AppSport

Livrables TD Backend 2026

1. Dépôt GitHub

🔗 https://github.com/gokunguru/AppSport

2. Description des cas d'usage

UC1 – Créer une session de sport

L'utilisateur crée une session de sport via l'API REST. Le backend valide les données, crée la session métier et la persiste en base H2.

Une fois la session enregistrée, un événement Kafka WorkoutCreatedEvent est publié afin de notifier la création de la session de manière asynchrone.

Endpoint : `POST /api/sessions`

UC2 – Ajouter un set à une session

L'utilisateur ajoute un set (exercice, nombre de répétitions, poids) à une session existante. Le backend récupère la session concernée, y ajoute le set, puis met à jour les données en base.

Endpoint : `POST /api/sessions/{id}/sets`

UC3 – Consulter toutes les sessions

L'utilisateur consulte la liste de toutes les sessions enregistrées. Les données sont récupérées depuis la base H2 et renvoyées sous forme de DTOs.

Endpoint : `GET /api/sessions`

UC4 – Consulter une session par identifiant

L'utilisateur consulte le détail d'une session spécifique via son identifiant. Les données sont récupérées depuis la base H2 et renvoyées sous forme de DTO.

Endpoint : `GET /api/sessions/{id}`

3. Répartition du travail

Le projet AppSport a été réalisé en binôme par Manyl et Kamil.

👤 Manyl	👤 Kamil
• **Configuration initiale du projet** • **Architecture Clean Architecture** • **UC1 et UC2** • **Modèle métier et persistance JPA/H2** • **Kafka Producer**	• UC3 et UC4 • Kafka Consumer • Dockerisation (Dockerfile, docker-compose) • Intégration et validation finale • Documentation

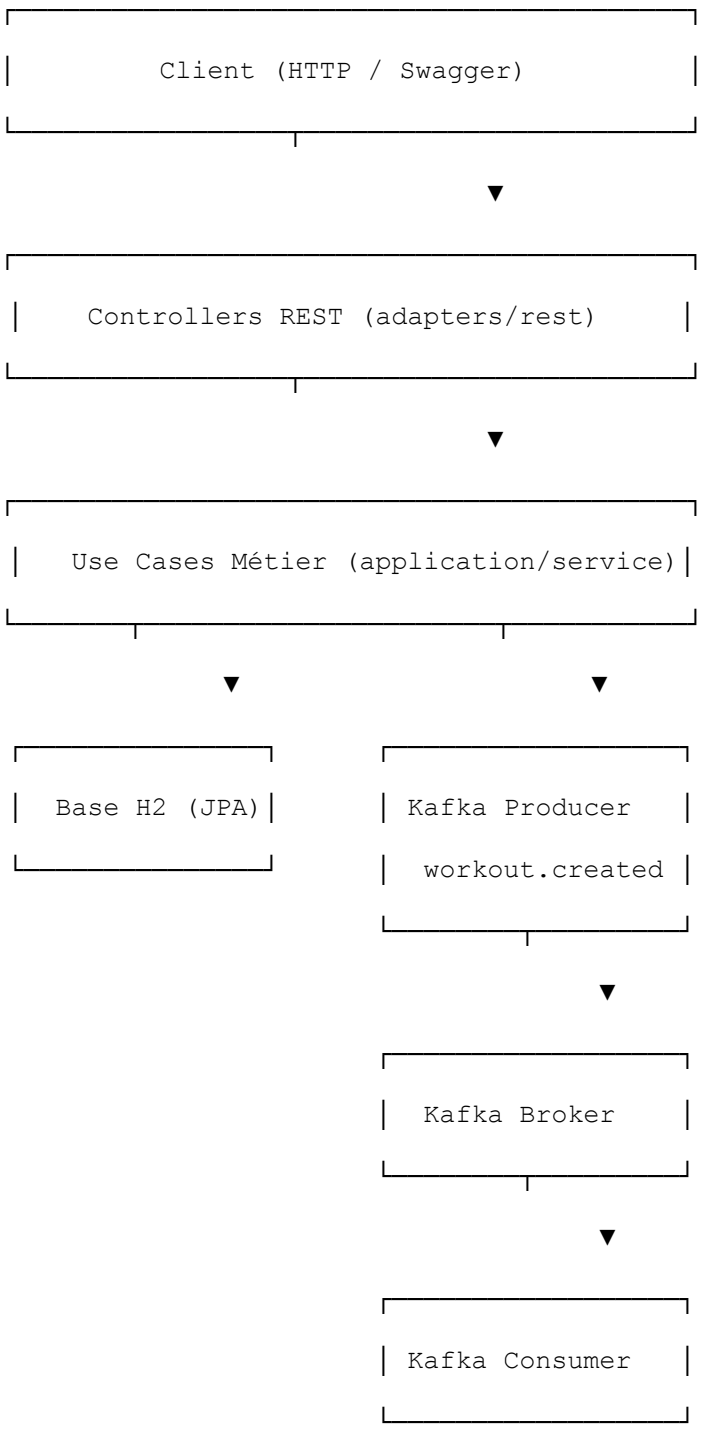
✅ Conclusion

Le travail a été réparti de manière complémentaire et progressive :

- **Manyl a conçu et implémenté les bases fonctionnelles et architecturales du projet**
- **Kamil a assuré l'intégration avancée, la communication événementielle complète et la dockerisation**

Le projet final est fonctionnel, cohérent et conforme aux exigences du TD Backend 2026.

4. Diagramme d'architecture

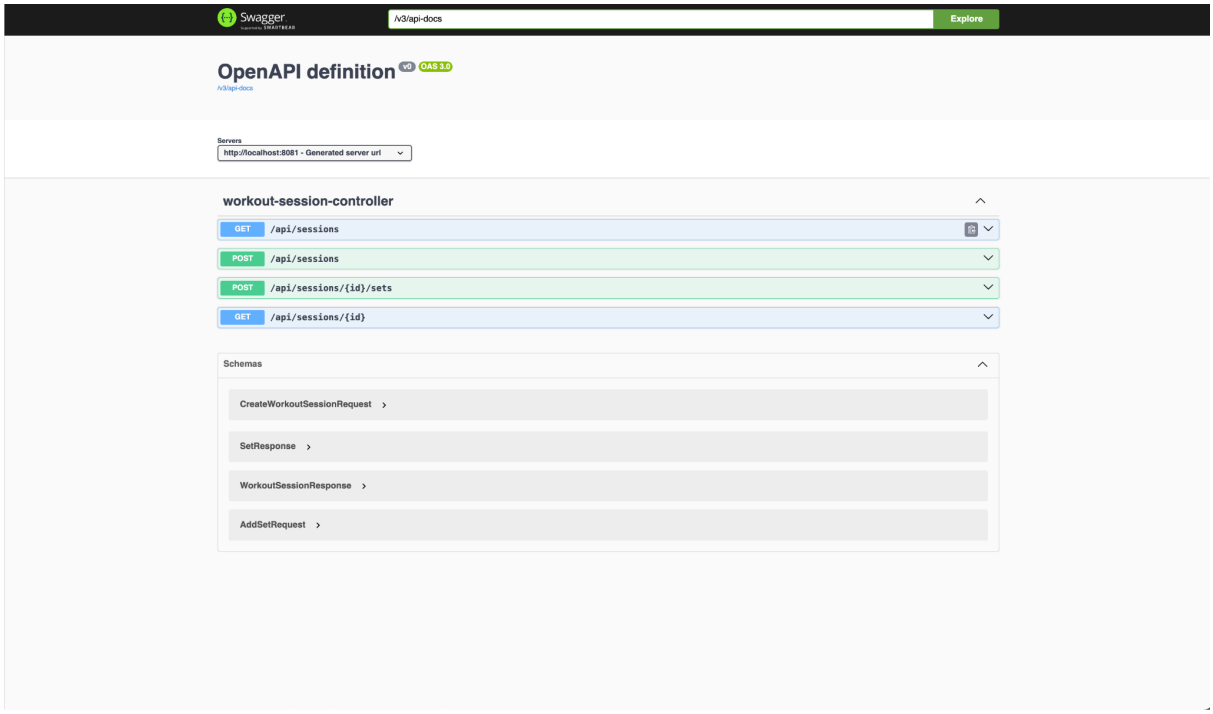


Principes clés

- Les controllers REST ne contiennent aucune logique métier
- Les use cases sont indépendants de Spring
- Kafka est isolé dans la couche infrastructure
- Communication événementielle asynchrone
- L'ensemble de l'application est exécuté via Docker Compose

5. Captures d'écran

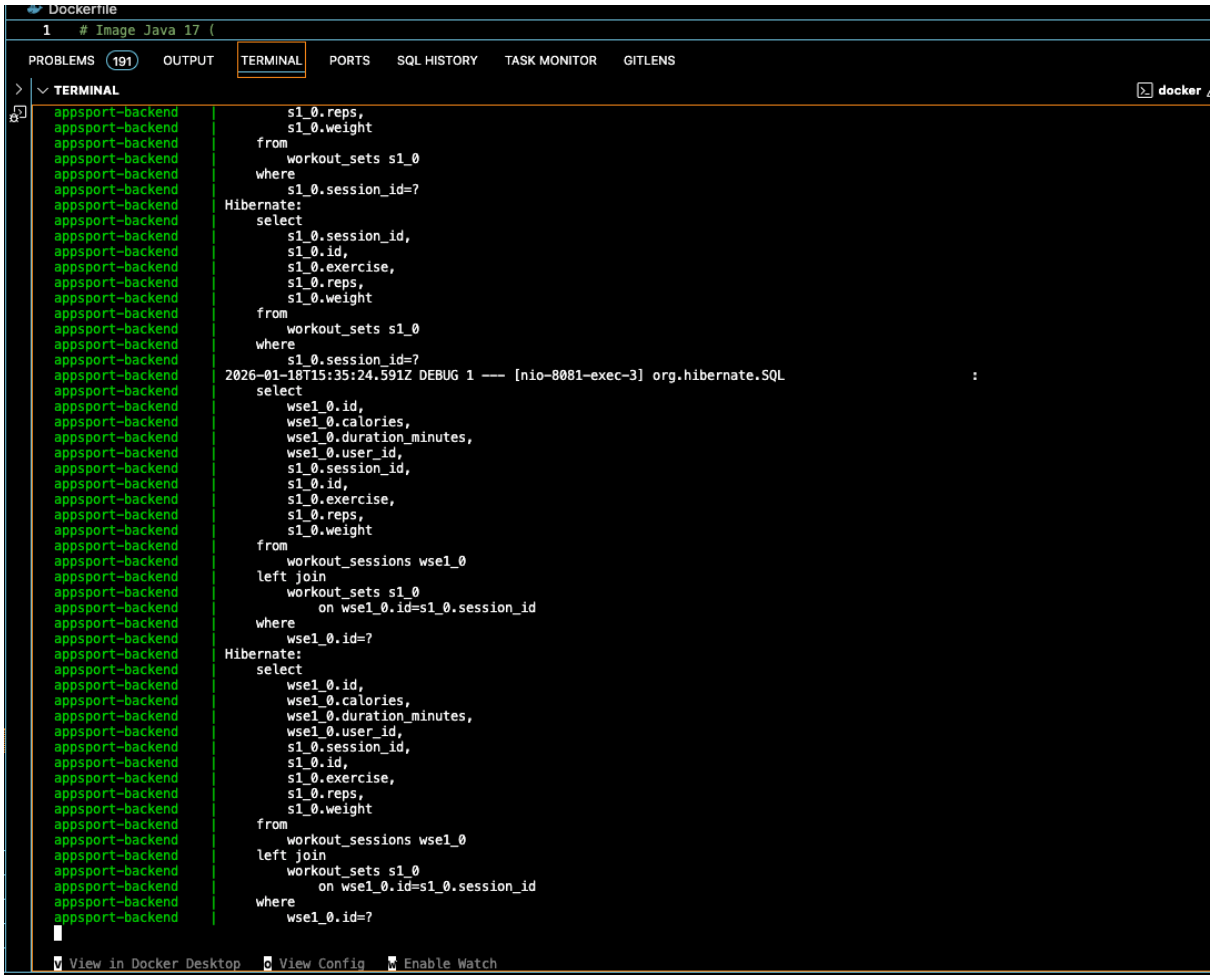
API REST & Swagger



Kafka

```
2026-01-18T16:28:50.405+01:00  INFO 81711 --- [io-8081-exec-10] o.a.k.c.t.i.KafkaMetricsCollector        : initializing Kafka metrics collector
2026-01-18T16:28:50.411+01:00  INFO 81711 --- [io-8081-exec-10] o.a.k.clients.producer.KafkaProducer     : [Producer clientId=producer-1] Instantiated an idempotent producer.
2026-01-18T16:28:50.423+01:00  INFO 81711 --- [io-8081-exec-10] o.a.kafka.common.utils.AppInfoParser     : Kafka version: 4.1.1
2026-01-18T16:28:50.423+01:00  INFO 81711 --- [io-8081-exec-10] o.a.kafka.common.utils.AppInfoParser     : Kafka commitId: be816b82d25370ce
2026-01-18T16:28:50.423+01:00  INFO 81711 --- [io-8081-exec-10] o.a.kafka.common.utils.AppInfoParser     : Kafka startTimeMs: 1768750130423
2026-01-18T16:28:50.439+01:00  INFO 81711 --- [ad | producer-1] org.apache.kafka.clients.Metadata        : [Producer clientId=producer-1] Cluster ID: Wg8ZO145TguBB4o-8qZV6Q
2026-01-18T16:28:50.577+01:00  INFO 81711 --- [ad | producer-1] o.a.k.c.p.internals.TransactionManager   : [Producer clientId=producer-1] ProducerId set to 0 with epoch 0
 Published to workout.created key=1
2026-01-18T16:28:50.614+01:00  INFO 81711 --- [ntainer#0-0-C-1] c.e.d.a.s.WorkoutEventConsumerService    : [KAFKA CONSUMER] Workout session received | sessionId=1 userId=1 calories=320
<==========---> 83% EXECUTING [1m 9s]
> :bootRun
```

Docker


Dockerfile
1 # Image Java 17 (
PROBLEMS 191
OUTPUT
TERMINAL
PORTS
SQL HISTORY
TASK MONITOR
GITLENS
TERMINAL
docker
appsport-backend | s1_0.reps,
appsport-backend | s1_0.weight
appsport-backend | from
appsport-backend | workout_sets s1_0
appsport-backend | where
appsport-backend | s1_0.session_id=?
appsport-backend | Hibernate:
appsport-backend | select
appsport-backend | s1_0.session_id,
appsport-backend | s1_0.id,
appsport-backend | s1_0.exercise,
appsport-backend | s1_0.reps,
appsport-backend | s1_0.weight
appsport-backend | from
appsport-backend | workout_sets s1_0
appsport-backend | where
appsport-backend | s1_0.session_id=?
appsport-backend | 2026-01-18T15:35:24.591Z DEBUG 1 --- [nio-8081-exec-3] org.hibernate.SQL :
appsport-backend | select
appsport-backend | wse1_0.id,
appsport-backend | wse1_0.calories,
appsport-backend | wse1_0.duration_minutes,
appsport-backend | wse1_0.user_id,
appsport-backend | s1_0.session_id,
appsport-backend | s1_0.id,
appsport-backend | s1_0.exercise,
appsport-backend | s1_0.reps,
appsport-backend | s1_0.weight
appsport-backend | from
appsport-backend | workout_sessions wse1_0
appsport-backend | left join
appsport-backend | workout_sets s1_0
appsport-backend | on wse1_0.id=s1_0.session_id
appsport-backend | where
appsport-backend | wse1_0.id=?
appsport-backend | Hibernate:
appsport-backend | select
appsport-backend | wse1_0.id,
appsport-backend | wse1_0.calories,
appsport-backend | wse1_0.duration_minutes,
appsport-backend | wse1_0.user_id,
appsport-backend | s1_0.session_id,
appsport-backend | s1_0.id,
appsport-backend | s1_0.exercise,
appsport-backend | s1_0.reps,
appsport-backend | s1_0.weight
appsport-backend | from
appsport-backend | workout_sessions wse1_0
appsport-backend | left join
appsport-backend | workout_sets s1_0
appsport-backend | on wse1_0.id=s1_0.session_id
appsport-backend | where
appsport-backend | wse1_0.id=?
View in Docker Desktop
View Config
Enable Watch