[1. Kiến trúc phần mềm là gì? Tầm quan trọng của nó? 2](#_Toc6817655)

[2. Mục đích và tầm quan trọng 2](#_Toc6817656)

[3. Mô hình (4+1) quan điểm cho kiến trúc pm 3](#_Toc6817657)

[4. 4 view+1 là như thế nào và các biểu đồ liên quan 3](#_Toc6817658)

[5. Các thuộc tính chất lượng 4](#_Toc6817659)

[6. Câu 1: MVC? Khó khăn? So sánh kiến trúc này với kiến trúc 3 Layer? 6](#_Toc6817660)

[7. Câu 2: Microservirces. So sánh Microservirces - Component Based Arc? 7](#_Toc6817661)

[8. Câu 3 3 Layer? So sánh kiến trúc này với kiến trúc tier? 10](#_Toc6817662)

[9. Câu 4: Mô hình Client – Server 12](#_Toc6817663)

[10. Câu 5 Mô hình MVP 14](#_Toc6817664)

[11. Câu 6 Mô hình SOA 14](#_Toc6817665)

[12. So sánh mô hình SOA với Microservice 17](#_Toc6817666)

**Hiệp**

**ĐỀ CƯƠNG MÔN KIẾN TRÚC PHẦN MỀM HIỆN ĐẠI**

**PHẦN 1:TỔNG QUAN**

1. Kiến trúc phần mềm là gì? Tầm quan trọng của nó?

* Là các cấu trúc của hệ thống đc tạo nên bởi: các cấu phần (thành phần, hệ thống con) và các mỗi liên hệ giữa các cấu phần
* VD: hệ thống thư viện

+ các cấu phần: cấu phần quản lý tài liệu(1), cấu phần phục vụ bạn đọc(2), cấu phần quan hệ giao dịch với các thư viện khác(3),...

+ quan hệ giữa các cấu phần: (2) với các cấu phần khác, (3) với các cấu phần khác, ...

* Ta có thể thiết kế kiến trúc phần mềm theo 2 cấp độ: nhỏ và lớn.
* Kiến trúc nhỏ liên quan đến các kiến trúc của chương trình máy tính riêng biệt. Tại mức này ta quan tâm phân rã một chương trình cụ thể thành các thành phần.
* Kiến trúc lớn liên quan đến các hệ thống phức hợp chứa các hệ thống khác, các chương trình và các cấu phần của chương trình. Các hệ thống này phân tán trên nhiều máy tính được nhiều đơn vị khác nhau quản lí.
* Trong quá trình thiết kế kiến trúc, mỗi cấu phần có thể được phân rã thành các cấu phần nhỏ hơn.
* Các cấu phần nhỏ còn có thể được phân rã tiếp thành các chi tiết, các đơn vị => quy trình này gọi là phân rã hoặc làm mịn thiết kế hệ thống.
* Tiếp cận này được gọi là tiếp cận trên xuống, tức là đi từ đại thể đến chi tiết, từ trừu tượng mức cao đến chi tiết.
* Thí dụ Cấu phần phục vụ bạn đọc được chi tiết hoá thành các cấu phần:
* Hướng dẫn đăng nhập;
* Hướng dẫn tìm kiếm;
* Trợ giúp (bạn đọc);
* Thống kê sở thích (bạn đọc);
* Đòi tài liệu quá hạn;

1. Mục đích và tầm quan trọng

* **Mục đích**
* là quy trình xác định các cấu phần và các **mối quan hệ giữa các cấu phần** trong hệ thống phần mềm.
* đòi hỏi sự thấu hiểu về hệ thống.
* Mục tiêu cuối cùng là xây dựng một cấu trúc tổng thể cho hệ thống.
* Là giai đoạn đầu tiên của một quy trình thiết kế hệ thống
* tiến hành song song với các hoạt động đặc tả.
* Đầu vào là các yêu cầu đối với hệ thống.
* Đầu ra là mô hình kiến trúc phản ánh tổ chức của hệ thống như một tập các cấu phần có liên hệ với nhau. Toàn bộ sản phẩm đầu ra tạo thành một bộ hồ sơ thiết kế kiến trúc.
* Tầm quan trọng
* Vì nó phản ánh **tính hiệu quả, tính tin cậy, tính khả chuyển** của hệ thống.
* **Cho phép tái sử dụng ở mức rộng:** Một mô hình kiến trúc hệ thống là một bản mô tả các cấu trúc với các cấu phần và các tương tác giữa các cấu phần đó.
* Khi gặp 1 số hệ thống có cấu trúc giống có thể tận dụng lại để phát triển cả một dòng sản phẩm.
* **VD** Các hệ thống phần mềm sau đây đều có chung một kiến trúc:
* Các hệ điều hành trên máy tính cá nhân.
* Các máy điện thọai di động.
* Các hệ thống quản lí bán hàng.
* Các hệ thống quản lí học tập
* **Mỗi cấu phần** thể hiện **một yêu cầu chức năng** của hệ thống.
* Các yêu cầu phi chức năng phụ thuộc phương thức tổ chức, vận dụng, khai thác cấu phần đó.
* các yêu cầu phi chức năng cũng được chi phối bởi các cấu phần riêng, nhưng kiến trúc của hệ thống gây ảnh hưởng quyết định.
* Kiến trúc phần mềm biểu thị các quyết định thiết kế dành cho hệ thống =>**stackholders** sẽ có cái nhìn tổng quan, sớm hơn về hệ thống ngay từ khi nó còn sơ khai. Từ đó sẽ có các đóng góp ý, đề xuất, phản bác của mình ngay từ đầu. Nếu không có quyết định sớm, thì khi phần mềm đã được xây dựng hoàn chỉnh hoặc khá hoàn chỉnh mà đột ngột xuất hiện các yêu cầu thay đổi từ phía nhóm này hoặc nhóm khác, ngoài việc **gây trì trệ** cho tiến độ công việc mà còn có thể gây ra **tâm lý căng thẳng, mâu thuẫn** giữa stackholders

1. Mô hình (4+1) quan điểm cho kiến trúc pm
   * **Quan điểm logic**: giúp thể hiện các đối tượng và lớp trừu tượng cơ bản trong hệ thống, thể hiện các yêu cầu về thực thể dưới dạng các quan niệm logic.
   * **Quan điểm tiến trình**. Vào thời điểm hệ thống hoạt động ta phải hình dung rõ các qui trình tương tác của hệ thống. Khi tuân thủ quan điểm tiến trình, điều quan trọng là phải hiểu được các yêu cầu phi chức năng như tính hiệu năng hoặc tính hữu dụng.
   * **Quan điểm phát triển**: giải thích được hệ thống sẽ được phân rã ra sao trong quá trình phát triển. Hữu ích đối với quản trị và lập trình viên.
   * **Quan điểm vật lí**: giúp xác định được phần cứng và phần mềm tương ứng trong hệ thống. Dành cho các kĩ sư hệ thống.
2. 4 view+1 là như thế nào và các biểu đồ liên quan

**?Vì sao gọi 4+1 mà k phải 5:** Vì góc nhìn UC view nằm giữa chi phối các góc nhìn còn lại. Và nhằm nhấn mạnh vai trò của nó

**PHYSICAL/COMPONENT VIEW (vật lý)**

Biểu đồ thành phần

**LOGICAL VIEW (logic)**

Lớp, trạng thái

**DEPLOYMENT VIEW**

**(triển khai)**

Biểu đồ triển khai

**PROCESS VIEW (truy cập)**

Trình tự, hoạt động

Sơ đồ hoạt động

* ***Use Case View*:** cung cấp góc nhìn về các ca sử dụng giúp chúng ta hiểu hệ thống có gì? ai dùng và dùng nó như thế nào.

Dành cho: tất cả stackholders (cả ng dùng cuối)

* ***Logical View*:** cung cấp góc nhìn về cấu trúc hệ thống, xem nó được tổ chức như thế nào. Bên trong nó có gì.

Dành cho: designer

* ***Process View:***cung cấp góc nhìn động về hệ thống, xem các thành phần trong hệ thống tương tác với nhau như thế nào.

Dành cho: Integrators

* ***Component View:***Cũng là một góc nhìn về cấu trúc giúp chúng ta hiểu cách phân bổ và sử dụng lại các thành phần trong hệ thống ra sao.

*Dành cho:* *Lập trình viên*

* ***Deployment View*:** cung cấp góc nhìn về triển khai hệ thống, nó cũng ảnh hưởng lớn đến kiến trúc hệ thống.

*Dành cho:* Deployment managers

1. Các thuộc tính chất lượng
   * + - **Chất lượng thiết kế:**

* **Tính toàn vẹn về khái niệm:** xác định tính nhất quán và sự gắn kết của thiết kế tổng thể. Điều này bao gồm cách các thành phần hoặc mô-đun được thiết kế, cũng như các yếu tố như kiểu mã hóa và đặt tên biến.
* **Bảo trì:** khả năng của hệ thống trải qua những thay đổi với mức độ dễ dàng. Những thay đổi này có thể ảnh hưởng đến các thành phần, dịch vụ, tính năng và giao diện khi thêm hoặc thay đổi chức năng, sửa lỗi và đáp ứng các yêu cầu kinh doanh mới.
* **Tái sử dụng:** xác định khả năng cho các thành phần và hệ thống con phù hợp để sử dụng trong các ứng dụng khác và trong các tình huống khác. Khả năng sử dụng lại giảm thiểu việc sao chép các thành phần và thời gian thực hiện.
  + - * **Chất lượng thực thi**
* **Tính ứng dụng**: xác định tỷ lệ thời gian mà hệ thống hoạt động và làm việc. Nó có thể được đo bằng tỷ lệ phần trăm của tổng thời gian ngừng hoạt động của hệ thống trong một khoảng thời gian được xác định trước. Tính khả dụng sẽ bị ảnh hưởng bởi lỗi hệ thống, sự cố cơ sở hạ tầng, các cuộc tấn công độc hại và tải hệ thống.
* **Khả năng tương tác:** khả năng của một hệ thống hoặc các hệ thống khác nhau hoạt động thành công bằng cách giao tiếp và trao đổi thông tin với các hệ thống bên ngoài khác được viết và điều hành bởi các bên ngoài. Một hệ thống có thể tương tác làm cho việc trao đổi và tái sử dụng thông tin trong nội bộ cũng như bên ngoài dễ dàng hơn.
* **Khả năng quản lý:** xác định mức độ dễ dàng cho quản trị viên hệ thống để quản lý ứng dụng, thường thông qua các công cụ đầy đủ và hữu ích được sử dụng trong các hệ thống giám sát và để gỡ lỗi và điều chỉnh hiệu suất.
* **Hiệu suất:** khả năng đáp ứng của hệ thống để thực hiện bất kỳ hành động nào trong một khoảng thời gian nhất định. Nó có thể được đo lường về độ trễ hoặc thông lượng. Độ trễ là thời gian thực hiện để phản hồi bất kỳ sự kiện nào. Thông lượng là số lượng sự kiện diễn ra trong một khoảng thời gian nhất định.
* **Độ tin cậy:** khả năng của một hệ thống duy trì hoạt động theo thời gian. Đo bằng xác suất mà một hệ thống sẽ không thực hiện các chức năng dự định của nó trong một khoảng thời gian xác định.
* **Khả năng mở rộng:** khả năng của một hệ thống có thể xử lý tăng tải mà không ảnh hưởng đến hiệu suất của hệ thống hoặc khả năng dễ dàng mở rộng.
* **Bảo mật:** khả năng của một hệ thống để ngăn chặn các hành động độc hại hoặc vô tình bên ngoài việc sử dụng được thiết kế và để ngăn chặn tiết lộ hoặc mất thông tin. Một hệ thống an toàn nhằm bảo vệ tài sản và ngăn chặn sửa đổi thông tin trái phép.
  + - * **Chất lượng sử dụng**
* **Khả năng sử dụng:** xác định mức độ ứng dụng đáp ứng yêu cầu của người dùng bằng cách trực quan, dễ bản địa hóa và toàn cầu hóa, cung cấp quyền truy cập tốt cho người dùng bị vô hiệu hóa và mang lại trải nghiệm tốt cho người dùng.
  + - * **Chất lượng hệ thống:**
* **Khả năng hỗ trợ:** khả năng hệ thống cung cấp thông tin hữu ích để xác định và giải quyết các vấn đề khi nó không hoạt động chính xác.
* **Khả năng kiểm thử:** thước đo mức độ dễ dàng để tạo các tiêu chí kiểm tra cho hệ thống và các thành phần và thực hiện các kiểm tra này để xác định xem các tiêu chí có được đáp ứng hay không. Khả năng kiểm tra tốt làm cho nhiều khả năng các lỗi trong hệ thống có thể được cách ly kịp thời và hiệu quả
  + - * **Chất lượng nghiệp vụ**
      * **Chất lượng kiến trúc**

**PHẦN 2: CÁC MÔ HÌNH KIẾN TRÚC**

1. Câu 1: MVC? Khó khăn? So sánh kiến trúc này với kiến trúc 3 Layer?

MVC tách riêng hiển thị dữ liệu và tương tác dữ liệu hệ thống. Hệ thống được tổ chức thành ba cấu phần logic tương tác lẫn nhau:

* **Model:** chứa tất cả các nghiệp vụ logic, phương thức xử lý, truy xuất database, đối tượng mô tả dữ liệu như các Class, hàm xử lý...
* **View** là phần đảm nhận hiển thị chính xác những gì cho người dùng.
* **Controller:** Giữ nhiệm vụ nhận điều hướng các yêu cầu từ người dùng và gọi đúng những phương thức xử lý chúng...
* **Cách làm việc**:
* Khi user tương tác với View bằng cách click vào button, user gửi yêu cầu đi.
* Controller nhận và điều hướng chúng đến phương thức xử lý ở Model.
* Cotroller nhận thông tin và thực thi yêu cầu
* Khi model hoàn tất việc xử lý, View sẽ nhận kết quả và hiển thị lại cho người dùng.
* **Ưu điểm**
* Có tính chuyên nghiệp hóa, có thể chia cho nhiều nhóm được đào tạo nhiều kỹ năng khác nhau, từ thiết kế cho đến lập trình đến tổ chức database.
* Giúp phát triển ứng dụng nhanh, đơn giản, dễ nâng cấp, bảo trì….
* **Khó khăn**:
* Đối với những hệ thống nhỏ gây cồng kềnh , tốn thời gian.
* Tốn thời gian trung chuyển giữa các lớp
* Xây dựng cầu kì, mất nhiều thời gian để xây dựng thư viện và cấu trúc.
* VD biểu đồ trình tự



* **So sánh kiến trúc này với kiến trúc 3 Layer**

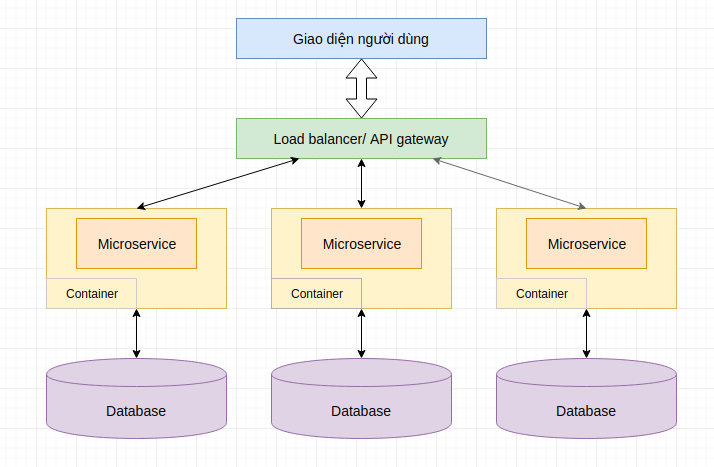
|  |  |
| --- | --- |
| **MVC** | * 1. **Layer** |
| * Sử dụng cho lập trình các **ứng dụng web: php, asp.net** * Các tầng có thể tương tác qua lại với nhau. | * Sử dụng trong lập trình ứng dụng, windowform. * Chỉ có Data Access Layer - Business Logic Layer **hoặc** là Business Logic Layer - Presentation Layers….. |

1. Câu 2: Microservirces. So sánh Microservirces - Component Based Arc?

* **Microservice** là một cách tiếp cận để phát triển ứng dụng, trong đó một ứng dụng lớn được xây dựng như một bộ dịch vụ nhỏ, tự trị.
* Mỗi dịch vụ đều độc lập và nên thực thi một khả năng nghiệp vụ riêng. mỗi dịch vụ là nhỏ, độc lập và kết nối lỏng lẻo
* Mỗi dịch vụ là một mã nguồn riêng biệt, chúng có thể được quản lý bởi nhóm phát triển nhỏ.
* Các dịch vụ có thể được triển khai độc lập. Một nhóm có thể cập nhật sự tồn tại của 1 dịch vụ mà không cần phải building hoặc triển khai lại toàn bộ hệ thống.
* Thay vì chia sẻ một cơ sở dữ liệu như trong ứng dụng Monolithic, mỗi microservice có cơ sở dữ liệu riêng.
* Có cơ sở dữ liệu cho mỗi dịch vụ là điều cần thiết nếu bạn muốn hưởng lợi từ microservice, bởi vì nó đảm bảo khớp nối lỏng lẻo.
* Hơn nữa, một dịch vụ có thể sử dụng một loại cơ sở dữ liệu phù hợp nhất với nhu cầu của nó.
* Các services được viết bằng các ngôn ngữ, frameworks và các phiên bản khác nhau
* Các dịch vụ giao tiếp với nhau bằng việc sử dụng APIs. Các chi tiết thực thi bên trong của mỗi dịch vụ sẽ bị ẩn khỏi các dịch vụ khác.
* Ngoài ra, còn có một số thành phần khác xuất hiện trong kiến trúc Microservices
  + **Management**: thành phần quản lý chịu trách nhiệm đặt các dịch vụ trên các nodes, xác định các lỗi, cân bằng lại các dịch vụ trên các nodes,..
  + **Services Discovery**: Duy trì một danh sách các dịch vụ và các nút mà chúng được đặt trên đó. Cho phép tra cứu dịch vụ để tìm điểm cuối cho các dịch vụ
  + **API Gateway**: API Gateway là điểm vào cho các Clients. Clients không gọi các dịch vụ trực tiếp. Thay vào đó, chúng gọi API Gateway, chuyển tiếp cuộc gọi đến các dịch vụ phù hợp ở mặt sau. Cổng API có thể tổng hợp các phản hồi từ một vài dịch vụ và trả về phản hồi tổng hợp.
  + **VD**

Về ứng dụng thương mại điện tử, bao gồm một số thành phần / mô-đun.

Xác định mỗi module là một dịch vụ được ghép lỏng lẻo tùy thuộc vào yêu cầu, có thể cộng tác với nhau dựa trên ngữ cảnh. Chúng tôi có thể có các dịch vụ sau cho một ứng dụng hoàn chỉnh:

* + Dịch vụ xác minh - Chịu trách nhiệm xác minh cho khách hàng.
  + Dịch vụ đặt hàng - nhận đơn đặt hàng và xử lý nó.
  + Dịch vụ Catalogue - Quản lý sản phẩm và kiểm tra hàng tồn kho sản phẩm.
  + Dịch vụ giỏ hàng - Quản lý giỏ hàng của người dùng, dịch vụ này có thể sử dụng dịch vụ Danh mục làm nguồn dữ liệu.
  + Dịch vụ thanh toán - Quản lý và ủy quyền thanh toán.
  + Dịch vụ vận chuyển – vận chuyển đặt hàng sản phẩm..
* **Ưu điểm của Microservices**
  + **Triển khai độc lập**: có thể cập nhật một dịch vụ mà không cần triển khai lại toàn bộ ứng dụng và quay lại bản cập nhật nếu có sự cố xảy ra. Sửa lỗi và phát hành tính năng sẽ được dễ dàng quản lý và ít rủi ro hơn.
  + **Phát triển độc lập**: Mỗi nhóm đơn có thể xây dựng, kiểm thử và triển khai một dịch vụ. Kết quả là luôn được đổi mới, tốc độ phát hành nhanh hơn.
  + **Khả năng kiểm tra tốt** hơn - các dịch vụ nhỏ hơn và nhanh hơn để kiểm tra
  + **Small, focused teams**. Mỗi đội chịu trách nhiệm cho một dịch vụ. Phạm vi nhỏ hơn của mỗi dịch vụ làm cho mã nguồn dễ hiểu hơn và dễ dàng hơn cho các thành viên nhóm mới phát triển. Hơn nữa mỗi nhóm có thể phát triển, triển khai và mở rộng quy mô dịch vụ của team độc lập hoàn toàn
  + Ứng dụng khởi động nhanh hơn => nhà phát triển làm việc hiệu quả hơn và tăng tốc triển khai
  + Cải thiện lỗi độc lập. Ví dụ: nếu có rò rỉ bộ nhớ trong một dịch vụ thì chỉ có dịch vụ đó bị ảnh hưởng. Các dịch vụ khác tiếp tục xử lý các yêu cầu. Để so sánh, một thành phần hoạt động sai của kiến ​​trúc nguyên khối có thể hạ gục toàn bộ hệ thống.
  + Microservice loại bỏ bất kỳ cam kết lâu dài đối với một technology stack (một tập hợp những phần mềm/công nghệ phối hợp chung với nhau, tạo thành một nền tảng để ứng dụng có thể hoạt động được). Khi phát triển một dịch vụ mới, bạn có thể chọn technology stack mới ⬄ khi thực hiện các thay đổi lớn đối với dịch vụ hiện có, có thể chọn new technology stack
* **Nhược điểm :**
* Các module giao tiếp qua mạng nên có thể tốc độ không cao bằng monolith. Ngoài ra, mỗi module phải tự giải quyết các vấn đề về [bảo mật](https://toidicodedao.com/2016/09/13/bao-mat-co-ban-phan-1/), transaction, lỗi kết nối, quản lý log files.
* Các IDE phát triển dành cho hướng xây dựng monolithics là chủ yếu =>k hỗ trợ phân tán nhiều
* Mỗi service sử dụng một database riêng, việc đảm bảo tính đồng nhất trong dữ liệu sẽ trở nên phức tạp.
* Sử dụng nhiều service nên việc theo dõi, quản lý các service này sẽ phức tạp hơn. Do vậy, một số tool/công nghệ ra đời để phục vụ cho việc này (Docker, Ansible, Salt, Octopus, …).
* ****Cần một đội ngũ có khả năng và trình độ: Software Architect để phân tách module, Techinical Leader để setup workflow, IT để setup [CI](https://toidicodedao.com/2015/08/27/giai-thich-don-gian-ve-ci-continuous-integration-tich-hop-lien-tuc/), deploy lên cloud…. kết thúc với nhiều ngôn ngữ khác và khung làm việc mà ứng dụng trở lên khó khăn để duy trì. Có thể hữu ích khi đặt một số tiêu chuẩn cho toàn dự án, mà không hạn chế quá mức tính linh hoạt của các nhóm. Điều này đặc biệt áp dụng cho chức năng xuyên suốt như đăng nhập
* **So sánh**

|  |  |
| --- | --- |
| **Microservice** | **CBA** |
| **-** lớn được xây dựng như một bộ dịch vụ nhỏ, tự trị  (lôi đống khái niệm vào) | **-** tập trung vào việc phân rã các thiết kế thành các thành phần chức năng hoặc các thành phần logic có chứa các sự kiện, và thuộc tính |

* **VD**

1. Câu 3 3 Layer? So sánh kiến trúc này với kiến trúc tier?

3 layer có tính logic (mỗi layer 1 công việc): Presentation Layers, Business Layers, và Data Layers (vì lượng thông tin thông tin gửi nhiều ta có thể dùng Data Tranfer Object (DTO) để chuyển đối tượng hoặc danh sách đối tượng giữa các tầng với nhau cho tiện). Các lớp giao tiếp với nhau thông qua các dịch vụ (services) mà mỗi lớp cung cấp để tạo nên ứng dụng, lớp này cũng không cần biết bên trong lớp kia làm gì mà chỉ cần biết lớp kia cung cấp dịch vụ gì cho mình và sử dụng nó mà thôi. Mô tả 3 lớp như sau:

* Presentation Layer (GUI): Lớp này có nhiệm vụ chính giao tiếp với người dùng. Nó gồm các thành phần giao diện (win form, web form…) và thực hiện các công việc như nhập liệu, hiển thị dữ liêu, tiếp nhận các event, kiểm tra tính đúng đắn dữ liệu trước và gọi lớp Business Logic Layer (BLL).
* Business Logic Layer (BLL): Layer này phân ra 2 thành nhiệm vụ:

• Đây là nơi đáp ứng các yêu cầu thao tác dữ liệu của GUI layer, xử lý chính nguồn dữ liệu từ Presentation Layer trước khi truyền xuống Data Access Layer và lưu xuống hệ quản trị CSDL.

• Đây còn là nơi kiểm tra các ràng buộc, tính toàn vẹn và hợp lệ dữ liệu, thực hiện tính toán và xử lý các yêu cầu nghiệp vụ, trước khi trả kết quả về Presentation Layer.

* Data Access Layer (DAL): Lớp này có chức năng giao tiếp với hệ quản trị CSDL chứa các methods (insert, update,...). Thường sử dụng các dịch vụ của SQL Server, Oracle,để thực hiện nhiệm vụ của mình. Đối với tập tin, lớp này thực hiện việc đọc, ghi tập tin theo yêu cầu của phần mềm. Việc thực hiện này do lớp xử lý gọi.
* **Tính chất**
* Giảm sự gắn kết giữa các thực thể phần mềm
* Tái sử dụng
* Chia sẻ trách nhiệm
* **Ưu điểm:**
  + Khả năng sử dụng lại cao,
  + Cấu trúc sáng sủa, rõ ràng,
  + Xác định loại lỗi dễ dàng,
  + Người lập trình có thể làm việc chung với nhau một cách dễ dàng,
  + Tính bảo mật cao hơn do có nhiều lớp,
  + Dễ dàng hơn trong việc thay thế và nâng cấp từng lớp mà không ảnh hưởng tới toàn bộ hệ thống,
  + Quản lý: tách từng lớp giúp cho việc xác định các mối quan hệ và cách tổ chức mã vào từng phần dễ quản lý hơn,

**Nhược điểm:**

* Việc truyền dữ liệu giữa các tầng sẽ **chậm hơn** vì phải truyền giữa các tiến trình khác nhau, dữ liệu cần phải được đóng gói -> truyền đi -> mở gói trc khi dùng.
* Trong thực tế việc tổ chức phân chia các tầng thường là khó vì tầng cao đôi khi cần tương tác với các tầng thấp không nằm trực tiếp dưới nó. Kiến trúc LA có thể gây ảnh hưởng đến năng suất vì mọi yêu cầu được chuyển đến tầng khác đều phải được kiểm soát và xử lí...
* Việc phát triển ứng dụng phức tạp hơn
* **So sánh kiến trúc 3layer này với kiến trúc tier**

|  |  |
| --- | --- |
| 3 Layer | 3 Tier |
| * 3 Layers có tính logic (mỗi layer có 1 công việc) * 3 lớp gồm những lớp sau: lớp thứ nhất là lớp DataAccess lớp thứ 2 là lớp Bussiness và lớp thứ 3 là lớp GUI, và những lớp này nằm trong tầng SERVER của mô hình 3-Tier. | -3 Tier có tính vật lý,  - là một dạng của client-server(mỗi tier có thể đặt chung 1 nơi hoặc nhiều nơi, kết nối với nhau qua Web services. Mỗi tier chạy trên một máy/cụm máy cụ thể.   * 3 tầng: Tầng thứ nhất là tầng DAO tầng thứ 2 là tầng SERVER tầng thứ 3 là tầng Client. |

* **Tổ chức code**

**Server Explore**: Data connections tạo DTO- đối tượng trao đổi dữ liệu

3 thư mục:

- BLL: các file chứa code xử lý dữ liệu trước hiển thi và trước lưu CSDL

- DAL: chứa các hàm thực thi truy vấn với CSDL

- GUI: các form giao diện

1. Câu 4: Mô hình Client – Server

* Được sử dụng rộng rãi trong các hệ thống phân tán, gồm các thành phần:
  + Server: phục vụ, cung cấp các chức năng
  + Client: yêu cầu, sử dụng các chức năng đó.
  + Mạng kết nối
  + **Mô tả**

Mối quan hệ giữa client và server:

* + client gửi một hoặc nhiều yêu cầu, chờ đợi trả lời.
  + Server thường thực hiện các xử lý cần thiết để phản hồi lại các yêu cầu. Các máy chủ có thể gửi phản hồi bằng cách sử dụng một loạt các giao thức và định dạng dữ liệu để truyền thông tin tới clients.
  + **Các máy trên mạng muốn giao tiếp với nhau thì phải có chung phương thức**
  + TCP (*Transmission Control Protocol*): thiết lập kết nối giữa các máy tính để truyền dữ liệu. Nó chia nhỏ dữ liệu ra thành những gói (*packet*) và đảm bảo việc truyền dữ liệu thành công.
  + IP (*Internet Protocol*): định tuyến (*route*) các gói dữ liệu khi chúng được truyền qua Internet, đảm bảo dữ liệu sẽ đến đúng nơi cần nhận.
  + HTTP (*HyperText Transfer Protocol*): cho phép trao đổi thông tin (chủ yếu ở dạng siêu văn bản) qua Internet.
  + FTP (*File Transfer Protocol*): cho phép trao đổi tập tin qua Internet...
  + **Yêu cầu server**
* Database Server là máy chủ cài đặt phần mềm hệ quản trị csdl: Sql server, MySql, Oracle,…
* Application Server: là một phần mềm cung cấp ứng dụng cho các máy trạm hoặc thiết bị, thông thường là qua mạng Internet.
* Web Server được dùng để xử lý các truy cập được gửi từ máy khách thông qua giao thức HTTP
* **Ưu điểm**
  + Server được phân tán trên mạng. Chức năng chung (dịch vụ in ấn chẳng hạn) có thể có sẵn cho tất cả các khách hàng và không cần thiết phải cài đặt toàn bộ các dịch vụ.
  + Bảo mật cao hơn. Tất cả dữ liệu được lưu trữ trên máy chủ, mà đồng nghĩa với việc kiểm soát dễ dàng hơn.
  + Truy cập dữ liệu tập trung: dữ liệu được lưu trữ tập trung trên server thay vì nằm rải rác trên nhiều máy, giúp đơn giản hóa việc truy xuất và cập nhật dữ liệu
  + Dễ bảo trì: nhờ khả năng quản lý tập trung mà công việc bảo trì cũng trở nên nhẹ nhàng hơn vì phần lớn việc bảo trì chỉ cần thực hiện trên server. Trong trường hợp hệ thống có nhiều server với thiết bị dự phòng, quá trình bảo trì (như sửa chữa, thay thế server) có thể diễn ra hoàn toàn trong suốt với phía client.
* **Nhược điểm**
  + Hiệu năng có thể không dự đoán trước được do nó phụ thuộc vào mạng cũng như hệ thống.
  + Có thể có các vấn đề về quản lý nếu server được sở hữu bởi các tổ chức khác nhau.
* Khi nào sử dụng
* Được sử dụng khi dữ liệu trong một cơ sở dữ liệu chia sẻ phải truy cập từ nhiều nơi
* Chương trình duyệt Web chạy dựa trên Internet hay một mạng cục bộ
* Microsoft Windows hoạt động ứng dụng hệ thống dựa trên các dịch vụ truy cập dữ liệu mạng;
* VD
* ứng dụng truy cập các dữ liệu từ xa (như chương trình chat và đọc e-mail, và các công cụ truy vấn cơ sở dữ liệu);
* và các công cụ và tiện ích để thao tác với hệ thống điều khiển từ xa (ví dụ như các công cụ quản lý hệ thống và các công cụ giám sát mạng).
* Biểu đồ thành phần
* Cách thức xây dựng socket sử dụng giao thức TCP

1. Câu 5 Mô hình MVP

MVP là một mô hình kiến trúc hướng giao diện người dùng, được thiết kế để tạo thuận lợi cho việc kiểm thử đơn vị (unit testing) và tăng tính tách biệt giữa tầng dữ liệu và tầng hiển thị dữ liệu trong mô hình MVC.

Đặc điểm của mô hình MVP:

* Phân phối: phần lớn các trách nhiệm được chia đều giữa Presenter và Model, View có rất ít vai trò, chủ yếu để hiển thị dữ liệu.
* Khả năng test: chúng ta có thể test phần lớn các code logic do ít phụ thuộc vào View.
* Tính dễ dùng: số lượng code gấp 2 so với mô hình MVC truyền thống, tuy nhiên về mặt ý tưởng thì mô hình MVP rất rõ ràng.

Hệ thống gồm 3 thành phần tương tác lẫn nhau :

**Model:** là lớp xử lý dữ liệu, nhiệm vụ của lớp này là lấy cùng dữ liệu từ database và network sau đó di chuyển đến Presenter thông qua lệnh callback.

**View:**Lớp chịu trách nhiệm xử lý các thành phần liên quan đến View, hiển thị data từ Model, tiếp nhận tương tác từ user gửi tới để Presenter xử lý các tương tác đó.

**Presenter:**Lớp**c**hịu trách nhiệm xử lý các business logic. Đây là lớp tương tác được với cả View và Model. Chịu trách nhiệm lấy data từ Model khi có thay đổi trên View, xử lý và binding data đã xử lý lên View.

|  |  |
| --- | --- |
| **MVP** | **MVC** |
| V là tầng duy nhất nhận tương tác | V, C đều tương tác với người dùng |
| P logic xử lý dữ liệu tương tác và dữ liệu hiển thị | C logic xử lý dữ liệu tương tác và dữ liệu hiển thị |
| Kết hợp làm giảm số lượng code và logic code 1 nơi | Code phân loại theo chức năng |
| Với phần mềm lớn số lượng code lớn làm rối việc đọc hiểu bảo trì | Với phần mềm lớn số lượng code lớn làm rối việc đọc hiểu nâng cấp cập nhật |
| Kiểm thử đơn vị dễ dàng | Kiểm thử tốn nhiều công sức, thời gian |

1. Câu 6 Mô hình SOA

SOA (Service-Oriented Architecture) - kiến trúc hướng dịch vụ. SOA là một dịch vụ được cung cấp rải rác ở nhiều nơi khác nhau và có thể truy cập thông qua môi trường mạng. => một hệ thống SOA là một tập hợp nhiều dịch vụ được cung cấp trên mạng, được tích hợp lại với nhau để cùng cộng tác thực hiện các tác vụ nào đó theo yêu cầu của khách hàng. **Dịch vụ (Service)** là yếu tố then chốt trong SOA. Có thể hiểu dịch vụ như là một loại module thực hiện một quy trình nghiệp vụ nào đó.

* **Mục đích**
* Giúp các ứng dụng có thể giao tiếp được với nhau mà không cần biết các chi tiết kỹ thuật bên trong. SOA là cấp độ cao hơn của phát triển ứng dụng, chú trọng đến quy trình nghiệp vụ và dùng giao tiếp chuẩn để giúp che đi sự phức tạp kỹ thuật bên dưới và định ra một chuẩn giao tiếp (dùng để gọi hàm dịch vụ) được định nghĩa rõ ràng và độc lập với nền tảng hệ thống và có thể tái sử dụng.

# **Vai trò**

# Có 2 vai trò chính. 2 tác nhân nhà cung cấp dịch vụ và người tiêu dùng dịch vụ. Một tác nhân phần mềm có thể đóng cả hai vai trò. Tầng người tiêu dùng là điểm mà người tiêu dùng (người dùng con người, các dịch vụ khác hoặc bên thứ ba) tương tác với Lớp nhà cung cấp và SOA bao gồm tất cả các dịch vụ được định nghĩa trong SOA.

* UDDI là một thư mục dịch vụ, nơi mà chúng ta có thể đăng ký và tìm kiếm các dịch vụ web.
* SOAP (Simple Object Access Protocol)
* **Ưu điểm**
* Sử dụng lại những thành phần có sẵn: giảm chi phí mua phần mềm mới, giảm tính dư thừa, sử dụng lại những thành phần sẵn có để xây dựng phần mềm mới, giảm chi phí phát triển cho từng phần độc lập của mỗi tính năng mới chưa có.
* Tăng tính linh hoạt và triển khai cài đặt nhờ tính loose coupling thuận tiện cho việc chỉnh sửa, nâng cấp hoặc mở rộng hệ thống
* Dễ dàng và nhanh chóng tạo ra các tiến trình nghiệp vụ từ các service đã có
* Thích ứng với những thay đổi trong tương lai
* Hỗ trợ đa thiết bị và đa nền tảng.
* Tăng khả năng mở rộng và khả năng sẵn sàng cung cấp.
* **Nhược điểm**
* Hệ thống phức tạp
* Khi xây dựng ứng dụng tổng hợp từ nhiều dịch vụ với tính tái sử dụng cao thì vấn đề bảo mật như: xác thực, phân quyền, bí mật, toàn vẹn dữ liệu, bảo vệ quyền riêng tư,,,. Trở thành bài toán hết sức phức tạp và đòi hỏi bằng những hướng tiếp cận bảo mật hoàn toàn mới so với các phương pháp bảo mật truyền thống
* **Lợi ích**
  + Về mặt kinh tế:
    - * Doanh nghiệp có điều kiện tập trung thời gian để tìm kiếm các giải pháp cho các bài toán liên quan đến kinh tế.
      * Thúc đẩy khả năng phát triển của hệ thống hiện có cũng như khả năng mở rộng của hệ thống trong tương lai.
  + Về mặt kỹ thuật:
    - Độc lập hệ thống: những service không phụ thuộc vào hệ thống và mạng cụ thể
    - Có khả năng tái sử dụng
    - Khả năng hồi đáp thích nghi tốt và nhanh hơn để đáp ứng với sự thay đổi về yêu cầu giao dịch
    - Cho phép dễ dàng triển khai chương trình, môi trường chạy và quản lý service dễ dàng hơn.
    - Những xác nhận và chứng minh của Service consumer về những tính năng sercurity dựa trên giao tiếp Service tốt hơn cơ chế kết nối chặt chẽ.

1. So sánh mô hình SOA với Microservice

|  |  |
| --- | --- |
| **Soa** | **Microservice** |
| Được xây dựng trên ý tưởng về cách tiếp cận kiến ​​trúc của chia sẻ nhiều nhất có thể | Được xây dựng dựa trên ý tưởng về cách tiếp cận kiến ​​trúc của chia sẻ ít nhất có thể |
| Quan trọng hơn về tái sử dụng chức năng kinh doanh | Quan trọng hơn về khái niệm bối cảnh giới hạn |
| Quản trị và tiêu chuẩn chung | Quản trị thoải mái, tập trung hơn vào sự hợp tác của mọi người và tự do lựa chọn |
| Sử dụng xe buýt dịch vụ doanh nghiệp (ESB) để liên lạc | Sử dụng hệ thống nhắn tin ít phức tạp và đơn giản |
| Hỗ trợ nhiều giao thức tin nhắn | Sử dụng các giao thức nhẹ như HTTP / REST & AMQP |
| Nền tảng chung cho tất cả các dịch vụ được triển khai cho nó | Máy chủ ứng dụng không thực sự được sử dụng. Các nền tảng như Node.JS có thể được sử dụng |
| Đa luồng với nhiều chi phí hơn để xử lý I / O | Một luồng thường sử dụng các tính năng Vòng lặp sự kiện để xử lý I / O không khóa |
| Sử dụng các container (Docker, Linux Container) ít phổ biến hơn | Container hoạt động rất tốt trong MSA |
| Tối đa hóa khả năng sử dụng lại dịch vụ ứng dụng | Tập trung hơn vào việc tách rời |
| Sử dụng cơ sở dữ liệu quan hệ truyền thống thường xuyên hơn | Sử dụng cơ sở dữ liệu hiện đại, không liên quan |
| Một sự thay đổi có hệ thống đòi hỏi phải sửa đổi nguyên khối | Thay đổi có hệ thống là tạo ra một dịch vụ mới |