Système Universitaire SOA

Architecture Orientée Services

Presenter par :

Hamzaoui Moetez

Saidane Med Lamine

Sommaire

01 CONTEXTE ET OBJECTIFS

02 ARCHITECTURE GLOBALE

03 SERVICES REST

04 SERVICES SOAP

05 PÉRIMÈTRE FONCTIONNEL

06 STACK TECHNIQUE

07 SÉCURITÉ - JWT

08 DÉPLOIEMENT DOCKER

07 AVANTAGES DE L'ARCHITECTURE SOA

08 CONCLUSION

Contexte et objectifs du projet

Contexte :

L'université souhaite moderniser son système d'information enadoptant une architecture orientée services (SOA) pour séparer les fonctionnalités en services indépendants.

Solution proposée :

- Mettre en place une architecture SOA complète
- Développer des services web hétérogènes (REST + SOAP)
- Assurer la sécurité via JWT
- Garantir l'interopérabilité entre technologies
- Préparer le déploiement conteneurisé (Docker)

Architecture globale du système

Architecture globale du système :

- Architecture SOA basée sur 6 services
- API Gateway comme point d’entrée unique
- Communication via REST & SOAP

Services REST

Auth Service

- Tech:
  - Spring Boot + JWT
- Endpoints:
  - POST /auth/register
  - POST /auth/login
  - GET /auth/me

Grade Service

- Tech:
  - Node.js / Express
- Endpoints:
  - POST /grades
  - GET /grades/student/:id
  - GET /grades/student/:id/moyenne

Student Service

- Tech:
  - Node.js / Express
- Endpoints:
  - GET /students
  - POST /students
  - PUT /students/:id
  - DELETE /students/:id

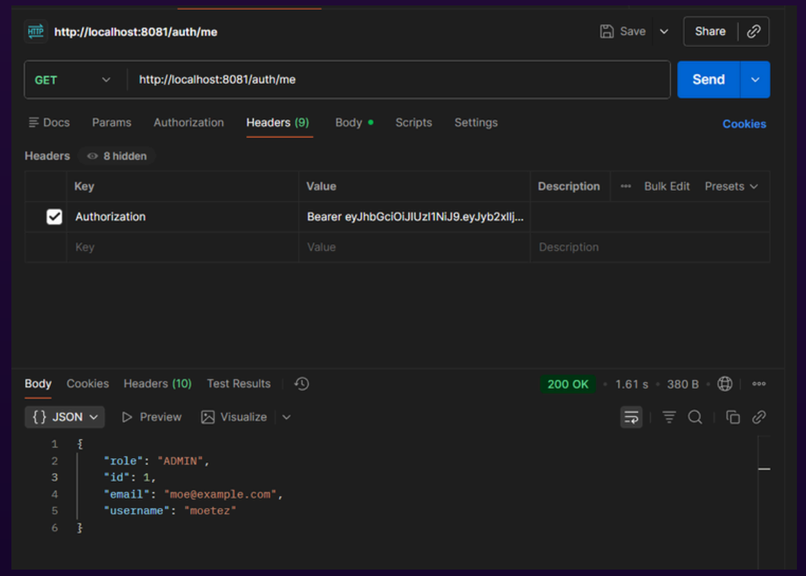
Sécurité

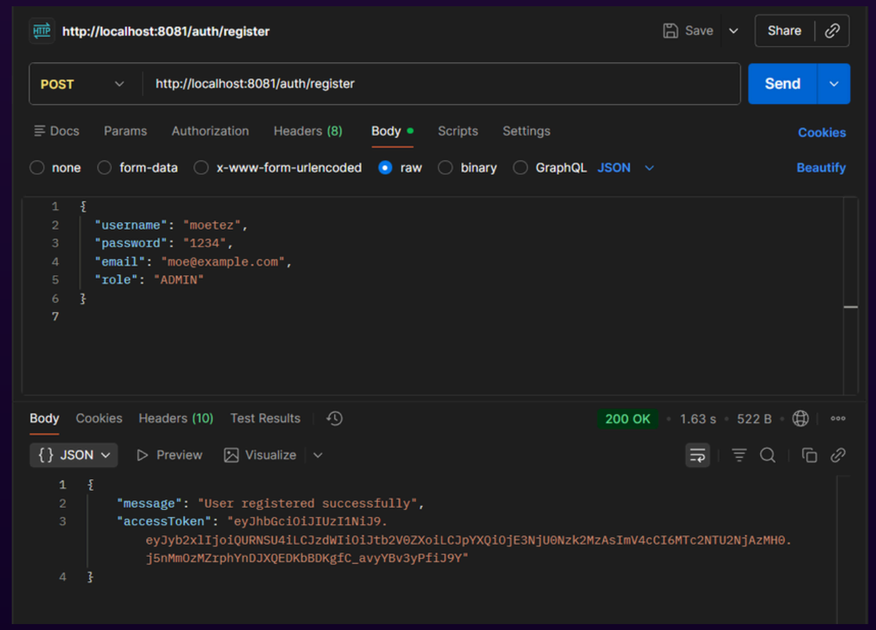
Tous les endpoints protégés par JWT

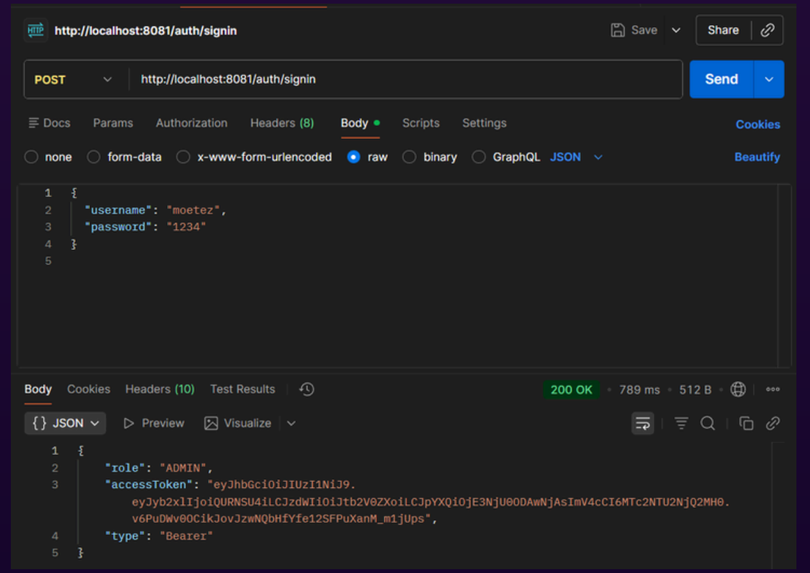
Authorization: Bearer <token>

Tests des services REST via Postman

Auth Service

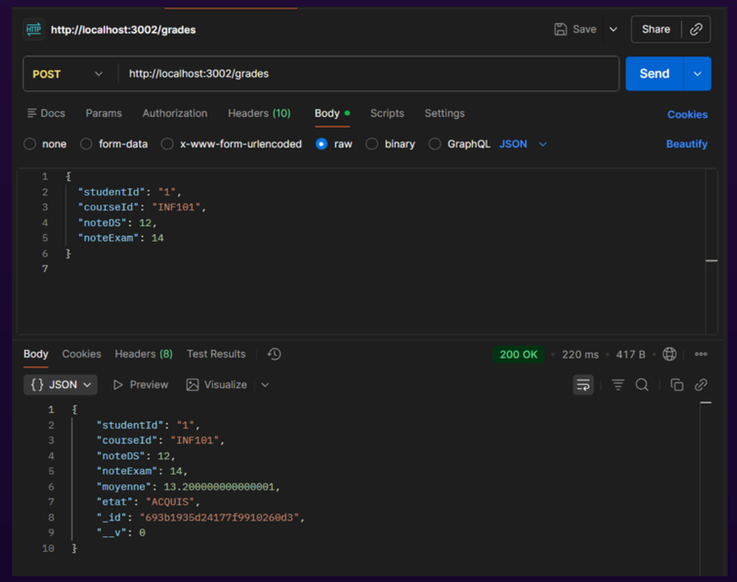


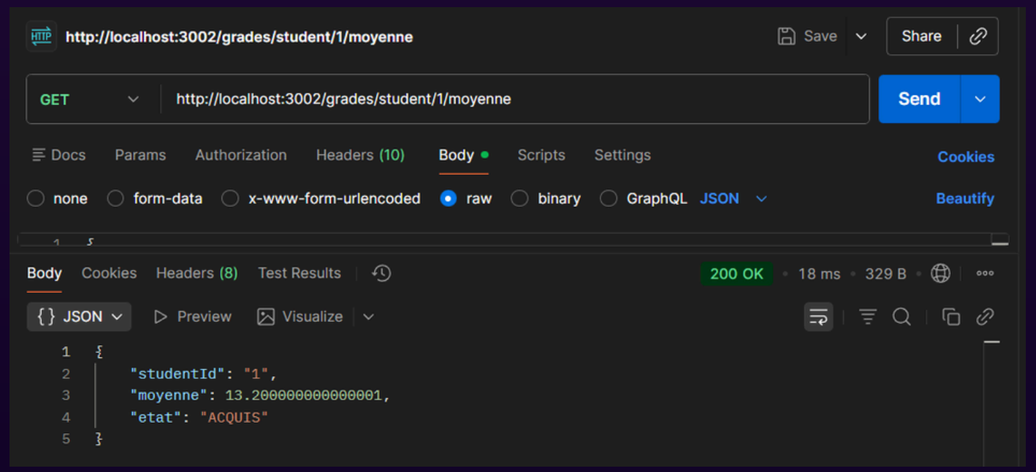


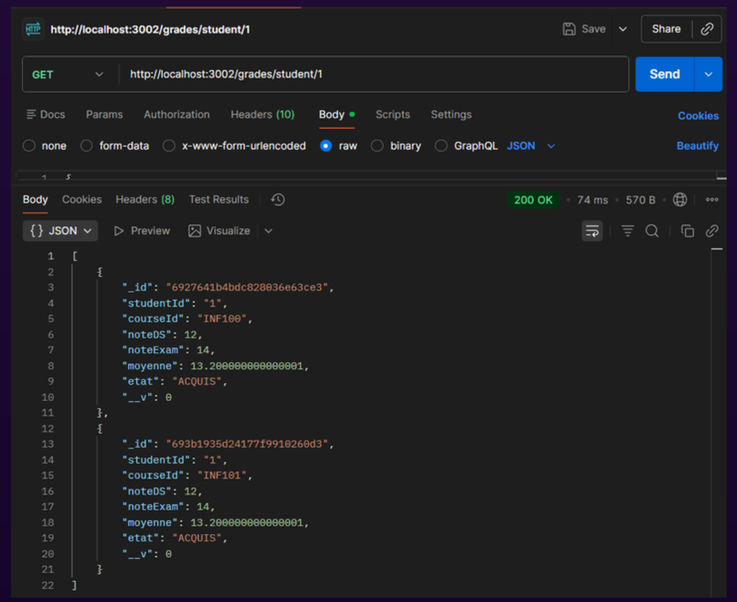


Tests des services REST via Postman

Grade Service

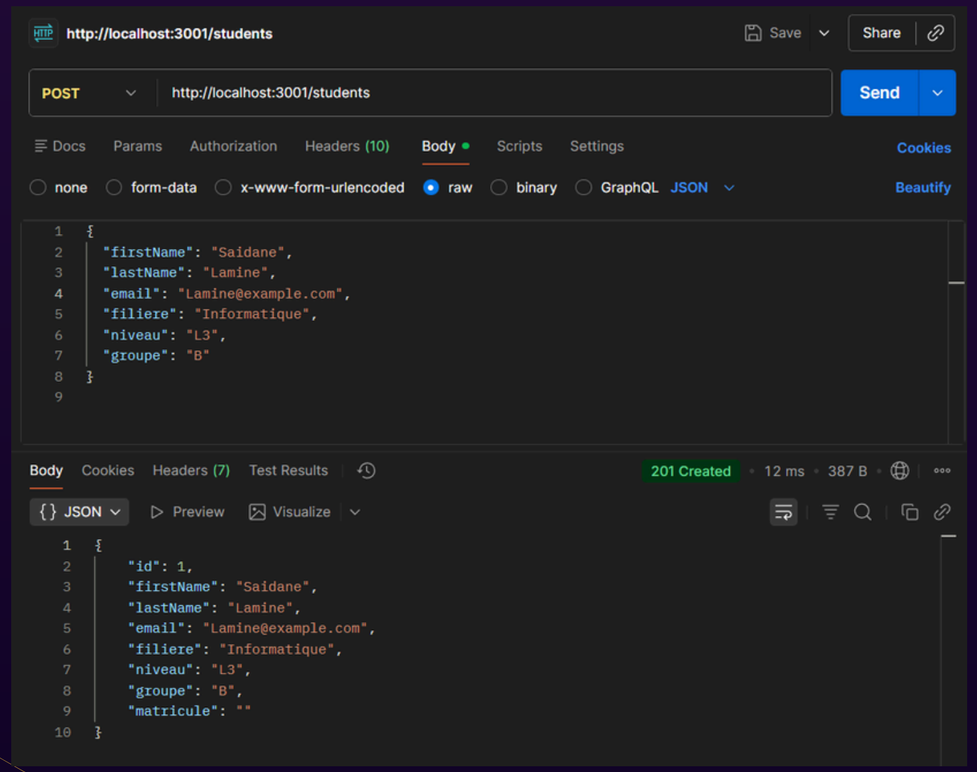


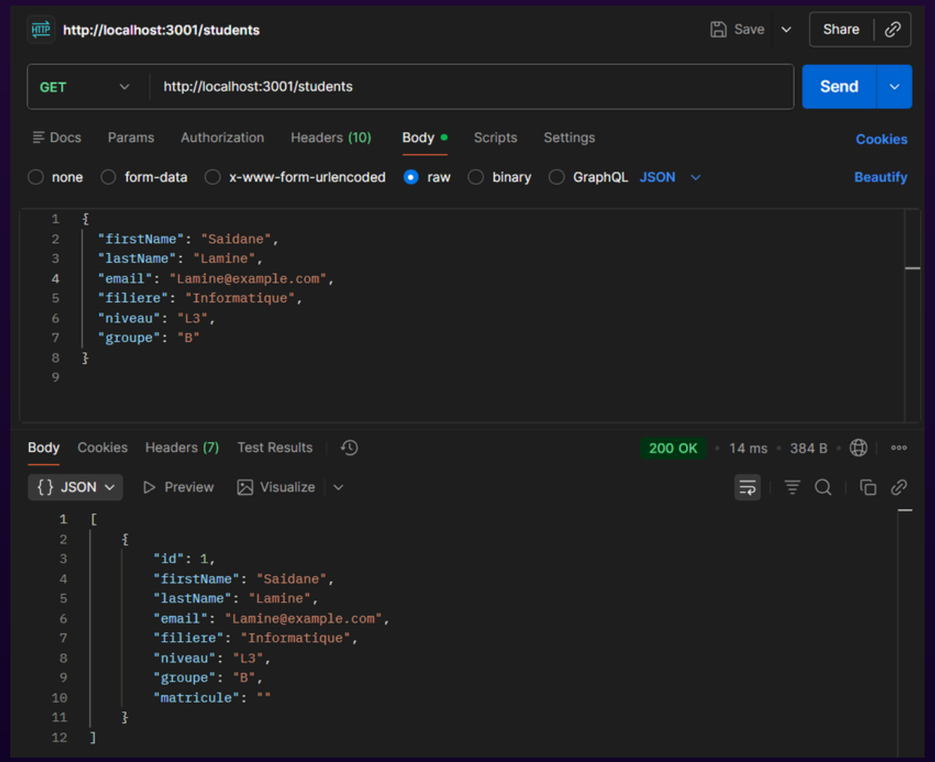




Tests des services REST via Postman

Student Service



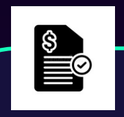


Services SOAP

Course Service

- **Tech:**
  - Spring Boot + SOAP/WSDL
- **Endpoints:**
  - addCourse(Course)
  - updateCourse(Course)
  - getCourseById(id)
  - listCourses()
  - listCourses()

Billing Service



- **Tech:**
  - Spring Boot + SOAP/WSDL
- **Endpoints:**
  - calculateFees(studentId)
  - getInvoice(studentId)
  - payInvoice(invoiceId)

Calcul: Total = Inscription + Scolarité + Pénalités

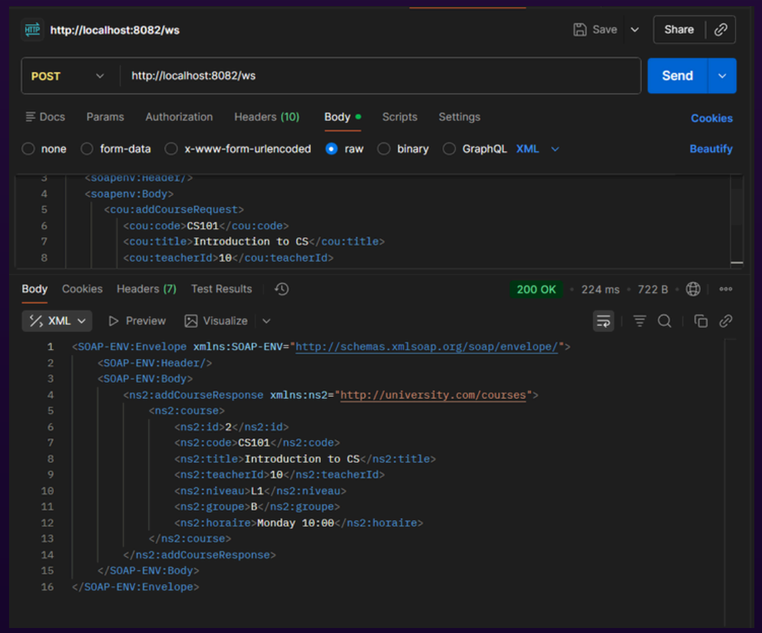
URL WSDL

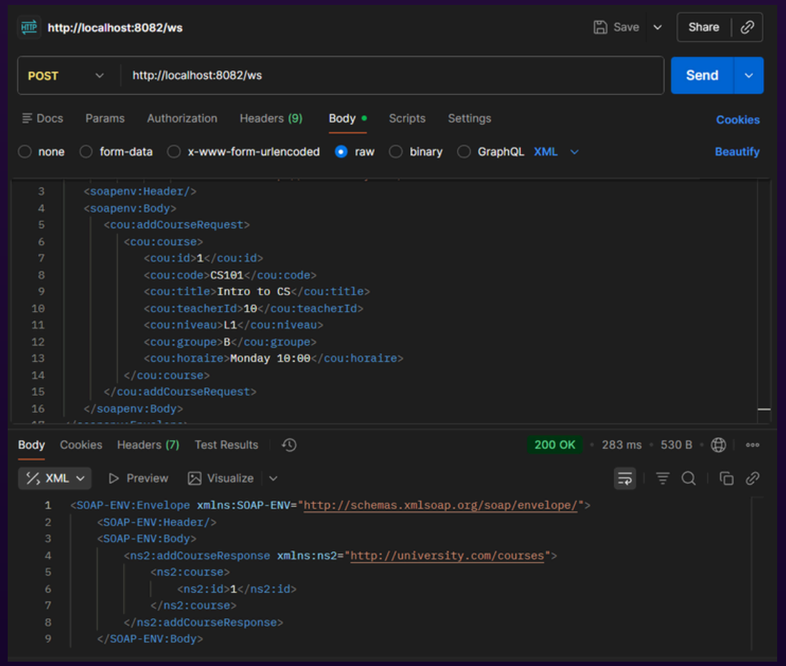
http://host:port/ws/courses?wsdl

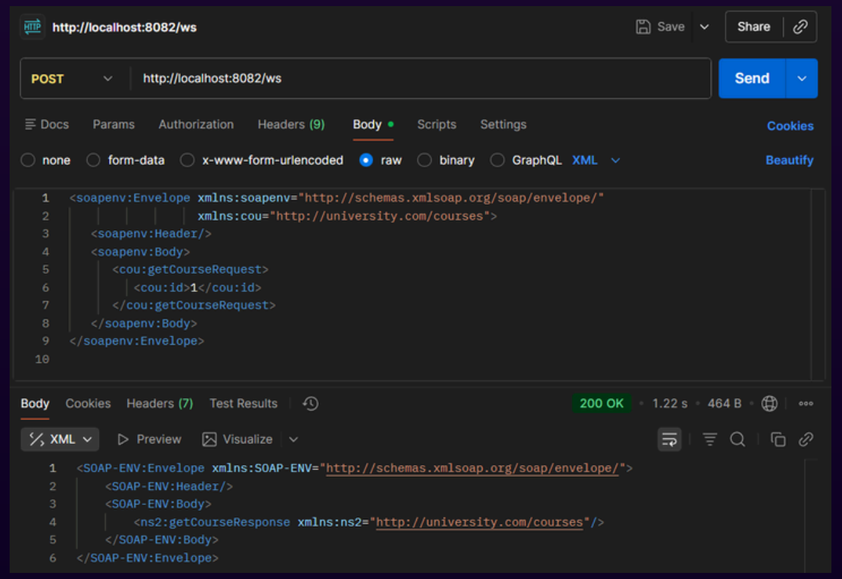
http://host:port/ws/billing?wsdl

Tests des services SOAP via Postman

Course Service

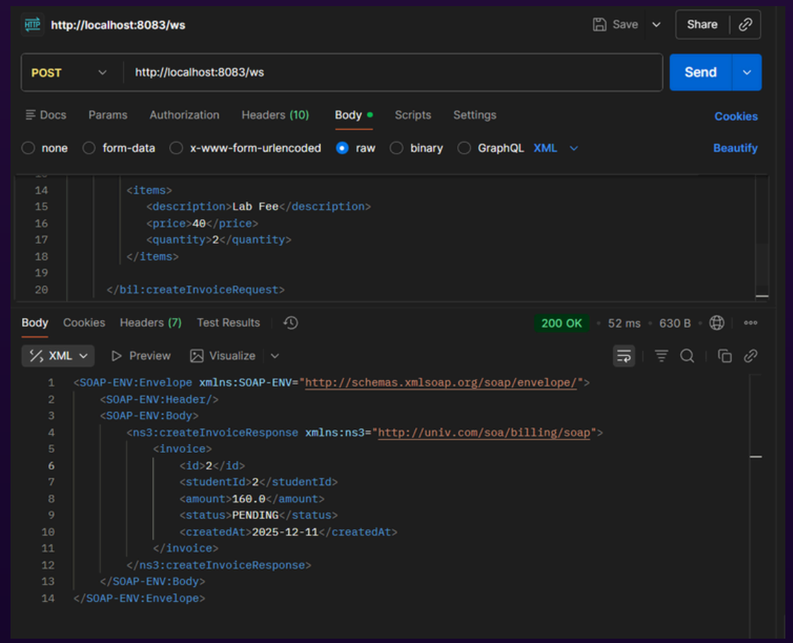


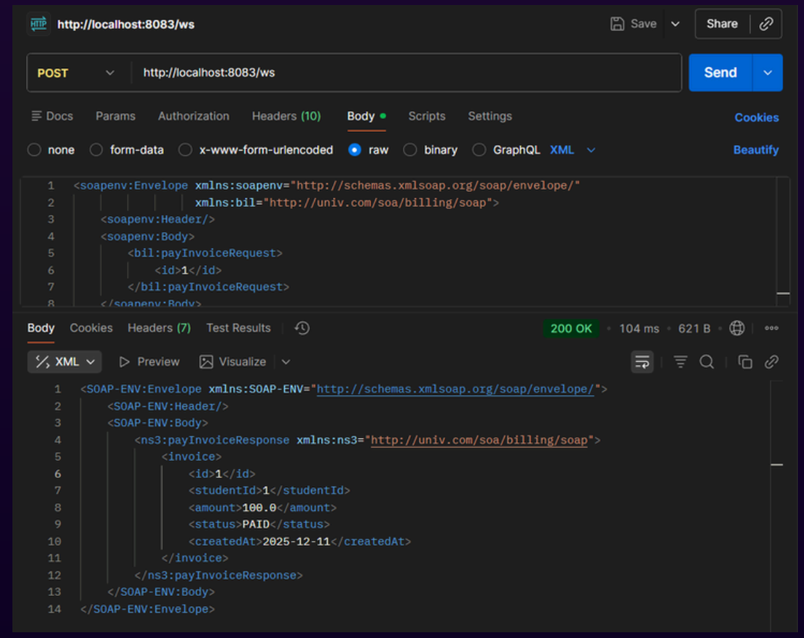


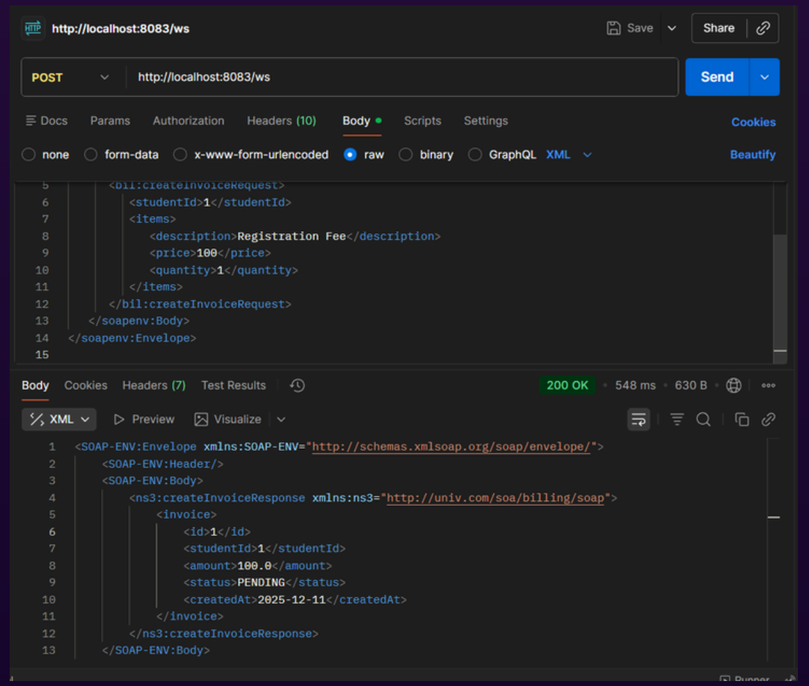


Tests des services SOAP via Postman

Billing Service







Périmètre Fonctionnel

Gestion Utilisateurs

- Création de comptes
- Authentification JWT
- Gestion des rôles

Gestion Notes

- Saisie des notes
- Calcul de moyennes
- État: PRINCIPALE/CONTROLE

Gestion Étudiants

- CRUD complet
- Filtrage par filière/niveau

Facturation

- Calcul des frais
- Gestion paiements
- Génération reçus

Périmètre Fonctionnel

Gestion Cours

- Création et modification
- Emploi du temps

API Gateway

- Point d'entrée unique
- Routage intelligent

Stack Technique

Spring Boot

Node.js

Express.js

JWT

SOAP/WSDL

REST API

Docker

Spring Cloud Gateway

MongoDB

Stack Technique

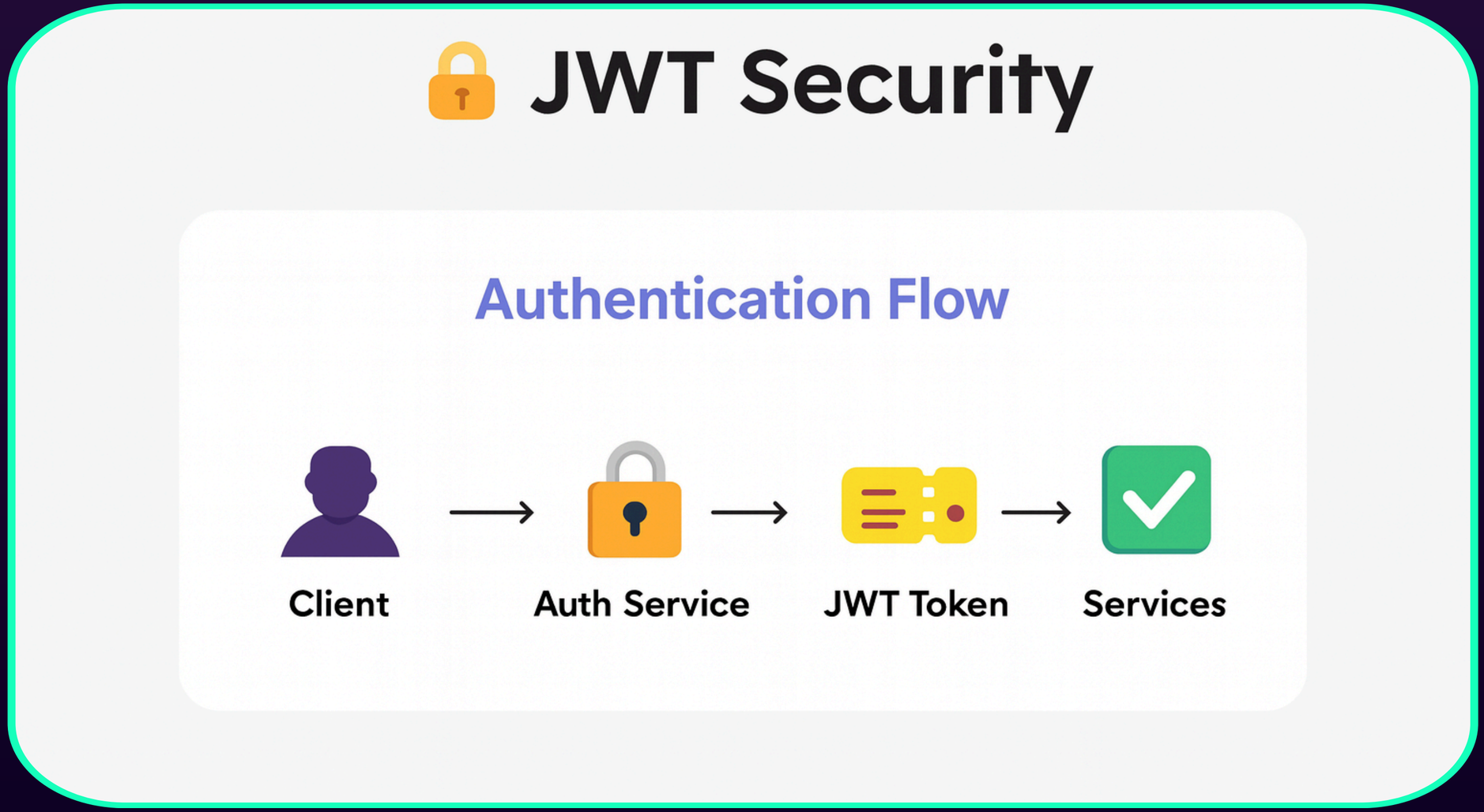
Commandes de Démarrage

```
# Auth Service
cd services/auth-service
mvn spring-boot:run

# Student Service
cd services/student-service
npm start

# Grade Service
cd services/grade-service
npm start
```

```
# API Gateway
cd services/api-gateway
mvn spring-boot:run

# Ou avec Docker Compose
docker-compose up --build
```

Sécurité - JWT

Flux d'Authentification



Sécurité - JWT

Fonctionnalités

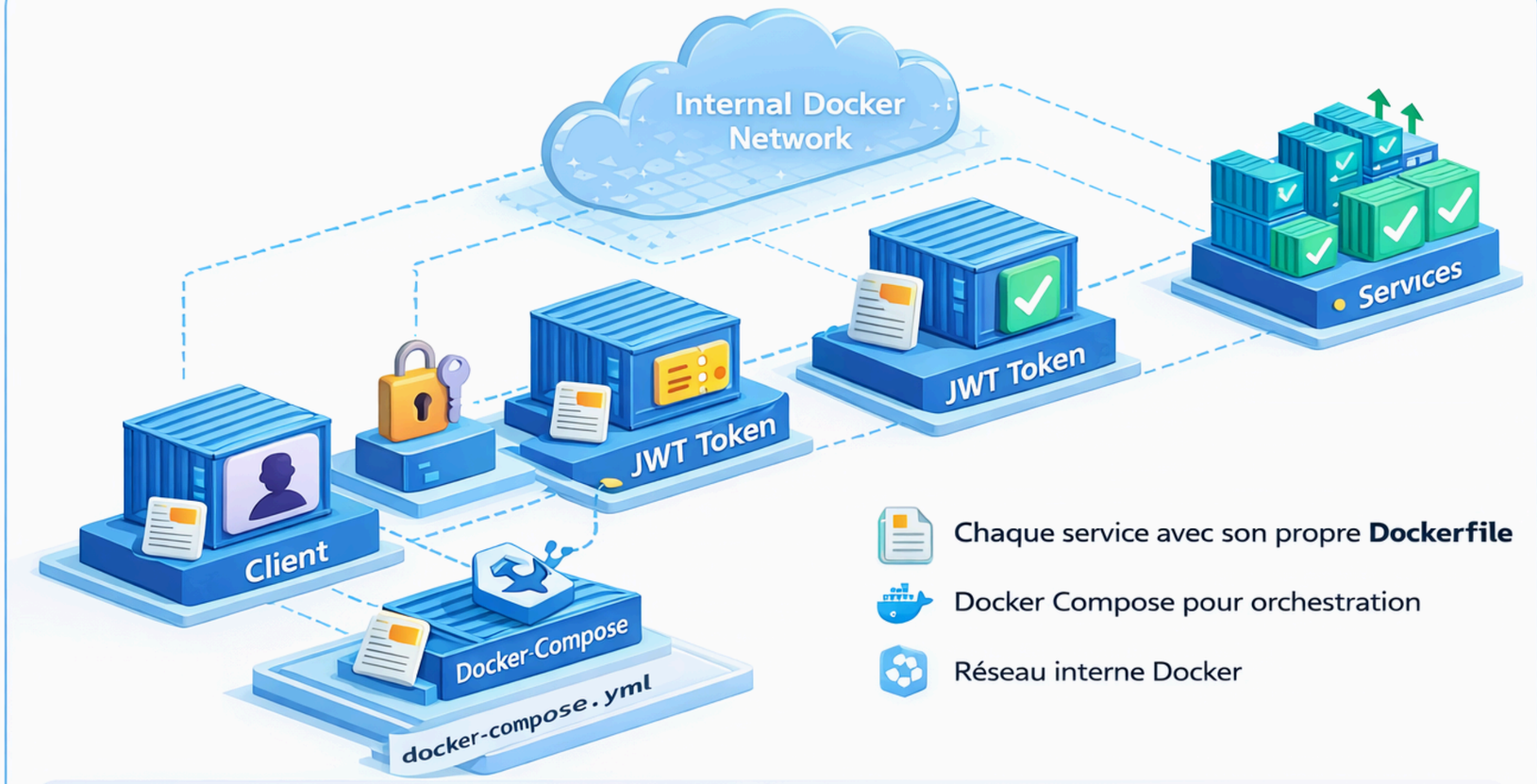
- Génération de token JWT
- Validation des tokens
- Gestion des rôles
- Passwords hashés (BCrypt)

Rôles Utilisateurs

- ADMIN - Gestion complète
- TEACHER - Saisie notes
- STUDENT - Consultation

Déploiement Docker

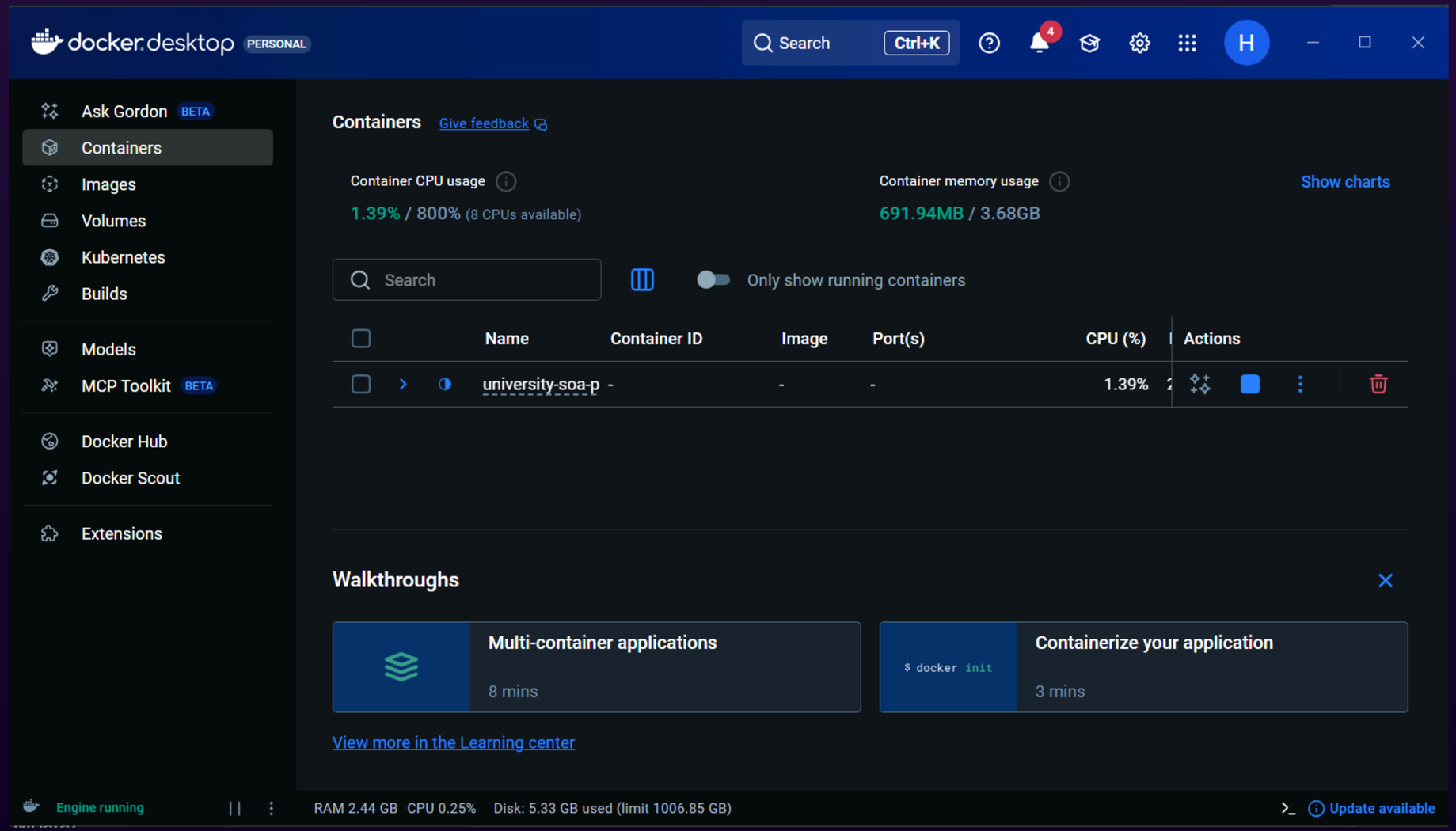
Architecture Docker



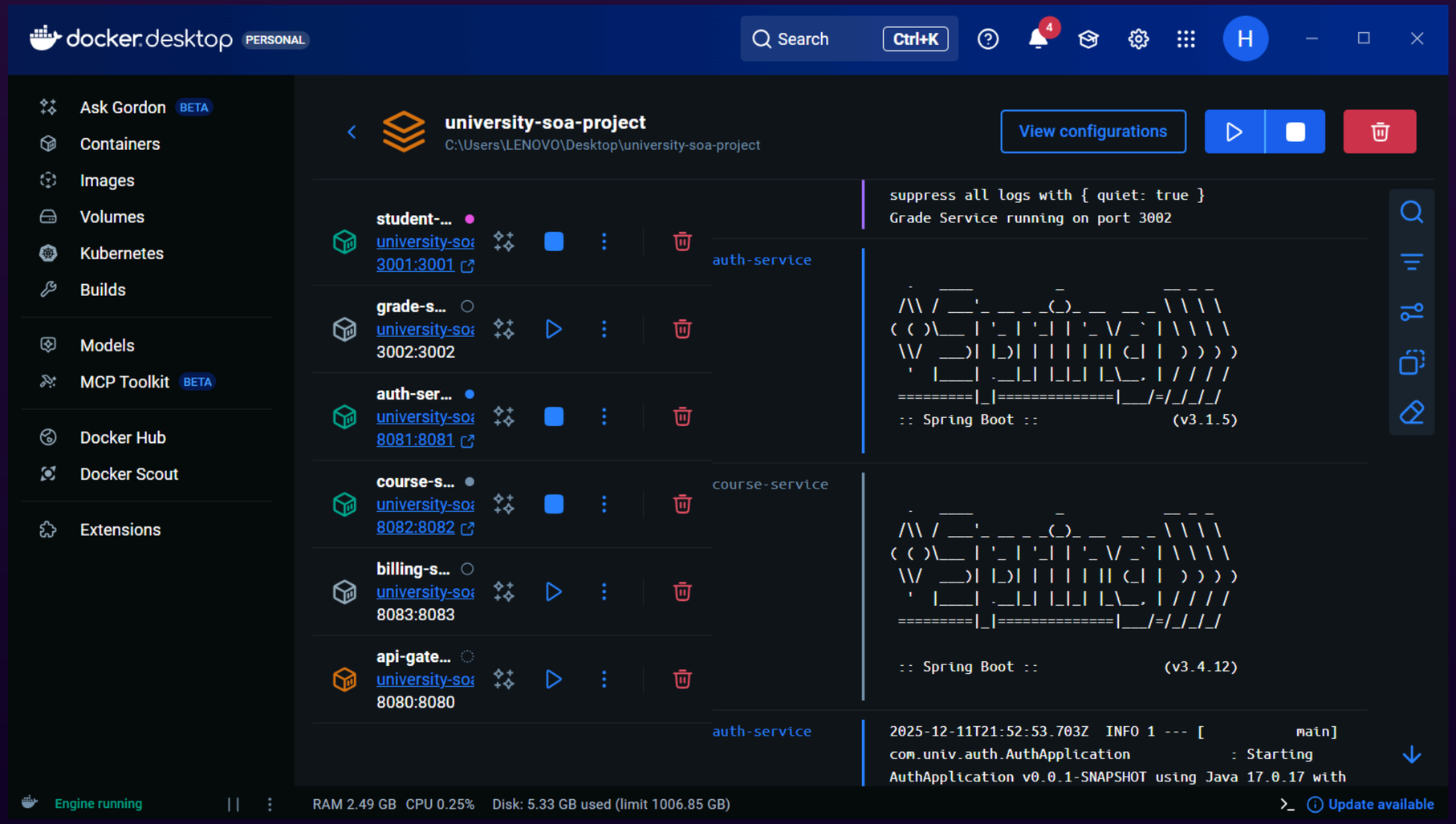
- Chaque service avec son propre **Dockerfile**
- Docker Compose pour orchestration

Réseau interne Docker

Déploiement Docker Dans Docker Desktop



Déploiement Docker Dans Docker Desktop



Avantages de l'Architecture SOA

Évolutivité

- Services indépendants pouvant évoluer séparément sans impact sur le système global

Maintenabilité

- Code modulaire plus facile à maintenir, tester et déboguer

Interopérabilité

- Technologies hétérogènes (Java, Node.js) qui communiquent via standards

Scalabilité

- Chaque service peut être scalé indépendamment selon les besoins

Sécurité

- Authentification centralisée et protection par JWT de tous les endpoints

Déploiement

- Conteneurisation facilitant le déploiement et la gestion des environnements

Conclusion

- Ce projet a permis de réaliser une architecture SOA sécurisée et scalable, intégrant des services REST et SOAP avec une authentification basée sur JWT. L’interopérabilité entre Spring Boot et Node.js ainsi que le déploiement via Docker ont assuré flexibilité et portabilité.

Merci pour votre attention !