Labo 3 – Exposition d'une API RESTful pour un système multi-magasins

Cours : **Architecture Logicielle (LOG430)**

Session : **Été 2025**

Date du laboratoire : **Semaine du 02 Juin 2025**

Le génie pour l'industrie

Contexte

À l'issue du Laboratoire 2, vous avez conçu et mis en œuvre une architecture logicielle évolutive permettant à une entreprise de gérer plusieurs magasins répartis géographiquement, un centre logistique, ainsi qu'une maison mère assurant les fonctions administratives.

Le Laboratoire 3 s'inscrit dans la continuité de ce travail, avec un nouvel objectif : exposer les principales fonctionnalités métier du système à travers une API RESTful. Ce type d'interface est aujourd'hui indispensable pour :

— Permettre l'interopérabilité avec des clients externes (applications web ou mobiles) ;

— Favoriser une séparation claire entre les couches front-end et back-end ;

— Préparer le système à une future évolution vers une architecture orientée microservices.

Dans ce laboratoire, vous devrez donc ajouter une couche d'API RESTful à votre système existant, en respectant les principes fondamentaux de l'architecture REST, tout en assurant une cohérence avec la logique métier et les contraintes techniques définies dans les laboratoires précédents.

Ce travail vous permettra de pratiquer l'exposition sécurisée d'un service applicatif, la documentation des API, ainsi que leur intégration dans une architecture logicielle avec CI/CD.

Objectifs d'apprentissage

— Concevoir et exposer une API RESTful respectant les bonnes pratiques.

— Séparer les couches d'accès, de présentation et de logique métier dans une architecture existante.

— Documenter l'API à l'aide d'outils comme Swagger (OpenAPI).

— Implémenter des contrôleurs REST pour différents cas d'usage métiers.

— Réaliser des tests d'API (Postman, Swagger UI, ou JUnit).

— Sécuriser les points d'entrée API (CORS, authentification de base, etc.).

— Intégrer les tests dans le pipeline CI/CD.

Tâches à réaliser

1. Extension de l'architecture avec une couche d'API REST

— Ajoutez une couche d'API REST exposant les fonctionnalités principales (consultation, génération de rapport, mise à jour).

— Respectez les principes du modèle MVC ou hexagonal pour bien isoler la logique métier.

— Définissez une structure claire pour vos routes REST.

2. Documentation des API

— Créez une documentation Swagger (OpenAPI 3.0) de vos endpoints.

— Décrivez les méthodes HTTP, les formats d'entrée/sortie, les statuts de réponse, et des exemples de requêtes.

— Intégrez une interface de visualisation comme Swagger UI ou Redoc.

3. Sécurité et accessibilité

— Activez et configurez le CORS pour permettre les appels depuis un client distant.

— Implémentez une authentification simple (token statique, Basic Auth ou JWT léger si connu).

4. Tests et validation

— Créez une collection de requêtes API à tester (via Postman ou Swagger).

— Développez des tests automatisés (MockMVC, JUnit ou équivalent) pour valider les endpoints.

— Ajoutez les tests à votre pipeline CI/CD.

5. Déploiement et démonstration

— Assurez un déploiement fonctionnel de l'API, accessible localement ou dans un conteneur Docker.

— Fournissez des instructions claires pour l'utilisation des endpoints.

6. Bonnes pratiques de conception d'une API RESTful

Afin de garantir la qualité, la cohérence et l'évolutivité de votre API, vous devez respecter les bonnes pratiques reconnues dans la conception d'APIs RESTful.

Respect des principes REST : Votre API doit suivre les contraintes fondamentales de l'architecture REST, notamment :

— Sans état (Stateless) : chaque requête contient toutes les informations nécessaires.

— Cacheabilité : les réponses doivent expliciter si elles sont mises en cache.

— Interface uniforme : notamment via URI, méthodes HTTP, et représentations standardisées.

— Système en couches : l'API ne doit pas exposer la complexité sous-jacente. HATEOAS (Hypermedia as the Engine of Application State) : inclure dans les réponses des liens vers les actions possibles liées à la ressource.

Structure des URIs : Utilisez une nomenclature cohérente et orientée ressource :

- /api/products → collection
- /api/products/123 → élément
- /api/stores/42/stock → ressource liée à une autre
- Pas de verbes dans les URIs (préférez ‘GET /products‘ à ‘/getProducts‘).

Utilisation appropriée des méthodes (verbes) HTTP :

- GET → Récupérer une ressource
- POST → Créer une nouvelle ressource
- PUT → Mettre à jour une ressource existante
- DELETE → Supprimer une ressource
- PATCH → Modifier partiellement une ressource

Codes de statut HTTP explicites : Répondez avec des statuts clairs et standards :

- 200 OK → pour une requête réussie
- 201 Created → après une création
- 204 No Content → après une suppression sans réponse
- 400 Bad Request → pour une erreur de validation
- 401 Unauthorized ou 403 Forbidden → selon le cas
- 404 Not Found → si la ressource n’existe pas
- 406 Not Acceptable → si le format demandé n’est pas supporté
- 500 → Internal Server Error‘ pour une défaillance serveur

Messages d’erreurs normalisés : Structurez vos messages d’erreurs pour aider les clients :

```
{
  "timestamp": "2025-06-02T10:21:00Z",
  "status": 400,
  "error": "Bad Request",
  "message": "Le champ 'name' est requis.",
  "path": "/api/products"
}
```

Format de réponse et versionnage :

- Utilisez **JSON** comme format par défaut.
- Prévoyez un **versionnage** clair (ex. : /api/v1/products).
- Supportez le **Content Negotiation** via le header `Accept`.

7. Pagination, filtrage, tri : pour les collections volumineuses :

- Pagination → /api/products ?page=2&size=20
- Filtrage et tri → /api/products ?category=coffee&sort=name,asc

Exemples de cas d'usage à exposer via l'API

Voici quelques exemples représentatifs de cas d'usage que votre API RESTful devrait permettre de traiter. Chaque cas illustre un scénario métier pertinent dans le contexte d'un système multi-magasins. Vous êtes encouragés à proposer et implémenter d'autres cas d'usage originaux en fonction de vos choix d'architecture et de conception. Faites preuve de créativité tout en respectant les principes REST !

- **UC1 – Générer un rapport consolidé des ventes**
  Permet d'obtenir un résumé agrégé des ventes réalisées dans tous les magasins pour une période donnée.
- **UC2 – Consulter le stock d'un magasin spécifique**
  Permet d'interroger le niveau de stock d'un magasin identifié par son ID.
- **UC3 – Visualiser les performances globales des magasins**
  Fournit un tableau de bord regroupant des indicateurs clés de performance (ventes, fréquentation, disponibilité des stocks).
- **UC4 – Mettre à jour les informations d'un produit**
  Permet de modifier les attributs d'un produit existant (nom, prix, stock, etc.) dans la base de données.

Livrables attendus (organisation du dépôt)

- **.Zip file du code source du Lab 3 :** dossier `lab3/` avec :
  - Contrôleurs REST et services associés
  - Fichier de documentation Swagger (YAML ou JSON)
  - Configuration sécurité/CORS
  - Collection de requêtes (Postman ou équivalent)
  - Tests automatisés
- **Documentation technique :**
  - Description des endpoints
  - Exemple de requêtes/réponses
  - Screenshots Swagger/Postman
  - Vue mise à jour du modèle 4+1 (si nécessaire)
- **Mise à jour du pipeline CI/CD pour les tests d'API.**
- **Instructions d'exécution dans le `README.md`.**

Conseils pédagogiques

— Favorisez la simplicité et la clarté dans la structure de vos routes et vos DTOs (Data Transfer Objects).

— Testez les endpoints dès leur création via Swagger ou Postman.

— Nommez explicitement les ressources REST selon les conventions.

— Conservez une séparation claire entre la logique métier et l'interface API.

— Restez cohérents avec les livrables des Labs 1 et 2 (modularité, nomenclature, tags Git).