**Kiến trúc microservices** (Microservices Architecture) là một kiểu kiến trúc phần mềm trong đó **ứng dụng được chia thành một tập hợp các dịch vụ nhỏ, độc lập**, mỗi dịch vụ đảm nhận một chức năng cụ thể trong toàn bộ hệ thống. Các dịch vụ này **giao tiếp với nhau thông qua giao thức nhẹ, thường là HTTP (REST API), gRPC hoặc messaging queue**.

**🔧 Đặc điểm chính của kiến trúc microservices**

1. **Tách biệt thành các dịch vụ độc lập**: Mỗi service là một đơn vị triển khai và vận hành riêng biệt.
2. **Triển khai độc lập**: Có thể build, deploy, scale từng service riêng lẻ mà không ảnh hưởng đến toàn hệ thống.
3. **Quản lý dữ liệu riêng biệt**: Mỗi service có thể có **database riêng**, không chia sẻ trực tiếp schema.
4. **Giao tiếp qua API**: Các service giao tiếp với nhau qua HTTP REST, gRPC hoặc message broker như RabbitMQ, Kafka.
5. **Khả năng mở rộng linh hoạt**: Có thể scale theo chiều ngang từng service theo nhu cầu thực tế.
6. **Tổ chức theo domain (DDD)**: Mỗi service tương ứng với một **business domain** (ví dụ: User Service, Order Service, Payment Service).
7. **Đa dạng công nghệ**: Mỗi service có thể sử dụng công nghệ, ngôn ngữ lập trình khác nhau (polyglot).

**⚙️ So sánh với kiến trúc Monolithic (nguyên khối)**

| **Tiêu chí** | **Monolithic** | **Microservices** |
| --- | --- | --- |
| Cấu trúc | Một khối duy nhất | Nhiều service nhỏ |
| Triển khai | Deploy toàn bộ ứng dụng | Deploy từng service |
| Mở rộng | Scale toàn ứng dụng | Scale theo từng service |
| Sửa lỗi | Dễ gây lỗi toàn hệ thống | Chỉ ảnh hưởng đến 1 service |
| Quản lý | Dễ khi hệ thống nhỏ | Phức tạp hơn khi nhiều service |
| Phù hợp với | Dự án nhỏ/trung bình | Dự án lớn, nhiều team |

**📦 Ví dụ kiến trúc microservices cho một hệ thống bán hàng online:**

* User Service: quản lý đăng ký, đăng nhập, thông tin người dùng.
* Product Service: quản lý sản phẩm, danh mục.
* Order Service: xử lý đơn hàng.
* Inventory Service: kiểm tra, cập nhật tồn kho.
* Payment Service: xử lý thanh toán.
* Notification Service: gửi email, SMS, push notification.

**🧩 Lợi ích**

* Dễ mở rộng và bảo trì
* Tăng tính chịu lỗi (fault-tolerant)
* Phát triển song song bởi nhiều nhóm
* Hỗ trợ CI/CD hiệu quả

**🚧 Thách thức**

* Quản lý nhiều service phức tạp hơn
* Cần giải quyết giao tiếp giữa các service (network, failover)
* Theo dõi (monitoring), logging, bảo mật phức tạp
* Quản lý giao dịch phân tán (distributed transaction)

Nếu bạn đang làm lập trình viên, và chuẩn bị xây dựng hệ thống dùng microservices, bạn nên tìm hiểu thêm về:

* **Service Discovery** (Eureka, Consul)
* **API Gateway** (Kong, NGINX, Spring Cloud Gateway)
* **Circuit Breaker** (Resilience4j, Hystrix)
* **Containerization & Orchestration** (Docker, Kubernetes)
* **Event-driven architecture** (Kafka, RabbitMQ)

Bạn muốn mình vẽ sơ đồ minh họa kiến trúc microservices không?

**Bạn đã nói:**

bạn có thể làm Database cho các ví dụ này không?

**ChatGPT đã nói:**

Chắc chắn rồi! Dưới đây là thiết kế **Database (DB) mẫu cho kiến trúc Microservices** với ví dụ hệ thống bán hàng online gồm các dịch vụ:

**🧩 Tổng quan các Service và Database riêng:**

| **Service** | **Mô tả** | **Database riêng** |
| --- | --- | --- |
| User Service | Quản lý người dùng, đăng ký, đăng nhập | user\_db |
| Product Service | Quản lý sản phẩm, danh mục | product\_db |
| Order Service | Quản lý đơn hàng, chi tiết đơn | order\_db |
| Inventory Service | Quản lý tồn kho sản phẩm | inventory\_db |
| Payment Service | Quản lý thanh toán | payment\_db |
| Notification Service | Gửi email, SMS... | notification\_db |

**1. 📂 User Service – user\_db**

sql

Sao chépChỉnh sửa

-- user\_db

CREATE TABLE users (

id UUID PRIMARY KEY,

email VARCHAR(100) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

full\_name VARCHAR(100),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**2. 🛒 Product Service – product\_db**

sql

Sao chépChỉnh sửa

-- product\_db

CREATE TABLE categories (

id UUID PRIMARY KEY,

name VARCHAR(100) NOT NULL

);

CREATE TABLE products (

id UUID PRIMARY KEY,

name VARCHAR(100) NOT NULL,

description TEXT,

price DECIMAL(10, 2) NOT NULL,

category\_id UUID,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (category\_id) REFERENCES categories(id)

);

**3. 🧾 Order Service – order\_db**

sql

Sao chépChỉnh sửa

-- order\_db

CREATE TABLE orders (

id UUID PRIMARY KEY,

user\_id UUID NOT NULL,

total\_amount DECIMAL(10, 2),

status VARCHAR(20), -- pending, completed, cancelled

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE order\_items (

id UUID PRIMARY KEY,

order\_id UUID NOT NULL,

product\_id UUID NOT NULL,

quantity INT NOT NULL,

price DECIMAL(10,2) NOT NULL,

FOREIGN KEY (order\_id) REFERENCES orders(id)

);

user\_id và product\_id là dữ liệu tham khảo từ services khác (phải fetch qua API, không join trực tiếp).

**4. 📦 Inventory Service – inventory\_db**

sql

Sao chépChỉnh sửa

-- inventory\_db

CREATE TABLE inventory (

product\_id UUID PRIMARY KEY,

quantity INT NOT NULL,

last\_updated TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**5. 💳 Payment Service – payment\_db**

sql

Sao chépChỉnh sửa

-- payment\_db

CREATE TABLE payments (

id UUID PRIMARY KEY,

order\_id UUID NOT NULL,

payment\_method VARCHAR(50),

status VARCHAR(20), -- paid, failed, refunded

amount DECIMAL(10, 2),

paid\_at TIMESTAMP

);

**6. 📧 Notification Service – notification\_db**

sql

Sao chépChỉnh sửa

-- notification\_db

CREATE TABLE notifications (

id UUID PRIMARY KEY,

user\_id UUID NOT NULL,

type VARCHAR(20), -- email, sms, push

content TEXT,

status VARCHAR(20), -- sent, failed, pending

sent\_at TIMESTAMP

);

**🧠 Một số lưu ý:**

* **Không có JOIN giữa các DB** → giao tiếp giữa service qua API/gRPC hoặc Event (Kafka, RabbitMQ).
* Các khóa ngoại chỉ mang tính **logic (soft link)**, không enforced bởi DB.
* Mỗi service có thể dùng **PostgreSQL, MySQL, MongoDB, v.v.** tùy yêu cầu.
* Các bảng nên có UUID để đảm bảo **tính toàn cục và không trùng lặp** giữa các service.

Bạn muốn mình vẽ sơ đồ ERD cho từng service hoặc toàn hệ thống không? Hay muốn file .sql export?

**Bạn đã nói:**

vậy làm sao để có UUID, nó là tự động ư? ví dụ trong mysql

**ChatGPT đã nói:**

Rất hay! UUID (Universally Unique Identifier) là một cách tạo ID duy nhất toàn cục (unique across systems), **thường được dùng trong kiến trúc microservices** vì:

* Tránh trùng ID giữa các service dùng database khác nhau.
* Không cần phụ thuộc auto-increment.
* Hữu ích cho hệ thống phân tán.

**✅ Cách tạo UUID trong MySQL**

**📌 1. Tạo UUID tự động khi insert**

MySQL hỗ trợ hàm UUID() để sinh UUID version 1 (hoặc version 4 tùy phiên bản).

**✅ Cách dùng:**

sql

Sao chépChỉnh sửa

INSERT INTO users (id, email, password\_hash)

VALUES (UUID(), 'abc@example.com', 'hashed\_password');

Trong đó UUID() sẽ tạo giá trị như: 550e8400-e29b-41d4-a716-446655440000

**📌 2. Đặt giá trị mặc định UUID khi tạo bảng**

Không thể dùng DEFAULT UUID() trực tiếp trong MySQL 5.x hoặc 8.x (vì UUID() không phải hàm **deterministic**). Tuy nhiên, có thể xử lý ở **application layer** (PHP, Node.js, Java, Python…) hoặc dùng trigger.

**✅ Cách dùng Trigger (nếu muốn auto UUID):**

sql

Sao chépChỉnh sửa

CREATE TABLE users (

id CHAR(36) PRIMARY KEY,

email VARCHAR(100),

password\_hash VARCHAR(255)

);

DELIMITER $$

CREATE TRIGGER before\_insert\_user

BEFORE INSERT ON users

FOR EACH ROW

BEGIN

IF NEW.id IS NULL THEN

SET NEW.id = UUID();

END IF;

END$$

DELIMITER ;

Khi insert không truyền id, trigger sẽ tự tạo UUID.

**📌 3. Dùng trong ứng dụng (khuyến nghị)**

Phổ biến hơn là sinh UUID từ app, vì:

* Bạn có thể dùng thư viện tạo UUID v4 (ngẫu nhiên) trong hầu hết ngôn ngữ:
  + Node.js: uuid package
  + Python: uuid.uuid4()
  + Java: UUID.randomUUID()
  + PHP: Ramsey\Uuid package

**Ví dụ trong Node.js:**

js

Sao chépChỉnh sửa

const { v4: uuidv4 } = require('uuid');

const id = uuidv4(); // 'c9bf9e57-1685-4c89-bafb-ff5af830be8a'

**📌 Cấu trúc cột UUID trong MySQL**

* Nếu dùng dạng chuỗi → dùng CHAR(36) hoặc VARCHAR(36)
* Nếu muốn tối ưu hơn → lưu dạng **BINARY(16)**, nhưng cần encode/decode khi dùng

sql

Sao chépChỉnh sửa

-- Đơn giản (dễ debug)

id CHAR(36)

-- Tối ưu không gian

id BINARY(16)

**💡 Tóm lại:**

| **Cách tạo UUID** | **Ưu điểm** | **Nhược điểm** |
| --- | --- | --- |
| Dùng UUID() trực tiếp | Nhanh, dễ dùng | Không đặt làm DEFAULT được |
| Dùng Trigger | Tự động hóa ở DB | Cần thêm trigger |
| Dùng từ ứng dụng | Linh hoạt, sinh UUID v4 | Logic tách khỏi DB |
| Dùng BINARY(16) | Tối ưu dung lượng | Phức tạp hơn khi thao tác |

Bạn muốn mình giúp tạo script SQL với UUID auto theo cách nào? Trigger, app hay UUID trực tiếp?

**Bạn đã nói:**

"- Tìm hiểu kiến trúc Microservices - Tìm hiểu Spring Boot, RESTful APIs và MySQL cho mỗi vi dịch vụ. - Tìm hiểu cách sử dụng RabbitMQ hoặc Kafka để giao tiếp bất đồng bộ - Tìm hiểu Eureka để cung cấp Discovery Service cho các vi dịch vụ - Tìm hiểu Spring Cloud Gateway để làm điểm truy cập duy nhất cho người dùng" với các công nghệ thế này thì tôi cần làm gì để tạo một project code?

**ChatGPT đã nói:**

Rất tốt! Bạn đang có định hướng triển khai một **hệ thống Microservices bằng Spring Boot** với các công nghệ chuẩn:

**✅ Tổng kết nhanh stack công nghệ:**

| **Thành phần** | **Công nghệ** |
| --- | --- |
| Vi dịch vụ (microservices) | Spring Boot + RESTful API + MySQL |
| Giao tiếp bất đồng bộ | RabbitMQ hoặc Kafka |
| Service Discovery | Eureka (Spring Cloud Netflix) |
| API Gateway | Spring Cloud Gateway |
| Giao tiếp nội bộ (synchronous) | REST/gRPC (Spring RestTemplate/WebClient) |
| Cấu hình tập trung (khuyến nghị thêm) | Spring Cloud Config Server |

**📌 Các bước cụ thể để bắt đầu tạo một project microservices**

**✅ Bước 1: Xác định các vi dịch vụ chính**

Ví dụ hệ thống bán hàng:

* user-service (quản lý người dùng)
* product-service (quản lý sản phẩm)
* order-service (xử lý đơn hàng)
* inventory-service (quản lý tồn kho)
* payment-service (xử lý thanh toán)
* notification-service (gửi email/SMS)
* api-gateway
* discovery-service (Eureka server)

**✅ Bước 2: Tạo project code**

Bạn có thể dùng Spring Initializr để tạo từng vi dịch vụ, hoặc dùng công cụ như Maven multi-module project (nếu muốn quản lý tập trung).

**🛠 Mỗi service cần:**

* Spring Web
* Spring Data JPA
* MySQL Driver
* Spring Boot DevTools
* Spring Cloud (Eureka Client, Config nếu cần)
* Kafka/RabbitMQ (tuỳ)

**✅ Bước 3: Cấu hình discovery-service (Eureka Server)**

Tạo riêng 1 project discovery-service:

properties

Sao chépChỉnh sửa

# application.properties

server.port=8761

spring.application.name=discovery-service

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

Thêm dependency: spring-cloud-starter-netflix-eureka-server

java

Sao chépChỉnh sửa

@EnableEurekaServer

@SpringBootApplication

public class DiscoveryServiceApplication {

public static void main(String[] args) {

SpringApplication.run(DiscoveryServiceApplication.class, args);

}

}

**✅ Bước 4: Tạo các Microservices và cấu hình kết nối Eureka + DB**

properties

Sao chépChỉnh sửa

# application.properties for user-service

server.port=8081

spring.application.name=user-service

# MySQL

spring.datasource.url=jdbc:mysql://localhost:3306/user\_db

spring.datasource.username=root

spring.datasource.password=1234

# Eureka

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

Đừng quên thêm: @EnableEurekaClient hoặc chỉ dùng spring-cloud-starter-netflix-eureka-client là đủ.

**✅ Bước 5: Tạo API Gateway (api-gateway)**

Tạo project api-gateway:

properties

Sao chépChỉnh sửa

server.port=8080

spring.application.name=api-gateway

# Eureka

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

# Route config (nếu dùng yml)

spring:

cloud:

gateway:

routes:

- id: user-service

uri: lb://user-service

predicates:

- Path=/users/\*\*

- id: order-service

uri: lb://order-service

predicates:

- Path=/orders/\*\*

Thêm dependency: spring-cloud-starter-gateway, spring-cloud-starter-netflix-eureka-client

**✅ Bước 6: Giao tiếp giữa các services**

**🔁 Synchronous: dùng RestTemplate hoặc WebClient**

java

Sao chépChỉnh sửa

// Gọi từ order-service sang product-service

@RestTemplate restTemplate = new RestTemplate();

ProductDTO product = restTemplate.getForObject("http://product-service/products/{id}", ProductDTO.class, id);

**🔁 Asynchronous: dùng RabbitMQ hoặc Kafka**

* Spring Boot hỗ trợ cả hai.
* Khuyên dùng **Kafka** nếu hệ thống lớn, có nhu cầu lưu log hay event-driven.
* Khuyên dùng **RabbitMQ** nếu bạn cần xử lý message đơn giản, dễ cấu hình.

**✅ Bước 7: Dockerize từng service (nếu cần)**

Tạo Dockerfile cho từng service và viết docker-compose.yml để chạy:

yaml

Sao chépChỉnh sửa

services:

discovery-service:

build: ./discovery-service

ports:

- "8761:8761"

user-service:

build: ./user-service

ports:

- "8081:8081"

depends\_on:

- discovery-service

**✅ Bước 8 (khuyên dùng): Config Server (quản lý cấu hình tập trung)**

Tạo config-server, các services sẽ lấy cấu hình từ Git hoặc file thay vì hardcode application.properties.

**📌 Kết quả sau khi hoàn tất:**

scss

Sao chépChỉnh sửa

[Spring Boot Service Layer]

├── user-service

├── product-service

├── order-service

├── inventory-service

├── payment-service

├── notification-service

├── api-gateway

└── discovery-service (Eureka)

**🔧 Bạn cần gì tiếp theo?**

Bạn có thể yêu cầu:

* 📁 Tạo cấu trúc project mẫu (zip/Maven)
* 🧪 Code API CRUD ví dụ cho một service
* 🐇 Cách tích hợp RabbitMQ/Kafka trong Spring Boot
* 🐳 Docker Compose để chạy toàn bộ hệ thống

Bạn muốn mình tạo bộ project mẫu user-service đầu tiên cho bạn không?

**Bạn đã nói:**

có cần cài java không?

**ChatGPT đã nói:**

**Có**, bạn **bắt buộc phải cài Java** nếu muốn phát triển hoặc chạy project Spring Boot, đặc biệt là kiến trúc microservices như bạn đang hướng đến.

**✅ Vì sao cần cài Java?**

Spring Boot được xây dựng trên nền tảng Java, nên:

* Compilation (biên dịch) cần JDK.
* Spring Boot chạy trên JVM (Java Virtual Machine).
* Maven/Gradle (các công cụ build project) đều cần Java.

**📌 Cần cài gì?**

**1. ✅ JDK (Java Development Kit)**

* Phiên bản đề xuất: **JDK 17** (Spring Boot 3.x yêu cầu Java 17 trở lên)
* Tải tại:  
  🔗 https://adoptium.net/ (gợi ý: Temurin JDK)  
  hoặc  
  🔗 <https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>

❗ **Chỉ JRE là không đủ**, bạn cần JDK để build + run Spring Boot app.

**2. ✅ Kiểm tra sau khi cài**

Mở terminal/cmd và chạy:

bash

Sao chépChỉnh sửa

java -version

Kết quả mong muốn:

nginx

Sao chépChỉnh sửa

java version "17.x.x"

**❓ Cần IDE không?**

Không bắt buộc, nhưng nên dùng một trong các IDE sau để phát triển nhanh:

| **IDE** | **Gợi ý** |
| --- | --- |
| IntelliJ IDEA | Phổ biến nhất, bản Community miễn phí |
| Eclipse | Miễn phí, nhiều plugin |
| VS Code | Nhẹ, cần cài Java Extension Pack |

**✅ Tổng kết**

| **Công cụ** | **Bắt buộc** | **Ghi chú** |
| --- | --- | --- |
| JDK 17+ | ✅ | Chạy Spring Boot |
| Maven/Gradle | ✅ | Build project |
| IDE | ❌ nhưng nên | Tăng tốc lập trình |

Bạn muốn mình gửi link cài chi tiết hoặc file Docker nếu không muốn cài Java trực tiếp trên máy không?

**Bạn đã nói:**

java version "1.8.0\_461" Java(TM) SE Runtime Environment (build 1.8.0\_461-b11) Java HotSpot(TM) 64-Bit Server VM (build 25.461-b11, mixed mode) tôi đang có thế này

**ChatGPT đã nói:**

Bạn đang dùng **Java 8 (1.8.0\_461)**, và **nó không còn đủ** để chạy **Spring Boot phiên bản mới** như:

* 🔴 **Spring Boot 3.x** (hiện tại) → yêu cầu **Java 17 trở lên**
* 🟡 **Spring Boot 2.7.x** → vẫn chạy được với **Java 8**, nhưng đã gần như "end of life"

**✅ Bạn có 2 lựa chọn:**

**🟢 1. Nâng cấp lên JDK 17+ (Khuyến nghị)**

Để dùng:

* Spring Boot 3.x (mới, ổn định, hiện đại)
* Spring Cloud mới nhất
* Các tính năng Java mới như var, records, switch pattern matching, v.v.

**👉 Cách làm:**

* Gỡ Java 8 hoặc cài song song Java 17
* Tải tại: https://adoptium.net/en-GB/temurin/releases/?version=17  
  (Chọn Windows x64 Installer nếu bạn dùng Windows)

Sau đó kiểm tra:

bash

Sao chépChỉnh sửa

java -version

Phải thấy kết quả như:

bash

Sao chépChỉnh sửa

java version "17.0.x"

**🟡 2. Giữ Java 8 và dùng Spring Boot 2.7.x (Tạm ổn)**

Nếu bạn chưa thể nâng cấp thì vẫn có thể dùng Spring Boot 2.7.x và Spring Cloud 2021.x, nhưng sẽ:

* Không chạy được các bản Spring Boot mới nhất
* Có thể không tương thích Kafka/RabbitMQ mới
* Ít được cộng đồng hỗ trợ về sau

**✅ Lời khuyên cho dự án Microservices:**

| **Yêu cầu** | **Java đề xuất** |
| --- | --- |
| Dự án mới, học hoặc production | **Java 17 hoặc 21** |
| Dự án cũ, Spring Boot 2.x | Java 8 (tạm được) |

Bạn muốn mình hướng dẫn cụ thể cách **gỡ Java 8 và cài Java 17**, hoặc **cấu hình máy dùng 2 version song song** không?

**Bạn đã nói:**

giúp tôi gỡ Java 8 và cài Java 17

**ChatGPT đã nói:**

Tuyệt! Mình sẽ hướng dẫn bạn cách **gỡ Java 8** và **cài Java 17** trên **Windows**, một cách sạch sẽ và chính xác.

**🧼 BƯỚC 1: GỠ JAVA 8**

**📌 Cách gỡ cài đặt Java 8:**

1. Mở **Control Panel** → Programs → Programs and Features
2. Tìm mục có tên:

mathematica

Sao chépChỉnh sửa

Java 8 Update 461

Java SE Runtime Environment 8...

1. Chọn và nhấn **Uninstall**

📝 Có thể bạn sẽ thấy 2 mục: JRE và JDK, hãy gỡ cả 2 nếu có.

**📥 BƯỚC 2: TẢI & CÀI JAVA 17**

**🔗 Link tải đề xuất (Temurin OpenJDK – uy tín, nhẹ, không cần đăng ký):**

👉 https://adoptium.net/en-GB/temurin/releases/?version=17

**Cách tải:**

* Chọn:
  + Version: **17 (LTS)**
  + Operating system: **Windows**
  + Architecture: **x64**
  + Package type: **Installer**
* Nhấn **Download**

**✅ Cài đặt:**

1. Chạy file .msi vừa tải
2. Chọn **"Set JAVA\_HOME"** (nếu có tùy chọn)
3. Cài như phần mềm bình thường (Next → Install)

**🧪 BƯỚC 3: KIỂM TRA LẠI JAVA**

**Mở Command Prompt (CMD) và nhập:**

bash

Sao chépChỉnh sửa

java -version

✅ Kết quả mong muốn:

mathematica

Sao chépChỉnh sửa

java version "17.0.x"

OpenJDK Runtime Environment Temurin-17...

**🔧 (Tùy chọn) Kiểm tra biến môi trường**

Vào **System Properties** → Advanced → Environment Variables:

**Kiểm tra:**

* JAVA\_HOME → trỏ đến thư mục như: C:\Program Files\Eclipse Adoptium\jdk-17.x.x
* Path → phải có: %JAVA\_HOME%\bin

Nếu bạn chưa thấy, hãy **tạo thủ công** biến môi trường:

| **Tên biến** | **Giá trị** |
| --- | --- |
| JAVA\_HOME | C:\Program Files\Eclipse Adoptium\jdk-17.0.x |
| Path | %JAVA\_HOME%\bin (nằm trong danh sách) |

**✅ XONG! Bạn đã có Java 17 để chạy Spring Boot 3**

Bạn muốn mình giúp tạo project Spring Boot 3 test thử với Java 17 không?

**Bạn đã nói:**

tôi vừa thống nhất lại với nhóm rằng dùng java 21

**ChatGPT đã nói:**

Tuyệt vời! Việc nhóm bạn quyết định dùng **Java 21 (LTS)** là **rất sáng suốt** – vì:

* ✅ Là bản **Long-Term Support (LTS)** mới nhất (được hỗ trợ lâu dài)
* ✅ Tương thích hoàn toàn với **Spring Boot 3.2+**
* ✅ Có nhiều tính năng hiện đại: **record**, **sealed class**, **pattern matching**, **virtual threads (preview)**

**🛠 Hướng dẫn CÀI ĐẶT JAVA 21 trên Windows**

**🔗 Bước 1: Tải JDK 21**

👉 Truy cập: https://adoptium.net/en-GB/temurin/releases/?version=21

* Chọn:
  + **Operating System**: Windows
  + **Architecture**: x64
  + **Package Type**: Installer (.msi)

➡️ Tải file .msi về máy

**🧭 Bước 2: Cài đặt Java 21**

1. Chạy file .msi
2. Tích chọn:
   * ✅ "Set JAVA\_HOME variable"
   * ✅ "Add to PATH"
3. Bấm **Next → Install**

**🧪 Bước 3: Kiểm tra Java đã cài**

Mở Command Prompt (CMD) và gõ:

bash

Sao chépChỉnh sửa

java -version

✔️ Kết quả mong muốn:

mathematica

Sao chépChỉnh sửa

java version "21.0.x"

OpenJDK Runtime Environment Temurin-21...

**🔧 (Tùy chọn) Kiểm tra biến môi trường**

Vào Control Panel → System → Advanced system settings → Environment Variables:

| **Biến** | **Giá trị** |
| --- | --- |
| JAVA\_HOME | C:\Program Files\Eclipse Adoptium\jdk-21 |
| Path | Có %JAVA\_HOME%\bin trong danh sách |

**💡 Gợi ý dùng thêm:**

Nếu bạn muốn dùng nhiều version Java cùng lúc (ví dụ: Java 8 và Java 21), có thể cài:

* **SDKMAN** (Linux/macOS)
* [**Jabba**](https://github.com/shyiko/jabba) (Windows via WSL)
* **jEnv** (macOS)
* Hoặc dùng công cụ như **IntelliJ IDEA** để chọn version Java theo từng project

**✅ Bây giờ bạn có thể:**

* Tạo project Spring Boot 3.x
* Dùng các tính năng Java mới (record, switch mới, sealed class, v.v.)
* Viết microservices hiện đại với hiệu suất tốt

👉 Bạn muốn mình tạo giúp project Spring Boot mẫu chạy với Java 21 (kèm cấu hình Maven) để bạn khởi động nhanh không?

**Bạn đã nói:**

- Tìm hiểu kiến trúc Microservices - Tìm hiểu Spring Boot, RESTful APIs và MySQL cho mỗi vi dịch vụ. - Tìm hiểu cách sử dụng RabbitMQ hoặc Kafka để giao tiếp bất đồng bộ - Tìm hiểu Eureka để cung cấp Discovery Service cho các vi dịch vụ - Tìm hiểu Spring Cloud Gateway để làm điểm truy cập duy nhất cho người dùng như đã đề cập ở trên, đây sẽ là những gì tôi dùng trong một project code backend service, hãy chỉ tôi tạo project đúng chuẩn những yêu cầu này nhé

**ChatGPT đã nói:**

Đã cập nhật bộ nhớ đã lưu

Tuyệt! Mình sẽ hướng dẫn bạn tạo **project backend microservices** đầy đủ theo chuẩn yêu cầu bạn đã thống nhất, gồm:

**🎯 Mục tiêu hệ thống**

Một hệ thống backend chia nhỏ thành nhiều vi dịch vụ, mỗi dịch vụ:

* Viết bằng **Spring Boot**
* Dùng **RESTful API** để giao tiếp đồng bộ
* Dùng **RabbitMQ hoặc Kafka** cho giao tiếp bất đồng bộ
* Dùng **MySQL** riêng cho mỗi service
* Dùng **Eureka** để quản lý Discovery
* Có **API Gateway (Spring Cloud Gateway)** làm điểm vào duy nhất
* Tất cả chạy trên **Java 21**

**📁 Cấu trúc tổng thể dự án**

sql

Sao chépChỉnh sửa

microservices-project/

├── api-gateway/

├── discovery-service/

├── user-service/

├── product-service/

├── order-service/

├── inventory-service/

├── notification-service/

├── docker-compose.yml

└── shared-libraries/ (tùy chọn: DTO chung, config, constants...)

**🚧 CÁC BƯỚC CHI TIẾT ĐỂ TẠO PROJECT**

**🔹 Bước 1: Tạo từng project với Spring Initializr**

Vào https://start.spring.io/ và chọn:

* **Java 21**
* **Maven**
* **Spring Boot 3.2.x**
* **Group**: com.example
* **Artifact**: user-service, product-service, ...
* **Dependencies** (cho mỗi service):
  + Spring Web
  + Spring Data JPA
  + MySQL Driver
  + Spring Boot DevTools
  + Spring Cloud Discovery Client
  + Spring for RabbitMQ (hoặc Kafka, nếu dùng Kafka)

Tạo riêng một project discovery-service và một api-gateway.

**🔹 Bước 2: Cấu hình discovery-service (Eureka Server)**

* Thêm dependency:
  + Spring Cloud Netflix Eureka Server
* Cấu hình trong application.yml:

yaml

Sao chépChỉnh sửa

server:

port: 8761

spring:

application:

name: discovery-service

eureka:

client:

register-with-eureka: false

fetch-registry: false

* Bật Eureka Server:

java

Sao chépChỉnh sửa

@SpringBootApplication

@EnableEurekaServer

public class DiscoveryServiceApplication {

public static void main(String[] args) {

SpringApplication.run(DiscoveryServiceApplication.class, args);

}

}

**🔹 Bước 3: Cấu hình các service khác để kết nối Eureka**

Trong application.yml mỗi service:

yaml

Sao chépChỉnh sửa

spring:

application:

name: user-service

datasource:

url: jdbc:mysql://localhost:3306/user\_db

username: root

password: 1234

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka

server:

port: 8081

**🔹 Bước 4: Cấu hình api-gateway**

Tạo api-gateway với các dependencies:

* Spring Cloud Gateway
* Eureka Client

yaml

Sao chépChỉnh sửa

spring:

application:

name: api-gateway

cloud:

gateway:

routes:

- id: user-service

uri: lb://user-service

predicates:

- Path=/users/\*\*

- id: product-service

uri: lb://product-service

predicates:

- Path=/products/\*\*

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka

server:

port: 8080

**🔹 Bước 5: Giao tiếp bất đồng bộ với RabbitMQ hoặc Kafka**

Tùy chọn 1: **RabbitMQ** (nhẹ, dễ dùng):

* Thêm dependency Spring for RabbitMQ
* Cấu hình:

yaml

Sao chépChỉnh sửa

spring:

rabbitmq:

host: localhost

port: 5672

username: guest

password: guest

Tùy chọn 2: **Kafka** (nên dùng khi hệ thống lớn, cần scale log):

yaml

Sao chépChỉnh sửa

spring:

kafka:

bootstrap-servers: localhost:9092

Mỗi service publish hoặc consume tùy vai trò:

* order-service publish event "order-created"
* inventory-service consume để giảm số lượng tồn kho
* notification-service consume để gửi email

**🔹 Bước 6: Docker hóa hệ thống**

Tạo Dockerfile cho từng service + docker-compose.yml chứa:

* Các service
* MySQL (mỗi DB)
* RabbitMQ / Kafka
* Eureka
* API Gateway

**✅ Tổng kết công nghệ trong project của bạn**

| **Thành phần** | **Công nghệ sử dụng** |
| --- | --- |
| Ngôn ngữ | Java 21 |
| Framework | Spring Boot 3.2+ |
| API | RESTful |
| DB | MySQL (mỗi service riêng) |
| Message Queue | RabbitMQ hoặc Kafka |
| Discovery | Eureka (Spring Cloud Netflix) |
| Gateway | Spring Cloud Gateway |
| Build Tool | Maven |
| DevOps | Docker + Docker Compose (gợi ý) |

**🚀 Bạn muốn mình giúp gì tiếp theo?**

Bạn có thể chọn một trong các mục sau để mình làm mẫu:

1. 🔧 Tạo bộ project sẵn để bạn tải về (ZIP hoặc GitHub)
2. ✏️ Viết 1 vi dịch vụ mẫu (CRUD đơn giản, ví dụ user-service)
3. 🐇 Kết nối RabbitMQ (hoặc Kafka)
4. 🐳 Viết file Docker Compose cho toàn hệ thống
5. 🔍 Sơ đồ tổng thể kiến trúc hệ thống (vẽ kiến trúc)
6. 📦 Viết shared-library dùng chung cho DTO / message event

Bạn muốn bắt đầu từ phần nào trước?

**Bạn đã nói:**

về spring boot thì tôi không thấy 3.2x

**ChatGPT đã nói:**

Đúng rồi, bạn đang nhìn thấy các phiên bản như 3.5.4, 3.4.8, hoặc 4.0.0-SNAPSHOT, **không thấy Spring Boot 3.2.x** là vì:

**❗ Trang Spring Initializr (https://start.spring.io/) luôn chỉ hiển thị một số phiên bản mới nhất và ổn định gần đây nhất, nên:**

* Các bản **Spring Boot 3.2.x** đã được thay thế bởi các bản **3.4.x** hoặc cao hơn (hiện tại là **3.5.4**).
* Nếu bạn dùng Java 21, thì **Spring Boot 3.5.4 là bản mới nhất, ổn định và tương thích tốt** → bạn **có thể chọn dùng 3.5.4 hoàn toàn bình thường**.

**✅ Spring Boot 3.5.4 dùng được không?**

* **Có**, nó là bản **ổn định** (không phải SNAPSHOT, không phải Milestone).
* Hỗ trợ tốt Java 21.
* Hỗ trợ Spring Cloud mới nhất (chẳng hạn Spring Cloud 2023.x).

**📌 Gợi ý chọn phiên bản**

| **Thành phần** | **Phiên bản đề xuất** |
| --- | --- |
| Spring Boot | ✅ **3.5.4** |
| Java | ✅ **21** |
| Spring Cloud | Tự động tương thích theo BOM của Boot 3.5.4 (thường là 2023.x) |

**🔧 Gợi ý cấu hình pom.xml để dùng Spring Boot 3.5.4**

Bạn không cần ghi rõ Spring Cloud version — chỉ cần dùng BOM:

xml

Sao chépChỉnh sửa

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.4</version>

</parent>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>2023.0.2</version> <!-- tương thích với Boot 3.5.x -->

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

**✅ Kết luận**

Bạn **cứ dùng Spring Boot 3.5.4** là đúng chuẩn hiện tại, không cần tìm Spring Boot 3.2 nữa.  
Đây là phiên bản tốt nhất hỗ trợ Java 21 và toàn bộ Spring Cloud stack (Eureka, Gateway, Kafka, ...).

👉 Bạn muốn mình giúp bạn tạo 1 project user-service mẫu bằng Spring Boot 3.5.4 + Java 21 để khởi đầu không?

**Bạn đã nói:**

hãy hướng dẫn lại từ đầu tạo project nhé

**ChatGPT đã nói:**

Dưới đây là **hướng dẫn từng bước** để tạo một project **microservice (ví dụ: user-service)** bằng **Spring Boot 3.5.4 + Java 21**, phù hợp với kiến trúc bạn đã thống nhất:

**🧱 1. Chuẩn bị công cụ**

**✅ Bắt buộc phải cài:**

| **Công cụ** | **Dùng để** |
| --- | --- |
| **Java 21 JDK** | Chạy code |
| **Maven** hoặc **Gradle** | Quản lý project |
| **IDE** (IntelliJ / VS Code / Eclipse) | Viết code |
| Docker (tuỳ chọn) | Chạy MySQL, RabbitMQ, Eureka, ... dễ dàng |

⚠️ Kiểm tra:

bash

Sao chépChỉnh sửa

java -version # phải là Java 21

mvn -v # nếu dùng Maven

**⚙️ 2. Tạo project user-service với Spring Boot**

**Cách nhanh nhất: Dùng https://start.spring.io/**

**Chọn cấu hình như sau**:

| **Mục** | **Giá trị nên chọn** |
| --- | --- |
| **Project** | Maven Project |
| **Language** | Java |
| **Spring Boot** | 3.5.4 ✅ |
| **Group** | com.example |
| **Artifact** | user-service |
| **Name** | user-service |
| **Description** | User Service |
| **Package name** | com.example.userservice |
| **Java version** | 21 ✅ |
| **Packaging** | Jar |
| **Dependencies** |  |
|  | ✅ Spring Web |
|  | ✅ Spring Boot DevTools |
|  | ✅ Spring Data JPA |
|  | ✅ MySQL Driver |
|  | ✅ Spring Boot Actuator |
|  | (sau này thêm Kafka/Rabbit, Eureka, Gateway nếu cần) |

Sau đó bấm **"GENERATE"** để tải về file .zip ⇒ giải nén vào máy.

**📁 3. Mở project trong IDE**

* Mở thư mục user-service trong IntelliJ hoặc VS Code
* Đợi nó tải dependencies (nếu dùng Maven thì chờ mvn install)

**🏗️ 4. Cấu hình cơ bản (file application.yml hoặc application.properties)**

Ví dụ dùng application.yml:

yaml

Sao chépChỉnh sửa

server:

port: 8081

spring:

datasource:

url: jdbc:mysql://localhost:3306/user\_db

username: root

password: your\_password

jpa:

hibernate:

ddl-auto: update

show-sql: true

database-platform: org.hibernate.dialect.MySQL8Dialect

Tạo database user\_db trong MySQL trước khi chạy.

**👨‍💻 5. Tạo entity, repository, controller ví dụ**

**User.java**

java

Sao chépChỉnh sửa

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

}

**UserRepository.java**

java

Sao chépChỉnh sửa

public interface UserRepository extends JpaRepository<User, Long> {}

**UserController.java**

java

Sao chépChỉnh sửa

@RestController

@RequestMapping("/api/users")

public class UserController {

@Autowired

private UserRepository userRepository;

@GetMapping

public List<User> getAllUsers() {

return userRepository.findAll();

}

}

**🚀 6. Chạy ứng dụng**

Chạy UserServiceApplication.java (file có @SpringBootApplication).

Nếu dùng Docker, có thể chạy MySQL container với:

bash

Sao chépChỉnh sửa

docker run --name mysql -e MYSQL\_ROOT\_PASSWORD=your\_password -e MYSQL\_DATABASE=user\_db -p 3306:3306 -d mysql:8

**🧩 Tiếp theo: Tạo các microservice khác**

Bạn sẽ tạo:

* user-service ✅ (xong)
* product-service
* order-service
* api-gateway
* discovery-service (Eureka)
* RabbitMQ hoặc Kafka config

Bạn muốn mình:

* ✳️ Tạo tiếp api-gateway và discovery-service
* 📦 Hoặc tạo Docker Compose cho tất cả?

Bạn chọn hướng đi nào tiếp theo?

**Bạn đã nói:**

khoan đã, như thế này đã đủ chưa, chưa có restfull apis?

**ChatGPT đã nói:**

Bạn chọn các dependencies như hình là **đúng rồi** và **đã bao gồm RESTful APIs** thông qua Spring Web. Cụ thể:

**✅ Ý nghĩa các dependencies bạn chọn:**

| **Dependency** | **Vai trò** |
| --- | --- |
| **Spring Web** | ✔️ **Đây chính là RESTful APIs** – cung cấp @RestController, @RequestMapping, xử lý HTTP, Tomcat embedded. |
| Spring Boot DevTools | Giúp reload nhanh khi thay đổi code. |
| Spring Data JPA | ORM – thao tác CSDL bằng object (Entity). |
| MySQL Driver | Kết nối đến MySQL database. |
| Eureka Discovery Client | Cho phép service đăng ký vào Discovery Server (Eureka). |
| Spring for RabbitMQ | Giao tiếp bất đồng bộ (publish/subscribe message). |

**🔁 RESTful API có chưa?**

**→ CÓ RỒI!**  
RESTful APIs được xây dựng dựa trên Spring Web, bạn chỉ cần tạo:

java

Sao chépChỉnh sửa

@RestController

@RequestMapping("/api/users")

public class UserController {

@GetMapping

public List<User> getAllUsers() {

// return danh sách user

}

@PostMapping

public User createUser(@RequestBody User user) {

// tạo mới user

}

}

Là bạn đang **viết API theo kiểu REST** rồi.

**📌 Kết luận**

✅ **Bạn đã chọn đúng dependencies cho project** user-service.

👉 Nếu bạn đồng ý, mình sẽ **hướng dẫn viết code đầy đủ cho user-service**, hoặc tiếp tục tạo eureka-server, api-gateway, v.v.

Bạn muốn mình tiếp theo phần nào?

**Bạn đã nói:**

có cần thêm gateway spring cloud routing không?

**ChatGPT đã nói:**

💡 **Câu trả lời: Có!**  
Bạn **cần thêm** dependency **"Spring Cloud Gateway"** (thường hiện tên là Gateway hoặc Spring Cloud Routing) **nếu bạn đang tạo API Gateway** trong hệ thống Microservices.

**🧭 Lý do tại sao cần Spring Cloud Gateway?**

Trong kiến trúc Microservices, **Gateway** đóng vai trò là **cổng vào duy nhất** cho toàn bộ các client:

* ✅ Chuyển tiếp request đến đúng service (routing).
* ✅ Có thể thêm các chức năng như logging, authentication, throttling.
* ✅ Giúp tách biệt frontend/client khỏi các vi dịch vụ cụ thể.

**📦 Khi nào thêm Spring Cloud Gateway?**

| **Trường hợp** | **Có thêm Spring Cloud Gateway không?** |
| --- | --- |
| Bạn đang tạo **API Gateway** | ✅ **Cần thêm** |
| Bạn đang tạo một vi dịch vụ như user-service, order-service... | ❌ **Không cần thêm** |

**📌 Kết luận**

* ❗ Nếu bạn đang tạo **API Gateway service**, thì **phải thêm** Spring Cloud Gateway (thường tên là **Gateway** hay **Spring Cloud Routing** trong Spring Initializr).
* ✅ Các service còn lại như user-service, product-service,... thì **không cần** thêm gateway.

👉 Bạn muốn tạo api-gateway bây giờ chứ? Mình có thể hướng dẫn chi tiết từng bước.

**Bạn đã nói:**

bạn hãy đọc kỹ lại nội dung yêu cầu, và xem có thiếu không, mục tiêu hiện tại của tôi là tạo các service, tuy nhiên vì có dùng gateway nên bây giờ nên làm gì?

**ChatGPT đã nói:**

Cảm ơn bạn đã cung cấp lại thông tin. Mình đã xem kỹ các dependency bạn chọn (ảnh mới nhất) và mục tiêu bạn nêu ra là:

✅ **Bạn đang tạo các service (vi dịch vụ)** trong kiến trúc Microservices  
✅ **Có sử dụng Gateway** (tức là sẽ có một service riêng để định tuyến request đến các vi dịch vụ)

**🎯 Tóm tắt định hướng**

| **Thành phần** | **Nhiệm vụ** | **Có cần thêm "Gateway"?** |
| --- | --- | --- |
| api-gateway | Chuyển tiếp request đến các vi dịch vụ khác | ✅ Có |
| user-service, order-service,... | Xử lý logic nghiệp vụ riêng lẻ (CRUD, DB, RabbitMQ...) | ❌ Không |

**📦 Phân tích danh sách dependency bạn đang có**

Ảnh bạn gửi cho thấy bạn **đang tạo một vi dịch vụ** (ví dụ user-service) chứ không phải api-gateway, vì:

**✅ Phù hợp với service:**

| **Dependency** | **Mục đích** |
| --- | --- |
| Spring Web | Xây dựng REST API (bạn có rồi ✅) |
| Spring Boot DevTools | Tự động reload khi code thay đổi (cho dev) ✅ |
| Spring Data JPA | Truy cập database theo kiểu ORM ✅ |
| MySQL Driver | Kết nối với MySQL ✅ |
| Eureka Discovery Client | Đăng ký với Eureka để các service tìm thấy nhau ✅ |
| Spring for RabbitMQ | Gửi/nhận message giữa các service (giao tiếp bất đồng bộ) ✅ |

**❗ Không nên có lúc này:**

| **Dependency** | **Lý do** |
| --- | --- |
| Gateway | Đây là phần **chỉ nên có trong dự án API Gateway**, **không nên có trong các service** như user-service, order-service,... |

**✅ Kết luận & Hướng dẫn tiếp theo**

**🎯 Hiện tại:**

* **Bạn đang tạo một vi dịch vụ** → giữ nguyên các dependency hiện tại ✅
* **XÓA Gateway ra khỏi vi dịch vụ này** ❌ (nó thuộc về service riêng tên api-gateway)

**🛠️ Tiếp theo, bạn nên:**

1. **Tạo riêng một project API Gateway**, thêm các dependency sau:
   * ✅ Spring Cloud Gateway
   * ✅ Eureka Discovery Client
   * ✅ Spring Boot DevTools (tuỳ chọn)
2. Trong project Gateway:
   * Cấu hình route chuyển tiếp request đến các vi dịch vụ (dựa trên application name đã đăng ký vào Eureka).
3. Trong mỗi vi dịch vụ (user-service, product-service,...):
   * Không cần Gateway
   * Giữ lại Eureka Client, JPA, MySQL, RabbitMQ, Web...

Bạn có muốn mình hướng dẫn cụ thể cách:

* 🧱 Tạo project cho user-service hoặc
* 🔀 Tạo api-gateway?

Bạn chỉ cần nói tên project bạn muốn làm trước, mình sẽ hướng dẫn từ đầu cho project đó.

**Bạn đã nói:**

hiện tôi đang còn lại các vi service: Service Addendum Service,Contract Service,Contract Service,Billing & Payment Service,Notification Service. hướng dẫn tôi tạo chúng

**ChatGPT đã nói:**

Đã cập nhật bộ nhớ đã lưu

Dưới đây là hướng dẫn tạo từng **microservice** một cách chuẩn chỉnh, dựa trên yêu cầu hệ thống bạn đang xây dựng:

**✅ Cấu hình ban đầu chung cho từng vi dịch vụ**

Bạn sẽ lặp lại các bước này cho **Addendum Service**, **Contract Service**, **Billing & Payment Service**, và **Notification Service**.

**1. Tạo project Spring Boot mới (dùng Spring Initializr)**

Tạo riêng từng service tại: https://start.spring.io

* **Java version**: 21
* **Packaging**: Jar
* **Dependencies**:
  + Spring Web
  + Spring Boot DevTools
  + Spring Data JPA
  + MySQL Driver
  + Eureka Discovery Client
  + Spring for RabbitMQ
  + **(Optional)** Lombok nếu bạn muốn code ngắn gọn

**2. Cấu trúc thư mục cho mỗi service**

Ví dụ: contract-service

css

Sao chépChỉnh sửa

contract-service/

├── src/

│ └── main/

│ ├── java/com/example/contract/

│ │ ├── controller/

│ │ ├── service/

│ │ ├── repository/

│ │ ├── model/

│ │ └── ContractServiceApplication.java

│ └── resources/

│ ├── application.yml

│ └── ...

├── pom.xml

**3. Cấu hình application.yml cho mỗi service**

yaml

Sao chépChỉnh sửa

server:

port: 808X # ví dụ contract là 8081, addendum là 8082

spring:

application:

name: contract-service

datasource:

url: jdbc:mysql://localhost:3306/contract\_db

username: root

password: your\_password

jpa:

hibernate:

ddl-auto: update

show-sql: true

database-platform: org.hibernate.dialect.MySQLDialect

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka/

**4. Các class cần tạo trong từng service**

**📦 model/Contract.java**

java

Sao chépChỉnh sửa

@Entity

public class Contract {

@Id @GeneratedValue

private Long id;

private String customerName;

private String contractType;

// Các thuộc tính khác

}

**📦 repository/ContractRepository.java**

java

Sao chépChỉnh sửa

public interface ContractRepository extends JpaRepository<Contract, Long> {

}

**📦 service/ContractService.java**

java

Sao chépChỉnh sửa

@Service

public class ContractService {

private final ContractRepository repository;

public ContractService(ContractRepository repository) {

this.repository = repository;

}

public List<Contract> getAllContracts() {

return repository.findAll();

}

}

**📦 controller/ContractController.java**

java

Sao chépChỉnh sửa

@RestController

@RequestMapping("/api/contracts")

public class ContractController {

private final ContractService service;

public ContractController(ContractService service) {

this.service = service;

}

@GetMapping

public List<Contract> getAll() {

return service.getAllContracts();

}

}

**5. Đăng ký với Eureka**

Bạn chỉ cần thêm annotation vào class ContractServiceApplication.java:

java

Sao chépChỉnh sửa

@EnableEurekaClient

@SpringBootApplication

public class ContractServiceApplication {

public static void main(String[] args) {

SpringApplication.run(ContractServiceApplication.class, args);

}

}

**6. Tích hợp với RabbitMQ (nếu cần gửi/nhận thông báo)**

Ở Notification Service:

java

Sao chépChỉnh sửa

@RabbitListener(queues = "notify-queue")

public void handleMessage(String message) {

// xử lý thông báo

}

Ở Billing Service:

java

Sao chépChỉnh sửa

@Autowired

private RabbitTemplate rabbitTemplate;

public void sendNotification(String message) {

rabbitTemplate.convertAndSend("notify-exchange", "notify-key", message);

}

**✅ Tiếp theo: Tạo Gateway và Eureka Server**

Sau khi tạo các vi dịch vụ, bạn cần:

* **Tạo Eureka Server** để quản lý các service
* **Tạo API Gateway** để định tuyến người dùng đến các service

Nếu bạn cần, mình sẽ hướng dẫn tiếp bước tạo Gateway và Eureka Server.

Bạn muốn mình hướng dẫn tạo từng service một hay cả bộ project cùng lúc?

**Bạn đã nói:**

project mete sẽ viết như thế nào, trước đó tôi đã có

**ChatGPT đã nói:**

Dựa trên ảnh bạn vừa gửi về Project Metadata, sau đây là cách viết mô tả **Project Metadata** chuẩn và hợp lý cho từng microservice trong hệ thống của bạn (đặc biệt là khi bạn đã có 1 số dịch vụ như user-service):

**✅ Mẫu viết Project Metadata cho các Microservices**

**1. User Service**

| **Trường** | **Giá trị điền** |
| --- | --- |
| **Group** | QLDV |
| **Artifact** | user-service |
| **Name** | user-service |
| **Description** | Manage User Service |
| **Package name** | qldv.userservice *(không nên dùng dấu -)* |
| **Packaging** | Jar |
| **Java** | 21 |

**2. Addendum Service**

| **Trường** | **Giá trị điền** |
| --- | --- |
| **Group** | QLDV |
| **Artifact** | addendum-service |
| **Name** | addendum-service |
| **Description** | Manage Contract Addendums |
| **Package name** | qldv.addendumservice |
| **Packaging** | Jar |
| **Java** | 21 |

**3. Contract Service**

| **Trường** | **Giá trị điền** |
| --- | --- |
| **Group** | QLDV |
| **Artifact** | contract-service |
| **Name** | contract-service |
| **Description** | Manage Contract Information |
| **Package name** | qldv.contractservice |
| **Packaging** | Jar |
| **Java** | 21 |

**4. Billing & Payment Service**

| **Trường** | **Giá trị điền** |
| --- | --- |
| **Group** | QLDV |
| **Artifact** | billing-payment-service |
| **Name** | billing-payment-service |
| **Description** | Handle Billing and Payments |
| **Package name** | qldv.billingpaymentservice |
| **Packaging** | Jar |
| **Java** | 21 |

**5. Notification Service**

| **Trường** | **Giá trị điền** |
| --- | --- |
| **Group** | QLDV |
| **Artifact** | notification-service |
| **Name** | notification-service |
| **Description** | Send Notifications to Users |
| **Package name** | qldv.notificationservice |
| **Packaging** | Jar |
| **Java** | 21 |

**❗ Lưu ý về Package name**

* **Tránh dùng dấu -** trong Package name vì Java không cho phép, hãy thay bằng dấu chấm (.).
* Bạn nên để theo cấu trúc: com.qldv.[service name] hoặc qldv.[service name] để đồng nhất.

Bạn muốn mình nén lại các cấu hình Spring Initializr thành từng tệp zip mẫu cho bạn tải về không? Hay bạn muốn tự tạo tay theo từng bước?

**Bạn đã nói:**

tôi đã tạo thành công và bây giờ đã có các file zip

**ChatGPT đã nói:**

Tốt lắm! 🎉 Việc bạn đã tạo thành công các project .zip cho từng microservice là một bước rất quan trọng trong việc khởi tạo kiến trúc **microservices**.

**✅ Bước tiếp theo: Giải nén & tổ chức workspace**

**1. Giải nén các file zip**

Tạo một thư mục gốc cho hệ thống, ví dụ:

sql

Sao chépChỉnh sửa

QLDV-System/

│

├── user-service/

├── contract-service/

├── addendum-service/

├── billing-payment-service/

├── notification-service/

Mỗi thư mục là một Spring Boot project độc lập.

**2. Mở bằng IDE (IntelliJ hoặc VSCode)**

* Với IntelliJ, chọn **"Open Folder"** và mở QLDV-System, IDE sẽ nhận diện các module Maven/Gradle.
* Với VSCode, có thể dùng plugin **Java Extension Pack** và mở toàn bộ folder gốc.

**✅ Bước tiếp theo bạn nên làm là:**

**🔧 1. Chạy thử từng service**

* Mỗi service nên chạy cổng riêng để tránh trùng:

properties

Sao chépChỉnh sửa

# application.properties của từng service

server.port=8081 # user-service

server.port=8082 # contract-service

server.port=8083 # addendum-service

...

**🔁 2. Thiết lập Spring Cloud Eureka (Service Discovery)**

Tạo thêm một service gọi là discovery-server để các vi dịch vụ đăng ký và tìm nhau.

**🌐 3. Thiết lập Spring Cloud Gateway**

Tạo thêm một service gọi là api-gateway để làm cổng duy nhất cho FE gọi API (routing theo path hoặc service ID).

**🗂 4. Tổ chức Git (nếu dùng)**

* Có thể dùng **Monorepo** (1 repo chứa nhiều service).
* Hoặc tách từng service thành repo riêng.

Nếu bạn muốn, mình có thể hướng dẫn tiếp:

* Tạo Discovery Server (Eureka)
* Tạo API Gateway (Spring Cloud Gateway)
* Giao tiếp giữa các service (REST / FeignClient / Kafka / RabbitMQ)
* Cấu hình DB và repository cơ bản
* Viết REST Controller đầu tiên

👉 Bạn muốn làm tiếp phần nào trước?

**Bạn đã nói:**

giúp tôi Tạo Discovery Server

**ChatGPT đã nói:**

Tuyệt vời. Dưới đây là hướng dẫn **tạo Discovery Server (Eureka Server)** bằng Spring Boot từ đầu:

**✅ Bước 1: Tạo project discovery-server**

**👉 Truy cập https://start.spring.io**

**Cấu hình:**

| **Trường** | **Giá trị** |
| --- | --- |
| Group | QLDV |
| Artifact | discovery-server |
| Name | discovery-server |
| Description | Service Registry using Eureka |
| Packaging | Jar |
| Java | 21 |

**Dependencies:**

* ✅ **Eureka Server**
* ✅ (Tùy chọn) **Spring Boot DevTools** *(nếu muốn reload nhanh)*

➡️ Nhấn **Generate** để tải .zip.

**✅ Bước 2: Giải nén và mở project trong IDE**

**✅ Bước 3: Cấu hình pom.xml**

Spring Initializr sẽ tự thêm dependency này nếu bạn chọn đúng:

xml

Sao chépChỉnh sửa

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

Bạn cũng cần thêm **Spring Cloud BOM** để tránh lỗi version:

xml

Sao chépChỉnh sửa

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>2023.0.1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

**✅ Bước 4: Tạo file cấu hình application.yml (hoặc .properties)**

yaml

Sao chépChỉnh sửa

server:

port: 8761

spring:

application:

name: discovery-server

eureka:

client:

register-with-eureka: false

fetch-registry: false

**✅ Bước 5: Bật Eureka Server trong main class**

java

Sao chépChỉnh sửa

package QLDV.discoveryserver;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

@EnableEurekaServer

public class DiscoveryServerApplication {

public static void main(String[] args) {

SpringApplication.run(DiscoveryServerApplication.class, args);

}

}

**✅ Bước 6: Chạy thử discovery-server**

Mở trình duyệt và truy cập:  
📍 http://localhost:8761

Bạn sẽ thấy giao diện web của Eureka!

**✅ Bước 7: Chuẩn bị để các microservice khác đăng ký vào**

Trong từng service (user-service, contract-service, v.v.), cần:

1. Thêm dependency spring-cloud-starter-netflix-eureka-client
2. Cấu hình eureka.client.service-url.defaultZone trong application.yml

Bạn muốn mình hướng dẫn luôn **cách cấu hình các service để kết nối tới Discovery Server** không?

**Bạn đã nói:**

khoan đã, eureka client là cho các vi service, còn eureka server cho này đúng không?

**ChatGPT đã nói:**

Đúng rồi, bạn hiểu **chính xác** 👇

**✅ Tóm tắt:**

| **Thành phần** | **Mục đích** |
| --- | --- |
| **Eureka Server** | Là **Discovery Server** – trung tâm đăng ký các microservices. |
| **Eureka Client** | Là các **vi dịch vụ (microservices)** – tự động **đăng ký vào Server**. |

**✅ Vai trò trong hệ thống của bạn:**

| **Dịch vụ** | **Loại** | **Vai trò** |
| --- | --- | --- |
| discovery-server | **Eureka Server** | Trung tâm lưu trữ danh sách các service đang hoạt động |
| user-service | **Eureka Client** | Gửi thông tin đăng ký đến discovery-server |
| contract-service | **Eureka Client** | Đăng ký với Eureka và lấy info các service khác |
| billing-payment-service | **Eureka Client** | Có thể gọi sang user-service hoặc contract-service thông qua tên |
| notification-service | **Eureka Client** | Gửi thông báo sau khi nhận dữ liệu từ các service khác |
| gateway-service (nếu có) | **Client + Gateway** | Cổng vào toàn hệ thống – định tuyến theo tên các service đã đăng ký |

Bạn muốn mình hướng dẫn luôn cách **các service đăng ký vào Eureka Server**, hay muốn setup **Spring Cloud Gateway** định tuyến trước?

**Bạn đã nói:**

ở bước 6, chạy thử, thì làm sao để chạy thử? tôi dùng vscod

**ChatGPT đã nói:**

Tốt! Bạn đang ở **bước 6: chạy thử Discovery Server (Eureka Server)** và sử dụng **VS Code**. Dưới đây là hướng dẫn chi tiết để chạy thử:

**✅ Bước 6: Chạy thử Discovery Server (Eureka Server)**

**🎯 Điều kiện trước:**

* Bạn đã tạo project discovery-server với Spring Initializr, có dependency **Eureka Server**.
* Bạn đã cấu hình đúng trong application.yml hoặc application.properties.
* Đang dùng **Java 21** và đã cài **Maven** (hoặc dùng wrapper ./mvnw).

**✅ Cách chạy trên VS Code**

**🔧 Cách 1: Dùng Terminal trong VS Code**

1. **Mở thư mục discovery-server** trong VS Code.
2. Mở Terminal (bấm Ctrl + ~ hoặc Terminal > New Terminal).
3. Gõ lệnh sau để chạy:

bash

Sao chépChỉnh sửa

./mvnw spring-boot:run

Nếu bạn dùng Windows mà lỗi ./mvnw, hãy dùng:

bash

Sao chépChỉnh sửa

mvnw.cmd spring-boot:run

**🔧 Cách 2: Chạy file DiscoveryServerApplication.java**

1. Mở file src/main/java/.../DiscoveryServerApplication.java (tên có thể khác).
2. Nếu có nút **Run | Debug** ở góc trên file → nhấn **Run**.

Nếu không có nút Run, bạn cần cài extension:

* **Java Extension Pack** (Cài trong Extensions Marketplace).
* Sau đó restart VS Code.

**✅ Kiểm tra chạy thành công**

1. Khi chạy thành công, log console sẽ in:

nginx

Sao chépChỉnh sửa

Started DiscoveryServerApplication in ... seconds

1. Mở trình duyệt truy cập:

arduino

Sao chépChỉnh sửa

http://localhost:8761/

Đây là **Eureka Dashboard**. Nếu bạn thấy một giao diện đơn giản với tiêu đề **Eureka** thì đã thành công!

**❓Vấn đề thường gặp**

| **Lỗi** | **Nguyên nhân & Cách khắc phục** |
| --- | --- |
| Port 8761 bị chiếm | Đổi port trong application.yml (server.port: 8762 chẳng hạn) |
| Không có Run | Debug button |
| Lỗi thiếu Java/Maven | Kiểm tra java -version và mvn -v hoạt động đúng chưa |

Bạn có thể gửi file application.yml nếu cần mình kiểm tra giúp.  
Muốn chạy **các vi service client** và kết nối tới Eureka sau không? Mình có thể hướng dẫn tiếp.

**Bạn đã nói:**

tôi dùng spring.application.name=discoveryserver server: port: 8762 spring: application: name: discovery-server eureka: client: register-with-eureka: false fetch-registry: false

**ChatGPT đã nói:**

Tốt, cấu hình application.yml của bạn **đã đúng chức năng của Eureka Server**, nhưng có **một lỗi nhỏ** do bạn khai báo spring.application.name **2 lần**, cụ thể:

**❌ Vấn đề:**

yaml

Sao chépChỉnh sửa

spring.application.name=discoveryserver # <-- Sai cú pháp YAML và dư thừa

Dòng này không hợp lệ (đây là cú pháp application.properties, không phải YAML) và trùng lặp với dòng đúng bên dưới:

yaml

Sao chépChỉnh sửa

spring:

application:

name: discovery-server

**✅ Cấu hình sửa lại application.yml chuẩn như sau:**

yaml

Sao chépChỉnh sửa

server:

port: 8762

spring:

application:

name: discovery-server

eureka:

client:

register-with-eureka: false

fetch-registry: false

**✅ Tiếp theo: Chạy lại Eureka Server**

1. Xóa dòng spring.application.name=discoveryserver sai.
2. Lưu file application.yml.
3. Mở terminal trong VS Code.
4. Chạy lệnh:

bash

Sao chépChỉnh sửa

./mvnw spring-boot:run

Hoặc nếu dùng Windows:

bash

Sao chépChỉnh sửa

mvnw.cmd spring-boot:run

**🔍 Kiểm tra:**

Sau khi chạy thành công, truy cập trình duyệt:

arduino

Sao chépChỉnh sửa

http://localhost:8762/

Bạn sẽ thấy giao diện như sau:

* Có tiêu đề **Eureka**.
* Một bảng (danh sách service hiện đăng ký).
* Hiện tại chưa có service nào, nên bảng sẽ rỗng.

Bạn có thể gửi ảnh màn hình nếu thấy không xuất hiện Eureka UI.  
Sẵn sàng chưa? Mình có thể hướng dẫn tiếp để tạo các **Eureka Client (vi dịch vụ)** đăng ký vào server này.

**Bạn đã nói:**

tôi đã chạy được, tiếp với bước 7 nhé

**ChatGPT đã nói:**

Tuyệt vời, bạn đã chạy thành công **Discovery Server (Eureka Server)** 🎉  
Bây giờ ta sẽ đến **Bước 7: Tạo các vi dịch vụ (Microservices) và đăng ký vào Eureka**.

**✅ Bước 7: Tạo các Microservices và đăng ký vào Eureka**

Bạn đã thống nhất có các vi dịch vụ:

1. **Addendum Service**
2. **Contract Service**
3. **Billing & Payment Service**
4. **Notification Service**

**🔧 Mỗi Microservice cần:**

* Spring Boot (Java 21)
* Spring Web
* Spring Boot DevTools (tuỳ chọn)
* Spring Data JPA + MySQL Driver
* Spring Cloud Discovery Client (Eureka Client)
* Spring Boot Actuator (giúp hiển thị thông tin trạng thái)

**🔨 A. Tạo từng service bằng Spring Initializr**

Lặp lại thao tác tạo project như bạn đã làm cho Discovery Server.

Tạo project ZIP tại: https://start.spring.io/

Ví dụ với **Contract Service**:

| **Field** | **Value** |
| --- | --- |
| Project | Maven |
| Language | Java |
| Spring Boot | 3.2.x (hoặc 3.1.x nếu chưa thấy 3.2) |
| Group | com.yourcompany |
| Artifact | contract-service |
| Name | ContractService |
| Java Version | 21 |
| Dependencies | **Spring Web**, **Spring Data JPA**, **MySQL Driver**, **Eureka Discovery Client**, **Spring Boot DevTools**, **Spring Boot Actuator** |

Tải file .zip và giải nén → mở bằng VS Code.

**⚙️ B. Cấu hình application.yml cho từng service**

Ví dụ với **Contract Service**:

yaml

Sao chépChỉnh sửa

server:

port: 8081

spring:

application:

name: contract-service

datasource:

url: jdbc:mysql://localhost:3306/contract\_db

username: root

password: your\_password

driver-class-name: com.mysql.cj.jdbc.Driver

jpa:

hibernate:

ddl-auto: update

show-sql: true

eureka:

client:

register-with-eureka: true

fetch-registry: true

service-url:

defaultZone: http://localhost:8762/eureka

⚠️ Đảm bảo MySQL đã chạy và bạn đã tạo database contract\_db

**🧩 C. Annotate main class với @EnableDiscoveryClient**

java

Sao chépChỉnh sửa

@SpringBootApplication

@EnableDiscoveryClient

public class ContractServiceApplication {

public static void main(String[] args) {

SpringApplication.run(ContractServiceApplication.class, args);

}

}

**🚀 D. Chạy thử service**

Trong terminal của VS Code, chạy lệnh:

bash

Sao chépChỉnh sửa

./mvnw spring-boot:run

Nếu dùng Windows: mvnw.cmd spring-boot:run

Sau vài giây, bạn mở lại Eureka Dashboard tại:

arduino

Sao chépChỉnh sửa

http://localhost:8762/

📌 **Bạn sẽ thấy contract-service đã hiển thị** trên trang Eureka.

**🧱 E. Lặp lại cho các service khác**

| **Service Name** | **Port** | **Database Name** |
| --- | --- | --- |
| addendum-service | 8082 | addendum\_db |
| billing-payment-service | 8083 | billing\_payment\_db |
| notification-service | 8084 | notification\_db |

**✅ Tiếp theo?**

Sau khi hoàn tất các service và đăng ký thành công vào Eureka, ta sẽ:

➡️ **Tạo RESTful API cho từng service**  
➡️ **Cấu hình RabbitMQ hoặc Kafka (giao tiếp bất đồng bộ)**  
➡️ **Thêm Spring Cloud Gateway để làm điểm truy cập duy nhất**

Bạn muốn mình hướng dẫn tiếp phần tạo **Contract Service API đầu tiên**, hay bạn tạo đủ 4 service trước rồi làm tiếp?

**Bạn đã nói:**

spring: datasource: url: jdbc:mysql://localhost:3306/contract\_db username: root password: your\_password driver-class-name: com.mysql.cj.jdbc.Driver jpa: hibernate: ddl-auto: update show-sql: true eureka: client: register-with-eureka: true fetch-registry: true service-url: defaultZone: http://localhost:8761/eureka ở bản này, tôi thấy dùng spring.datasource sẽ được, giúp tôi đổi hết phần này. ví dụ dùng server.port=8081, thay vìs erver: port: 8081

2/2

**ChatGPT đã nói:**

Chắc chắn rồi! Bạn muốn chuyển toàn bộ file cấu hình application.yml sang dùng **định dạng application.properties** thay vì YAML. Dưới đây là nội dung tương đương từ YAML sang .properties:

**✅ application.properties cho contract-service (port 8081)**

properties

Sao chépChỉnh sửa

# ===== Server =====

server.port=8081

# ===== Spring Application =====

spring.application.name=contract-service

# ===== Datasource =====

spring.datasource.url=jdbc:mysql://localhost:3306/contract\_db

spring.datasource.username=root

spring.datasource.password=your\_password

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# ===== JPA (Hibernate) =====

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

# ===== Eureka Client =====

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

✅ Bạn có thể **xoá file application.yml** nếu đã chuyển hoàn toàn sang .properties.

🎯 Sau khi chỉnh xong, chỉ cần chạy lại project bằng:

bash

Sao chépChỉnh sửa

./mvnw spring-boot:run

Nếu bạn cần template cho các service khác (port và DB khác nhau), mình có thể giúp viết luôn. Bạn muốn không?

**Bạn đã nói:**

spring.datasource.url=jdbc:mysql://localhost:3306/user\_management\_db spring.datasource.username=root spring.datasource.password=long@091103 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver tôi dùng chuỗi đúng mà 2025-07-29T13:22:20.060+07:00 INFO 18316 --- [userService] [ restartedMain] org.hibernate.orm.connections.pooling : HHH10001005: Database info: Database JDBC URL [Connecting through datasource 'HikariDataSource (HikariPool-1)'] Database driver: undefined/unknown Database version: 8.0.39 Autocommit mode: undefined/unknown Isolation level: undefined/unknown Minimum pool size: undefined/unknown Maximum pool size: undefined/unknown

**ChatGPT đã nói:**

Bạn đã cấu hình chuỗi kết nối JDBC đúng cú pháp, nhưng lỗi "Database driver: undefined/unknown" và "Autocommit mode: undefined/unknown" cho thấy Hibernate **không thể xác định rõ driver hoặc thông tin kết nối**. Lỗi này thường do **thiếu dependency** trong file pom.xml.

**✅ Kiểm tra lại pom.xml**

Bạn cần chắc chắn pom.xml đã khai báo đủ **MySQL Connector** và **Spring Data JPA**:

xml

Sao chépChỉnh sửa

<dependencies>

<!-- Spring Data JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- MySQL JDBC Driver -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Eureka Client -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

</dependencies>

**✅ Ngoài ra, kiểm tra thêm:**

**1. Đúng driver class**

properties

Sao chépChỉnh sửa

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

Bạn đã dùng đúng (cj là MySQL Connector Java mới – tốt ✅).

**2. Dùng đúng phiên bản MySQL**

Spring Boot 3.x (Java 17+) nên dùng MySQL 8.x hoặc cao hơn, bạn đang dùng 8.0.39 là ổn.

**3. Chạy thử lệnh SQL để kiểm tra DB**

Bạn có thể vào terminal chạy thử:

bash

Sao chépChỉnh sửa

mysql -u root -p

# nhập mật khẩu

use user\_management\_db;

Nếu DB hoạt động bình thường thì xác nhận là **driver thiếu** là nguyên nhân chính.

**✅ Tổng kết**

**Hành động bạn cần làm ngay:**

1. Kiểm tra và thêm đoạn mysql-connector-j vào pom.xml như trên.
2. Chạy lại lệnh: ./mvnw clean install rồi ./mvnw spring-boot:run
3. Nếu vẫn lỗi, gửi lại toàn bộ pom.xml mình sẽ kiểm tra giúp bạn.

Bạn có muốn mình giúp kiểm tra pom.xml không?

**Bạn đã nói:**

việc unknow như vậy có quan trọng không?

**ChatGPT đã nói:**

Câu trả lời ngắn gọn: **Có thể không quan trọng nếu ứng dụng vẫn hoạt động bình thường, nhưng thường là dấu hiệu của một vấn đề tiềm ẩn.**

**✅ Khi nào *"Database driver: undefined/unknown"* không quan trọng?**

* Nếu **ứng dụng kết nối được DB**, truy vấn và ghi dữ liệu **vẫn hoạt động bình thường**.
* Đây có thể là **log cảnh báo** do Hibernate không lấy được thông tin driver từ JDBC (thường là do cấu hình pool hoặc thiếu metadata từ driver).
* Không ảnh hưởng tới tính năng chính nếu bạn chỉ dùng Hibernate/JPA đơn giản.

**⚠️ Khi nào nó quan trọng và cần sửa ngay?**

* Nếu ứng dụng bị lỗi khi truy cập DB (truy vấn thất bại, hoặc không tạo được bảng).
* Nếu bạn dùng một số tính năng nâng cao như:
  + Quản lý transaction chuyên sâu.
  + Thay đổi isolation level.
  + Monitoring connection pool.
* Nếu bạn định deploy thật lên production: Cảnh báo này nên được xử lý để đảm bảo hệ thống ổn định.

**🛠 Gợi ý:**

Nếu bạn **muốn yên tâm và làm chuẩn**, nên khắc phục nó bằng cách:

* **Đảm bảo mysql-connector-j** nằm trong pom.xml.
* Đảm bảo đang dùng **Spring Boot starter JPA** chính xác.
* Kiểm tra application.properties hoặc application.yml không có cấu hình sai nào khác.

**✅ Tổng kết:**

| **Trường hợp** | **Ứng dụng chạy bình thường** | **Có ảnh hưởng nghiêm trọng** |
| --- | --- | --- |
| Dự án học tập, test | ❌ Không đáng lo | ❌ Không cần sửa ngay |
| Dự án nhóm hoặc sắp triển khai | ⚠️ Nên sửa | ✅ Cần sửa để tránh rủi ro |
| Sử dụng tính năng nâng cao DB | ✅ Cần sửa | ✅ Cần sửa ngay |

Nếu bạn chia sẻ thêm nội dung pom.xml và application.properties/yml, mình có thể giúp xác nhận toàn bộ setup đã tối ưu chưa. Bạn có muốn không?

**Bạn đã nói:**

hiện tại giúp tôi cấu hình rabibitmq

**ChatGPT đã nói:**

Để cấu hình **RabbitMQ** cho dự án Spring Boot Microservices, bạn cần thực hiện các bước sau. Mình sẽ hướng dẫn chi tiết cách:

**🔧 1. Cài đặt RabbitMQ (nếu bạn chưa có)**

**✅ Cách nhanh nhất (dành cho local):**

**Yêu cầu: Docker**

bash

Sao chépChỉnh sửa

docker run -d --hostname rabbitmq --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management

* 5672: Cổng để ứng dụng Spring kết nối.
* 15672: Giao diện web (RabbitMQ UI) tại http://localhost:15672.
* Tài khoản mặc định: guest / guest

**📦 2. Thêm dependency vào pom.xml**

xml

Sao chépChỉnh sửa

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

**⚙️ 3. Cấu hình trong application.yml hoặc application.properties**

**Ví dụ với application.yml:**

yaml

Sao chépChỉnh sửa

spring:

rabbitmq:

host: localhost

port: 5672

username: guest

password: guest

**💬 4. Tạo cấu hình Queue, Exchange, Binding**

Bạn nên tạo class cấu hình RabbitMQ:

java

Sao chépChỉnh sửa

@Configuration

public class RabbitMQConfig {

public static final String QUEUE = "notification\_queue";

public static final String EXCHANGE = "notification\_exchange";

public static final String ROUTING\_KEY = "notification\_routing\_key";

@Bean

public Queue queue() {

return new Queue(QUEUE, true);

}

@Bean

public TopicExchange exchange() {

return new TopicExchange(EXCHANGE);

}

@Bean

public Binding binding(Queue queue, TopicExchange exchange) {

return BindingBuilder.bind(queue).to(exchange).with(ROUTING\_KEY);

}

}

**📤 5. Producer (Gửi message)**

java

Sao chépChỉnh sửa

@Service

public class NotificationProducer {

private final RabbitTemplate rabbitTemplate;

public NotificationProducer(RabbitTemplate rabbitTemplate) {

this.rabbitTemplate = rabbitTemplate;

}

public void sendNotification(String message) {

rabbitTemplate.convertAndSend(

RabbitMQConfig.EXCHANGE,

RabbitMQConfig.ROUTING\_KEY,

message

);

}

}

**📥 6. Consumer (Nhận message)**

java

Sao chépChỉnh sửa

@Service

public class NotificationConsumer {

@RabbitListener(queues = RabbitMQConfig.QUEUE)

public void receiveNotification(String message) {

System.out.println("Received message: " + message);

}

}

**✅ 7. Chạy thử**

* Mở RabbitMQ UI tại http://localhost:15672 và đăng nhập.
* Chạy Producer để gửi message.
* Consumer sẽ tự động nhận message từ hàng đợi.

Bạn đang dùng VS Code và Java 21, nên có thể kiểm tra:

bash

Sao chépChỉnh sửa

./mvnw spring-boot:run

Bạn muốn mình giúp cấu hình cụ thể cho service nào trước (ví dụ NotificationService)?

**Bạn đã nói:**

tôi dùng application.properties

**ChatGPT đã nói:**

Tuyệt vời. Nếu bạn dùng application.properties, thì dưới đây là cấu hình **RabbitMQ** tương đương với file YAML trước đó:

**✅ 1. Thêm cấu hình RabbitMQ vào application.properties (áp dụng cho mỗi service có giao tiếp bằng RabbitMQ):**

properties

Sao chépChỉnh sửa

# RabbitMQ config

spring.rabbitmq.host=localhost

spring.rabbitmq.port=5672

spring.rabbitmq.username=guest

spring.rabbitmq.password=guest

**✅ 2. Tạo class cấu hình RabbitMQ (RabbitMQConfig.java)**

java

Sao chépChỉnh sửa

import org.springframework.amqp.core.\*;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class RabbitMQConfig {

public static final String QUEUE = "notification\_queue";

public static final String EXCHANGE = "notification\_exchange";

public static final String ROUTING\_KEY = "notification\_routing\_key";

@Bean

public Queue queue() {

return new Queue(QUEUE, true); // durable = true

}

@Bean

public TopicExchange exchange() {

return new TopicExchange(EXCHANGE);

}

@Bean

public Binding binding(Queue queue, TopicExchange exchange) {

return BindingBuilder

.bind(queue)

.to(exchange)

.with(ROUTING\_KEY);

}

}

**✅ 3. Gửi message – Producer (NotificationProducer.java)**

java

Sao chépChỉnh sửa

import org.springframework.amqp.rabbit.core.RabbitTemplate;

import org.springframework.stereotype.Service;

@Service

public class NotificationProducer {

private final RabbitTemplate rabbitTemplate;

public NotificationProducer(RabbitTemplate rabbitTemplate) {

this.rabbitTemplate = rabbitTemplate;

}

public void sendNotification(String message) {

rabbitTemplate.convertAndSend(

RabbitMQConfig.EXCHANGE,

RabbitMQConfig.ROUTING\_KEY,

message

);

}

}

**✅ 4. Nhận message – Consumer (NotificationConsumer.java)**

java

Sao chépChỉnh sửa

import org.springframework.amqp.rabbit.annotation.RabbitListener;

import org.springframework.stereotype.Service;

@Service

public class NotificationConsumer {

@RabbitListener(queues = RabbitMQConfig.QUEUE)

public void receiveMessage(String message) {

System.out.println("📩 Received notification: " + message);

}

}

**📌 Lưu ý quan trọng:**

* Cả Producer và Consumer phải cấu hình cùng một QUEUE, EXCHANGE, ROUTING\_KEY.
* Bạn có thể thử gửi message từ một controller để test:

java

Sao chépChỉnh sửa

@RestController

@RequestMapping("/api/notifications")

public class NotificationController {

private final NotificationProducer notificationProducer;

public NotificationController(NotificationProducer notificationProducer) {

this.notificationProducer = notificationProducer;

}

@PostMapping

public ResponseEntity<String> sendMessage(@RequestBody String message) {

notificationProducer.sendNotification(message);

return ResponseEntity.ok("Message sent to queue");

}

}

Nếu bạn cần mình giúp tạo đoạn cấu hình cho một service cụ thể (Notification, Billing, Contract, v.v.) thì gửi tên service hoặc phần bạn đang làm nhé!

**Bạn đã nói:**

khhoan đã, tôi không biết rằng tôi đã cài rabbitmq và docker hay chưa

**ChatGPT đã nói:**

Không sao — mình sẽ giúp bạn kiểm tra và nếu chưa có thì hướng dẫn cách cài đặt từ đầu. Vì bạn dùng **Windows + VSCode**, mình sẽ hướng dẫn chi tiết theo hướng này nhé:

**✅ 1. Kiểm tra bạn đã cài Docker chưa**

Mở terminal/cmd/powershell và gõ:

bash

Sao chépChỉnh sửa

docker --version

* Nếu thấy kết quả kiểu như Docker version 24.0.5, build xxx → bạn đã cài.
* Nếu báo lỗi 'docker' is not recognized → bạn **chưa cài**, xem bước bên dưới.

**✅ 2. Cài Docker (nếu chưa có)**

1. Truy cập: https://www.docker.com/products/docker-desktop/
2. Tải Docker Desktop cho Windows (bản phù hợp với Win 10/11).
3. Cài đặt, khởi động lại máy nếu được yêu cầu.
4. Sau khi cài xong, mở Docker Desktop và bật Docker (biểu tượng con cá voi ở taskbar phải **không đỏ**).

**✅ 3. Chạy RabbitMQ bằng Docker**

Sau khi có Docker, bạn có thể chạy RabbitMQ bằng lệnh sau:

bash

Sao chépChỉnh sửa

docker run -d --hostname rabbitmq-host \

--name rabbitmq \

-p 5672:5672 -p 15672:15672 \

rabbitmq:3-management

Giải thích:

* 5672: cổng giao tiếp AMQP (service dùng)
* 15672: cổng giao diện web quản lý

Sau khi chạy, bạn mở trình duyệt vào:

arduino

Sao chépChỉnh sửa

http://localhost:15672

* Username: guest
* Password: guest

**✅ 4. Kiểm tra container đang chạy**

bash

Sao chépChỉnh sửa

docker ps

Nếu thấy container tên rabbitmq đang chạy → thành công.

**✅ 5. Kiểm tra từ Spring Boot**

Khi đã có RabbitMQ, ứng dụng Spring Boot của bạn sẽ kết nối được nếu dùng cấu hình:

properties

Sao chépChỉnh sửa

spring.rabbitmq.host=localhost

spring.rabbitmq.port=5672

spring.rabbitmq.username=guest

spring.rabbitmq.password=guest

Bạn có thể gửi cho mình kết quả của:

* docker --version
* docker ps

hoặc nếu muốn, mình có thể hướng dẫn thêm cài RabbitMQ **không cần Docker** (cài trực tiếp trên Windows), tuy nhiên dùng Docker là cách đơn giản và phổ biến hơn nhiều.

**Bạn đã nói:**

ở bước 3 thì chạy trên đâu?

**ChatGPT đã nói:**

Rất tốt! Ở **bước 3**, bạn sẽ **chạy lệnh Docker** để khởi động RabbitMQ. Có 2 cách chạy tùy bạn quen dùng cái nào:

**✅ Cách 1: Mở Windows Terminal / PowerShell / CMD**

1. Nhấn Windows + S → gõ "cmd" hoặc "powershell" hoặc "terminal" rồi **chạy với quyền Administrator** (nếu được).
2. Gõ lệnh sau để chạy RabbitMQ container:

bash

Sao chépChỉnh sửa

docker run -d --hostname rabbitmq-host \

--name rabbitmq \

-p 5672:5672 -p 15672:15672 \

rabbitmq:3-management

💡 Lưu ý: Nếu bạn đã từng chạy container tên rabbitmq rồi, hãy gỡ container cũ bằng:

bash

Sao chépChỉnh sửa

docker rm -f rabbitmq

rồi chạy lại lệnh docker run phía trên.

**✅ Cách 2: Terminal bên trong VSCode**

Nếu bạn đang dùng **VSCode** và đã cài Docker Desktop, bạn có thể:

1. Mở một project bất kỳ trong VSCode.
2. Nhấn Ctrl + ~ (hoặc vào menu **Terminal → New Terminal**).
3. Gõ lại lệnh Docker như trên:

bash

Sao chépChỉnh sửa

docker run -d --hostname rabbitmq-host \

--name rabbitmq \

-p 5672:5672 -p 15672:15672 \

rabbitmq:3-management

**✅ Sau khi chạy xong**

* Truy cập: <http://localhost:15672>
* Đăng nhập:
  + **Username:** guest
  + **Password:** guest

Nếu bạn thấy RabbitMQ UI hiển thị → thành công 🎉

📌 Bạn hãy thử chạy lệnh, rồi báo lại cho mình xem có hiện lỗi gì không nhé. Nếu bạn chưa cài Docker Desktop, mình sẽ hướng dẫn lại bước cài.

**Bạn đã nói:**

Thêm Spring Cloud Gateway để làm điểm truy cập duy nhất, hiện tại giúp tôi

**ChatGPT đã nói:**

Để thêm **Spring Cloud Gateway** làm điểm truy cập duy nhất cho hệ thống Microservices của bạn, bạn cần làm các bước sau:

**🧱 1. Tạo project Spring Boot cho Gateway**

Bạn có thể dùng Spring Initializr để tạo hoặc dùng VS Code/IntelliJ như sau:

**Cấu hình:**

* **Project:** Maven
* **Language:** Java
* **Spring Boot:** chọn bản mới nhất 3.x (nếu bạn đã dùng Java 21, hãy chọn Spring Boot >= 3.2)
* **Dependencies:**
  + Spring Cloud Gateway
  + Eureka Discovery Client
  + Spring Boot DevTools (nếu thích)

👉 Đặt tên project là api-gateway chẳng hạn.

**📁 2. Cấu trúc project sau khi tạo**

Cấu trúc thư mục có thể như sau:

css

Sao chépChỉnh sửa

api-gateway/

├── src/

│ └── main/

│ ├── java/com/example/apigateway/

│ │ └── ApiGatewayApplication.java

│ └── resources/

│ └── application.properties

├── pom.xml

**⚙️ 3. Cấu hình application.properties**

Ví dụ cấu hình cơ bản:

properties

Sao chépChỉnh sửa

server.port=8080

spring.application.name=api-gateway

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

spring.cloud.gateway.discovery.locator.enabled=true

spring.cloud.gateway.discovery.locator.lower-case-service-id=true

🔁 Cấu hình trên giúp Gateway tự động định tuyến đến các dịch vụ đã đăng ký với Eureka.

**🧪 4. Ví dụ định tuyến thủ công (nếu không dùng discovery)**

Bạn có thể định nghĩa tay các route:

properties

Sao chépChỉnh sửa

spring.cloud.gateway.routes[0].id=contract-service

spring.cloud.gateway.routes[0].uri=http://localhost:8081

spring.cloud.gateway.routes[0].predicates[0]=Path=/contracts/\*\*

spring.cloud.gateway.routes[1].id=notification-service

spring.cloud.gateway.routes[1].uri=http://localhost:8083

spring.cloud.gateway.routes[1].predicates[0]=Path=/notifications/\*\*

**🚀 5. Chạy Gateway**

Bạn chạy file ApiGatewayApplication.java như một ứng dụng Spring Boot bình thường:

bash

Sao chépChỉnh sửa

./mvnw spring-boot:run

Nếu chạy thành công, bạn có thể gửi request qua cổng 8080:

* http://localhost:8080/contracts/abc sẽ chuyển đến Contract Service.
* http://localhost:8080/notifications/xzy sẽ chuyển đến Notification Service.

**📌 Lưu ý**

* Tất cả các microservices của bạn phải **đăng ký với Eureka Server** (discovery-server) trước.
* Gateway phải **chạy sau Eureka**, và sẽ định tuyến được sau 1-2 giây khi các service đã đăng ký xong.

Bạn có muốn mình **giúp tạo Gateway project chi tiết ngay bây giờ** (tạo bằng Spring Initializr hoặc VS Code) không?

**Bạn đã nói:**

Tất cả các microservices của bạn phải đăng ký với Eureka Server (discovery-server) trước. làm sao để đăng ký

**ChatGPT đã nói:**

Để **đăng ký một Microservice với Eureka Server (Discovery Server)**, bạn cần **thêm các cấu hình và dependency cần thiết** trong mỗi project vi dịch vụ. Dưới đây là hướng dẫn cụ thể:

**✅ 1. Thêm dependency Eureka Client**

Mở pom.xml của mỗi microservice và **thêm dependency sau**:

xml

Sao chépChỉnh sửa

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

**Đừng quên** thêm Spring Cloud BOM (nếu chưa có):

xml

Sao chépChỉnh sửa

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>2023.0.1</version> <!-- Phiên bản tương thích với Spring Boot 3.x -->

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

**⚙️ 2. Cấu hình application.properties (hoặc application.yml)**

Trong từng vi dịch vụ (ví dụ: contract-service, billing-service, ...), bạn thêm cấu hình sau:

properties

Sao chépChỉnh sửa

server.port=8081

spring.application.name=contract-service

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

📌 **Lưu ý:** spring.application.name phải khác nhau cho từng service!

**📌 3. Thêm annotation @EnableEurekaClient (không bắt buộc)**

Với Spring Boot 2.1 trở lên, bạn **không bắt buộc** thêm @EnableEurekaClient, nhưng nếu muốn rõ ràng:

java

Sao chépChỉnh sửa

@SpringBootApplication

@EnableEurekaClient

public class ContractServiceApplication {

public static void main(String[] args) {

SpringApplication.run(ContractServiceApplication.class, args);

}

}

**🚀 4. Chạy các Service sau khi Discovery Server chạy**

1. Đảm bảo **Discovery Server (Eureka Server)** đang chạy tại http://localhost:8761
2. Chạy từng microservice:

bash

Sao chépChỉnh sửa

./mvnw spring-boot:run

1. Truy cập: <http://localhost:8761> và bạn sẽ thấy:
   * CONTRACT-SERVICE
   * NOTIFICATION-SERVICE
   * BILLING-SERVICE
   * ... đã đăng ký thành công.

**🧪 Ví dụ cấu hình cho một service**

**application.properties của notification-service:**

properties

Sao chépChỉnh sửa

server.port=8083

spring.application.name=notification-service

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

Nếu bạn muốn mình **kiểm tra cấu hình hoặc chỉ ra lỗi cụ thể với 1 service nào đó**, hãy gửi application.properties của nó cho mình.

**Bạn đã nói:**

tuyệt, vậy về cơ bản là đã xong phần cấu hình các micoservices rồi đúng không?

**ChatGPT đã nói:**

Đúng rồi, **về cơ bản bạn đã hoàn thành xong phần cấu hình các Microservices để tích hợp với Discovery Server (Eureka)** rồi. Cụ thể:

**✅ Tóm tắt những gì bạn đã hoàn thành:**

**🔹 1. Discovery Server (Eureka Server)**

* Đã tạo và chạy trên localhost:8761
* Được cấu hình không đăng ký chính nó (register-with-eureka: false)

**🔹 2. Các Microservices**

* Bạn đã tạo các vi dịch vụ: Contract Service, Addendum Service, Billing & Payment Service, Notification Service, v.v.
* Mỗi service:
  + Có spring.application.name riêng biệt
  + Có server.port riêng
  + Có cấu hình để đăng ký với Eureka

**🔹 3. Tích hợp với Eureka**

* Đã thêm dependency spring-cloud-starter-netflix-eureka-client
* Cấu hình eureka.client.service-url.defaultZone
* Đăng ký thành công các dịch vụ (xác nhận trên trang UI của Eureka)

**🟡 Gợi ý các bước tiếp theo**

Bây giờ bạn có thể triển khai các phần khác trong hệ thống:

**🔸 1. Tạo RESTful APIs cho từng service**

* Dùng @RestController, @RequestMapping, v.v.
* Dùng @Service, @Repository để chia tách rõ ràng các tầng
* Kết nối đến database (MySQL)

**🔸 2. Giao tiếp giữa các service**

* Dùng **RestTemplate**, **WebClient** hoặc **OpenFeign** (nên dùng OpenFeign + Eureka)

**🔸 3. Sử dụng RabbitMQ hoặc Kafka**

* Để giao tiếp bất đồng bộ giữa các services (sẽ triển khai sau)

**🔸 4. Spring Cloud Gateway**

* Đang setup Gateway để làm API entry point (bạn đang chuẩn bị phần này)
* Route các request đến các vi dịch vụ dựa trên path và serviceId

**✅ Tổng kết**

👉 **Đúng vậy, bạn đã xong phần cấu hình các Microservices về mặt tích hợp Discovery (Eureka)**.

Khi bạn sẵn sàng, mình có thể tiếp tục hướng dẫn:

* Viết RESTful API cho từng vi dịch vụ
* Định nghĩa contract giữa các service
* Triển khai RabbitMQ hoặc Kafka
* Hoặc cấu hình Spring Cloud Gateway cho routing

Bạn muốn đi tiếp với phần nào?

**Bạn đã nói:**

hiện tại tôi muốn tạo một project reactjs để làm phần frontend, tuy nhiên tôi chưa có gì cả, hãy chỉ tôi

**ChatGPT đã nói:**

Tuyệt! Bây giờ bạn sẽ bắt đầu tạo một **project ReactJS** để làm phần **frontend** cho hệ thống microservices của bạn. Dưới đây là hướng dẫn **từng bước từ đầu**:

**✅ Bước 1: Cài đặt công cụ cần thiết**

Bạn cần cài:

**1. Node.js (bao gồm cả npm)**

* Truy cập: https://nodejs.org
* Tải bản **LTS** (Long Term Support), ví dụ: Node.js 18.x hoặc 20.x
* Sau khi cài xong, kiểm tra:

bash

Sao chépChỉnh sửa

node -v

npm -v

**✅ Bước 2: Tạo project ReactJS mới**

Mở terminal hoặc VSCode, chạy lệnh sau:

bash

Sao chépChỉnh sửa

npx create-react-app frontend

⚠️ frontend là tên thư mục của dự án, bạn có thể thay đổi nếu muốn.

**Di chuyển vào thư mục project:**

bash

Sao chépChỉnh sửa

cd frontend

**✅ Bước 3: Chạy thử React app**

bash

Sao chépChỉnh sửa

npm start

* Trình duyệt sẽ tự mở http://localhost:3000
* Bạn sẽ thấy trang React mặc định hiện ra ✅

**✅ Bước 4: Cấu trúc thư mục ReactJS cơ bản**

Sau khi khởi tạo xong, bạn sẽ thấy:

pgsql

Sao chépChỉnh sửa

frontend/

├── public/

├── src/

│ ├── App.js

│ ├── index.js

├── package.json

Bạn sẽ phát triển giao diện và logic trong thư mục src/.

**✅ Bước 5: Gợi ý cấu trúc dự án cho microservices**

Giả sử bạn có các service:

* User Service
* Contract Service
* Billing Service
* Notification Service

Bạn nên chia route/giao diện thành:

css

Sao chépChỉnh sửa

src/

├── pages/

│ ├── Users.jsx

│ ├── Contracts.jsx

│ ├── Billings.jsx

│ ├── Notifications.jsx

├── services/

│ ├── userService.js

│ ├── contractService.js

├── App.js

**✅ Bước 6: Gọi API từ backend (qua Gateway)**

**Cài Axios (gọi HTTP API):**

bash

Sao chépChỉnh sửa

npm install axios

**Ví dụ gọi API trong userService.js:**

js

Sao chépChỉnh sửa

import axios from 'axios';

const API\_BASE\_URL = 'http://localhost:8080'; // Spring Cloud Gateway

export const getAllUsers = () => {

return axios.get(`${API\_BASE\_URL}/users`);

};

⚠️ Dùng đúng path routing trong Gateway nhé!

**✅ Bước 7: Tùy chỉnh App.js**

jsx

Sao chépChỉnh sửa

import React from 'react';

import Users from './pages/Users';

function App() {

return (

<div>

<h1>My Microservice Frontend</h1>

<Users />

</div>

);

}

export default App;

**✅ Bước 8: Cho phép React gọi API từ backend (CORS)**

Đừng quên cấu hình CORS trong backend (@CrossOrigin(origins = "http://localhost:3000")) hoặc cấu hình global.

**✅ Bước tiếp theo?**

Bạn có thể chọn một trong các hướng:

* Thiết kế giao diện từng trang (Users, Contracts, ...)
* Gọi thử API (nếu backend đã có REST API)
* Cấu hình react-router-dom cho SPA (Single Page App)
* Kết nối xác thực, gửi thông báo qua Notification Service, v.v.

Bạn muốn mình hướng dẫn tiếp phần nào? Thiết kế trang đầu tiên, gọi API thật hay setup routing?

**Bạn đã nói:**

bây giờ bạn có thể giúp tôi mô tả lại một luồng mẫu từ frontend đến apigateway-eureka-microservice-database không?

**ChatGPT đã nói:**

Tuyệt! Mình sẽ giúp bạn mô tả lại **luồng xử lý mẫu** từ **React frontend** → **API Gateway** → **Eureka Discovery** → **Microservice** → **Database**, phù hợp với kiến trúc bạn đang xây dựng.

**🧭 Mô tả kiến trúc tổng thể**

Luồng mẫu sẽ như sau:

scss

Sao chépChỉnh sửa

ReactJS (Frontend)

↓ (HTTP Request)

Spring Cloud Gateway (API Gateway)

↓ (Tìm service)

Eureka Server (Service Registry)

↓ (Chuyển tiếp request)

Microservice (e.g., User Service)

↓ (Xử lý business logic)

MySQL Database

**✅ Ví dụ cụ thể: Truy xuất danh sách người dùng**

**⚙️ Thành phần:**

* **Frontend (React)**: Gửi yêu cầu GET /users
* **Gateway**: Định tuyến /users đến user-service
* **Eureka**: Đã đăng ký user-service nên gateway biết địa chỉ
* **User Service**: Gọi DB lấy danh sách user
* **Database**: Trả về dữ liệu
* **React**: Nhận và hiển thị

**🪜 Các bước chi tiết**

**1. 🔵 React gửi yêu cầu API**

js

Sao chépChỉnh sửa

axios.get("http://localhost:8080/users")

→ gửi request tới API Gateway.

**2. 🟢 API Gateway định tuyến**

**Trong application.yml của Gateway**:

yaml

Sao chépChỉnh sửa

spring:

cloud:

gateway:

routes:

- id: user-service

uri: lb://user-service

predicates:

- Path=/users/\*\*

→ Tất cả các request /users/\*\* sẽ chuyển đến service tên là user-service.

**3. 🟡 Gateway tìm thông tin từ Eureka**

Gateway sử dụng DiscoveryClient để hỏi Eureka:  
→ “Service user-service đang chạy ở đâu?”

Eureka trả về địa chỉ thực của microservice, ví dụ:

text

Sao chépChỉnh sửa

http://localhost:8081

**4. 🔴 Chuyển tiếp tới User Service**

Gateway tự động forward request đến:

http

Sao chépChỉnh sửa

http://localhost:8081/users

**5. 🟠 User Service xử lý logic**

Trong UserController.java:

java

Sao chépChỉnh sửa

@GetMapping("/users")

public List<User> getAllUsers() {

return userRepository.findAll(); // Truy vấn MySQL

}

**6. 🟣 MySQL xử lý truy vấn**

Spring Data JPA gọi SQL:

sql

Sao chépChỉnh sửa

SELECT \* FROM users;

→ Trả về danh sách users

**7. 🔁 Dữ liệu trả về**

Chuỗi trả về như:

json

Sao chépChỉnh sửa

[

{ "id": 1, "name": "Alice" },

{ "id": 2, "name": "Bob" }

]

→ Quay ngược lại qua:  
User Service → Gateway → React

**8. 🖥️ React hiển thị kết quả**

Trong React component:

js

Sao chépChỉnh sửa

useEffect(() => {

axios.get("http://localhost:8080/users").then(res => {

setUsers(res.data);

});

}, []);

→ Hiển thị lên giao diện người dùng.

**✅ Tóm tắt lại**

| **Giai đoạn** | **Vai trò** |
| --- | --- |
| ReactJS | Gửi yêu cầu HTTP đến Gateway |
| Spring Cloud Gateway | Nhận request, dựa vào route chuyển đến microservice |
| Eureka Server | Cho Gateway biết địa chỉ thực của user-service |
| User Service | Nhận request, xử lý logic, truy vấn DB |
| MySQL | Trả về dữ liệu thực |
| ReactJS | Hiển thị dữ liệu trả về từ backend |

Bạn có muốn mình **vẽ sơ đồ** minh họa cho trực quan không? Hoặc hướng dẫn bạn thử **test toàn bộ luồng này bằng Postman hoặc React**?

**Bạn đã nói:**

tôi không dùng .yml mà dùng .properties

**ChatGPT đã nói:**

Tuyệt vời, nếu bạn đang dùng application.properties thay vì application.yml, thì phần cấu hình **API Gateway** để định tuyến đến các microservice sẽ được viết như sau:

**🔁 Ví dụ cấu hình định tuyến trong application.properties cho Spring Cloud Gateway**

**🗂️ File: application.properties của Gateway Service**

properties

Sao chépChỉnh sửa

server.port=8080

spring.application.name=api-gateway

# Kích hoạt discovery client

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

# ROUTING CONFIG (chuyển từ YAML sang Properties)

spring.cloud.gateway.routes[0].id=user-service

spring.cloud.gateway.routes[0].uri=lb://user-service

spring.cloud.gateway.routes[0].predicates[0]=Path=/users/\*\*

spring.cloud.gateway.routes[1].id=contract-service

spring.cloud.gateway.routes[1].uri=lb://contract-service

spring.cloud.gateway.routes[1].predicates[0]=Path=/contracts/\*\*

🔍 lb:// là viết tắt của **load balancing**, tức là nó sẽ lấy thông tin từ Eureka để biết user-service đang ở đâu.

**🧪 Test thử luồng sau khi cấu hình**

Sau khi bạn:

* Các microservice như user-service, contract-service, ... đã cấu hình application.name và đăng ký vào Eureka.
* Eureka server chạy ở localhost:8761
* Gateway chạy ở localhost:8080

Bạn có thể gửi request bằng Postman hoặc từ ReactJS tới:

bash

Sao chépChỉnh sửa

GET http://localhost:8080/users

→ sẽ được chuyển đến user-service

**✅ Checklist cho từng service**

**Ví dụ cho user-service (application.properties):**

properties

Sao chépChỉnh sửa

server.port=8081

spring.application.name=user-service

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

Nếu bạn muốn, mình có thể giúp bạn tạo **mẫu cấu hình đầy đủ cho các services** còn lại luôn. Bạn cần mình làm tiếp phần nào? (contract-service, billing-service, addendum-service, etc.?)

**Bạn đã nói:**

id ở đây lấy ở đâu, tên aftifact id của project à

**ChatGPT đã nói:**

Trong cấu hình spring.cloud.gateway.routes[n].id=..., phần id là **tên định danh tùy chọn** mà bạn tự đặt cho mỗi route — **không liên quan trực tiếp đến artifactId của project**, nhưng bạn **nên đặt sao cho dễ hiểu** và nhất quán với spring.application.name của service mục tiêu.

**🔎 Ý nghĩa của các phần trong route:**

properties

Sao chépChỉnh sửa

spring.cloud.gateway.routes[0].id=user-service # <-- ID tùy đặt để phân biệt route

spring.cloud.gateway.routes[0].uri=lb://user-service # <-- phải khớp với spring.application.name của microservice

spring.cloud.gateway.routes[0].predicates[0]=Path=/users/\*\*

| **Thành phần** | **Giải thích** |
| --- | --- |
| id | Tên định danh cho route, bạn tự đặt (nên giống hoặc gần giống tên service) |
| uri | lb://<service-name> là bắt buộc phải **khớp với spring.application.name** |
| predicates | Điều kiện để định tuyến, ví dụ như đường dẫn URL (/users/\*\*) |

**✅ Ví dụ cụ thể:**

Giả sử bạn có service như sau:

**user-service (một project Spring Boot riêng):**

properties

Sao chépChỉnh sửa

spring.application.name=user-service

**Gateway cấu hình:**

properties

Sao chépChỉnh sửa

spring.cloud.gateway.routes[0].id=user-api

spring.cloud.gateway.routes[0].uri=lb://user-service

spring.cloud.gateway.routes[0].predicates[0]=Path=/users/\*\*

Ở đây:

* id=user-api là bạn đặt cho dễ nhớ
* uri=lb://user-service thì **"user-service" phải đúng với spring.application.name của service đó**
* /users/\*\* là đường dẫn sẽ bắt và route về user-service

👉 **Tóm lại:**

* id là do bạn đặt, chỉ cần duy nhất trong gateway.
* uri bắt buộc phải đúng với spring.application.name của vi dịch vụ tương ứng.

Nếu bạn cần, mình có thể giúp bạn kiểm tra toàn bộ cấu hình các vi dịch vụ với gateway.

**Bạn đã nói:**

spring.cloud.gateway.routes[0].id=user-service # <-- ID tùy đặt để phân biệt route spring.cloud.gateway.routes[0].uri=lb://user-service # <-- phải khớp với spring.application.name của microservice spring.cloud.gateway.routes[0].predicates[0]=Path=/users/\*\* spring.cloud.gateway.routes[0].id=contract-service spring.cloud.gateway.routes[0].uri=http://localhost:8081 spring.cloud.gateway.routes[0].predicates[0]=Path=/contracts/\*\* hai kiểu khai báo này, cách nào tốt hơn

**ChatGPT đã nói:**

Cả hai cách bạn đưa ra đều hoạt động, nhưng **mỗi cách có mục đích khác nhau** — và **trong kiến trúc microservices dùng Eureka Discovery + Spring Cloud Gateway**, cách tốt hơn **nên là dùng lb://<service-name>**, cụ thể như sau:

**✅ Cách nên dùng: LoadBalancer (Eureka Discovery)**

properties

Sao chépChỉnh sửa

spring.cloud.gateway.routes[0].id=contract-service

spring.cloud.gateway.routes[0].uri=lb://contract-service

spring.cloud.gateway.routes[0].predicates[0]=Path=/contracts/\*\*

**Ưu điểm:**

* Tự động phân giải địa chỉ từ Eureka Server.
* Hỗ trợ load balancing nếu có nhiều instance của service.
* Linh hoạt khi triển khai (ví dụ khi deploy nhiều instance lên cloud, container...).

**Yêu cầu:**

* Service phải đăng ký với Eureka (spring.application.name=contract-service, và eureka.client.register-with-eureka=true...).

**🟡 Cách có thể dùng nhưng kém linh hoạt: Gọi trực tiếp URL**

properties

Sao chépChỉnh sửa

spring.cloud.gateway.routes[0].id=contract-service

spring.cloud.gateway.routes[0].uri=http://localhost:8081

spring.cloud.gateway.routes[0].predicates[0]=Path=/contracts/\*\*

**Nhược điểm:**

* Gateway phải biết chính xác địa chỉ IP/port của service.
* Không hỗ trợ load balancing, failover.
* Không linh hoạt khi service scale hoặc đổi cổng/IP.

**✅ Kết luận**

| **Tiêu chí** | **lb://... (Eureka)** | **http://localhost:port** |
| --- | --- | --- |
| Tích hợp với Eureka | ✅ Có | ❌ Không |
| Tự động định tuyến | ✅ Có | ❌ Không |
| Hỗ trợ load balancing | ✅ Có | ❌ Không |
| Dễ mở rộng & bảo trì | ✅ Cao | ❌ Thấp |
| Thường dùng trong microservices | ✅ Chuẩn | ❌ Tạm thời/debug |

👉 **Vì vậy: nếu bạn đang dùng Eureka Discovery, thì nên luôn dùng lb://<service-name> để Gateway định tuyến.**  
Bạn chỉ nên dùng http://localhost:8081 trong trường hợp **tạm thời để debug** hoặc khi **không dùng Eureka**.

Nếu bạn cần mình kiểm tra hoặc viết mẫu đầy đủ cấu hình .properties cho Gateway dùng lb://, cứ gửi các service name là mình sẽ giúp.

**Bạn đã nói:**

spring.application.name=apigateway server.port=8080 eureka.client.service-url.defaultZone=http://localhost:8761/eureka eureka.client.register-with-eureka=true eureka.client.fetch-registry=true spring.cloud.gateway.discovery.locator.enabled=true spring.cloud.gateway.discovery.locator.lower-case-service-id=true spring.cloud.gateway.routes[0].id=user-service # <-- ID tùy đặt để phân biệt route spring.cloud.gateway.routes[0].uri=lb://user-service # <-- phải khớp với spring.application.name của microservice spring.cloud.gateway.routes[0].predicates[0]=Path=/users/\*\* giúp tôi kiểm tra thử xem config thế này đủ chưa nhé

**ChatGPT đã nói:**

Cấu hình application.properties bạn đưa ra cho **API Gateway** là **gần như hoàn chỉnh và đúng chuẩn**, dùng Spring Cloud Gateway + Eureka. Dưới đây là phần đánh giá chi tiết và một vài đề xuất nhỏ để hoàn thiện hơn.

**✅ Cấu hình hiện tại của bạn: GẦN ĐÚNG**

properties

Sao chépChỉnh sửa

spring.application.name=apigateway

server.port=8080

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

spring.cloud.gateway.discovery.locator.enabled=true

spring.cloud.gateway.discovery.locator.lower-case-service-id=true

spring.cloud.gateway.routes[0].id=user-service

spring.cloud.gateway.routes[0].uri=lb://user-service

spring.cloud.gateway.routes[0].predicates[0]=Path=/users/\*\*

**✅ Phân tích các phần chính:**

| **Thành phần cấu hình** | **Mô tả** |
| --- | --- |
| spring.application.name=apigateway | Đặt tên cho API Gateway |
| server.port=8080 | Port chạy Gateway |
| eureka.client.\* | Cấu hình để API Gateway đăng ký vào Eureka |
| spring.cloud.gateway.discovery.locator.enabled | Tự động định tuyến dựa theo các service đã đăng ký trong Eureka |
| spring.cloud.gateway.routes[...] | Định nghĩa route thủ công (tốt cho các path rõ ràng như /users/\*\*) |

**🟡 Gợi ý bổ sung:**

1. **Nếu bạn đã bật discovery.locator.enabled=true, thì bạn KHÔNG bắt buộc phải khai báo thủ công routes[0] nữa**, vì Gateway sẽ **tự động** định tuyến theo spring.application.name của service đó.  
   Nhưng việc khai báo thủ công giúp bạn **kiểm soát tốt hơn**, ví dụ đổi path /api/users/\*\* → user-service.
2. **Gợi ý cấu trúc cấu hình rõ ràng hơn**:

properties

Sao chépChỉnh sửa

# Application info

spring.application.name=apigateway

server.port=8080

# Eureka config

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

# Gateway config

spring.cloud.gateway.discovery.locator.enabled=true

spring.cloud.gateway.discovery.locator.lower-case-service-id=true

# Manual Route to user-service

spring.cloud.gateway.routes[0].id=user-service

spring.cloud.gateway.routes[0].uri=lb://user-service

spring.cloud.gateway.routes[0].predicates[0]=Path=/users/\*\*

**✅ Kết luận:**

✔️ **Cấu hình hiện tại là đúng và có thể chạy tốt** với điều kiện:

* Service user-service có spring.application.name=user-service và đã **đăng ký lên Eureka thành công**.
* Eureka Server đang chạy ở http://localhost:8761.

Nếu bạn muốn:

* Thêm các route khác (/contracts/\*\*, /bills/\*\*, v.v.)
* Dùng prefix /api/\*\*
* Viết application.yml thay vì .properties (vì dễ đọc hơn cho các route)

thì mình có thể hỗ trợ cụ thể hơn. Cứ nói nhé!

**Bạn đã nói:**

spring.application.name=addendumService server.port=8082 # ===== Datasource ===== spring.datasource.url=jdbc:mysql://localhost:3306/service\_addendum\_db spring.datasource.username=root spring.datasource.password=long@091103 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver # ===== JPA (Hibernate) ===== spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true # ===== Eureka Client ===== eureka.client.register-with-eureka=true eureka.client.fetch-registry=true eureka.client.service-url.defaultZone=http://localhost:8761/eureka còn đây là .properties của một service, đã đủ chưa. khi viết mẫu một java để tôi sẽ thử viết như thế nào để dùng nó

**ChatGPT đã nói:**

Cấu hình .properties bạn cung cấp cho **Addendum Service** là **đầy đủ và hợp lệ** để:

* Kết nối được với MySQL
* Sử dụng Spring Data JPA
* Đăng ký lên **Eureka Discovery Server**
* Chạy như một **Spring Boot microservice độc lập**

**✅ Tóm tắt những gì bạn đã cấu hình đúng:**

properties

Sao chépChỉnh sửa

spring.application.name=addendumService # Đặt tên để dùng với Eureka và Gateway

server.port=8082 # Cổng riêng biệt

spring.datasource.\* # Cấu hình kết nối MySQL

spring.jpa.\* # Hibernate config

eureka.client.\* # Eureka client để đăng ký vào discovery-server

**✅ Giờ viết một class Java mẫu để thử nghiệm kết nối:**

Giả sử bạn có 1 bảng addendums trong MySQL, hãy làm thử như sau:

**1. Entity: Addendum.java**

java

Sao chépChỉnh sửa

package com.example.addendumservice.model;

import jakarta.persistence.\*;

@Entity

public class Addendum {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String description;

// Getters and Setters

}

**2. Repository: AddendumRepository.java**

java

Sao chépChỉnh sửa

package com.example.addendumservice.repository;

import com.example.addendumservice.model.Addendum;

import org.springframework.data.jpa.repository.JpaRepository;

public interface AddendumRepository extends JpaRepository<Addendum, Long> {

}

**3. REST Controller: AddendumController.java**

java

Sao chépChỉnh sửa

package com.example.addendumservice.controller;

import com.example.addendumservice.model.Addendum;

import com.example.addendumservice.repository.AddendumRepository;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/addendums")

public class AddendumController {

private final AddendumRepository repository;

public AddendumController(AddendumRepository repository) {

this.repository = repository;

}

@GetMapping

public List<Addendum> getAll() {

return repository.findAll();

}

@PostMapping

public Addendum create(@RequestBody Addendum addendum) {

return repository.save(addendum);

}

}

**4. Main App: AddendumServiceApplication.java**

java

Sao chépChỉnh sửa

package com.example.addendumservice;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class AddendumServiceApplication {

public static void main(String[] args) {

SpringApplication.run(AddendumServiceApplication.class, args);

}

}

**🧪 Thử nghiệm:**

1. Chạy Eureka Discovery Server.
2. Chạy AddendumService.
3. Truy cập:
   * http://localhost:8082/addendums (GET → danh sách)
   * Gửi POST JSON qua Postman để tạo:

json

Sao chépChỉnh sửa

{

"name": "Phụ lục A",

"description": "Chi tiết phụ lục A"

}

1. Kiểm tra tại http://localhost:8761/ để xem service đã đăng ký.

✅ Như vậy bạn đã có một microservice hoạt động đầy đủ: kết nối DB + đăng ký Eureka + có REST API.

Nếu bạn cần viết thêm các service khác như vậy, có thể dùng lại cấu trúc tương tự. Muốn mình tạo luôn ContractService, UserService, PaymentService mẫu như thế không?