**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM**



**ỨNG DỤNG QUẢN LÝ THIẾT BỊ TÁI CHẾ**

***Nhóm 13 - Sinh viên thực hiện***

1. Dương Thế Vinh - 20107651
2. Trần Nguyễn Hoàng Nam - 20095731
3. Nguyễn Ngọc Tuấn - 20103581

**GVHD : TS. Võ Văn Hải**

1. Tổng quan

Kiến trúc phần mềm đóng một vài trò rất quan trọng trong vòng đời phát triển phần mềm. Các thiết kế kiến trúc cung cấp một bản thiết kế và hướng dẫn để phát triển một hệ thống phần mềm dựa trên đặc tả phân tích yêu cầu của nó. Thiết kế kiến trúc là hiện thân của những quyết định sớm nhất có tác động quyết định đến sự thành công cuối cùng của sản phẩm phần mềm. Thiết kế cho thấy các thành phần hệ thống được cấu trúc như thế nào và cách chúng hoạt động cùng nhau. Một thiết kế kiến trúc cũng phải bao gồm các yêu cầu chức năng và phi chức năng của phần mềm. Nó phục vụ như một kế hoạch đánh giá và thực hiện để phát triển phần mềm và cải tiến phần mềm.

Nhiều nhà nghiên cứu đã giải thích về kiến trúc phần mềm, và họ có các quan điểm khác nhau về cách trình bày tốt nhất về kiến trúc của hệ thống phần mềm. Không có cách giải thích nào là sai; mỗi kiến trúc có giá trị riêng. Định nghĩa của Bass L., và cộng sự nắm giữ được điểm cốt yếu mà kiến trúc phần mềm đòi hỏi:

"Kiến trúc phần mềm của một chương trình hoặc hệ thống tính toán là cấu trúc hoặc các cấu trúc của hệ thống đó, gồm các thành phần của phần mềm, các thuộc tính có thể trông thấy được từ bên ngoài của các thành phần này, và các mối quan hệ giữa chúng." (Bass, Clements, Kazman) Một định nghĩa khác về Kiến trúc phần mềm được đưa ra bởi: Philippe Kruchten, Grady Booch, Kurt Bittner, và Rich Reitman bắt nguồn và tinh chỉnh một định nghĩa về kiến trúc dựa trên tác phẩm của Mary Shaw và David Garlan (Shaw and Garlan 1996). Định nghĩa của họ là: "Kiến trúc phần mềm bao gồm một loạt các quyết định quan trọng về việc tổ chức một hệ thống phần mềm bao gồm việc lựa chọn các yếu tố cấu trúc và giao diện của chúng bằng cách tạo ra hệ thống; hành vi được xác định trong hợp tác giữa các yếu tố đó; thành phần của các yếu tố cấu trúc và hành vi này vào các hệ thống con lớn hơn; và một mẫu kiến trúc hướng dẫn tổ chức này. Kiến trúc phần mềm cũng bao hàm các chức năng, khả năng sử dụng, khả năng phục hồi, hiệu năng, tái sử dụng, tính dễ hiểu, các ràng buộc về kinh tế và công nghệ, sự cân bằng và mối quan tâm thẩm mỹ”

Hay

"Kiến trúc phần mềm là một tập hợp các quyết định mà nếu làm không đúng sẽ là nguyên nhân làm cho dự án của bạn thất bại" (Eoin Woods)

IEEE Std (Standard) 1471 định nghĩa kiến trúc hệ thống là "tổ chức cơ bản của một hệ thống thể hiện trong các phần tử của nó, mối quan hệ của chúng với nhau và với môi trường và các nguyên tắc hướng dẫn thiết kế và phát triển của nó" (Maier, Emery, Hilliard).

Theo Martin Fowler cũng đã chỉ ra một số chủ đề phổ biến khi giải thích về kiến trúc như sau: việc phân rã hệ thống ở mức cao thành các bộ phận, những quyết định khó thay đổi, nhiều kiến trúc trong hệ thống, cái gì có thể thay đổi trong vòng đời của hệ thống…

Trong các định nghĩa này, các phần tử kiến trúc có thể là một mô-đun, hệ thống con, đối tượng, hoặc phần mềm nhị phân (binary) như là một thành phần DLL, một thành phần JavaBean, EJB, CORBA, hoặc web, hoặc thậm chí là cả một hệ thống. Trong cuốn sách này, chúng tôi sử dụng "phần tử - element" để chỉ các đơn vị chung của kiến trúc phần mềm, và chúng tôi sử dụng "thành phần - component" làm từ đồng nghĩa của nó trong các cuộc thảo luận liên quan đến kiến trúc phần mềm.

1. Mô tả đề bài

Để minh họa tính linh hoạt và sức mạnh của phong cách kiến trúc dựa trên dịch vụ, hãy xem xét ví dụ thực tế về hệ thống tái chế điện tử được sử dụng để tái chế các thiết bị điện tử cũ (chẳng hạn như điện thoại di động iPhone hoặc Galaxy). Quy trình xử lý tái chế các thiết bị điện tử cũ hoạt động như sau: đầu tiên, khách hàng hỏi công ty (thông qua trang web hoặc ki-ốt) họ có thể nhận được bao nhiêu tiền cho thiết bị điện tử cũ (gọi là báo giá). Nếu hài lòng, khách hàng sẽ gửi thiết bị điện tử đến công ty tái chế, công ty tái chế sẽ nhận lại thiết bị vật lý (gọi là nhận). Sau khi nhận được, công ty tái chế sẽ đánh giá thiết bị để xác định xem thiết bị có ở tình trạng hoạt động tốt hay không (gọi là đánh giá). Nếu thiết bị hoạt động tốt, công ty sẽ gửi cho khách hàng số tiền đã hứa mua thiết bị (gọi là kế toán). Thông qua quá trình này, khách hàng có thể vào website bất cứ lúc nào để kiểm tra tình trạng của mặt hàng (gọi là tình trạng mặt hàng). Dựa trên đánh giá, thiết bị sau đó sẽ được tái chế bằng cách tiêu hủy một cách an toàn hoặc bán lại (gọi là tái chế). Cuối cùng, công ty định kỳ thực hiện các báo cáo tài chính và hoạt động theo lịch trình và đột xuất dựa trên hoạt động tái chế (được gọi là báo cáo).

1. Chọn kiến trúc

1.Chọn kiến trúc

* Layered Architecture (Kiến trúc phân tầng)

2.Lý do chọn kiến trúc

Là một kiểu kiến trúc phần mềm phổ biến trong phát triển ứng dụng với lí do được chọn để áp dụng vào dự án:

+ Các lớp có thể được sử dụng lại trong các dự án khác nhau mà không cần phải viết lại mã.

+ Các tầng có thể được thay đổi hoặc thêm mới mà không ảnh hưởng đến các tầng khác, giúp dễ dàng mở rộng hệ thống.

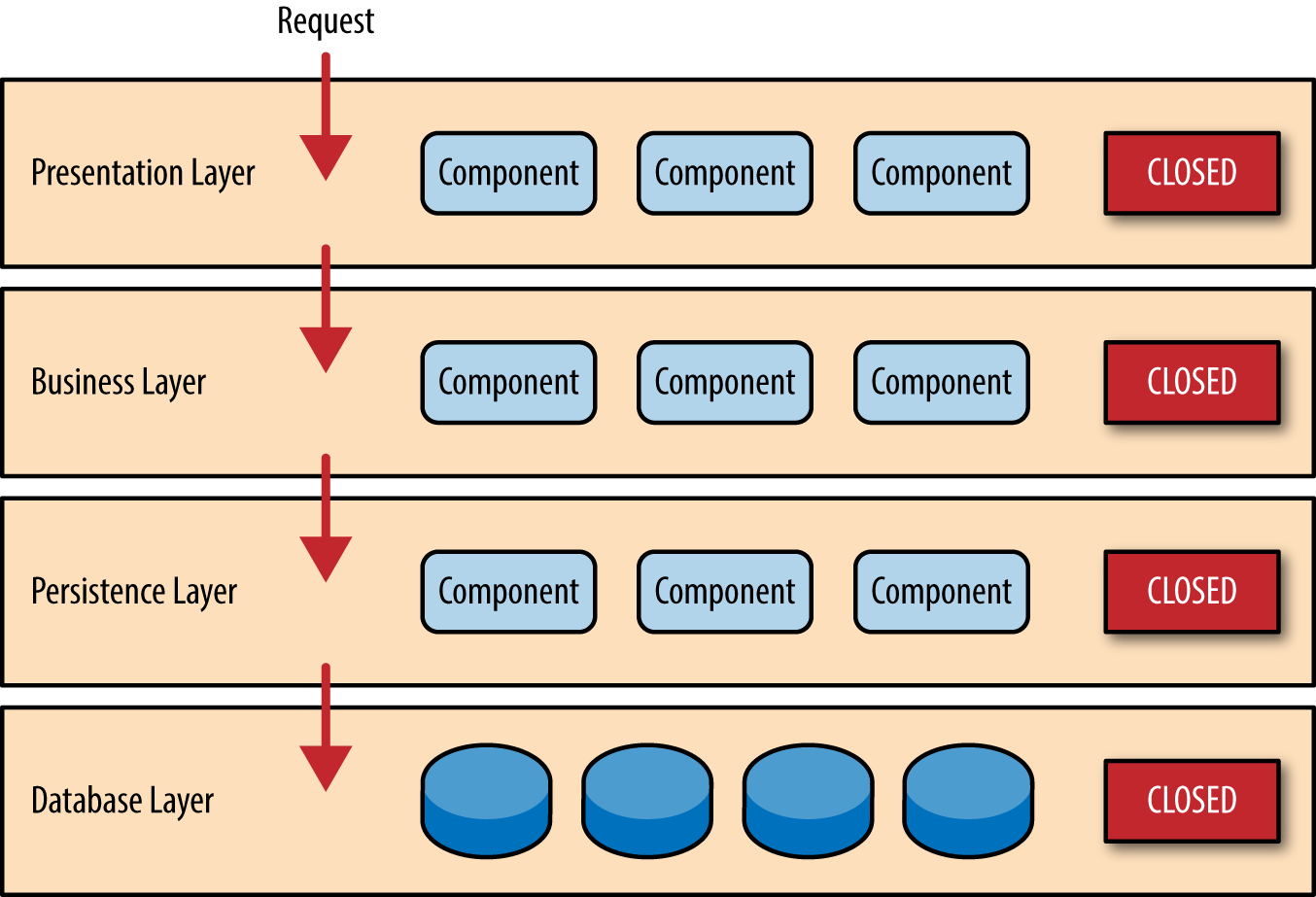
+ Các tầng riêng biệt giúp tách rời các phần của hệ thống, làm cho mã dễ bảo trì và phát triển lại.

+Các tầng với trách nhiệm độc lập giúp quản lý bảo mật hiệu quả hơn.

1. Trình bày kiến trúc

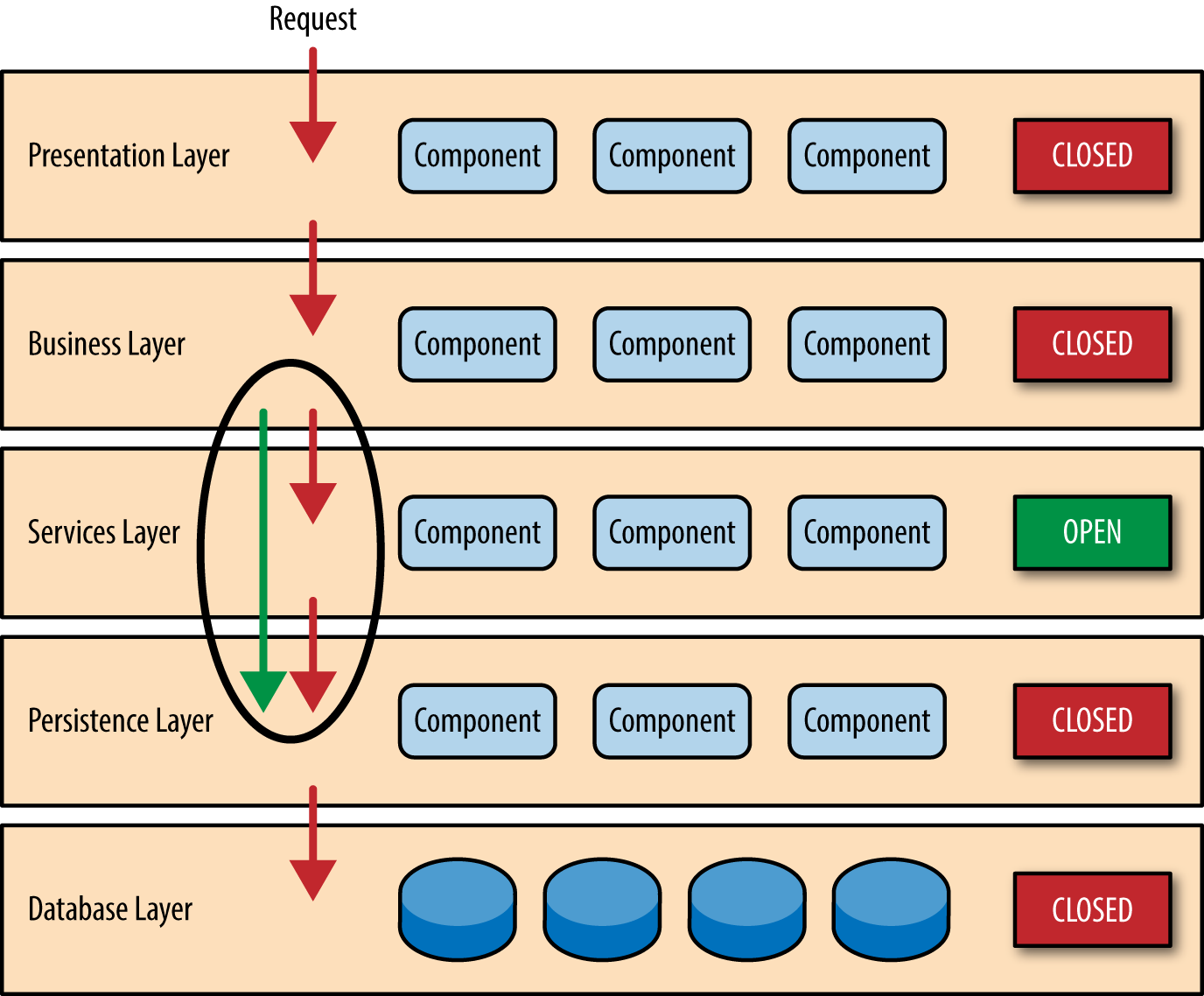
# 1.1 Khái niệm

* Kiến trúc phân tầng(hay còn được gọi là kiến trúc n - tier ) là kiến trúc phổ biến nhất. Kiến trúc này được xem là chuẩn không chính thức cho các ứng dụng Java EE và được biết đến rộng rãi bởi hầu hết kiến trúc sư, nhà thiết kế và nhà phát triển. Kiến trúc này quen thuộc với các cơ cấu tổ chức và truyền thông CNTT truyền thống, được tìm thấy ở hầu hết các công ty khiến nó trở thành một lựa chọn tự nhiên cho các công ty phát triển ứng dụng doanh nghiệp.
* Các thành phần bên trong kiến trúc phân tầng được tổ chức thành các tầng nằm ngang , mỗi tầng thực hiện một vai trò cụ thể trong ứng dụng, ví dụ như tầng trình diễn(presentation) hay tầng nghiệp vụ (business). Mặc dù kiến trúc phân tầng không quy định số lượng hay các loại tầng phải tồn tại, hầu hết các kiến trúc phân tầng bao gồm 4 tầng: tầng trình diễn(presentation layer), tầng nghiệp vụ(business layer), tầng lưu trữ(persistence layer), tầng cơ sở dữ liệu(database layer).
* Tầng Trình diễn (Presentation Layer):
* Tầng này chịu trách nhiệm cho việc hiển thị thông tin cho người dùng và tương tác với họ thông qua giao diện người dùng.
* Nó thường bao gồm các thành phần như giao diện người dùng (UI), các thành phần đồ họa, mã JavaScript, và các loại tập tin CSS.
* Nhiệm vụ chính là đảm bảo giao diện người dùng được hiển thị một cách đúng đắn và hợp lý, cũng như giao tiếp với các tầng dưới đây để lấy và gửi dữ liệu.
* Tầng Nghiệp vụ (Business Layer):
* Tầng này chứa logic kinh doanh của ứng dụng. Nó xử lý dữ liệu, thực hiện các quy trình kinh doanh, và quản lý các luồng logic của ứng dụng.
* Các thành phần ở đây thường là các lớp và đối tượng mô hình hóa các quy trình kinh doanh và các thực thể trong hệ thống.
* Nhiệm vụ của tầng này là đảm bảo rằng dữ liệu được xử lý một cách chính xác theo logic kinh doanh đã được định nghĩa.
* Tầng Lưu trữ (Persistence Layer):
* Tầng này liên quan đến việc lưu trữ dữ liệu vào và lấy dữ liệu từ các nguồn lưu trữ khác nhau như cơ sở dữ liệu, file hệ thống, hoặc bất kỳ nguồn nào có thể lưu trữ dữ liệu.
* Các thành phần ở đây thường là các lớp và các phương thức để thực hiện các hoạt động CRUD (Create, Read, Update, Delete) với dữ liệu.
* Nhiệm vụ chính của tầng này là đảm bảo rằng dữ liệu được lưu trữ và truy xuất một cách hiệu quả và đáng tin cậy.
* Tầng Cơ sở dữ liệu (Database Layer):
* Tầng này là nơi thực sự lưu trữ dữ liệu. Đây có thể là các hệ quản trị cơ sở dữ liệu (DBMS) như MySQL, PostgreSQL, MongoDB, etc.
* Nhiệm vụ chính của tầng này là quản lý các cơ sở dữ liệu, bao gồm tạo, xóa, cập nhật cơ sở dữ liệu và thực hiện các truy vấn để lấy dữ liệu cần thiết cho ứng dụng.
* Lưu ý trong [Hình 1-2](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html#sapr_0102_img) rằng mỗi lớp trong kiến trúc được đánh dấu là đã *đóng*. Đây là một khái niệm rất quan trọng trong mô hình kiến trúc phân lớp. Một lớp đóng có nghĩa là khi một yêu cầu di chuyển từ lớp này sang lớp khác, nó phải đi qua lớp ngay bên dưới nó để đến lớp tiếp theo bên dưới lớp đó. Ví dụ: một yêu cầu bắt nguồn từ lớp trình bày trước tiên phải đi qua lớp nghiệp vụ và sau đó đến lớp bền bỉ trước khi cuối cùng nhấn lớp cơ sở dữ liệu.



Hình 1-2. Lớp đóng và yêu cầu quyền truy cập

* Vậy tại sao không cho phép lớp trình bày truy cập trực tiếp vào lớp bền bỉ hoặc lớp cơ sở dữ liệu? Xét cho cùng, truy cập cơ sở dữ liệu trực tiếp từ lớp trình bày nhanh hơn nhiều so với việc đi qua một loạt các lớp không cần thiết chỉ để truy xuất hoặc lưu thông tin cơ sở dữ liệu. Câu trả lời cho câu hỏi này nằm trong một khái niệm chính được gọi là *các lớp cô lập*.
* Các lớp của khái niệm cô lập có nghĩa là những thay đổi được thực hiện trong một lớp của kiến trúc thường không ảnh hưởng hoặc ảnh hưởng đến các thành phần trong các lớp khác: sự thay đổi được cô lập với các thành phần trong lớp đó và có thể là một lớp liên quan khác (chẳng hạn như lớp bền bỉ chứa SQL). Nếu bạn cho phép lớp trình bày truy cập trực tiếp vào lớp bền vững, thì những thay đổi được thực hiện đối với SQL trong lớp bền bỉ sẽ ảnh hưởng đến cả lớp nghiệp vụ và lớp trình bày, do đó tạo ra một ứng dụng được kết hợp rất chặt chẽ với nhiều sự phụ thuộc lẫn nhau giữa các thành phần. Kiểu kiến trúc này sau đó trở nên rất khó và tốn kém để thay đổi.
* Các lớp của khái niệm cách ly cũng có nghĩa là mỗi lớp độc lập với các lớp khác, do đó có ít hoặc không có kiến thức về hoạt động bên trong của các lớp khác trong kiến trúc. Để hiểu được sức mạnh và tầm quan trọng của khái niệm này, hãy xem xét một nỗ lực tái cấu trúc lớn để chuyển đổi khung trình bày từ JSP (Java Server Pages) sang JSF (Java Server Faces). Giả sử rằng các hợp đồng (ví dụ: mô hình) được sử dụng giữa lớp trình bày và lớp kinh doanh vẫn giữ nguyên, lớp kinh doanh không bị ảnh hưởng bởi việc tái cấu trúc và vẫn hoàn toàn độc lập với loại khung giao diện người dùng được sử dụng bởi lớp trình bày.
* Trong khi các lớp kín tạo điều kiện cho các lớp cách ly và do đó giúp cô lập sự thay đổi trong kiến trúc, có những lúc nó có ý nghĩa đối với các lớp nhất định được mở. Ví dụ: giả sử bạn muốn thêm một lớp dịch vụ chia sẻ vào kiến trúc chứa các thành phần dịch vụ phổ biến được truy cập bởi các thành phần trong lớp nghiệp vụ (ví dụ: các lớp tiện ích dữ liệu và chuỗi hoặc các lớp kiểm tra và ghi nhật ký). Tạo một lớp dịch vụ thường là một ý tưởng hay trong trường hợp này vì về mặt kiến trúc, nó hạn chế quyền truy cập vào các dịch vụ được chia sẻ cho lớp doanh nghiệp (chứ không phải lớp trình bày). Nếu không có một lớp riêng biệt, không có gì về mặt kiến trúc hạn chế lớp trình bày truy cập vào các dịch vụ phổ biến này, gây khó khăn cho việc quản lý hạn chế truy cập này.
* Trong ví dụ này, lớp dịch vụ mới có thể sẽ nằm *bên dưới* lớp nghiệp vụ để chỉ ra rằng các thành phần trong lớp dịch vụ này không thể truy cập được từ lớp trình bày. Tuy nhiên, điều này trình bày một vấn đề ở chỗ lớp nghiệp vụ bây giờ được yêu cầu phải đi qua lớp dịch vụ để đến lớp bền bỉ, điều này không có ý nghĩa gì cả. Đây là một vấn đề lâu đời với kiến trúc phân lớp, và được giải quyết bằng cách tạo ra các lớp mở trong kiến trúc.
* Như minh họa trong [Hình 1-3](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html#sapr_0103_img), lớp dịch vụ trong trường hợp này được đánh dấu là mở, có nghĩa là các yêu cầu được phép bỏ qua lớp mở này và đi trực tiếp đến lớp bên dưới nó. Trong ví dụ sau, vì lớp dịch vụ là mở, lớp nghiệp vụ bây giờ được phép bỏ qua nó và đi trực tiếp đến lớp bền vững, điều này hoàn toàn hợp lý.



###### Hình 1-3. Mở các lớp và luồng yêu cầu

* Tận dụng khái niệm về các lớp mở và đóng giúp xác định mối quan hệ giữa các lớp kiến trúc và luồng yêu cầu và cũng cung cấp cho các nhà thiết kế và nhà phát triển thông tin cần thiết để hiểu các hạn chế truy cập lớp khác nhau trong kiến trúc. Việc không ghi lại hoặc truyền đạt đúng lớp nào trong kiến trúc là mở và đóng (và tại sao) thường dẫn đến các kiến trúc được kết hợp chặt chẽ và giòn rất khó kiểm tra, bảo trì và triển khai.

# 2.Ưu điểm

* Các lớp có thể được sử dụng lại trong các dự án khác nhau mà không cần phải viết lại mã.
* Các tầng có thể được thay đổi hoặc thêm mới mà không ảnh hưởng đến các tầng khác, giúp dễ dàng mở rộng hệ thống.
* Các tầng riêng biệt giúp tách rời các phần của hệ thống, làm cho mã dễ bảo trì và phát triển lại.
* Các tầng với trách nhiệm độc lập giúp quản lý bảo mật hiệu quả hơn.

# 3.Nhược điểm

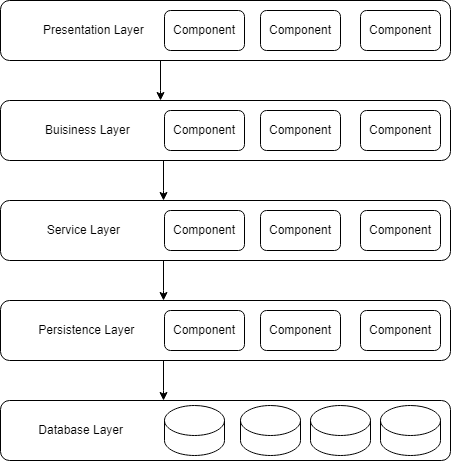
* Khó khăn trong việc thay đổi : Khi có yêu cầu thay đổi, đặc biệt là ở tầng logic ứng dụng, có thể cần phải thay đổi ở nhiều lớp hoặc module, điều này có thể gây ra sự phức tạp và rủi ro về tính ổn định của hệ thống.
* Khả năng tái sử dụng thấp : Một số logic xử lý có thể bị ràng buộc vào tầng cụ thể, khiến cho việc tái sử dụng code trở nên khó khăn. Điều này có thể dẫn đến việc lặp lại code hoặc khó khăn trong việc chia sẻ logic giữa các ứng dụng khác nhau.
* Rủi ro về hiệu suất:Trong một số trường hợp, việc gọi qua lại giữa các lớp có thể làm giảm hiệu suất của hệ thống, đặc biệt là khi có quá nhiều lớp hoặc khi truy vấn dữ liệu phải đi qua nhiều tầng.
* Khó khăn trong việc kiểm thử và gỡ lỗi : Việc tách biệt các thành phần thành các lớp riêng biệt có thể làm tăng độ phức tạp của quá trình kiểm thử và gỡ lỗi. Đặc biệt là khi có sự phụ thuộc mạnh mẽ giữa các lớp và tầng.

# 4.Cân nhắc

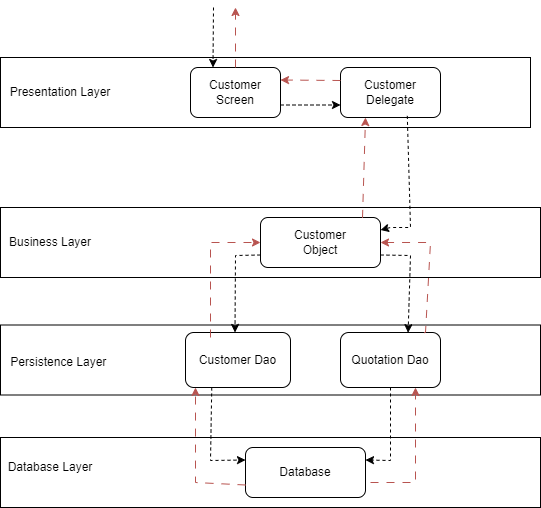
* Mẫu kiến trúc phân lớp là một mẫu có mục đích chung vững chắc, làm cho nó trở thành điểm khởi đầu tốt cho hầu hết các ứng dụng, đặc biệt khi bạn không chắc chắn mẫu kiến trúc nào phù hợp nhất cho ứng dụng của mình. Tuy nhiên, có một vài điều cần xem xét từ quan điểm kiến trúc khi chọn mẫu này.
* Điều đầu tiên cần chú ý là những gì được gọi là *kiến trúc hố sụt chống lại mô hình*. Phản mẫu này mô tả tình huống trong đó các yêu cầu chảy qua nhiều lớp của kiến trúc dưới dạng xử lý truyền qua đơn giản với rất ít hoặc không có logic được thực hiện trong mỗi lớp. Ví dụ: giả sử lớp trình bày phản hồi yêu cầu từ người dùng để truy xuất dữ liệu khách hàng. Lớp trình bày chuyển yêu cầu đến lớp nghiệp vụ, lớp này chỉ cần chuyển yêu cầu đến lớp bền vững, sau đó thực hiện một lệnh gọi SQL đơn giản đến lớp cơ sở dữ liệu để truy xuất dữ liệu khách hàng. Dữ liệu sau đó được chuyển tất cả các cách sao lưu lên ngăn xếp mà không cần xử lý hoặc logic bổ sung để tổng hợp, tính toán hoặc chuyển đổi dữ liệu.
* Mỗi kiến trúc phân lớp sẽ có ít nhất một số kịch bản rơi vào hố sụt kiến trúc chống lại mô hình. Tuy nhiên, chìa khóa là phân tích tỷ lệ phần trăm yêu cầu thuộc danh mục này. Quy tắc 80-20 thường là một thực hành tốt để tuân theo để xác định xem bạn có đang gặp phải mô hình hố sụt kiến trúc hay không. Thông thường, có khoảng 20 phần trăm các yêu cầu dưới dạng xử lý chuyển qua đơn giản và 80 phần trăm các yêu cầu có một số logic nghiệp vụ được liên kết với yêu cầu. Tuy nhiên, nếu bạn thấy rằng tỷ lệ này bị đảo ngược và phần lớn các yêu cầu của bạn là xử lý chuyển qua đơn giản, bạn có thể muốn xem xét làm cho một số lớp kiến trúc mở, hãy nhớ rằng sẽ khó kiểm soát thay đổi hơn do thiếu cách ly lớp.
* Một cân nhắc khác với mẫu kiến trúc phân lớp là nó có xu hướng cho vay đối với các ứng dụng nguyên khối, ngay cả khi bạn chia lớp trình bày và các lớp kinh doanh thành các đơn vị có thể triển khai riêng biệt. Mặc dù điều này có thể không phải là mối quan tâm đối với một số ứng dụng, nhưng nó đặt ra một số vấn đề tiềm ẩn về triển khai, độ mạnh mẽ và độ tin cậy chung, hiệu suất và khả năng mở rộng.

1. Thiết kế

# 1.Thiết kế vẽ



# 2.Thiết kế chi tiết



* Màn hình khách hàng có nhiệm vụ tiếp nhận yêu cầu và hiển thị thông tin về sự đánh giá thiết bị của khách hàng bên phía công ty.Nó không biết dữ liệu lấy ra ở đâu , được lấy ra như thế nào hoặc phải truy vấn bao nhiêu bảng cơ sở dữ liệu để lấy ra dữ liệu.Khi màn hình khách hàng yêu cầu lấy thông tin đánh giá về một cá nhân cụ thể nào đó , nó được chuyển tiếp đến mô-đun đại biểu khách hàng .Mô-đun này chịu trách nhiệm biết mô-đun nào trong lớp nghiệp vụ có thể xử lý yêu cầu đó cũng như cách truy cập mô-đun đó và dữ liệu nào nó cần. Đối *tượng khách hàng* trong lớp nghiệp vụ có trách nhiệm tổng hợp tất cả thông tin cần thiết theo yêu cầu nghiệp vụ (trong trường hợp này là để lấy thông tin đánh giá thiết bị của khách hàng).Mô-đun này gọi mô- *đun dao khách hàng* (đối tượng truy cập dữ liệu) trong lớp lưu trữ lâu dài để lấy dữ liệu về đánh giá thiết bị và mô- *đun dao đánh giá* để lấy thông tin về đánh giá một thiết bị của khách hàng.Các mô-đun này lần lược thực thi các câu lệnh SQL để lấy dữ liệu tương ứng và chuyển nó trở lại đối tượng khách hàng trong lớp nghiệp vụ.Khi đối tượng khách hàng nhận được dữ liệu , nó sẽ tổng hợp dữ liệu và chuyển thông tin đó trở lại cho đại biểu ,sau đó chuyển dữ liệu đó đến màn hình khách hàng để hiển thị cho người dùng
* Từ góc độ công nghệ , có hàng tá cách để triển khai mô-đun này. Ví dụ: Trong nền tảng Java , màn hình khách hàng có thể là màn hình Java Server Faces (JSF) được kết hợp với đại biểu khách hàng dưới dạng thành phần Bean được quản lý.Đối tượng khách hàng trong lớp nghiệp vụ có thể là một Spring Bean cục bộ hoặc Bean EJB3 từ xa . Các đối tượng truy cập dữ liệu được minh họa ví dụ trước có thể được triển khai dưới dạng POJO đơn giản (Đối tượng Java thuần túy),tệp MyBatis XML Mapper hoặc thậm chí các đối tượng đóng gói lệnh gọi JDBC thô hoặc truy vấn Hibernate. Từ góc độ nền tảng của Microsoft, màn hình khách hàng có thể là mô-đun ASP (trang máy chủ hoạt động) sử dụng .NET framework để truy cập các mô-đun C# trong lớp nghiệp vụ, với các mô-đun truy cập dữ liệu khách hàng và đơn đặt hàng được triển khai dưới dạng ADO (Đối tượng dữ liệu ActiveX).

# 3.Thiết kế tổng quan

### Tổng quan

* Kiến trúc phân tầng, hay còn gọi là kiến trúc n-tier, được lựa chọn cho hệ thống tái chế điện tử vì sự linh hoạt, khả năng mở rộng, và dễ bảo trì. Hệ thống sẽ được chia thành bốn tầng chính: Tầng trình diễn (Presentation Layer), Tầng nghiệp vụ (Business Layer), Tầng lưu trữ (Persistence Layer), và Tầng cơ sở dữ liệu (Database Layer).

### Sơ đồ Kiến trúc

### Tầng Trình diễn (Presentation Layer)

* Thành phần: Giao diện người dùng, HTML, CSS, JavaScript, AJAX.
* Chức năng: Hiển thị thông tin và tương tác với người dùng. Gửi yêu cầu và nhận phản hồi từ Tầng nghiệp vụ.

### Tầng Nghiệp vụ (Business Layer)

* Thành phần: Các dịch vụ nghiệp vụ
* Chức năng: Xử lý logic nghiệp vụ, quản lý luồng dữ liệu giữa Tầng trình diễn và Tầng lưu trữ.

### Tầng Lưu trữ (Persistence Layer)

* Thành phần: DAO (Data Access Objects),
* Chức năng: Quản lý việc truy cập và lưu trữ dữ liệu. Thực hiện các thao tác CRUD với cơ sở dữ liệu.

### Tầng Cơ sở dữ liệu (Database Layer)

* Thành phần: Hệ quản trị cơ sở dữ liệu SQL Server
* Chức năng: Lưu trữ dữ liệu thực tế. Xử lý các truy vấn và thao tác dữ liệu.

## a.Ví dụ Cụ thể:

Để minh họa cách hệ thống tái chế điện tử hoạt động, chúng ta sẽ sử dụng một quy trình mẫu:

#### Báo giá (Quote):

* Khách hàng truy cập trang web hoặc ki-ốt và yêu cầu báo giá cho thiết bị điện tử cũ của họ.
* Yêu cầu này sẽ được gửi từ Tầng trình diễn đến Tầng nghiệp vụ.
* Tầng nghiệp vụ sẽ xử lý yêu cầu và gọi các phương thức trong Tầng lưu trữ để truy xuất dữ liệu cần thiết từ Tầng cơ sở dữ liệu.
* Kết quả báo giá sẽ được gửi ngược lại Tầng trình diễn để hiển thị cho khách hàng.

#### Nhận (Receive):

* Nếu khách hàng chấp nhận báo giá, họ sẽ gửi thiết bị đến công ty tái chế.
* Thông tin về việc nhận thiết bị sẽ được cập nhật trong hệ thống thông qua Tầng trình diễn.
* Tầng nghiệp vụ sẽ xử lý việc cập nhật này và lưu trữ thông tin vào cơ sở dữ liệu thông qua Tầng lưu trữ.

#### Đánh giá (Evaluate):

* Sau khi nhận được thiết bị, công ty tái chế sẽ đánh giá tình trạng của thiết bị.
* Kết quả đánh giá sẽ được nhập vào hệ thống qua Tầng trình diễn và lưu trữ vào cơ sở dữ liệu qua Tầng nghiệp vụ và Tầng lưu trữ.

#### Kế toán (Accounting):

* Nếu thiết bị hoạt động tốt, hệ thống sẽ xử lý việc thanh toán cho khách hàng.
* Thông tin thanh toán sẽ được quản lý bởi Tầng nghiệp vụ và cập nhật vào cơ sở dữ liệu.

#### Tái chế (Recycle):

* Dựa trên kết quả đánh giá, thiết bị sẽ được tái chế hoặc bán lại.
* Quá trình này sẽ được quản lý bởi Tầng nghiệp vụ và cập nhật vào cơ sở dữ liệu.

#### Báo cáo (Reporting):

* Hệ thống sẽ tạo các báo cáo tài chính và hoạt động dựa trên dữ liệu thu thập từ quá trình tái chế.
* Các báo cáo này sẽ được tạo ra bởi Tầng nghiệp vụ và có thể được truy xuất từ Tầng trình diễn.
* Công nghệ và Công cụ Sử dụng
* Front-end:Java Swing
* Back-end: JDBC lưu trữ dữ liệu.
* Database: SQL Server
* Frameworks và Libraries:sqljdbc4-3.0.jar

## b. Quy trình Tương tác giữa Các Tầng

#### Gửi Yêu cầu:

* Người dùng gửi yêu cầu từ giao diện người dùng (UI).
* Yêu cầu được gửi đến Tầng nghiệp vụ thông qua các API

#### Xử lý Logic Nghiệp vụ:

* Tầng nghiệp vụ nhận yêu cầu và xử lý logic nghiệp vụ.
* Nếu cần dữ liệu, tầng này sẽ gọi các phương thức từ Tầng lưu trữ.

#### Truy xuất và Lưu trữ Dữ liệu:

* Tầng lưu trữ thực hiện các thao tác truy xuất hoặc lưu trữ dữ liệu thông qua các DAO (Data Access Objects)

Các thao tác này tương tác trực tiếp với cơ sở dữ liệu.

#### Trả Kết quả:

* Dữ liệu hoặc kết quả từ Tầng lưu trữ được gửi lại cho Tầng nghiệp vụ.

Tầng nghiệp vụ tổng hợp và gửi kết quả trở lại Tầng trình diễn.

#### Hiển thị Kết quả:

* Kết quả cuối cùng được Tầng trình diễn nhận và hiển thị cho người dùng.

## c. Các Quyết định Thiết kế Quan trọng

* Tách biệt các Tầng: Mỗi tầng có trách nhiệm riêng biệt, giúp dễ dàng bảo trì và mở rộng hệ thống.
* API : Sử dụng API giúp tương tác giữa các tầng và với hệ thống bên ngoài một cách linh hoạt.

Kết luận

* Kiến trúc phân tầng cung cấp một khung tổ chức rõ ràng và hiệu quả cho hệ thống tái chế điện tử. Bằng cách tách biệt các tầng với trách nhiệm riêng biệt, hệ thống dễ bảo trì, mở rộng và bảo mật. Quy trình tương tác giữa các tầng được thiết kế để đảm bảo rằng mỗi yêu cầu của người dùng được xử lý một cách hiệu quả và chính xác, từ khi gửi yêu cầu đến khi nhận kết quả.