# HOMEWORK - 1

1. **Domain-Driven Design là gì (DDD)?**

là một phương pháp thiết kế phần mềm tập trung vào domain (lĩnh vực nghiệp vụ cụ thể) và logic cốt lõi của ứng dụng. Mục tiêu chính của DDD là đảm bảo rằng thiết kế và phát triển phần mềm phản ánh chính xác các yêu cầu nghiệp vụ và mô hình hóa chúng một cách rõ ràng

1. **Liệt kê các thành phần chính và nguyên tắc của DDD**
   1. Các thành phần chính:

* Entity (Thực thể): Là các đối tượng có danh tính duy nhất (unique identity) và vòng đời riêng.
* Value Object (Đối tượng giá trị): Là các đối tượng không có danh tính mà chỉ dựa trên giá trị của chúng.
* Aggregate (Tập hợp): Là một nhóm các Entity và/hoặc Value Object liên kết với nhau và được xử lý như một đơn vị duy nhất.
* Repository (Kho lưu trữ): Là một lớp hoặc giao diện chịu trách nhiệm truy xuất và lưu trữ dữ liệu của các Aggregate từ cơ sở dữ liệu.
* Service (Dịch vụ): Được sử dụng để thực hiện các hành động hoặc logic không thuộc về một thực thể hay giá trị cụ thể.
* Bounded Context (Ngữ cảnh giới hạn): Là một khu vực cụ thể trong domain nơi mà các thuật ngữ và mô hình có ý nghĩa cụ thể.
* Ubiquitous Language (Ngôn ngữ phổ quát): Là một ngôn ngữ chung được sử dụng bởi các chuyên gia nghiệp vụ và nhà phát triển, nhằm giảm thiểu hiểu lầm.
  1. Nguyên tắc
* Tập trung vào domain và nghiệp vụ cốt lõi: Phần mềm được thiết kế để giải quyết các vấn đề quan trọng trong domain, đảm bảo rằng các yêu cầu nghiệp vụ là trọng tâm.
* Phát triển dựa trên sự hợp tác: Đội ngũ phát triển và các chuyên gia nghiệp vụ phải làm việc chặt chẽ với nhau để hiểu và mô hình hóa domain.
* Mô hình domain là cốt lõi của thiết kế: Domain phải được mô hình hóa một cách chính xác và phải là cơ sở cho tất cả các quyết định về kiến trúc phần mềm.
* Chia domain thành các ngữ cảnh nhỏ: Sử dụng Bounded Context để chia hệ thống thành các phần độc lập nhằm giảm độ phức tạp.
* Sử dụng ngôn ngữ phổ quát: Giảm thiểu sự mơ hồ trong giao tiếp thông qua việc sử dụng một ngôn ngữ chung.
* Tính nhất quán và toàn vẹn dữ liệu: Bảo đảm tính nhất quán trong từng Aggregate và sử dụng cơ chế kiểm soát giao dịch khi cần thiết.

1. **Đưa ra ví dụ về DDD và giải thích**

Giả sử bạn đang thiết kế một hệ thống thương mại điện tử (e-commerce system). Trong hệ thống này, bạn có một domain là "Đặt hàng trực tuyến" (Online Order Management).

Áp dụng DDD vào ví dụ:

* Domain chính: Quản lý đặt hàng.
* Ngôn ngữ phổ quát (Ubiquitous Language): Các thuật ngữ như Order (Đơn hàng), Product (Sản phẩm), Customer (Khách hàng), Order Status (Trạng thái đơn hàng), Payment (Thanh toán) được định nghĩa và thống nhất giữa các chuyên gia nghiệp vụ và nhà phát triển.

Entity (Thực thể):

* Customer (Khách hàng):
  + Mỗi khách hàng được xác định bởi một Customer ID duy nhất.
  + Thuộc tính: CustomerID, Name, Email.
* Address (Địa chỉ):
  + Chỉ có ý nghĩa dựa trên giá trị của nó.
  + Thuộc tính: Street, City, PostalCode.
* Order (Đơn hàng):
  + Là một Aggregate. Order ID là Aggregate Root.
  + Bao gồm: Customer, danh sách Product, trạng thái đơn hàng (Pending, Shipped, Delivered), và thông tin thanh toán.
* OrderRepository: Lớp này quản lý việc lưu trữ và truy xuất dữ liệu của các đơn hàng từ cơ sở dữ liệu.

Service (Dịch vụ):

* PaymentService: Dịch vụ xử lý thanh toán cho các đơn hàng, ví dụ xác nhận thẻ tín dụng hoặc liên kết ví điện tử.

Bounded Context (Ngữ cảnh giới hạn):

* Ngữ cảnh "Quản lý đơn hàng": Quản lý các thông tin liên quan đến đặt hàng.
* Ngữ cảnh "Xử lý thanh toán":
  + Quản lý và xác minh giao dịch thanh toán.
  + Hai ngữ cảnh này có thể giao tiếp thông qua các API hoặc message queues.

Một ví dụ cụ thể

* Customer (Khách hàng) thực hiện một đơn hàng qua ứng dụng.
* Hệ thống tạo một Order (Đơn hàng), gán các sản phẩm từ giỏ hàng của khách hàng.
* PaymentService (Dịch vụ thanh toán) xử lý giao dịch.
* Trạng thái của Order được cập nhật thông qua OrderRepository.
* Khi đơn hàng được giao, trạng thái chuyển sang Delivered.

Giải thích ví dụ:

* DDD trong ví dụ trên giúp chia hệ thống thành các phần dễ hiểu và độc lập, sát với nghiệp vụ thực tế.
* Sử dụng Bounded Context giúp giảm sự phụ thuộc giữa các phần của hệ thống (ví dụ: quản lý đơn hàng và xử lý thanh toán).
* Ubiquitous Language đảm bảo rằng mọi người trong nhóm phát triển đều hiểu các thuật ngữ quan trọng và sử dụng chúng một cách nhất quán.
* Kết quả là hệ thống dễ mở rộng, dễ bảo trì và phù hợp với yêu cầu nghiệp vụ.

1. **Mô tả sự khác biệt giữa các thuật ngữ Coupling và Cohesion**

|  |  |  |
| --- | --- | --- |
| **Đặc điểm** | **Coupling** | **Cohesion** |
| Ý nghĩa | Đo mức độ phụ thuộc giữa các module. | Đo mức độ các phần trong một module liên quan với nhau. |
| Mức độ mong muốn | Thấp là tốt | Cao là tốt |
| Tác động khi thay đổi | Coupling cao làm thay đổi một module ảnh hưởng đến module khác. | Cohesion thấp làm module khó hiểu và khó bảo trì. |
| Mục tiêu thiết kế | Giảm sự phụ thuộc giữa các module. | Đảm bảo mỗi module thực hiện một nhiệm vụ cụ thể. |

1. **Dựa vào các bài giảng, mô tả các thành phần chính của Microservices thì thành phần nào quan trọng nhất với các dự án cuối kỳ.**

Điều này phụ thuộc vào bản chất và yêu cầu cụ thể của dự án, nhưng trong hầu hết các trường hợp, API Gateway và Service Communication thường là hai thành phần quan trọng nhất.

* API Gateway:
  + Đây là điểm truy cập chính cho người dùng hoặc ứng dụng đến hệ thống của bạn. Nếu không có API Gateway, việc điều phối và quản lý các dịch vụ sẽ trở nên phức tạp.
  + Ví dụ: Trong một ứng dụng đặt vé xe, API Gateway sẽ nhận các yêu cầu như tìm kiếm chuyến xe, đặt vé, hoặc thanh toán.
* Service Communication: Đảm bảo các dịch vụ trong hệ thống có thể giao tiếp hiệu quả, đặc biệt trong các hệ thống cần phối hợp nhiều dịch vụ (như thanh toán, quản lý vé, và quản lý khách hàng).

1. **Các kho lưu trữ công khai trên GitHub thường được sử dụng để chia sẻ phần mềm mã nguồn mở. Để thực sự làm cho kho lưu trữ của bạn trở thành mã nguồn mở, bạn cần cấp phép cho nó, cho phép người khác tự do sử dụng, thay đổi và phân phối phần mềm. So sánh và mô tả các đặc điểm chính của các loại giấy phép sau trên GitHub:**
   1. **Apache License 2.0**

* Đặc điểm chính:
  + Cho phép sử dụng, chỉnh sửa, phân phối và thương mại hóa mã nguồn.
  + Yêu cầu bảo lưu bản quyền và thông báo giấy phép trong mã nguồn phân phối lại.
  + Cung cấp điều khoản bảo vệ bản quyền sáng chế (patent rights), giúp tránh các tranh chấp pháp lý liên quan đến bằng sáng chế.
* Hạn chế:

Không yêu cầu các thay đổi phải được công khai (không phải copyleft).

* Tính ứng dụng:

Phù hợp với các dự án thương mại cần bảo vệ quyền sở hữu trí tuệ.

* 1. **GNU General Public License v3.0**
* Đặc điểm chính:
  + Là giấy phép mạnh mẽ với yêu cầu "copyleft": Nếu một dự án sử dụng mã nguồn GPL, mọi phần mềm phái sinh cũng phải được cấp phép dưới GPL.
  + Yêu cầu cung cấp mã nguồn của phần mềm phái sinh.
  + Tăng cường bảo vệ người dùng khỏi các hạn chế kỹ thuật số (Digital Rights Management - DRM).
* Hạn chế: Không phù hợp với các dự án thương mại vì yêu cầu chia sẻ mã nguồn phái sinh.
* Tính ứng dụng: Phù hợp với các dự án cộng đồng, khuyến khích chia sẻ mã nguồn.
  1. **MIT License**
* Đặc điểm chính:
  + Là giấy phép đơn giản và phổ biến nhất.
  + Cho phép sử dụng, chỉnh sửa, phân phối và thương mại hóa mã nguồn mà không yêu cầu chia sẻ mã nguồn.
  + Chỉ yêu cầu giữ lại thông báo bản quyền trong mã phân phối.
* Hạn chế: Không bảo vệ người dùng khỏi tranh chấp bằng sáng chế hoặc hạn chế DRM.
* Tính ứng dụng: Phù hợp với cả dự án cá nhân và thương mại khi bạn muốn tối đa hóa tính tự do cho người dùng.
  1. **Mozilla Public License 2.0**
* Đặc điểm chính:
  + Là giấy phép dạng "copyleft yếu" (weak copyleft).
  + Cho phép mã nguồn có thể được liên kết với các phần mềm khác có giấy phép khác, nhưng mọi thay đổi đối với mã nguồn MPL phải được công khai.
  + Cung cấp quyền bảo vệ sáng chế giống Apache License.
* Hạn chế: Tạo ra sự phân chia giữa mã nguồn công khai và phần mềm sở hữu riêng.
* Tính ứng dụng: Phù hợp với các dự án có mã nguồn muốn chia sẻ nhưng vẫn có thể tích hợp vào các phần mềm độc quyền.
  1. **Open Software License 3**
* Đặc điểm chính:
  + Là một giấy phép "copyleft mạnh" tương tự GPL nhưng có thêm các điều khoản pháp lý về giao kết hợp đồng.
  + Yêu cầu mã nguồn phái sinh phải được công khai và cấp phép dưới OSL.
  + Cung cấp quyền bảo vệ sáng chế.
* Hạn chế: Ít phổ biến hơn các giấy phép khác như GPL hoặc MIT.
* Tính ứng dụng: Thích hợp cho các dự án yêu cầu mạnh mẽ về tính pháp lý và công khai mã nguồn phái sinh.