[**Phần 1** 2](#_Toc60857123)

[**1.** **Giải thích mô hình MVC, 3 layer (P, B, D)** 2](#_Toc60857124)

[***a.*** ***MVC*** 2](#_Toc60857125)

[***b.*** ***3 Layer (3 lớp)*** 5](#_Toc60857126)

[**2.** **Nhận xét ưu điểm và hạn chế của mỗi mô hình** 8](#_Toc60857127)

[***a.*** ***MVC*** 8](#_Toc60857128)

[***b.*** ***3-Layer*** 9](#_Toc60857129)

[3. Mô hình nào phù hợp với application, web 10](#_Toc60857130)

[4. So sánh giống và khác nhau giữa 2 mô hình MVC, 3 layer 10](#_Toc60857131)

[5. Ví dụ cách viết code của một chức năng nào đó (Hiển thị danh sách, thêm, xoá, sửa, tìm kiếm) theo mô hình MVC hoặc 3 layer 11](#_Toc60857132)

[***a.*** ***3 Layer (VD todolist)*** 11](#_Toc60857133)

[***b.*** ***MVC*** 16](#_Toc60857134)

[***MVC (Ví dụ: Todolist hiển thị tất cả công việc, thêm xóa sửa):*** 16](#_Toc60857135)

[6. ORM 20](#_Toc60857136)

[**a.** **ORM là gì?** 20](#_Toc60857137)

[**b.** **Hoạt động** 21](#_Toc60857138)

[**c.** **Ưu, nhược điểm** 21](#_Toc60857139)

[***d.*** ***Tại sao nên cài đặt ORM để truy xuất dữ liệu*** 22](#_Toc60857140)

[**Phần 2** 22](#_Toc60857141)

[*I. Quản lý bán hàng (MVC)* 22](#_Toc60857142)

[*II. Quản lý bán hàng (3 lớp)* 29](#_Toc60857143)

# **Phần 1**

1. **Giải thích mô hình MVC, 3 layer (P, B, D)**
2. ***MVC***
   * *Mục đích* hướng đến kiến trúc MVC trong thiết kế ứng dụng Web

* Loại bỏ những hạn chế của ASP.NET Web Forms
* Nhu cầu xây dựng các Thành phần Ghép nối thấp (Loosely-coupled Components)
  + Loosely-coupled: Thiết kế các Lớp có vai trò riêng biệt và giảm sự liên quan lẫn nhau trong chương trình, nghĩa là giảm thiểu sự ràng buộc, phụ thuộc.
  + Hướng phát triển:
    - Các thành phần riêng lẻ không trực tiếp phụ thuộc lẫn nhau
    - Các thành phần có thể được thay thế khi cần thiết
  + Khả năng Kiểm thử
    - Khi kiểm thử, tránh việc gọi trực tiếp xuống CSDL, bằng cách thay đổi Thành phần truy xuất dữ liệu bằng một Thành phần trả về dữ liệu giả
  + Bảo trì
    - Thay đổi càng ít Thành phần càng tốt
  + *Khái niệm*
* MVC là mô hình thiết kế ứng dụng, chia ứng dụng thành 3 thành phần
  + Model, View, Controller
* Mỗi thành phần đều có một công việc cụ thể. Điều quan trọng là chúng không quan tâm đến cách thức hoạt động của các lớp còn lại
* MVC là chuẩn xây dựng web được ứng dụng nhiều nhất để xây dựng các dự án có khả năng scale và mở rộng
* Kết hợp tốt với phương pháp luận Thiết kế hướng đối tượng (OOP). Vì mỗi Model, View, Controller được xem như là các object, và tái sử dụng trong chương trình

Diagram

Description automatically generated

* + *Chi tiết các lớp*
* Model
  + Bao gồm mọi logic liên quan đến dữ liệu
  + Chứa các class bao gồm:
    - Thuộc tính thể hiện các trường của dữ liệu
    - Phương thức xử lý nghiệp vụ
  + Đối tượng thuộc lớp Model có thể là dữ liệu truyền giữa View với Controller; Có thể là những dữ liệu khác liên quan đến nghiệp vụ
* Model thể hiện dưới hình thức là một cơ sở dữ liệu hoặc có khi chỉ đơn giản là một [file XML](https://monamedia.co/file-xml-la-gi/) bình thường. Model thể hiện rõ các thao tác với cơ sở dữ liệu như cho phép xem, truy xuất, xử lý dữ liệu,…
  + *VD: object 'Customer' nắm giữ các thông tin khách hàng lấy từ database, có thể cập nhật nó và lưu xuống database, hoặc lấy nó để hiển thị cho người dùng*
* View
  + Thành phần xử lý mọi logic liên quan đến Giao diện người dùng của ứng dụng
* Nơi mà người dùng có thể lấy được thông tin dữ liệu của MVC thông qua các thao tác truy vấn như tìm kiếm hoặc sử dụng thông qua các website.
  + Đảm nhận việc hiển thị kết quả xử lý, tạo ra trang HTML để đưa về phía người dùng
  + Thường được gán cùng với một Model để hiển thị thông tin của Model đó.
  + Tuy nhiên, View không có mối quan hệ trực tiếp với Controller, cũng không được lấy dữ liệu từ Controller mà chỉ hiển thị yêu cầu chuyển cho Controller mà thôi.
  + VD: Trang chi tiết của 'Customer' sẽ hiển thị các thông tin của khách hàng cùng với các control cho phép chỉnh sửa thông tin như textbox, combobox,... Nút “delete” được tạo bởi View khi người dùng nhấn vào nút đó sẽ có một hành động trong Controller.
* Controller
  + Đóng vai trò điều phối hoạt động giữa View và Model
  + Tiếp nhận yêu cầu của người dùng
  + Truy vấn hoặc cập nhật dữ liệu thông qua Model và tương tác với View để xuất ra kết quả cho người dùng
  + *Sơ đồ hoạt động*

Mô tả sơ đồ: Bạn có thể hình dung, khi  một yêu cầu từ máy client gửi tới server, Controller sẽ thực hiện nhiệm vụ của mình đó là tiếp nhận và xử lý yêu cầu. Trong trường hợp cần thiết, nó có thể liên hệ Model – bộ phận làm việc với database để hỗ trợ.

Khi xử lý xong yêu cầu, kết quả sẽ được trả về View. Tại View sẽ tạo mã HTML thành giao diện và trả về hiển thị trên trình duyệt.

Diagram

Description automatically generated

* Ví dụ: Chức năng thêm bài viết mới trong trang quản trị website. Nơi đây có 2 trường nhập về tiêu đề và nội dung bài viết, thì trong đó:

View sẽ hiển thị ra phần nhập form tiêu đề và nội dung.

Controller lấy dữ liệu từ 2 trường và gửi tới Model.

Model lúc này sẽ nhận dữ liệu từ Controller để lưu trữ trong cơ sở dữ liệu.

Mỗi bộ phận thực hiện chức năng nhất định, nhưng chúng có sự thống nhất, liên kết với nhau tạo nên **mô hình MVC**. Mô hình này tương đối nhẹ. Nó có thể tích hợp được nhiều tính năng có trong ASP.NET hiện giờ. Ví dụ như authentication (quá trình xác thực).

* + Cách xử lý yêu cầu (Workflow)
* User tương tác trên View, có thể là click vào một link, ghi URL hoặc thực hiện submit form => Thực hiện HTTP Request đến server
* Với URL của request, server sẽ route yêu cầu đến Controller thích hợp, Controller Action (phương thức thuộc Controller) thích hợp
* Controller cùng với yêu cầu của người dùng (nếu là submit form, sẽ có dữ liệu mà người dùng gửi đi), thực hiện tương tác với Model để tiếp tục xử lý
* Model thực hiện xử lý nghiệp vụ, kết nối với kho dữ liệu để thực hiện truy xuất, cập nhật tuỳ thuộc vào yêu cầu. Sau đó trả kết quả về Controller
* Với kết quả xử lý, Controller truyền kết quả đó cho View để xuất kết quả cho người dùng

1. ***3 Layer (3 lớp)***
   * Khái niệm

* Kiến trúc phân chia chương trình thành 3 lớp xử lý. Mỗi lớp đảm nhiệm công việc riêng biệt
* 3 lớp được sắp xếp chồng từ trên xuống dưới theo thứ tự: Presentation Layer, Business Logic Layer, Data Access Layer
* Luồng dữ liệu đi từ trên xuống dưới và từ dưới lên trên
* Mô hình này phát huy hiệu quả nhất khi bạn xây dựng một hệ thống lớn, việc quản lý code và xử lý dữ liệu lỗi dễ dàng hơn.

***Lưu ý khi xây dựng mô hình 3 lớp:***

* Cần một solution riêng cho project.
* Cần 3 project khác nhau để làm nên 3 lớp, tên Project đặt như sau:
* Lớp GUI: (VD: QuanLy\_GUI)
* Lớp Business: (VD: QuanLy\_BUS)
* Lớp Data Access: (VD: QuanLy\_DAL)
* Lớp DTO: (VD: QuanLy\_DTO)
  + Chi tiết các lớp
* Presentation Layer (GUI)
  + Chứa các thành phần hiển thị cũng như các file như ASPX, ASCX, MasterPages, CSS, hoặc WinForms
  + NOTE:
  + Giao tiếp với người dùng
    - Nhập
      * Trên form (nhập tay)
      * Import (xlsx, csv) => Batch Input
    - Lấy / Hiển thị
      * Hiển thị trên các field của WinForm
      * Export (xlsx, DevExpress GridView, ...)
* Kiểm tra tính đúng đắn dữ liệu trước khi gọi lớp Business Logic Layer (BLL).
  + Là các lớp form
  + Chiếm 50% code của chương trình
* Ví dụ : textbox, button, combobox, … UI Process Components : là thành phần chịu trách nhiệm quản lý các quá trình chuyển đổi giữa các UI…
* Ví dụ : Sắp xếp quá trình kiểm tra thông tin khách hàng:

1.Hiển thị màn hình tra cứu ID

2.Hiển thị màn hình thông tin chi tiết khách hàng tương ứng

3.Hiển thị màn hình liên lạc với khách hàng.

* Business Logic Layer (BLL)
  + Chứa các logic nghiệp vụ và tập các quy tắc hoạt động
  + Tương tác với lớp DAL
    - Lấy dữ liệu được lọc dựa theo điều kiện nghiệp vụ
    - Lưu lại cập nhật dữ liệu sau khi thực hiện chỉnh sửa theo yêu cầu nghiệp vụ
    - Thực hiện Tính toán và Xác thức dữ liệu
  + BLL thường trả dữ liệu xử lý về cho lớp cao hơn nó (như lớp GUI)
  + Có thể bao gồm cả Xử lý lỗi (error handling), logging
  + NOTE:
    - Suy nghĩ, tạo ra từ thực thể
    - Đối tượng trung tâm của bài toán
    - Thay thế cho:
      * DTO (Model), chỉ chứa các thuộc tính => Property
      * Service => Function
    - Không đơn thuần chứa các hàm đọc ghi, phải có logic nghiệp vụ như Tính toán, Điều kiện
* Data Access Layer (DAL)
  + Một tập các lớp được dùng để lưu trữ các phương thức truy xuất dữ liệu như CRUD
  + Công việc chính là tương tác với lớp dữ liệu (Data layer - như CSDL)
  + Không nên chứa các logic nghiệp vụ
  + Đóng vai trò như lớp bổ trợ (ultility, helper) cho việc truy vấn và cập nhật dữ liệu với một Data store
    - Truy xuất dữ liệu
    - Dữ liệu lưu trong CSDL (Data storage) và lưu trong App (RAM) là 2 phạm trù khác nhau về mặt vật lý
    - => Việc đổi dữ liệu từ CSDL sang App và ngược lại được thực hiện trong DAL
    - Mục đích: Thay đổi CSDL => Chỉ cần thay đổi code ở DAL
    - Dù vậy, Interface cho DAL không thay đổi
      * VD: KQ trả về vẫn là model hoặc là list của 1 model
  + Sơ đồ hoạt động
    - Diagram

      Description automatically generated
  + Cách xử lý yêu cầu (Workflow)
* Chương trình khởi động với giao diện (GUI)
* Lớp BLL có thể được gọi thông qua giai đoạn khởi tạo của GUI hoặc do người dùng kích hoạt các event như bấm vào button, nhập liệu, ... Gọi thông qua các phương thức mà BLL đưa ra
* Với yêu cầu (phương thức nhất định) với tham số đính kèm, BLL thực hiện xử lý nghiệp vụ. Nếu cần thiết, gọi lớp DAL để truy xuất dữ liệu
* DAL thực hiện các thao tác truy vấn kho dữ liệu, trả kết quả về BLL
* BLL nhận được kết quả, cuối cùng trả kết quả trực tiếp về GUI Một khi gặp lỗi (các trường hợp không đúng dữ liệu) thì đang ở layer nào thì quăng lên trên layer cao hơn nó 1 bậc cho tới GUI thì sẽ quăng ra cho người dùng biết - Các dữ liệu được trung chuyển giữa các Layer thông qua một đối tượng gọi là Data Transfer Object (DTO), đơn giản đây chỉ là các Class đại diện cho các đối tượng được lưu trữ trong Database.

1. **Nhận xét ưu điểm và hạn chế của mỗi mô hình**
2. ***MVC***
   * Ưu điểm
     + - Quy trình phát triển nhanh hơn
         * Hỗ trợ phát triển nhanh chóng và song song
         * Các lập trình viên, nhóm lập trình có thể phát triển các bộ phận khác nhau cùng một lúc (một nhóm đảm nhiệm xử lý, một nhóm đảm nhiệm giao diện). Chương trình được phát triển nhanh gấp 3 lần so với bình thường
       - Khả năng cung cấp nhiều View cho một Model
       - Khả năng trùng lặp code ra là ít
       - Hỗ trợ kĩ thuật bất đồng bộ, đảm bảo chương trình chạy với tốc độ nhanh nhất có thể
       - Thay đổi không làm ảnh hưởng đến kiến trúc
         * Giao diện là thành phần được thay đổi thường xuyên nhất. Việc thêm một View mới không làm ảnh hưởng đến các thành phần khác vì tính phân chia trách nhiệm rõ rệt của MVC
       - Model của MVC trả về dữ liệu mà không cần định dạng
         * Cùng một Model đó có thể được hiển thị ở nhiều dạng khác nhau như HTML, Flash, DreamViewer,...
       - Tối ưu SEO: MVC cho phép điều chỉnh URL dẫn đến Controller (routing) tuỳ thích và SEO-friendly
       - Dễ dàng tích hợp JavaScript, jQuery (nhất là sử dụng các thư viện JS bên thứ 3)
       - Tạo mô hình chuẩn cho dự án, giúp cho việc tiếp cận với ứng dụng dễ dàng hơn
       - Trình tự xử lý rõ ràng, nhiệm vụ riêng biệt, độc lập với các thành phần khác và các thành phần có thể tương tác được với nhau.
       - Quy hoạch các thành phần riêng biệt giúp cho quá trình phát triển, quản lý, vận hành, bảo trì ứng dụng thuận lợi hơn, đồng thời dễ dàng kiểm soát được luồng xử lý của ứng dụng.
       - Mô hình triển khai rõ ràng, mạch lạc, xử lý nghiệp vụ tốt, dễ dàng triển khai các ứng dụng vừa.
   * Hạn chế
     + Mô hình chỉ thích hợp sử dụng cho các dự án lớn. Áp dụng vào các dự án nhỏ sẽ trở thành công cụ cồng kềnh gây tốn thời gian trong quá trình phát triển cũng như thời gian trung chuyển dữ liệu
     + Phân chia công việc và nghiệp vụ giữa các thành phần không đồng đều, trong đó Model phải xử lý rất nhiều tác vụ
     + Việc kiểm thử riêng lẻ View rất khó khăn vì View phụ thuộc vào cả Controller và Model (nếu không có chúng, sẽ không có yêu cầu nào dẫn đến View cũng như View không có dữ liệu). Vì vậy khi cần kiểm thử ta phải giả lập cả Controller và Model
     + Đối với các ứng dụng quy mô lớn, quy trình xử lý nghiệp vụ có tính phức tạp cao, lượng dữ liệu lớn thì mô hình MVC trở nên không còn khả dụng

### ***3-Layer***

* + Ưu điểm
    - * Phân loại rõ ràng các lớp có các nhiệm vụ khác nhau. Từ đó ta có thể quản lý và bảo trì project tốt hơn
      * Dễ dàng phân loại các hành động tại BLL
      * Dễ dàng phân loại các hàm truy xuất tại DAL, phân loại hàm theo table
      * Việc phân chia thành từng lớp giúp cho code được tường minh hơn. Nhờ vào việc chia ra từng lớp đảm nhận các chức năng khác nhau và riêng biệt như giao diện, xử lý, truy vấn thay vì để tất cả lại một chỗ. Nhằm giảm sự kết dính.
      * Dễ bảo trì khi được phân chia, thì một thành phần của hệ thống sẽ dễ thay đổi. Việc thay đổi này có thể được cô lập trong 1 lớp, hoặc ảnh hưởng đến lớp gần nhất mà không ảnh hưởng đến cả chương trình.
      * Dễ phát triển, tái sử dụng: khi chúng ta muốn thêm một chức năng nào đó thì việc lập trình theo một mô hình sẽ dễ dàng hơn vì chúng ta đã có chuẩn để tuân theo. Và việc sử dụng lại  khi có sự thay đổi giữa hai môi trường ( Winform sang Webfrom ) thì chỉ việc thay đổi lại lớp GUI.
      * Dễ bàn giao. Nếu mọi người đều theo một quy chuẩn đã được định sẵn, thì công việc bàn giao, tương tác với nhau sẽ dễ dàng hơn và tiết kiệm được nhiều thời gian.
      * Dễ phân phối khối lượng công việc. Mỗi một nhóm, một bộ phận sẽ nhận một nhiệm vụ trong mô hình 3 lớp. Việc phân chia rõ ràng như thế sẽ giúp các lập trình viên kiểm soát được khối lượng công việc của mình.
  + Hạn chế
* Việc truyền dữ liệu giữa các tầng sẽ chậm hơn vì phải truyền giữa các tiến trình khác nhau (IPC), dữ liệu cần phải được đóng gói -> truyền đi -> mở gói trước khi có thể dùng được.
* Việc phát triển ứng dụng phức tạp hơn.

## Mô hình nào phù hợp với application, web

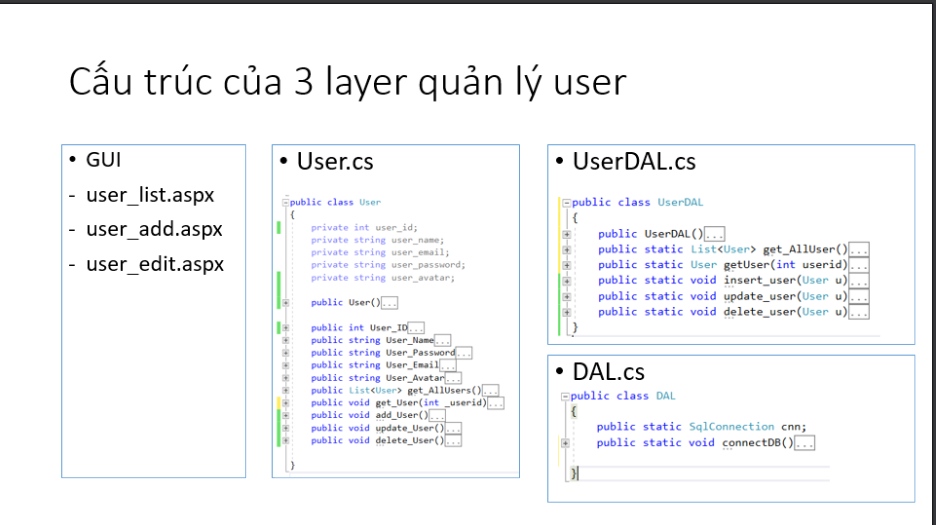
* 3-Layer phù hợp với Application, đặc biệt là các ứng dụng quản lý
* Trong khi đó, MVC được ưu chuộng nhiều trong lập trình website vì tính linh hoạt và không bắt buộc các DTO như 3-Layer .

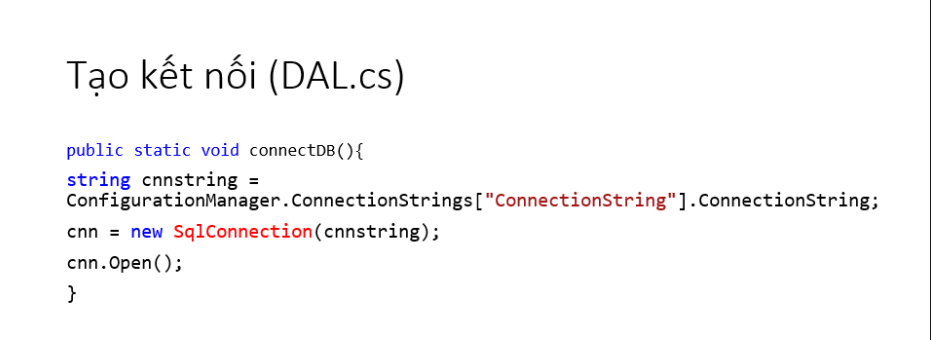
## So sánh giống và khác nhau giữa 2 mô hình MVC, 3 layer

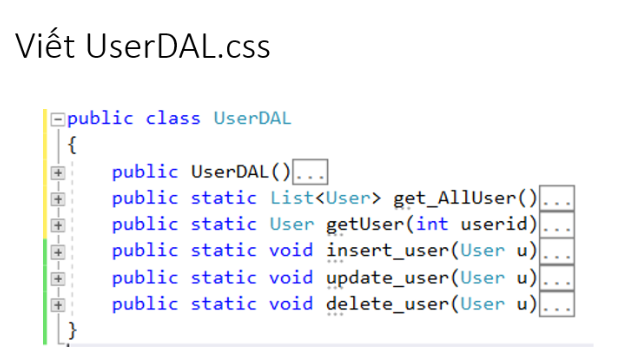
* Giống nhau
  + Cả hai đều là mô hình kiến trúc áp dụng trong lập trình được tách thành 3 thành phần.
  + Tách biệt sự phụ thuộc giữa ngôn ngữ lập trình, môi trường phát triển, xử lý logic.
  + Các thành phần ở mỗi mô hình có nhiều điểm tương đồng như Presentation Layer và View..
  + Tách rời programming core/business logic ra khỏi những phụ thuộc về tài nguyên và môi trường. Presentation Layer (PL) thể hiện giống như chức năng của View và Controller. Business Layer (BL) và Data Access Layer (DL) thể hiện giống như chức năng của Model. Như thế nhìn ở góc độ này, thì MVC tương đương với 3-layer (tất nhiên có chồng chéo như hình vẽ)
* Khác nhau
* Trong mô hình 3 lớp, quá trình đi theo chiều dọc, bắt đầu từ PL, sang BL, rồi tới DL, và từ DL, chạy ngược lại BL rồi quay ra lại PL.
* Còn trong mô hình MCD, dữ liệu được nhận bởi View, View sẽ chuyển cho Controller cập nhật vào Model, rồi sau đó dữ liệu trong Model sẽ được đưa lại cho View mà không thông qua Controller, do vậy luồng xử lý này có hình tam giác.
  + Khác biệt rõ nhất là cách xử lý yêu cầu (workflow) của 2 mô hình
    - MVC phức tạp hơn, cần thông qua routing để xác định controller, và trả kết quả thông qua việc tạo View hiển thị cho người dùng
    - 3-Layer đơn thuần chỉ là gọi hàm thông qua hàm và nhận kết quả trực tiếp
  + Ở MVC, thành phần Model giữ chức năng của cả lớp Business và Data Access trong mô hình 3 Layer

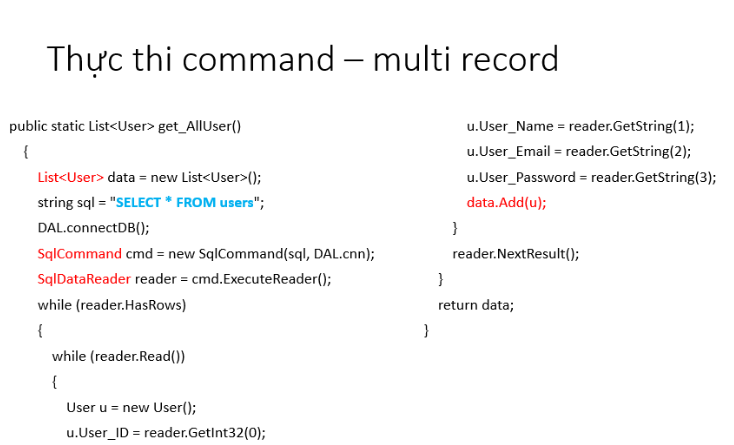
## Ví dụ cách viết code của một chức năng nào đó (Hiển thị danh sách, thêm, xoá, sửa, tìm kiếm) theo mô hình MVC hoặc 3 layer

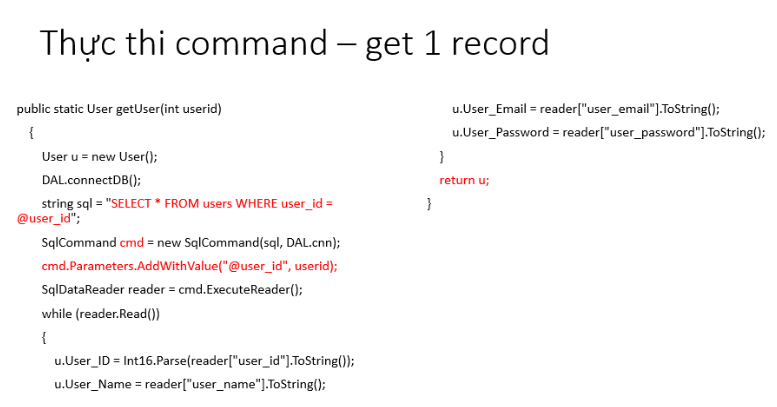
* 1. ***3 Layer (VD todolist)***



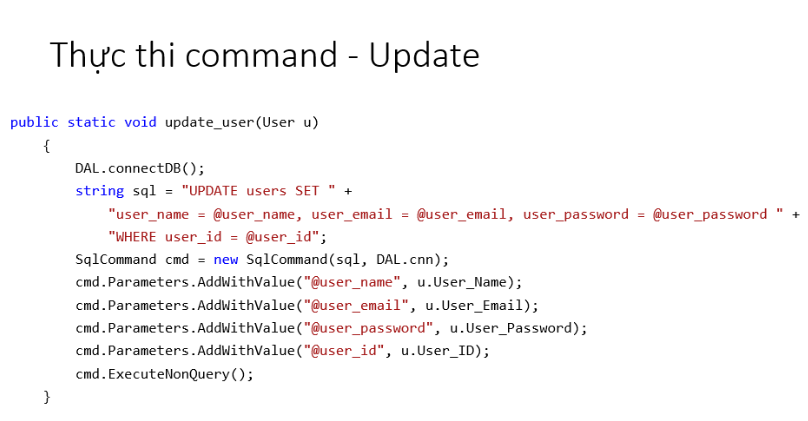




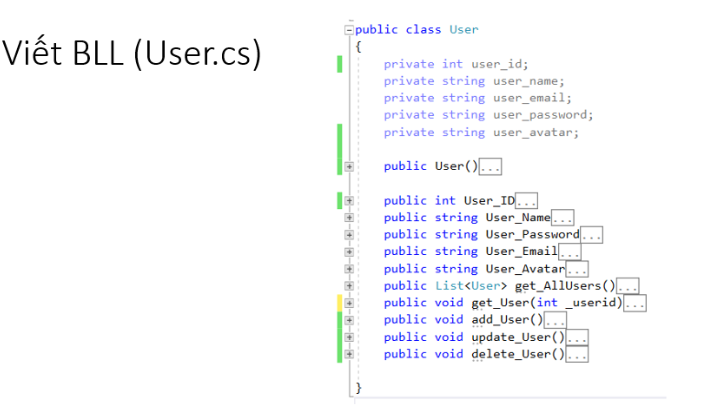


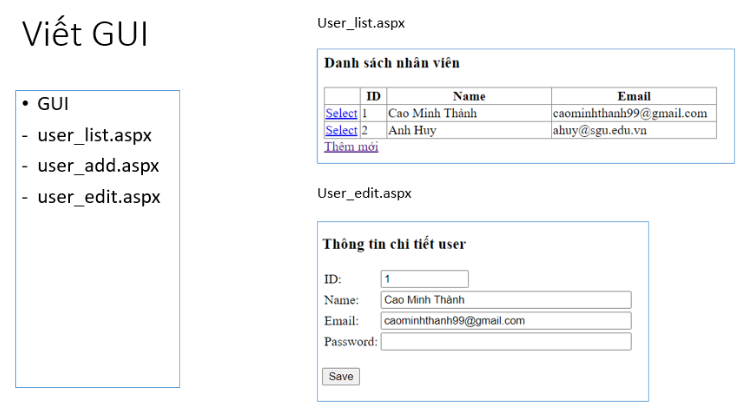








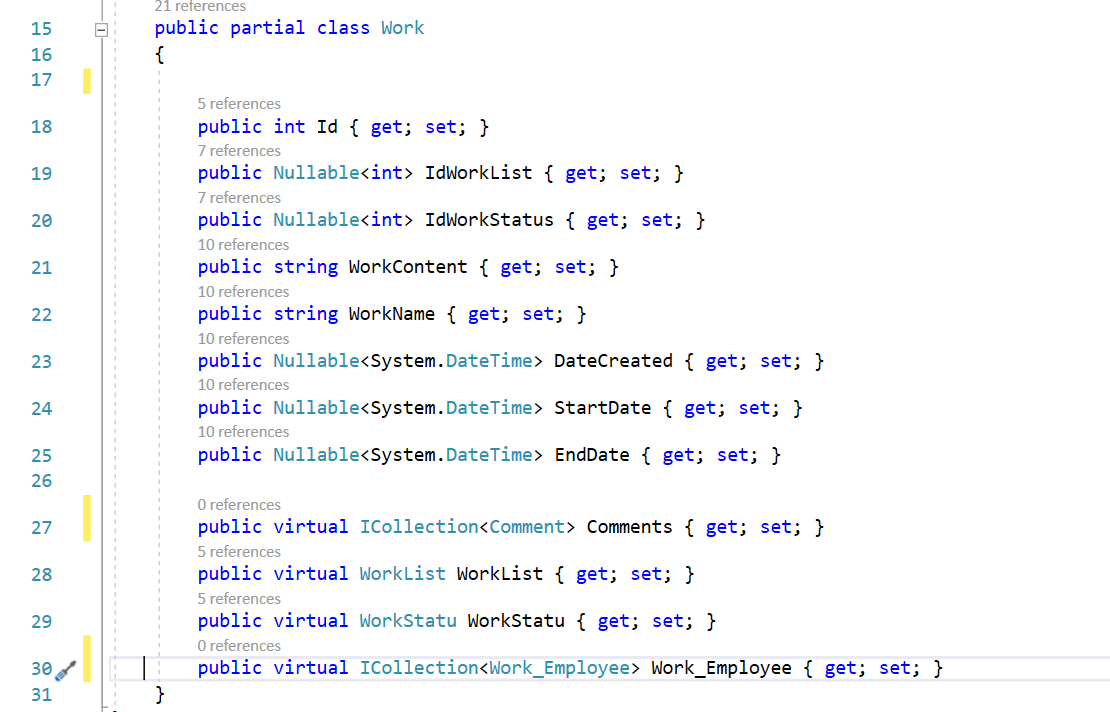




### ***MVC***

### ***MVC (Ví dụ: Todolist hiển thị tất cả công việc, thêm xóa sửa):***

**Model:** Sử dụng Database first thêm vào có sẵn model.

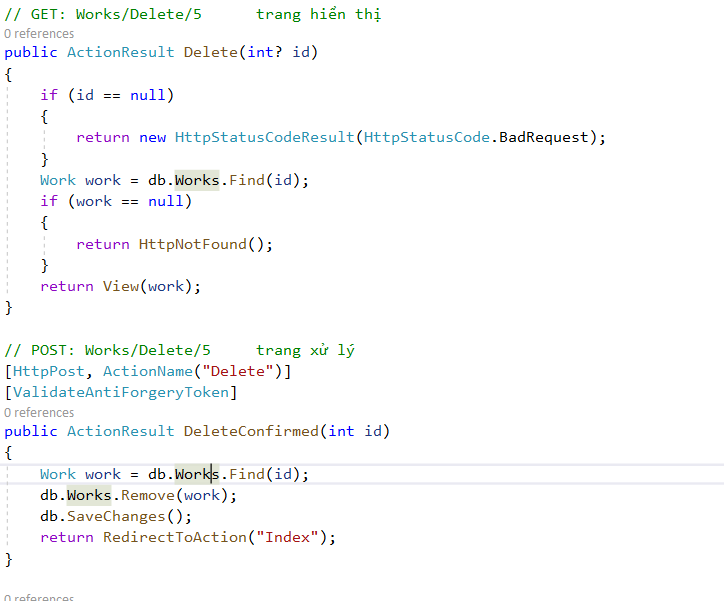


**Controller:**









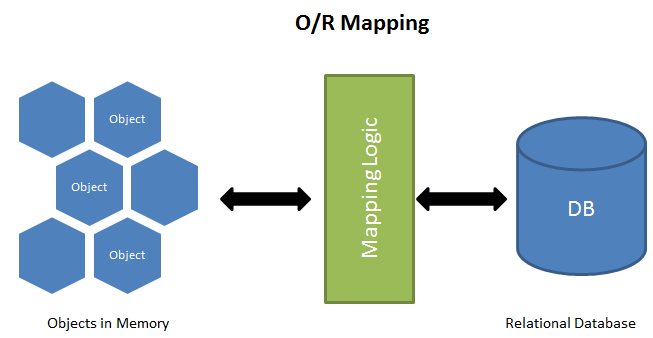
**View:** hiển thị danh sách tất cả Work



## ORM

### **ORM là gì?**

ORM là 1 kỹ thuật lập trình giúp ánh xạ các record dữ liệu trong hệ quản trị cơ sở dữ liệu sang dạng đối tượng đang định nghĩa trong các class - một khái niệm phổ biến được sử dụng trong tất cả các ngôn ngữ hiện đại ngày nay như: Java, PHP, Ruby.



**Hoạt động**

ORM giải quyết vấn đề đồng bộ giữa kiểu dữ liệu trả về khác nhau. Một bên là cơ sở dữ liệu, ***ở đó dữ liệu được thể hiện dưới dạng tập hợp các bản ghi.*** Một bên là các đối tượng, ***ở đó dữ liệu đc hiện thị dưới dạng object***



**Ưu, nhược điểm**

***Ưu điểm***

Tuân thủ nguyên tắc [DRY](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself), tập trung hóa code của bạn tại một chỗ, và việc chỉnh sửa, maintain, tái sử dụng lại code sẽ dễ dàng hơn

Rất nhiều thứ được thực hiện "tự động", liên quan tới việc xử lí dữ liệu, trong ví dụ trên chúng ta không cần quan tâm đến việc dùng vòng để convert data từ mysql ra, Chỉ cần biết đầu vào là keyword đầu ra trả lại 1 danh sách các quyển sách có author=Linus

Việc sử dụng ORM làm bạn bắt buộc phải viết code theo mô hình MVC, khiến code của bạn dễ sửa đổi, bảo trì hơn.

Bạn không cần phải biết quá nhiều về MYSQL, bạn vẫn có thể tiếp cận dễ dàng với ORM.

Model k ràng buộc chặt chẽ với ứng dụng, hay nói cách khác, Model thì linh hoạt, vì vậy bạn có thể thay đổi hoặc sử dụng nó bất cứ chỗ nào

ORM cho phép bạn tận dụng ưu điểm của OOP như kế thừa dữ liệu mà không phải đau đầu

***Nhược điểm***

Với mỗi framework sẽ có 1 thư viện ORM khác nhau, tất nhiên là bạn phải mất thời gian học nó. Và vì chúng là thư viện nên sẽ rất "nặng".

Performance ổn đối với các truy vấn thông thường, nhưng Raw SQL sẽ luôn làm tốt hơn với các dự án lớn.

ORM được coi là abstracts của DB. Nếu bạn không quan tâm đến những gì thực sự xảy ra khi sử dụng ORM. Nó có thể là cái bẫy với những bài toán N + 1 query.

***Tại sao nên cài đặt ORM để truy xuất dữ liệu***

**Portable – tính năng động**: ORM được sử dụng để bạn viết cấu trúc của bạn một lần và lớp ORM sẽ xử lý câu lệnh cuối cùng phù hợp với DBMS được cấu hình. Đây là một lợi thế tuyệt vời khi thao tác truy xuất dữ liệu đơn giản như giới hạn được thêm vào dưới dạng ‘limit 0,100’ ở cuối câu lệnh **Select trong MySQL**, trong khi đó với cách truy xuất thông thường bạn phải viết là ‘Select Top 100 From Table’ trong MS SQL.

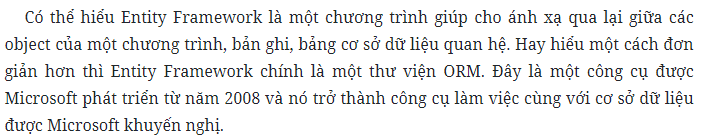
**Nesting Of Data – truy xuất lồng dữ liệu**:  trong trường hợp database bạn có nhiều bảng và các bảng này liên hệ rối rắm phức tạp về dữ liệu thì ORM sẽ tự động lấy dữ liệu cho bạn một cách đơn giản(ở đây đang nói tới việc đơn giản trong truy xuất dữ liệu, không bàn về vấn đề tối ưu truy xuất)

**Single Language** – **không cần biết SQL:**thật vậy với nguyên lý thiết kế là ánh xạ toàn bộ dữ liệu lấy được từ DBMS sang bộ nhớ nên việc thao tác truy suất bây giờ chỉ phụ thuộc vào ngôn ngữ lập trình bạn đang xử dụng, bạn chẳng cần quan tâm phía đằng sau của ORM sẽ làm gì sinh ra mã SQL như thế nào khi truy xuất SQL, và kết quả là chúng ta chỉ cần nhuần nhuyễn ngôn ngữ lập trình đang dùng.

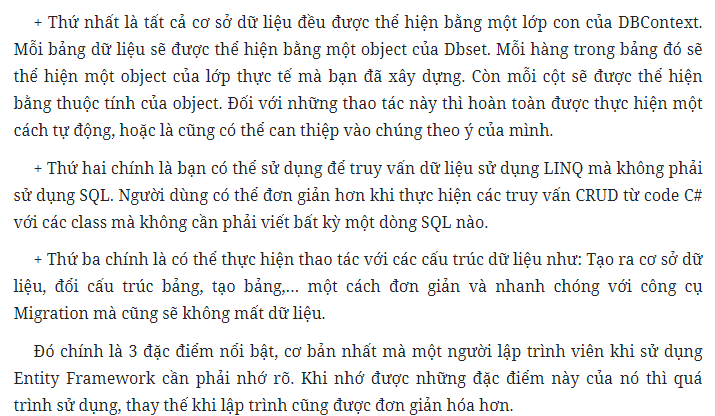
**Adding is like modifying – thêm sửa dữ liệu là như nhau:**đối với ORM, nó không phân biệt giữa thêm mới và cập nhật mọi tác vụ có liên quan đến sửa đổi hay chèn dữ liệu đều được xem là định nghĩa thêm mới, hai tác vụ này được xem như là một

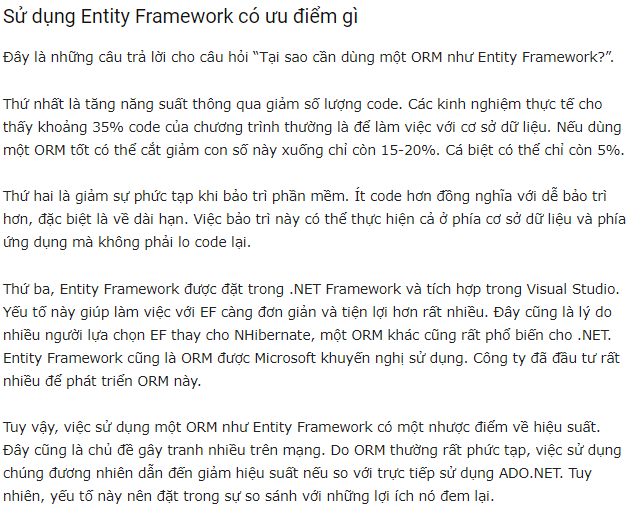
## Entity Framework

**Khái niệm**

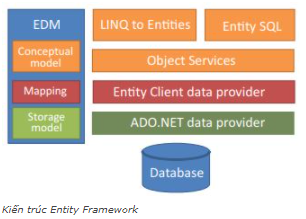


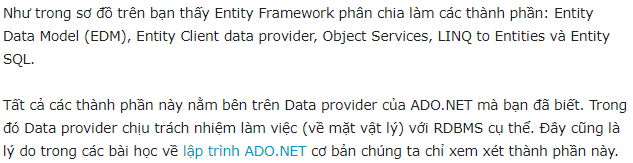
**Đặc điểm**

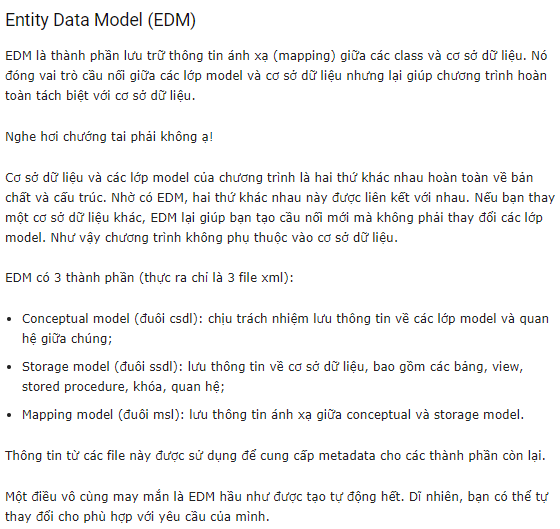


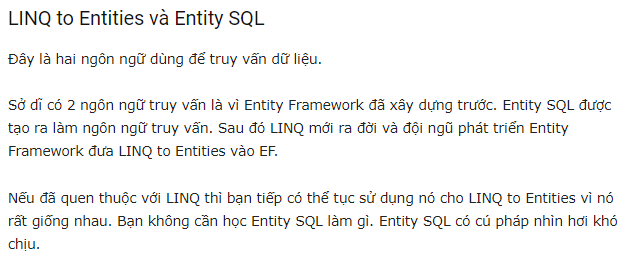


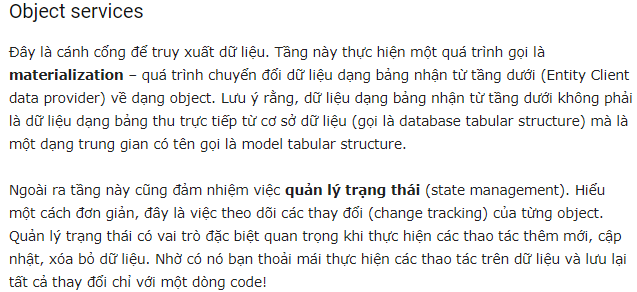
Kiến trúc entity framework

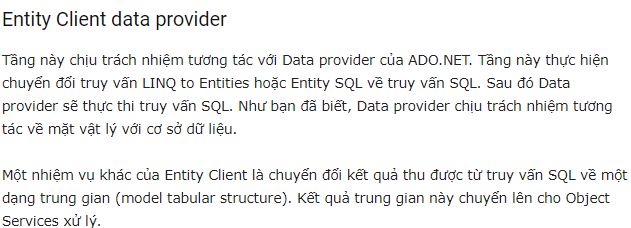




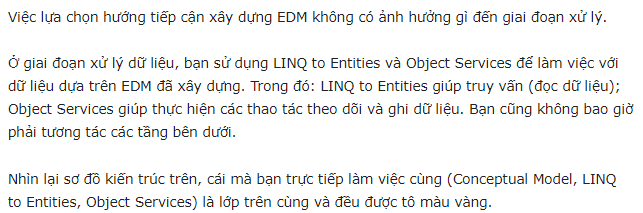


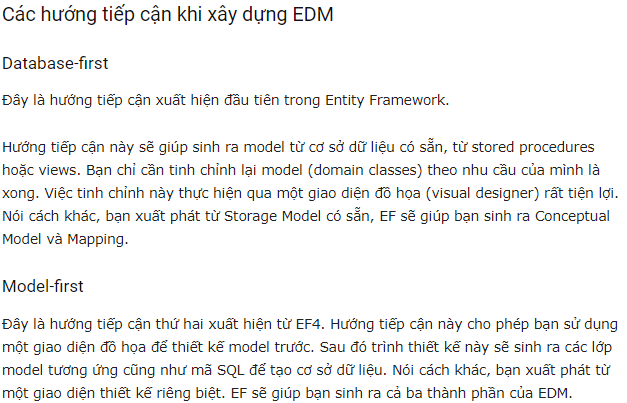


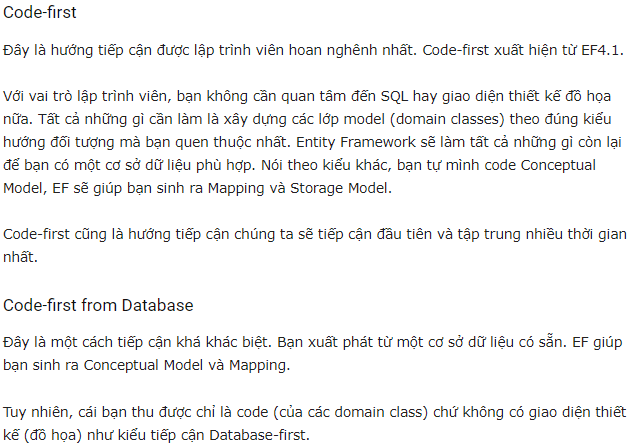








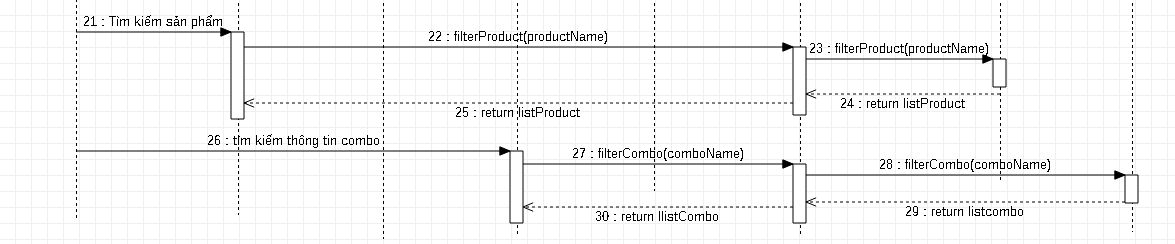
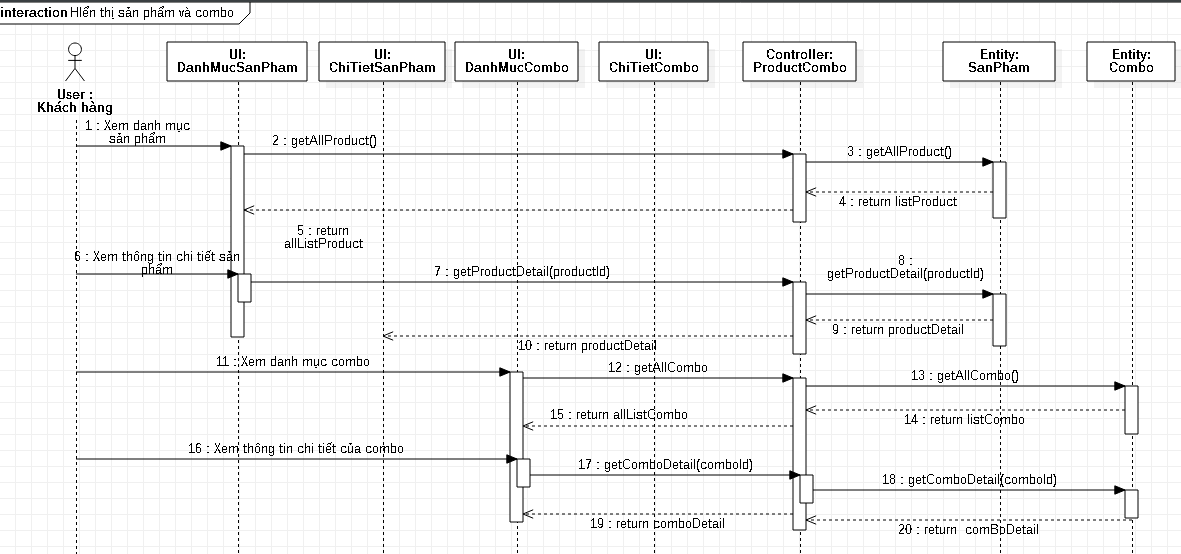




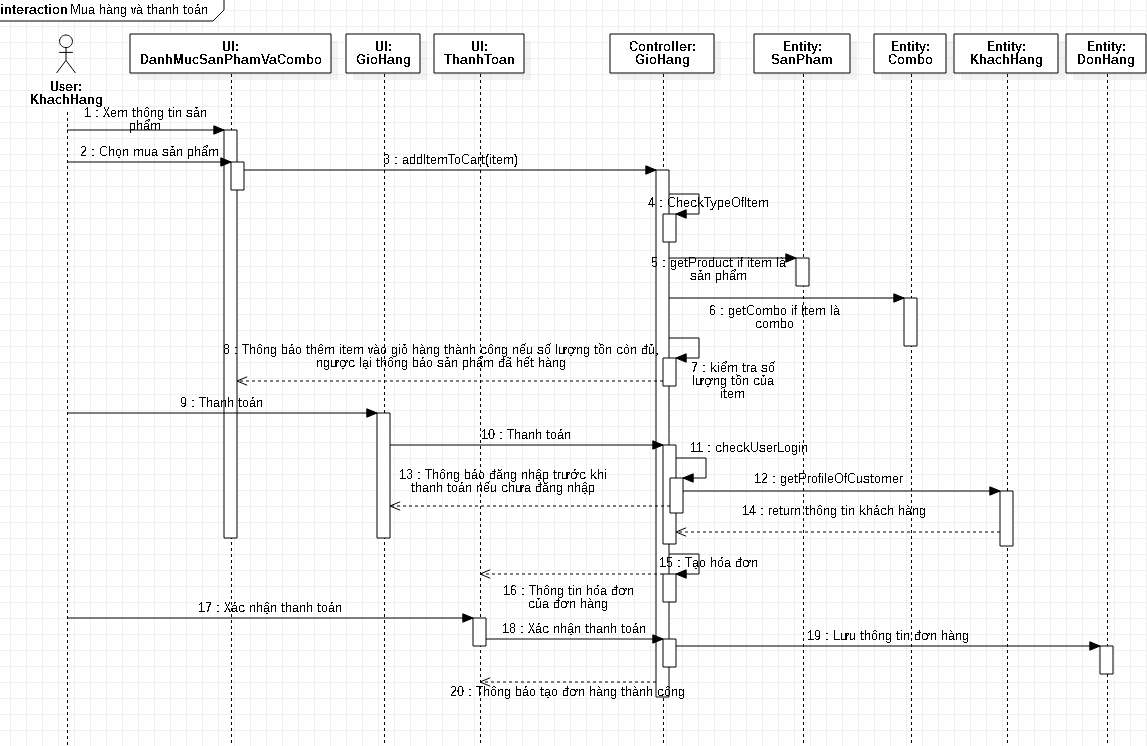
# **Phần 2**

## *I. Quản lý bán hàng (MVC)*

**Sequence hiển thị sản phẩm và combo**

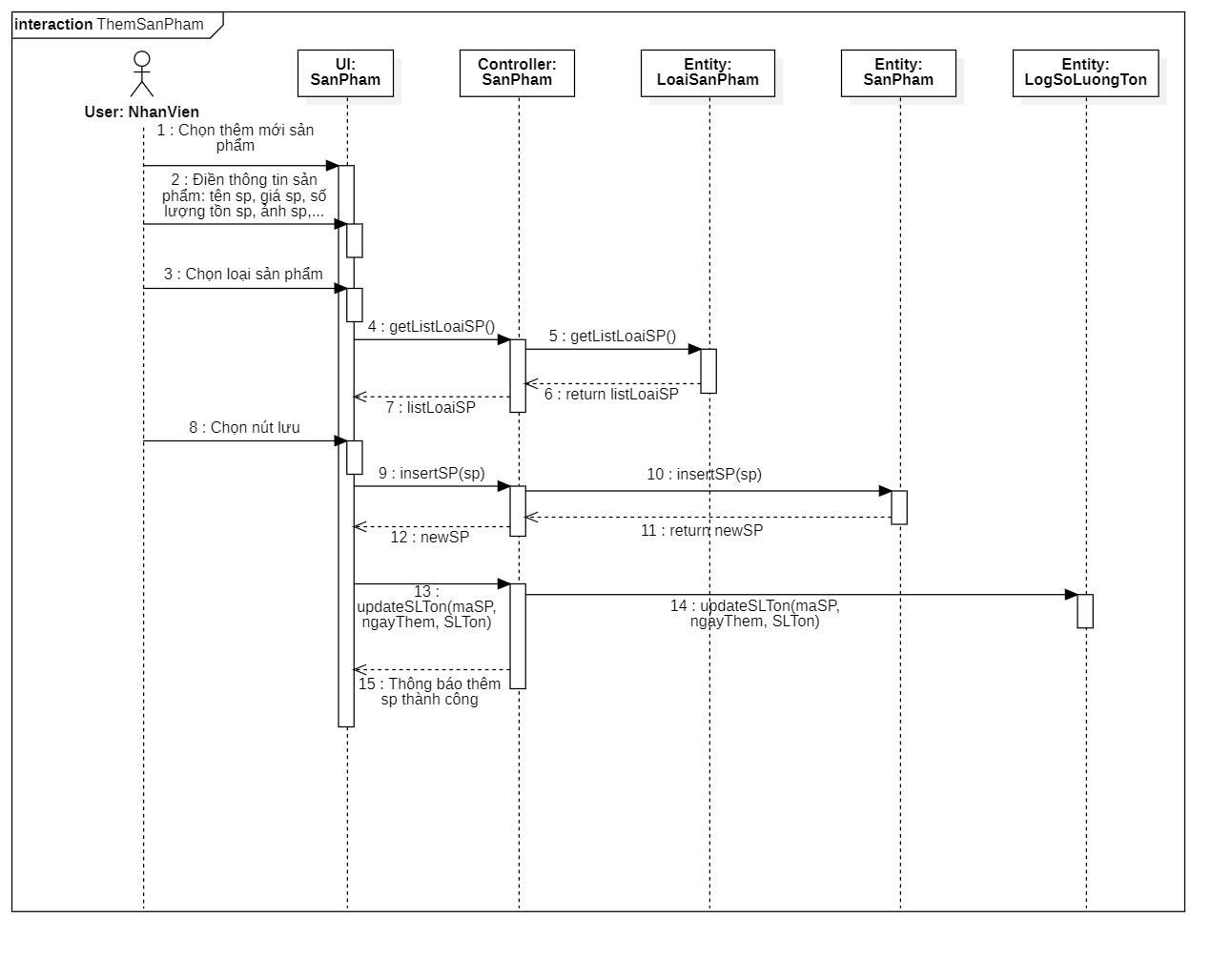


**2. Sequence thanh toán đặt hàng**

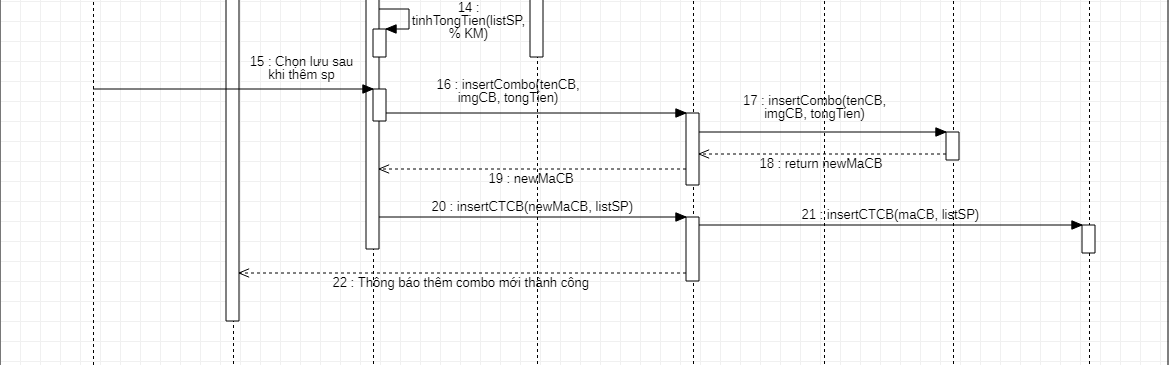
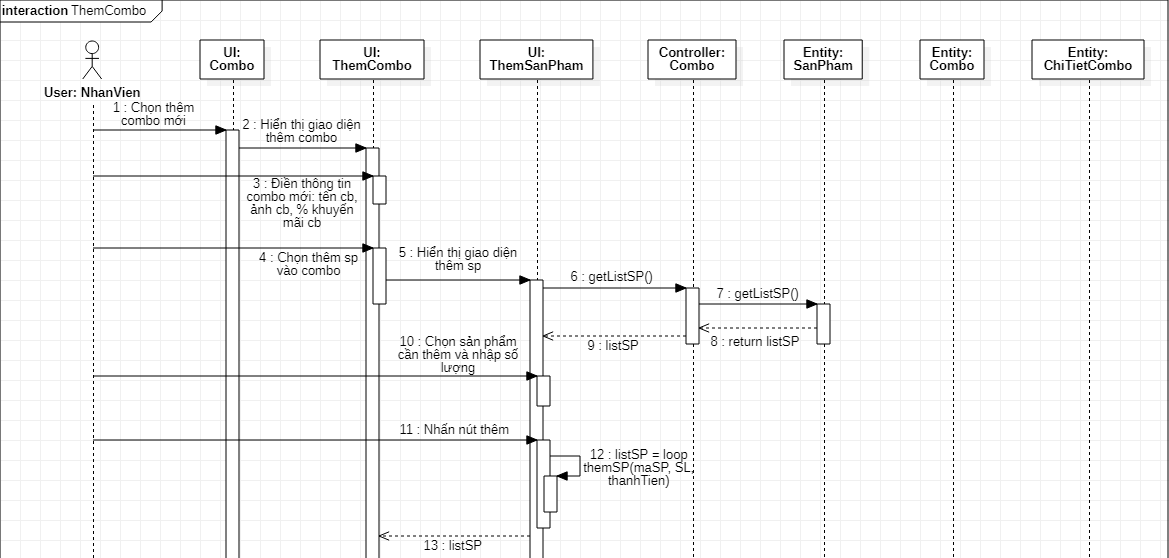


**3. Thêm sản phẩm, thêm combo**

**3.1. Thêm sản phẩm**

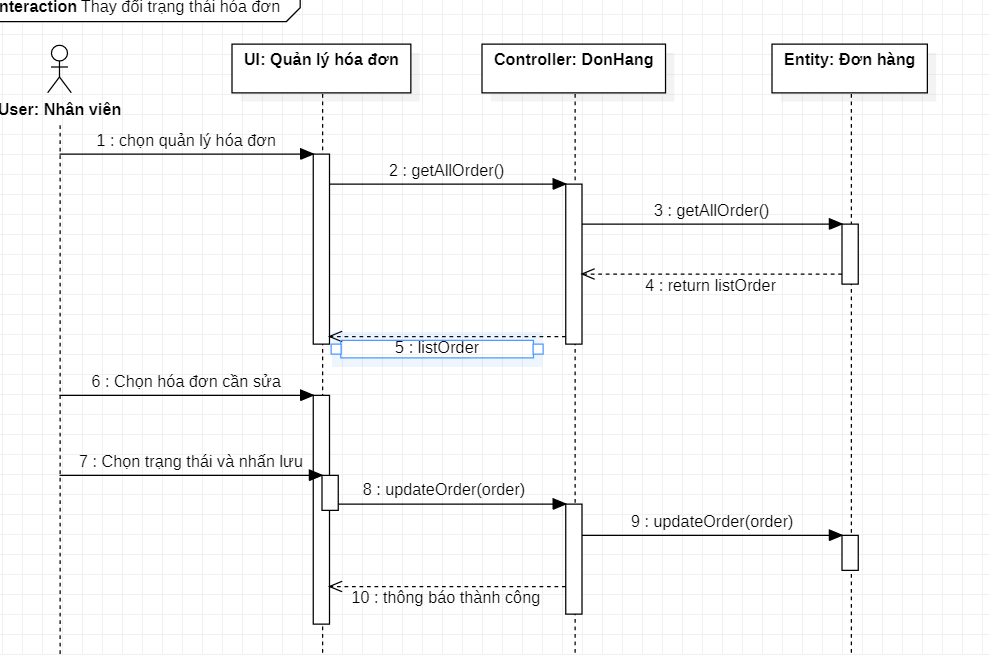


**3.2. Thêm combo**

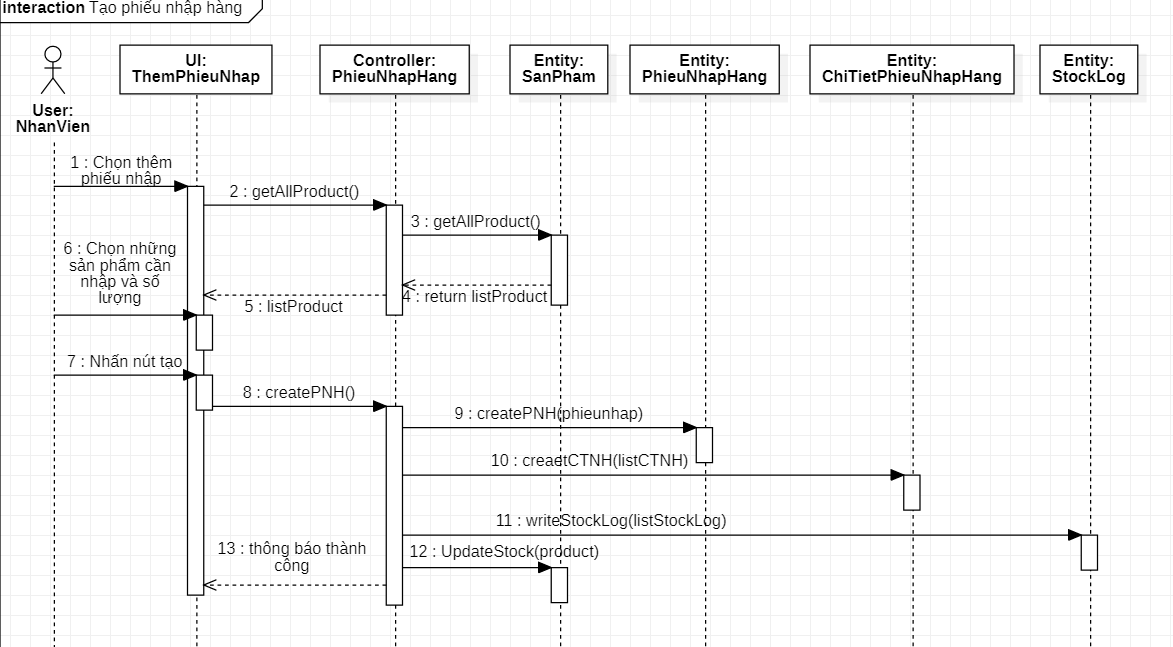


**4. Sửa combo**

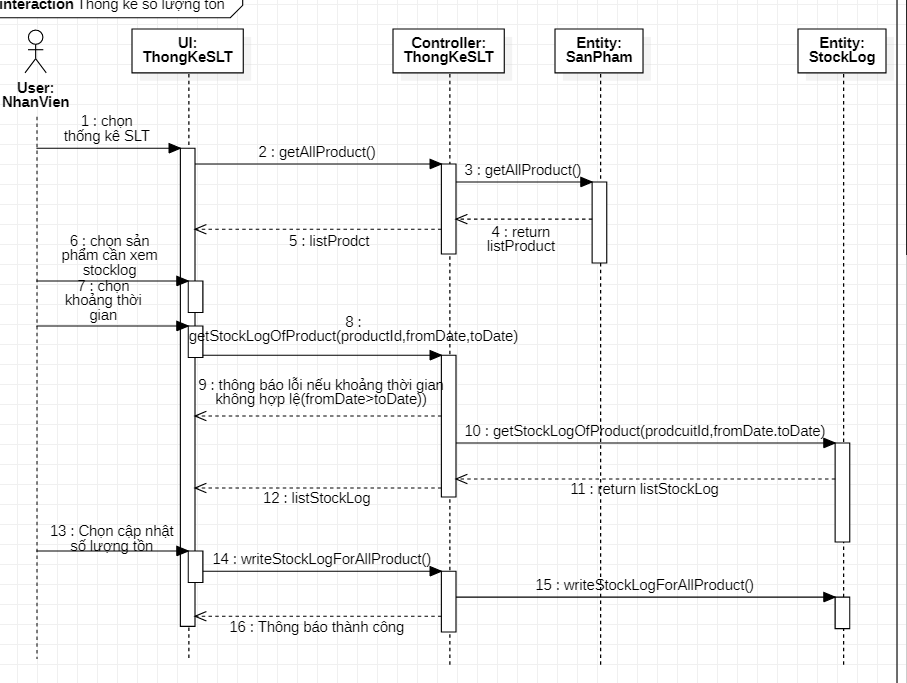
**5. Thay đổi trạng thái đơn hàng**



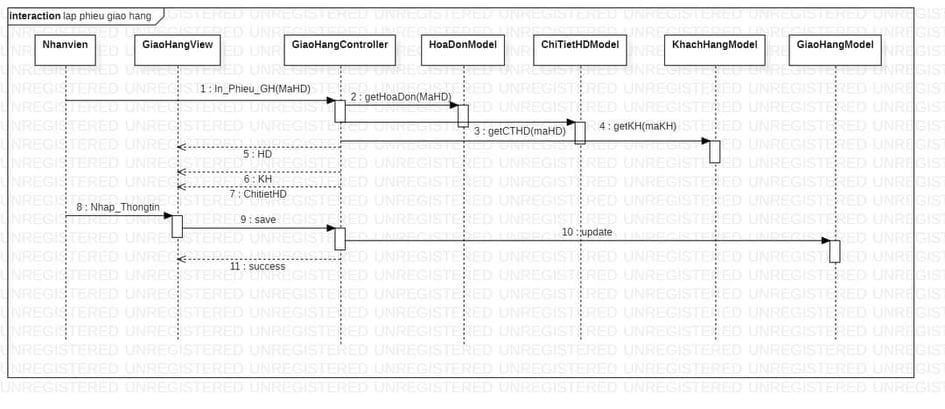
1. **Tạo phiếu nhập hàng**

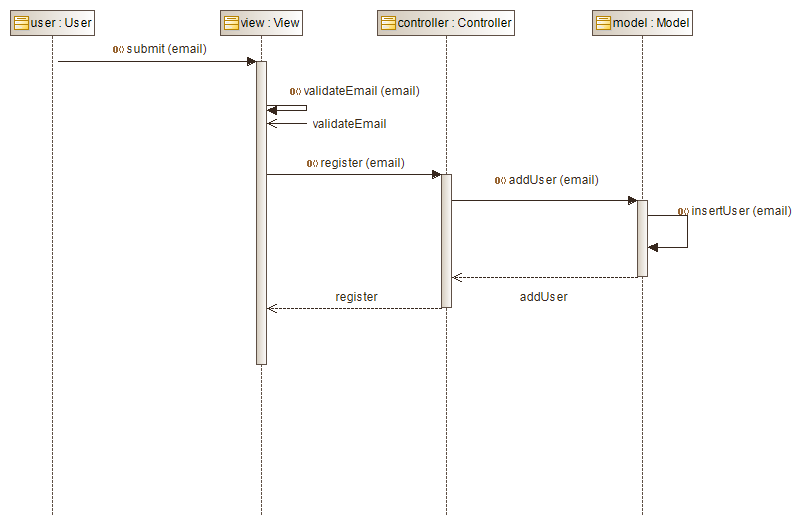


**7. Thống kê tồn kho**



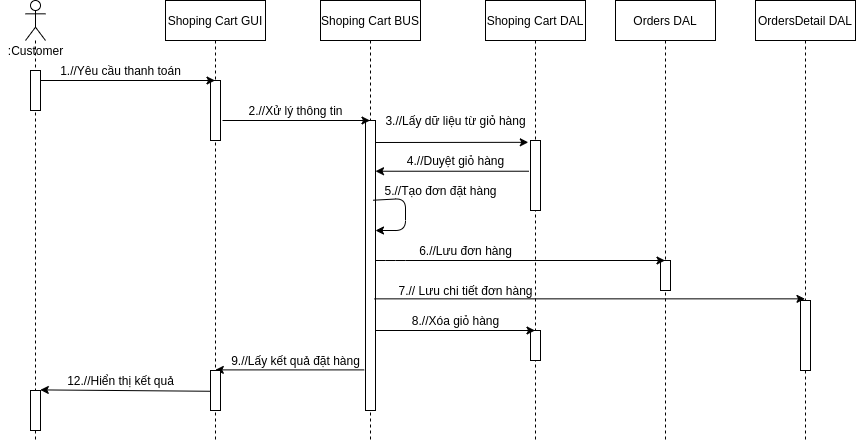
**8. Lập phiếu giao hàng**



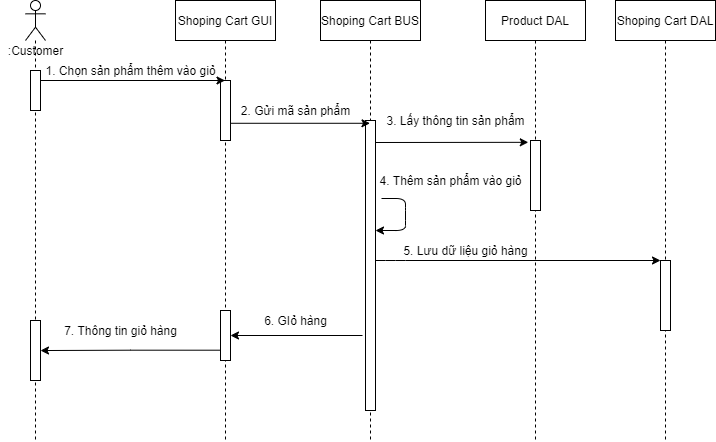


## *II. Quản lý bán hàng (3 lớp)*

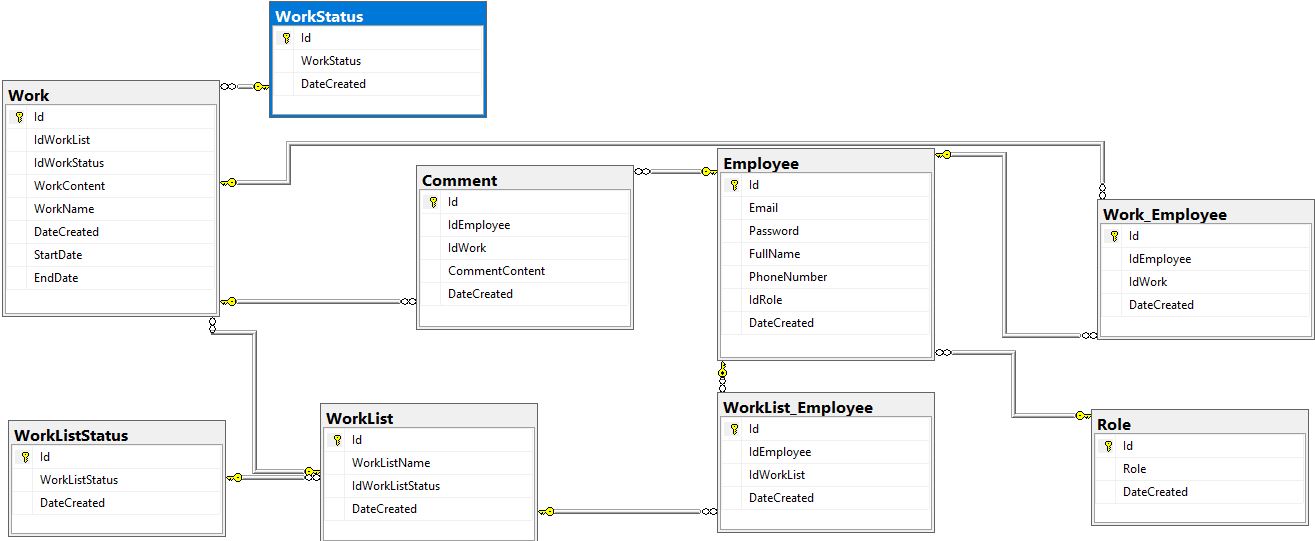
* + - **Chức năng đặt hàng**



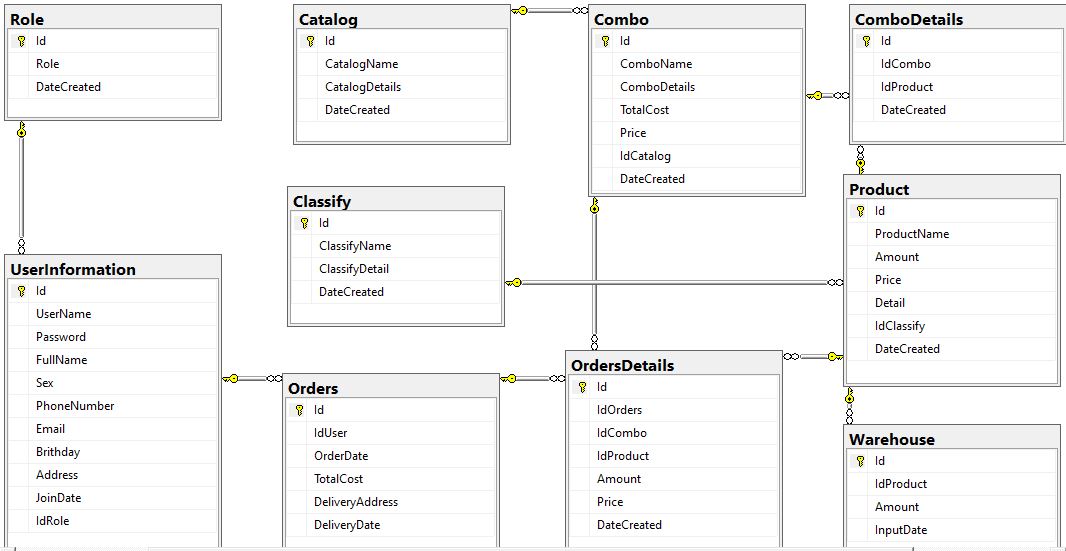
* + - **Chức năng thêm hàng vào giỏ**



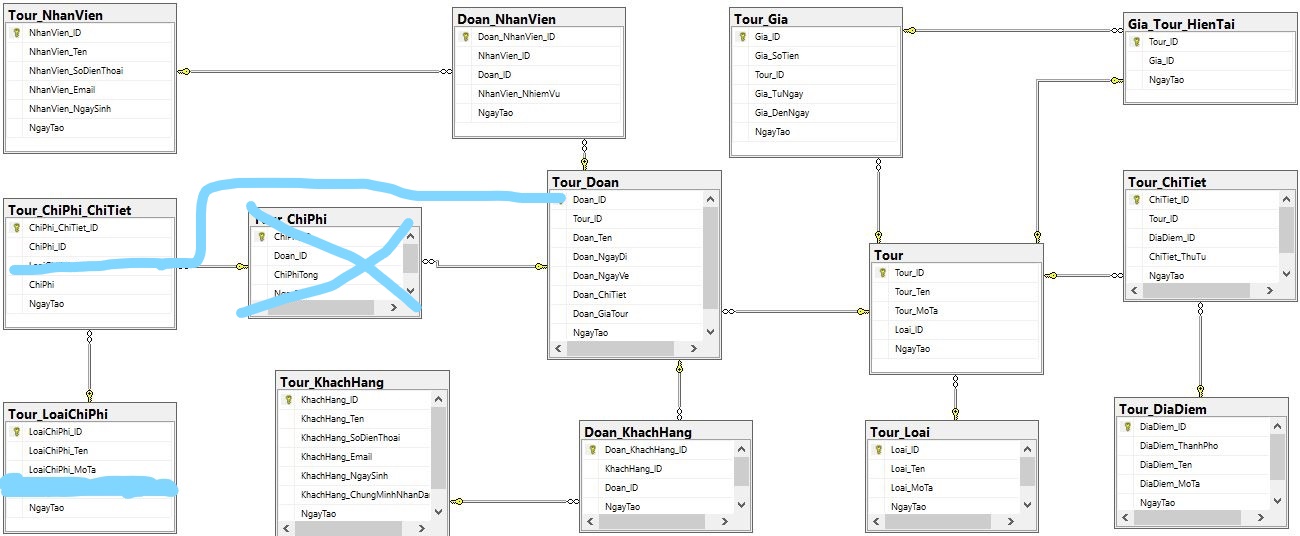
**DB của Todolist**

****

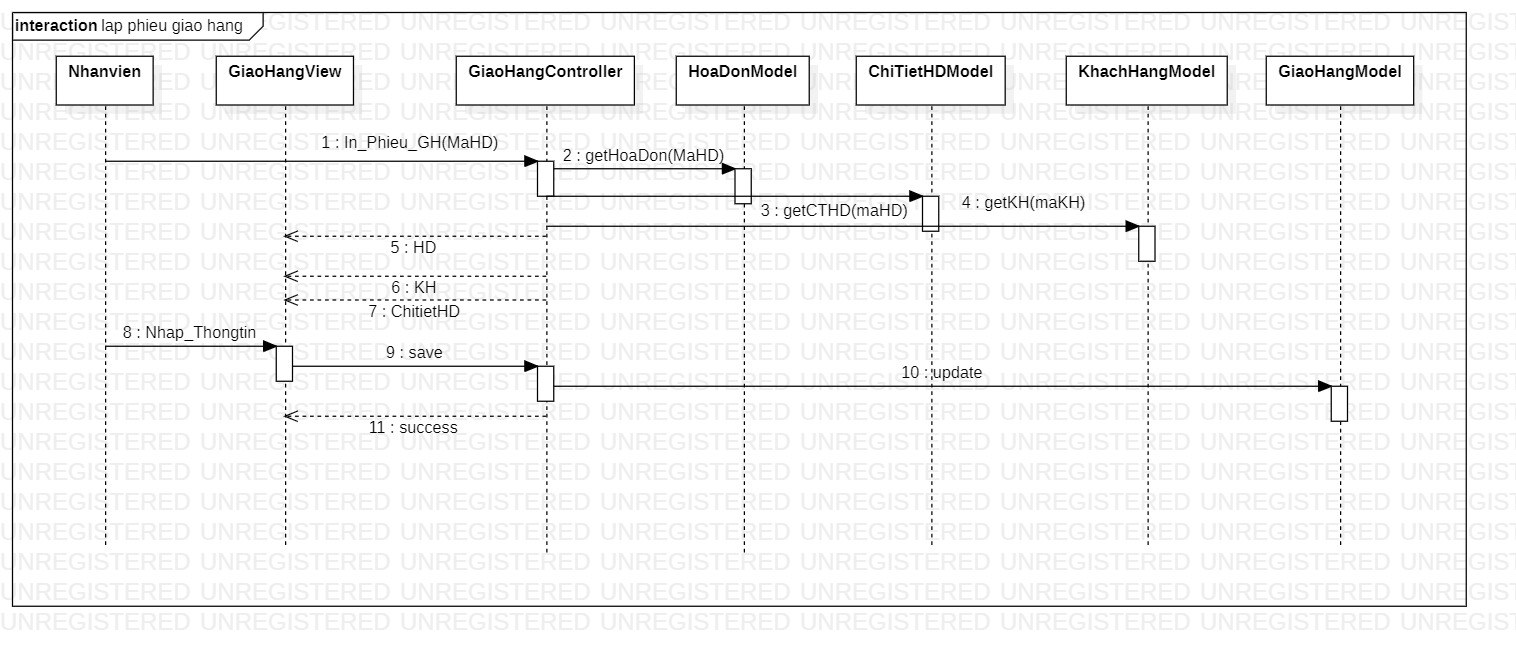
**DB của Quản lý bán hàng**

****

**DB của Tour**

****

**Ví dụ của thầy**

****

