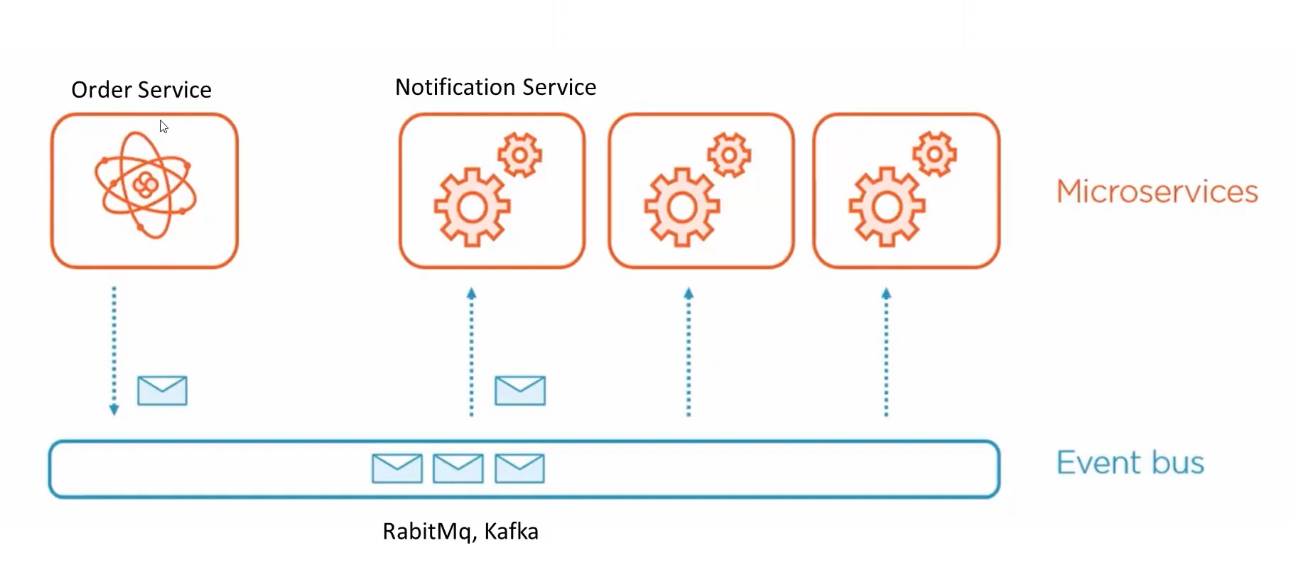
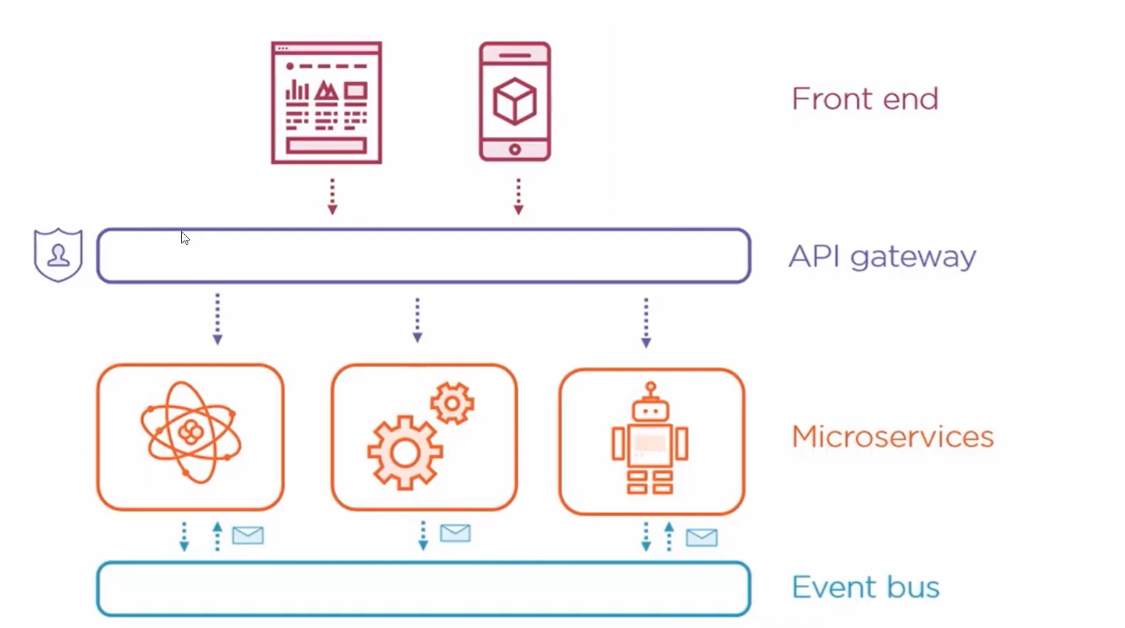
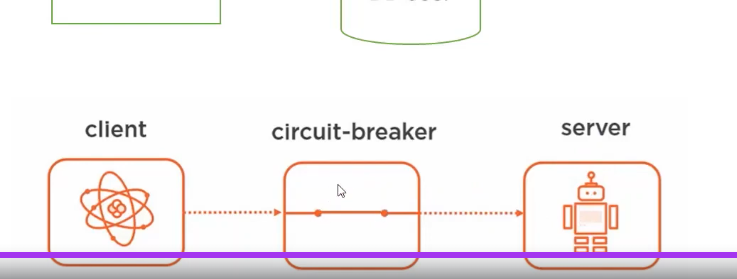
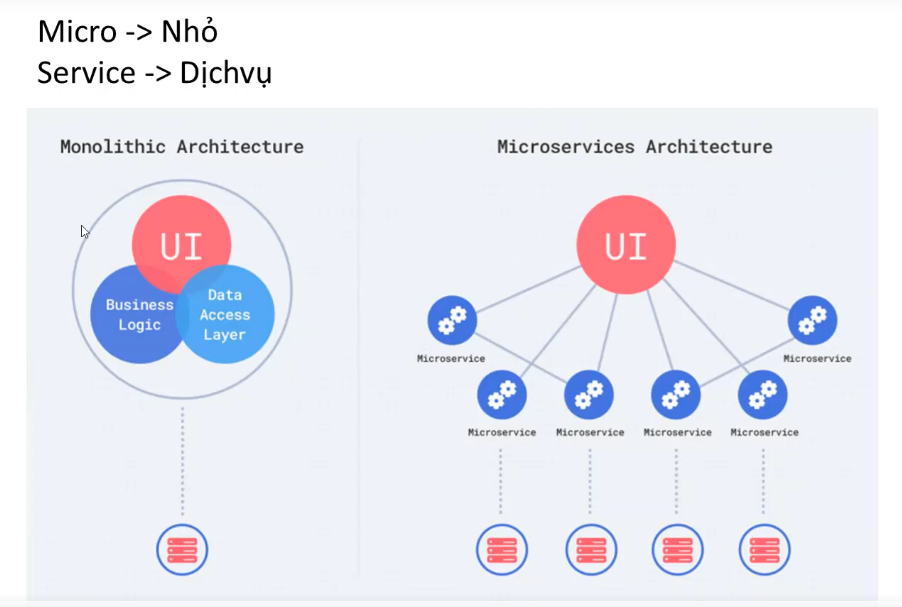
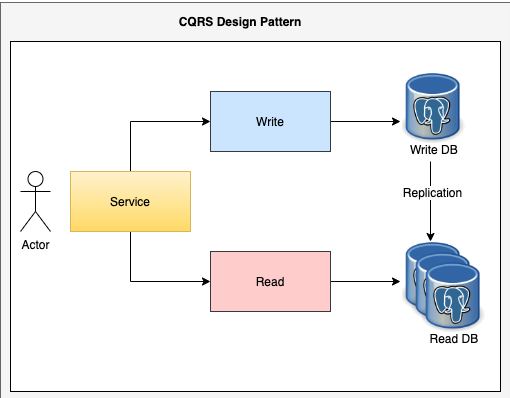
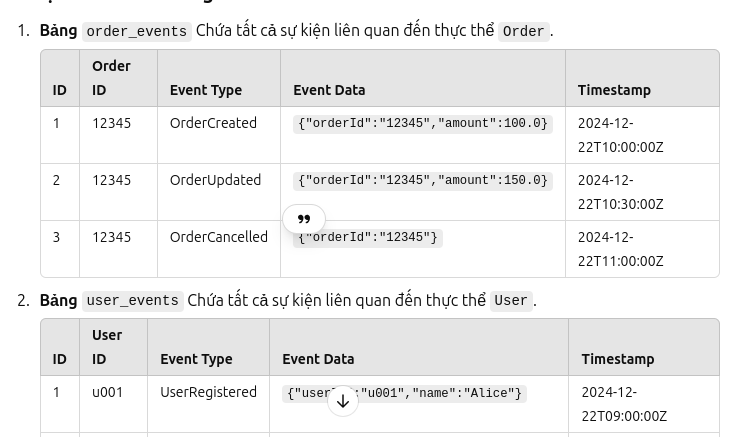
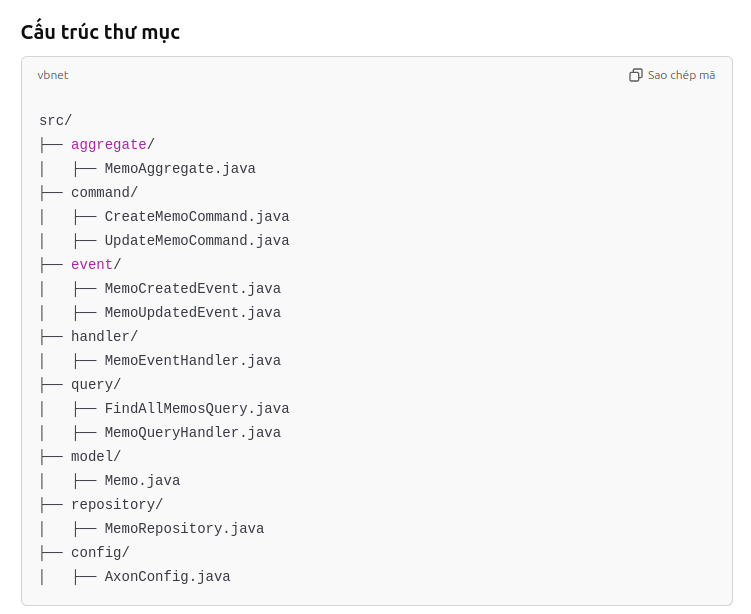
Tổng Quan

1. Tổng quan
   1. API gateway () : điều khiển request client đến từng service
   2. Disconvery server: trong hệ thống **Service Discovery** đóng vai trò trung tâm để quản lý và điều phối các dịch vụ.
      1. Tổng hợp thông tin từ các dịch vụ đăng ký: địa chỉ IP, cổng (port)....
      2. Xử lý các yêu cầu từ Service Consumer: lấy danh sách các service client
      3. Quản lý trạng thái của các dịch vụ
      4. Hỗ trợ Load Balancing
   3. config server: config chung
   4. event sourcing: giao tiếp giữa các service
   5. SAGA: quản lý transation
   6. distributed tracing: quản lý các request, track
   7. Circuit breaker ngắt các request hư hại đến các serviceNếu 1 service bị lỗi thì sẽ bị ngắt
2. microservice
   1. Mỗi microservice đều phải có CSDL riêng và độc lập
3. service discovery
   1. <https://spring.io/guides/gs/service-registration-and-discovery>
   2. eureka.client.register-with-eureka=false
   3. eureka.client.fetch-registry=false

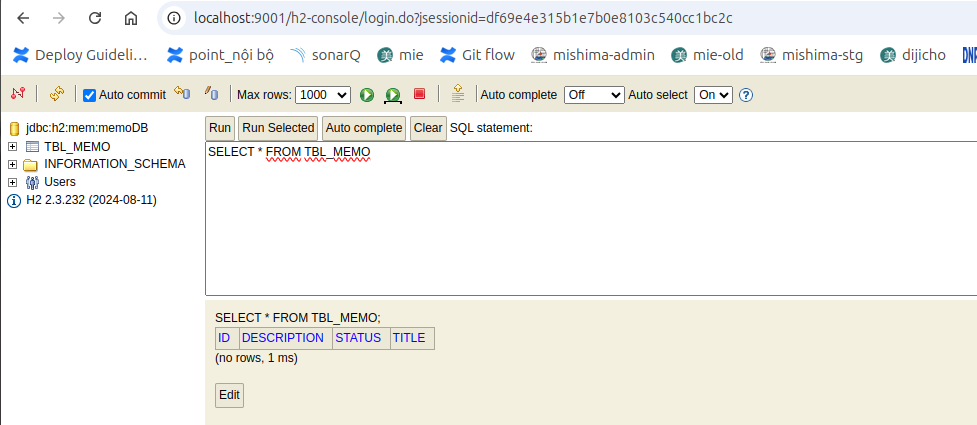
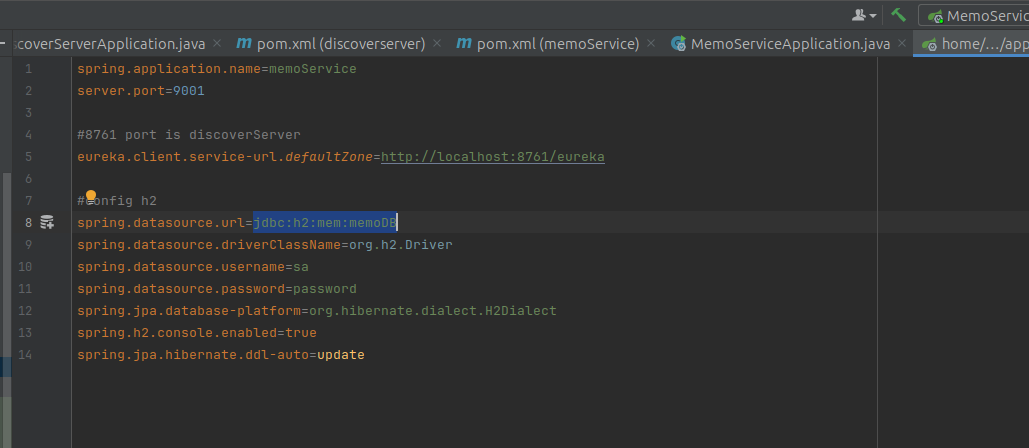
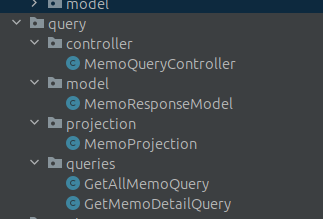
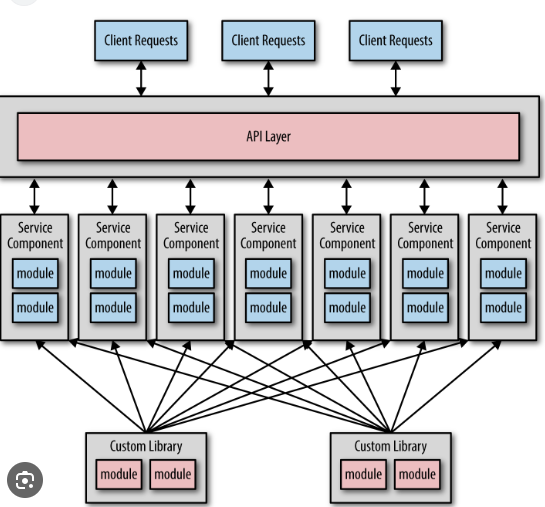
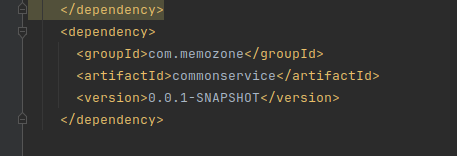
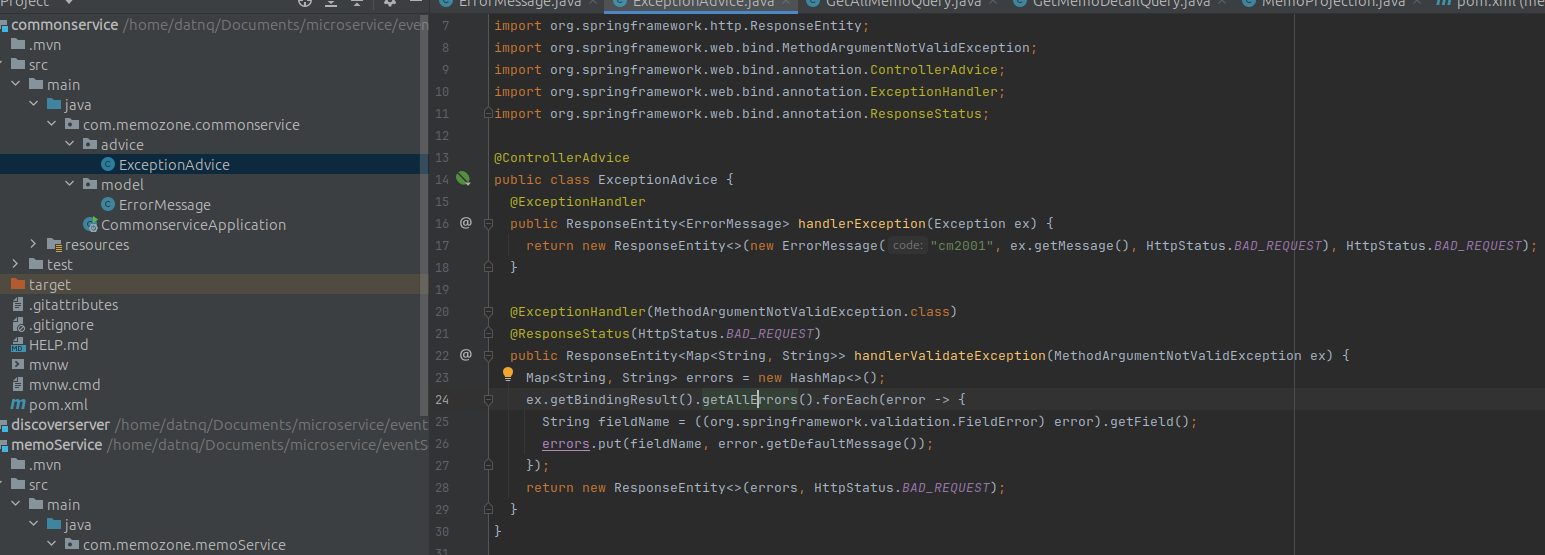
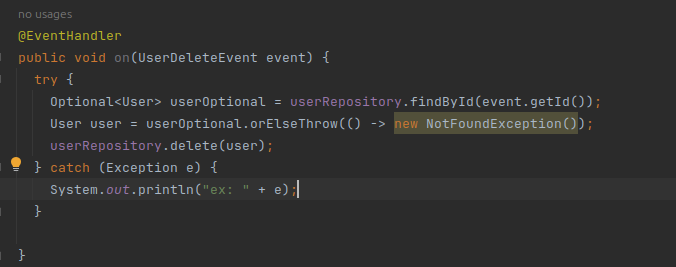
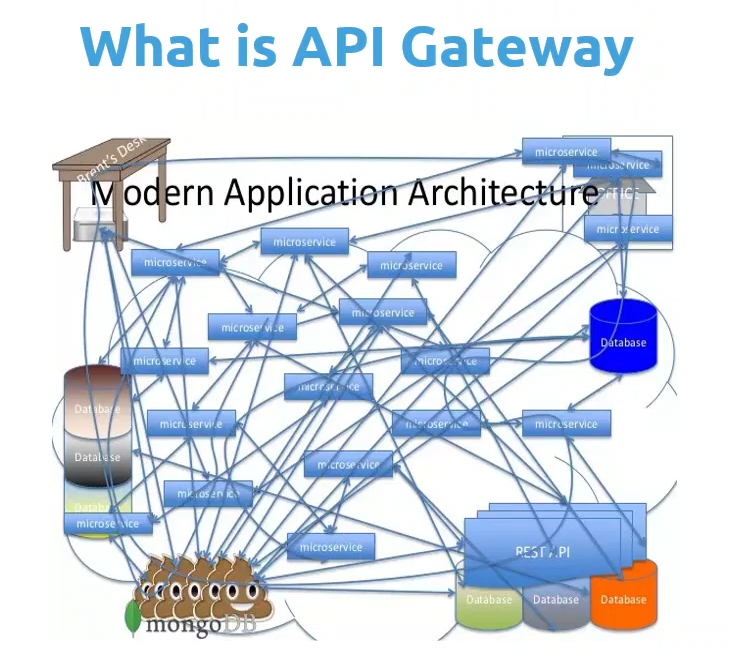
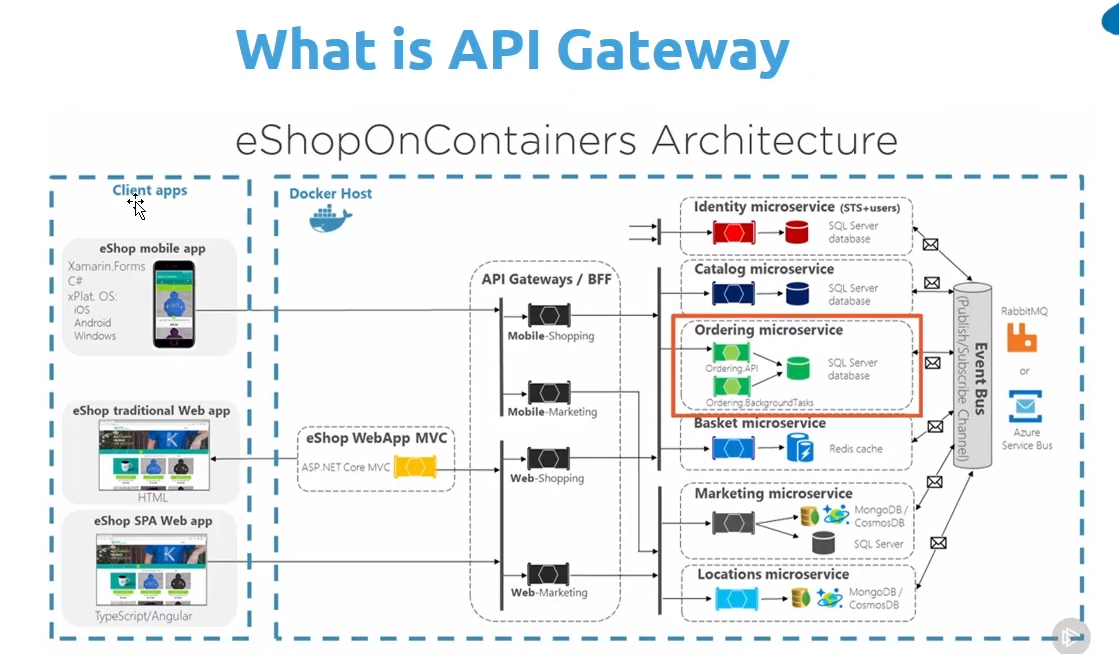
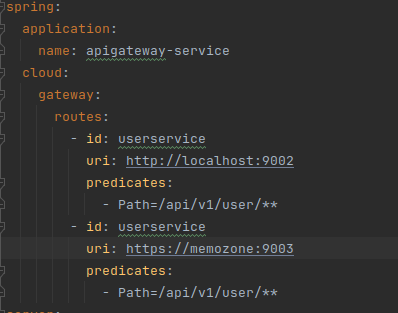
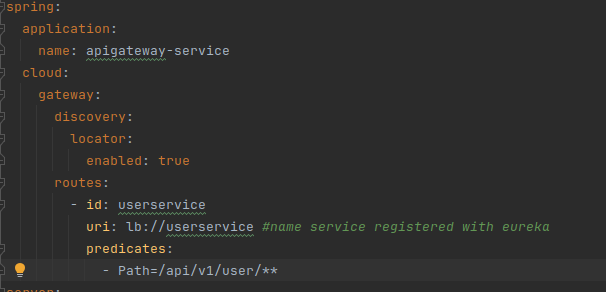
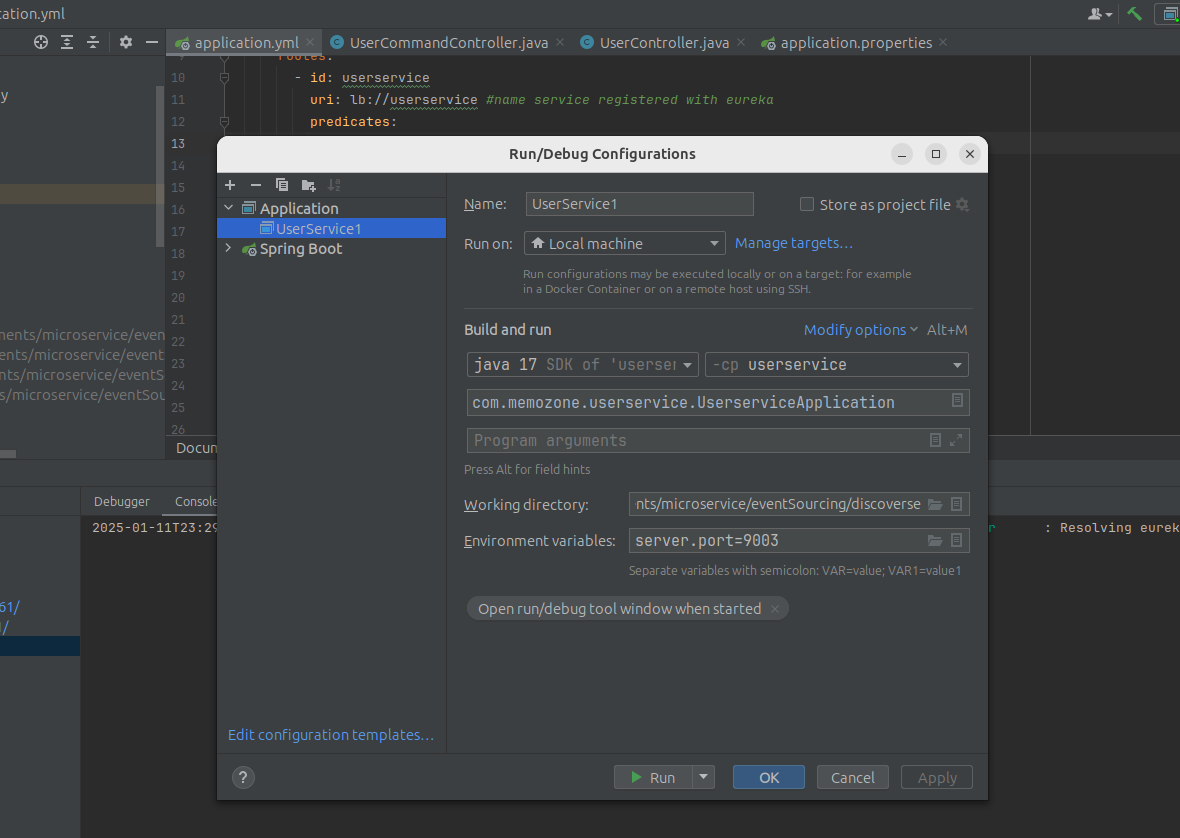
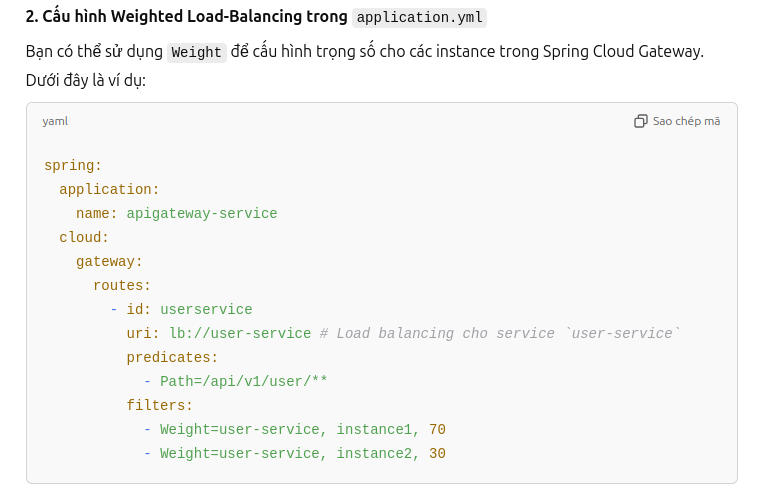
Architecture and Pattern

# Một số Architecture

* DDD Layered Architecture (Kiến trúc phân lớp)
* Microservices Architecture
* Monolithic Architecture (mvc - một khối)
* Event-Driven Architecture
* CQRS (Command Query Responsibility Segregation)
* Serverless Architecture
* ….

1. DDD layered **architecture** (kiến trúc thì khác với pattern (pattern thì là mức thấp hơn)nha)
   1. Link: <https://www.youtube.com/watch?v=WFlIoNHD_Yo>  
       <https://www.youtube.com/watch?v=WFlIoNHD_Yo&list=PLw0w5s5b9NK63Wa9AvcjpH4o3qOZhmWMq&index=19&t=201s>  
      <https://viblo.asia/p/gioi-thieu-design-pattern-domain-driven-design-ddd-Qbq5Q423lD8>
   2. Khái niệm là 1 các tiếp cận thiết kế phần mềm. DDD đề xuất thay vì tập trung vào các khía cạnh kỹ thuật trước thì dự án nên được tổ chức xung quanh các khái niện nghiệp vụ thực sự của hệ thống.
   3. User interface giống controller
   4. Application: chứa service
   5. Domain là interface
      1. ubiquitous language (ngôn ngữ chung): là ngôn ngữ chung để khác hàng cùng devlop cùng dễ hiểu
      2. aggreagate nhóm các đối tượng dữ liệu
   6. Infrastructure: liên kết DB, các dịch vụ khác (hạ tầng của hệ thống)
2. CQRS **architecture**
   1. **là tách riêng thằng đọc và thằng ghi ra**, mỗi thành phần xử lý một trách nhiệm riêng.
   2. 
   3. Có 2 thành phần chính là query và command
   4. **Query** Là các thao tác dùng đọc dữ liệu -Real
   5. **Command** Là cac thao tác dùng để ghi dữ liệu - Write
   6. Hoạt động là lồng đọc sẻ ở DB riêng và luồng write sẽ ở 1 server riêng => 2 DB sẽ có 1 cơ chế để liên kết và đồng bộ nhau. => từ đó luồng đọc có thể triển khai noSQL để triển khai nhanh đọc, còn write thì sẽ là PSQL hoặc MySQL để toàn vẹn dữ liệu
   7. 
3. Event Sourching **architecture**
   1. Khái niện:là một mô hình lưu trữ và xử lý dữ liệu, trong đó mọi thay đổi trạng thái của hệ thống được lưu trữ dưới dạng một chuỗi sự kiện bất biến (immutable events).   
      Lưu lại các hoạt động vào event, mục đích để quản lý và tái tạo nếu có lỗi
   2. Event Store (Kho lưu trữ sự kiện)
   3. Aggregate (Tập hợp): Chứa các lớp đại diện cho **Aggregate** trong Axon Framework.
   4. Event: Chứa các lớp sự kiện được phát hành khi trạng thái aggregate thay đổi.
   5. Command (Lệnh): Chứa các lớp đại diện cho các lệnh trong CQRS (Create, Update, Delete, v.v.).
   6. Event Handler (Trình xử lý sự kiện): Chứa các lớp để xử lý sự kiện và ghi chúng vào cơ sở dữ liệu
   7. Projection (Bản chiếu)
   8. Query: Chứa các lớp để định nghĩa các query và xử lý chúng.
   9. Model: Chứa các lớp đại diện cho mô hình dữ liệu (thực thể JPA hoặc các DTO).
   10. Repository: Chứa các interface repository để thao tác với cơ sở dữ liệu.
   11. 
4. Axon framework
   1. là một **Java framework** được thiết kế để hỗ trợ triển khai các kiến trúc phức tạp như **Event Sourcing**, **CQRS (Command Query Responsibility Segregation)**, và **Domain-Driven Design (DDD)**. Axon cung cấp các công cụ, thư viện, và cơ chế để dễ dàng quản lý các sự kiện, các lệnh (commands), và các truy vấn (queries) trong hệ thống.  
      <https://docs.axoniq.io/axon-framework-reference/development/>
   2. Nói đơn giản là ko có axon thì ta phải tự làm bằng tay hết
   3. Cài đặt bằng download file: giải nén tìm file jar rồi chạy như chạy 1 chương trình java
   4. Cài đặt bằng docker: chạy lệnh để tải image và run controlelr
      1. hoặc chạy lệnh: docker run -d -p 8024:8024 -p 8124:8124 --name axonserver axoniq/axonserver (Khi bạn chạy một container Docker với lệnh docker run, nếu image được chỉ định không tồn tại trên máy của bạn, Docker sẽ tự động tải image đó từ **Docker Hub** (hoặc registry khác mà bạn đã cấu hình, nếu có).)
   5. 

**Example**

1. H2 data base
   1. là một cơ sở dữ liệu quan hệ (Relational Database) được viết bằng Java, thường được sử dụng trong các dự án phát triển phần mềm bởi tính linh hoạt và dễ dàng cấu hình
   2. cung cấp giao diện
   3. **In-memory:** Dữ liệu chỉ tồn tại trong RAM và bị mất khi ứng dụng dừng.
   4. **Embedded:** Dữ liệu được lưu trữ trong file trên đĩa (file-based).
   5. 
2. jpa: object ánh sạ lên table
3. Command bus
   1. command: **Command** là một đối tượng dữ liệu (DTO) đại diện cho ý định thực hiện một hành động nào đó. Nó là một phần của **Write Model** trong kiến trúc **CQRS**.
   2. Command bus: **Command Bus** là cơ chế trung gian chịu trách nhiệm gửi Command từ người phát ra (Command Dispatcher) đến nơi xử lý (Command Handler).
      1. SimpleCommandBus: Cấu hình đơn giản để chạy trên một ứng dụng duy nhất.
      2. DistributedCommandBus: Để xử lý trong môi trường phân tán.
4. Aggreagte
   1. là thực thể hoặc nhóm thực thể
5. Query: lấy dữ liệu, Query được thiết kế để thực hiện các truy vấn đọc (query side) trong mô hình CQRS
   1. Một số loại query
      1. **Standard Queries** (Truy vấn tiêu chuẩn): Đây là loại truy vấn đơn giản và phổ biến nhất, dùng để lấy dữ liệu từ hệ thống. Mỗi truy vấn sẽ trả về một kết quả tương ứng.
      2. **Subscription Queries** (Truy vấn đăng ký): **Loại truy vấn này được dùng để lắng nghe các thay đổi của dữ liệu theo thời gian thực.** Kết quả ban đầu được trả về ngay, sau đó là các cập nhật liên tục khi dữ liệu thay đổi.
      3. **Scatter-Gather Queries** (Truy vấn phân tán - thu thập): Dùng để gửi truy vấn đến nhiều handler khác nhau và tập hợp tất cả các kết quả trả về. Loại truy vấn này phù hợp khi bạn cần tổng hợp dữ liệu từ nhiều nguồn
   2. Cấu trúc:
      1. 
      2. Model: Định nghĩa dữ liệu (Aggregate, Entity, DTO)
      3. projection: xử lý logic để lấy dữ liệu **@QueryHandler**
      4. queries: lưu trữ thông tin param đùng để filter
6. shared libraries
   1. viết các function dùng để tái sử dụng
   2. cơ bản nó giống như depend import vào trong các service
      1. build thành file jar và import vào project 
7. spring aop - lập trình khía cạnh
8. Xử lý ngoại lệ
   1. sử dụng spring aop viết xử lý exception vào service common
   2. 
   3. 
   4. nếu chúng ta throw ra lỗi mà ko try catch thì các event lỗi sẽ đi vào hàng đợi khi start lại ứng dụng thì các event lỗi sẽ tự restry
9. Xây dựng userService
10. swagger
    1. Mh dùng cách **Springdoc OpenAPI**  [https://springdoc.org/](https://springdoc.org/*)
    2. @OpenAPIDefinition ể dùng để cung cấp thông tin cấu hình và mô tả cho API của bạn.
    3. 
    4. @Operation (@Tag để note tên) => custom cho thông tin api, miêu tả thông tin api
    5. 
    6. @Hidden => che dấu enpoint
    7. (chú thích LDAP)
11. Xấy dựng api get way
    1. 
    2. 
    3. Nó hoạt động như một "cửa ngõ" duy nhất (single entry point) cho các client (như trình duyệt web, ứng dụng di động, hoặc dịch vụ khác) để tương tác với backend, đặc biệt là với nhiều microservices.
    4. Chức năng chính
       1. Định tuyến yêu cầu (Request Routing):Một API Gateway có thể nhận yêu cầu /api/users và định tuyến đến microservice quản lý người dùng, hoặc /api/orders đến microservice quản lý đơn hàng.
       2. Tích hợp (Aggregation): một ứng dụng hiển thị trang hồ sơ người dùng có thể lấy thông tin từ microservice người dùng, microservice giao dịch, và microservice thông báo.
       3. Bảo mậ: Kiểm soát tỷ lệ truy cập (Rate Limiting) và ngăn chặn tấn công DDoS. Xác thực (Authentication) và phân quyền (Authorization).
       4. Quản lý lỗi và chịu tải:API Gateway có thể xử lý các lỗi như yêu cầu thất bại, giới hạn lại số lượng yêu cầu, hoặc cung cấp các tính năng dự phòng như Circuit Breaker
       5. Cải thiện hiệu suất:Bộ nhớ đệm (Caching): Lưu trữ các phản hồi tạm thời để giảm tải cho backend.
       6. Một số thằng phổ biến aws,kong .. hoặc tự build 1 cái api getway (<https://spring.io/projects/spring-cloud-gateway>)
    5. Cách hoặt động: **Đăng ký service với Eureka** => **API Gateway truy vấn Eureka**
       1. ****Ví dụ cấu hình 2 instane cho api gateway bằng cách thủ công
       2. Dùng load Balancing
          1. Api gateway dựa vào thằng eureka để lấy service. Theo mặc định thì thằng load balancing lấy theo thuật toán ***Round robin***Tức là nó sẽ điều hướng tới lần lượt từng instane (có thể cấu hình các thuật toán). Có thể test trên local bằng cách chạy thêm 1 application nhưng khác point 
    6. Một số thuật toán load balance
       1. **Round robin**: lần lượt theo vòng
       2. Sticky round robin: Phân theo session, thằng A đang dùng instane IA thì các request của nó cứ bào instane IA thoii
       3. **weighted round robin**: phân phối tài nguyên theo trọng số, mỗi 1 instance được gán 1 weight (trong số) thằng nào có số weight lớn thì sẽ được ưu tiên phân bố (kiểm có 1 con server mạnh hơn mấy con khác thì muốn buff mạnh cho thằng này)
       4. IP/URL Hash: (giống sesion) băm ip/url thằng mã rồi gắn vào 1 server, rồi cứ thằng ip/url sẽ ánh xạ tới server đích thằng nào giống mới cái mã băm cũ thì nó chui vào (nếu khác hàng truy cập nhiều url thì sẽ ko quản lý dk)
       5. **least connection**: chuyển yêu cầu đến máy chủ ít được kết nối nhất
       6. least time: dựa theo thời gian phản hồi của máy chủ. thằng nào phản hồi nhanh thì chuyển request tới đó
    7. Cách config <https://docs.spring.io/spring-cloud-commons/docs/current/reference/html/#spring-cloud-loadbalancer>
       1. VD: 
    8. Rate limiting: bảo vệ api