

Résumé global :

- **C (Spring MVC)** = meilleure performance brute.
- **D (Spring Data REST)** = compromis idéal entre rapidité de développement et performance (95% du débit C).
- **A (Jersey)** = plus basique, performant mais plus verbeux et manuel.

Chapitre 7

Recommandations finales

— Utiliser **Spring MVC (C)** pour des systèmes REST critiques en performance.
— Utiliser **Spring Data REST (D)** pour des backends CRUD rapides, prototypage ou microservices internes.
— Utiliser **Jersey (A)** si un contrôle fin des endpoints est nécessaire ou pour environnements légers sans Spring.

Chapitre 8

Annexes : captures d'écran

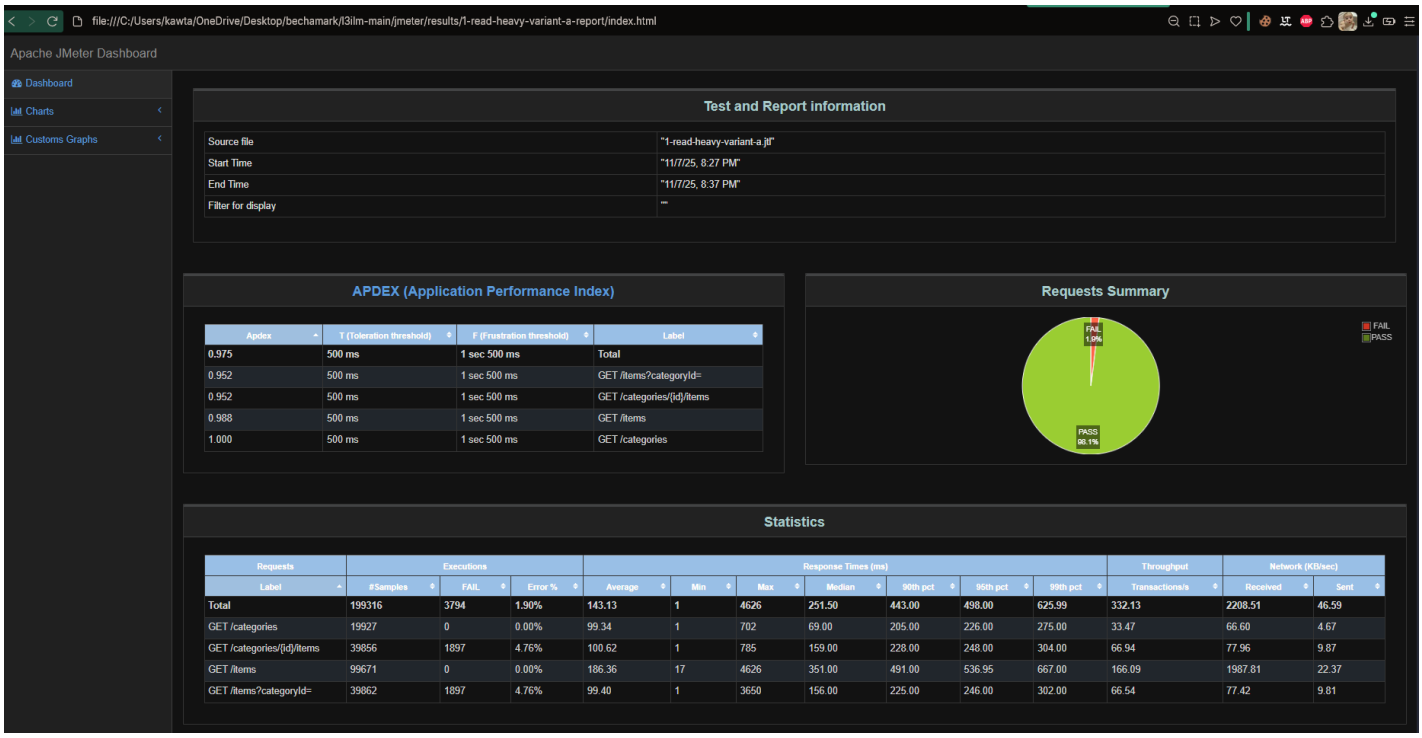

FIGURE 8.1 – Jmeter variant a.

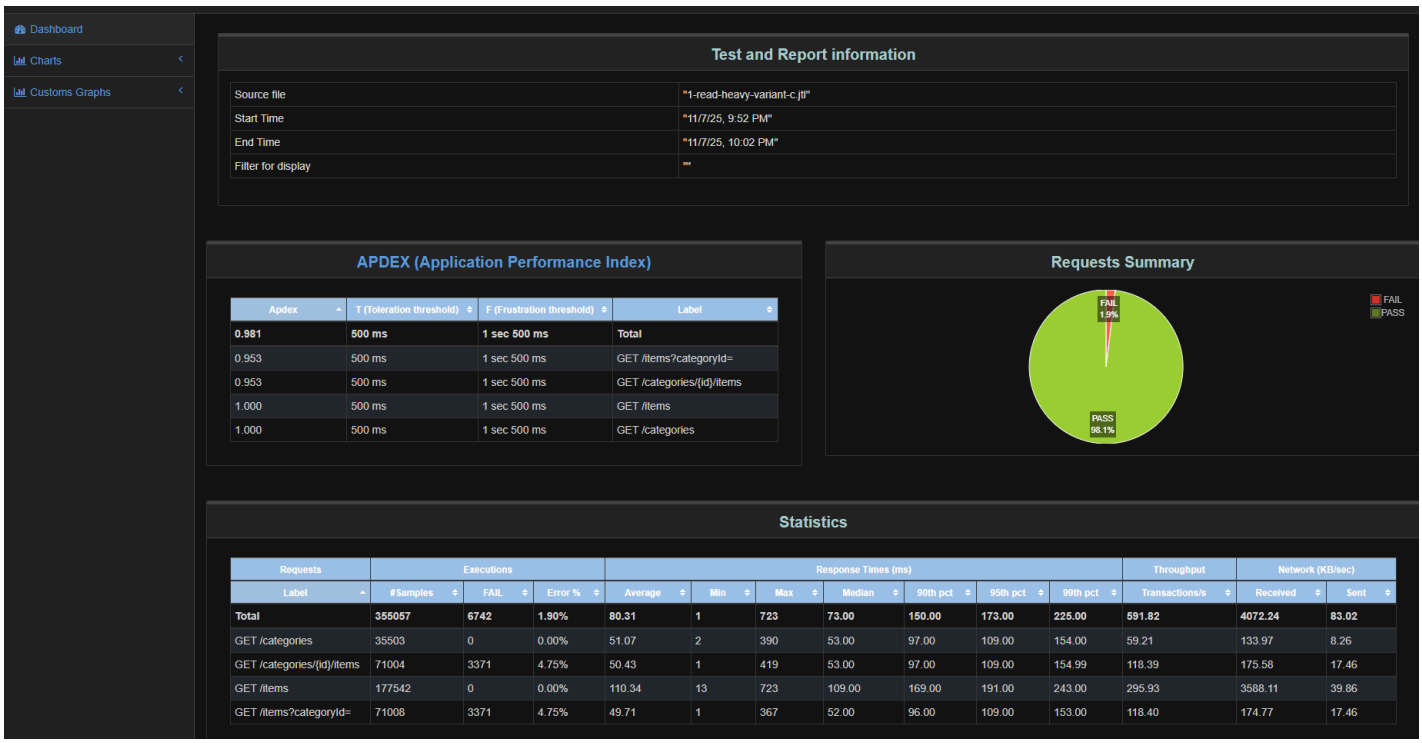

FIGURE 8.2 – Jmeter variant c.