BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

------o0o------

BÁO CÁO BÀI TẬP LỚN LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Tên nhóm : Nhóm 28

Bài tập lớn : Số 19

Họ và tên sinh viên :

Phạm Mạnh Hùng	KTMT04_K64	MSSV: 20194293
Lê Tôn Năng	KTMT04_K64	MSSV: 20194339
Đồng Duy Tiến	KTMT04_K64	MSSV: 20194381

Giảng viên hướng dẫn: GV. Đỗ Thị Ngọc Diệp

Hà Nội, ngày 20 tháng 5 năm 2021

BẢNG DANH SÁCH NHÓM VÀ PHÂN CÔNG CÔNG VIỆC

STT	Họ và tên	Mã sinh viên	Email	Công việc	Mức độ hoàn thành	Ghi chú
01	Phạm Mạnh Hùng	20194293	hung.pm194293@sis.hust.edu.vn	- Tìm hiểu: Singleton, mô hình 3 Layers. - Lập trình DAL, BUS. - Viết báo cáo phần thiết kế chương trình, thiết kế chi tiết.	100%	Trưởng nhóm
02	Lê Tôn Năng	20194339	nang.lt194339@sis.hust.edu.vn	- Tìm hiểu: Winforms, mô hình 3 Layers. - Lập trình GUI, đăng nhập. - Viết báo cáo phần ưu nhược điểm, thiết kế chi tiết.	100%	
03	Đồng Duy Tiến	20194381	tien.dd194381@sis.hust.edu.vn	- Tìm hiểu: UML, C#, mô hình 3 Layers. - Lập trình DTO. - Viết báo cáo phần tổng quan, vẽ sơ đồ lớp, thiết kế chi tiết.	100%	

MỤC LỤC

Trang

I. Tổng quan

1. Giới thiệu bài toán

Bài toán Quản lý khu trưng bày:

Một khu trưng bày cho thuê các gian hàng khác nhau gồm: gian hàng cao cấp và gian hàng tiêu chuẩn. Thông tin chung gian hàng gồm: mã gian hàng, diện tích, vị trí gian hàng. Thông tin riêng cho gian hàng cao cấp gồm: số lượng quạt làm mát, số lượng bàn ghế theo kèm. Thông tin riêng cho gian hàng tiêu chuẩn gồm: chất liệu mái che, chất liệu vách ngăn.

Quản lý khách thuê gian hàng gồm các thông tin: tên, địa chỉ, thời gian bắt đầu/kết thúc thuê, mã gian hàng, tiền đặt cọc.

Xây dựng một hệ thống quản lý việc thuê gian hàng gồm các chức năng:

- Thêm, bớt, sửa, xóa một gian hàng trong khu trưng bày.
- Thống kê số lượng gian hàng được thuê ở thời điểm hiện tại.
- Tìm kiếm gian hàng theo mã gian hàng, hiển thị toàn bộ thông tin về gian hàng.
- Biết rằng chi phí thuê gian hàng tiêu chuẩn = diện tích gian hàng * 100.000đ/ngày. Chi phí thuê gian hàng cao cấp = diện tích gian hàng * 120.000đ/ngày + số lượng quạt làm mát * 50.000đ/ngày.

  Viết phương thức đa hình tính chi phí cho thuê gian hàng theo từng loại.

  Tính doanh thu của khu trưng bày theo một khoảng thời gian nhập vào.

2. Giới thiệu chương trình

2.1. Thông tin chung về chương trình

- Link source code: [IT3100_124175_BaiTapLon_Nhom28_BTL19 - Google Drive]

- Link github: PhamHung2020/BTL_OOP: Bài tập lớn OOP (github.com)
- Ngôn ngữ sử dụng: C#
- Công nghệ GUI: Winform
- Nền tảng: .NET Framework 4.7.2 (đã cài đặt sẵn trên hệ điều hành Windows 10 version 1803 trở lên).
- Công cụ lập trình: Visual Studio 2019 Community
  - Link cài đặt: Download Visual Studio 2019 for Windows & Mac (microsoft.com)
  - Hướng dẫn cài đặt: How to Download and Install Visual Studio - YouTube. Khi lựa chọn các gói hỗ trợ lập trình trong Visual Studio Installer, chọn gói .NET desktop development.
- Các thư viện sử dụng tiêu biểu có sẵn: Winform, LinQ.

2.2. Một số lưu ý khi chạy chương trình

- Khi mở chương trình bằng Visual Studio 2019 và mở các file thuộc phần GUI, có thể sẽ hiện lên cửa sổ Activate. Người sử dụng cần đợi 10 giây và tắt cửa số đó để có thể truy cập các file GUI.
- Khi chạy chương trình, do có chức năng đăng nhập nên người dùng cần đăng nhập với tài khoản:
  - Username: nh3
  - Password: 123456

2.3. Các tài liệu đi kèm source code trong file .zip

- README.txt: ghi chú phiên bản của môi trường lập trình, chạy chương trình.

- API BUS: thông tin chi tiết về các phương thức mà GUI có thể gọi đến sử dụng. Tài liệu này được cung cấp cho thành viên đảm nhiệm phần GUI của nhóm sử dụng.
- Biểu đồ lớp UML: biểu đồ lớp đã được đính kèm dưới dạng file .pdf riêng để dễ quan sát hơn.
- Chú ý: nên giải nén file .zip bằng WinRar.

II. Phân tích và thiết kế kiến trúc chương trình

1. Thiết kế chương trình

1.1. Tổng quan về mô hình 3 Layers

Mô hình 3 Layers là một mô hình lập trình trong đó phân chia các thành phần trong hệ thống thành các layer. Các thành phần (thường là các class) có cùng chức năng sẽ được sẽ được nhóm lại thành một layer, mỗi layer đảm nhiệm một chức năng nhất định.

Mô hình 3 Layer giúp phân chia công việc một cách rõ ràng, từ đó giúp người lập trình quản lý và bảo trì project tốt hơn. Mô hình này khá phổ biến đối với sinh viên và thường được sử dụng trong lập trình desktop app C#.

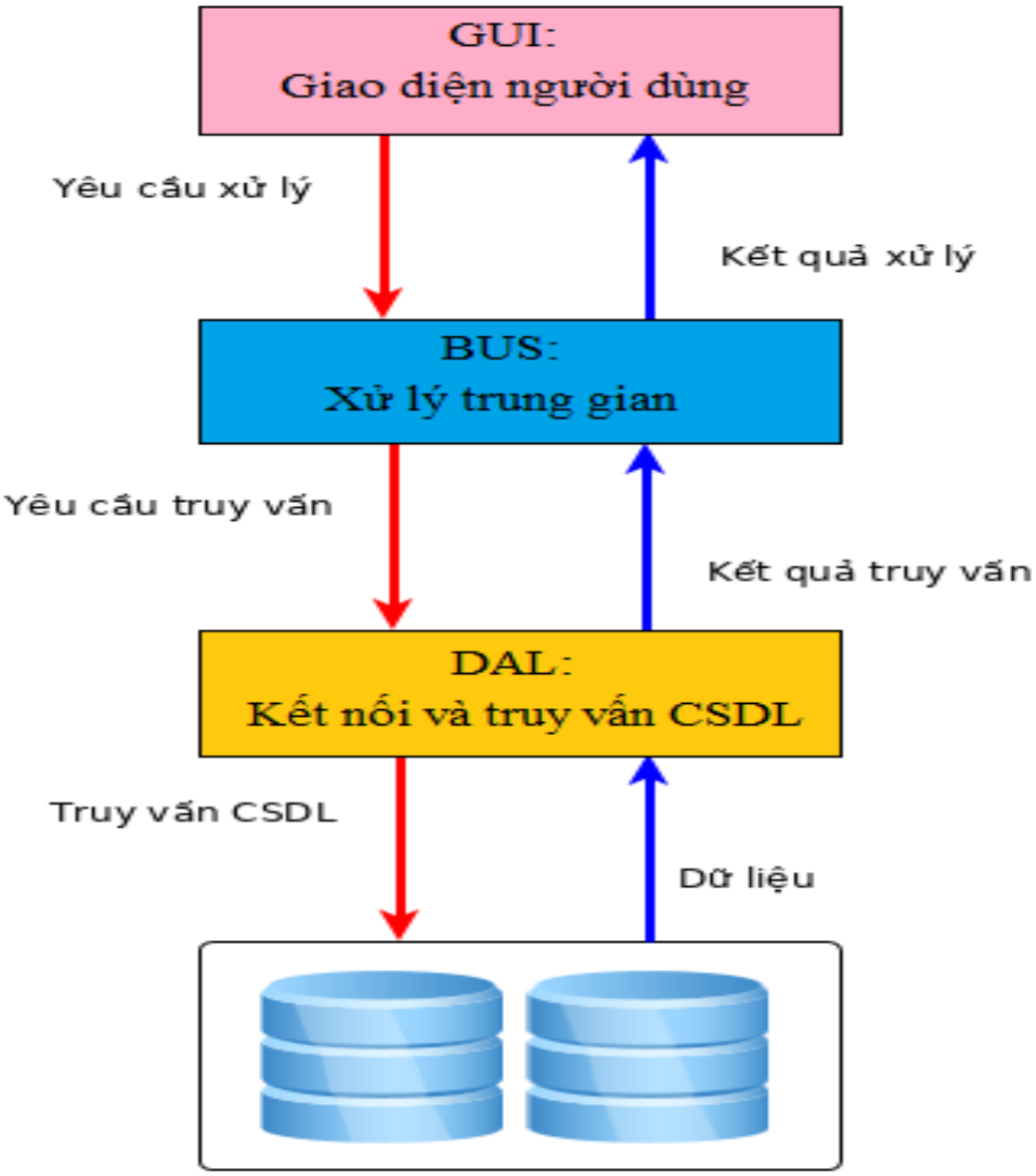
Mô hình 3 Layer gồm các thành phần sau:

- **Presentation Layer (GUI):** hiển thị giao diện và tương tác trực tiếp với người dùng như nhập dữ liệu, hiển thị dữ liệu, ... và gọi đến BUS để xử lý.
- **Business Logic Layer (BUS):**
  - Nhận dữ liệu và yêu cầu từ GUI, xử lý dữ liệu và truyền xuống cho DAL; hoặc lấy dữ liệu từ DAL, xử lý và truyền lên cho GUI hiển thị.
  - Công việc xử lý dữ liệu bao gồm: kiểm tra tính hợp lệ của dữ liệu, các ràng buộc, tính toán, yêu cầu của nghiệp vụ mà bài toán đặt ra.

- **Data Access Layer (DAL):** là layer duy nhất được giao tiếp trực tiếp với nguồn dữ liệu (database, file, ...), thực hiện các công việc lưu trữ và truy vấn dữ liệu theo yêu cầu từ BUS.
- **Data Transfer Object (DTO):** đây không phải là một layer, mà là tập hợp gồm các class ánh xạ với các đối tượng cần quản lý (Ví dụ: Gian hàng, Khách hàng, ...) và các cấu trúc dữ liệu (List<T>, HashSet<T>, ...) quản lý các đối tượng này. DTO vừa đóng vai trò là dữ liệu được lưu trữ trong quá trình chạy chương trình, vừa là đối tượng được các layer sử dụng để trao đổi thông tin.

Các layer có nhiều cách đặt tên khác nhau, trong project này các layer được đặt tên lần lượt là GUI, BUS, DAL.

Các class thuộc vào layer nào thì sẽ có tên của layer đó vào sau tên của class.



Hình 1 : Mô hình 3 Layers

1.2. Áp dụng mô hình 3 Layers vào bài toán

Trong project này, khu trưng bày được thiết kế bao gồm nhiều tầng (cụ thể là 4), mỗi tầng có một số lượng vị trí gian hàng nhất định, các vị trí được đánh số thứ tự. Khi đó, vị trí gian hàng sẽ là sự kết hợp của số tầng và số thứ tự đó. Ví dụ: một gian hàng nằm tại tầng 2, vị trí số 2 sẽ có vị trí gian hàng là 202. Mã gian hàng sẽ dựa vào loại gian hàng và

vị trí gian hàng đó. Ví dụ: gian hàng tiêu chuẩn (cao cấp) có vị trí 202 sẽ có mã là TC202 (CC202).

Từ yêu cầu bài toán, ta có các đối tượng cần quản lý sau : *Gian hàng, Gian hàng cao cấp, Gian hàng tiêu chuẩn, Khách hàng*. Ngoài ra, để giới hạn đối tượng sử dụng chương trình, cần có thêm một đối tượng là *tài khoản* người dùng.

Với mỗi đối tượng cần quản lý, ta sẽ cần một class để mô tả đối tượng đó, và class đó sẽ nằm trong phần DTO. Tiếp đến, để lưu trữ và quản lý danh sách các đối tượng, ta sẽ cần đến một class khác tương ứng thuộc phần DAL. Cuối cùng, để xử lý nghiệp vụ trung gian cũng như cung cấp các chức năng liên quan đến đối tượng, ta cần class tương ứng trong BUS.

Chương trình sau khi áp dụng mô hình này sẽ gồm các thành phần sau:

- **DTO**: gồm các class
  - *GianHangDTO (abstract):* lưu trữ các thông tin về một gian hàng nói chung.
  - *GianHangTieuChuanDTO:* kế thừa từ GianHangDTO, lưu trữ thông tin về một gian hàng tiêu chuẩn.
  - *GianHangCaoCapDTO:* kế thừa từ GianHangDTO, lưu trữ thông tin về một gian hàng cao cấp.
  - *KhachHangDTO:* lưu trữ thông tin về một khách hàng.
  - *TaiKhoanDTO:* lưu trữ thông tin về một tài khoản.

- **DAL**: gồm các class
  - *GianHangDAL:* chứa danh sách các gian hàng, chịu trách nhiệm lưu trữ và quản lý thông tin về các gian hàng.
  - *KhachHangDAL:* chứa danh sách khách hàng, chịu trách nhiệm lưu trữ và quản lý thông tin về các khách hàng.

- *TaiKhoanDAL:* chứa danh sách các tài khoản, chịu trách nhiệm lưu trữ và quản lý thông tin về các tài khoản.

- **BUS**: gồm các class
  - *GianHangBUS:* có chức năng xử lý dữ liệu, nghiệp vụ liên quan đến các gian hàng.
  - *KhachHangBUS:* có chức năng xử lý dữ liệu, nghiệp vụ liên quan đến các khách hàng.
  - *KhuTrungBayBUS:* có chức năng xử lý dữ liệu, nghiệp vụ liên quan đến khu trưng bày nói chung, và các vấn đề liên quan đến cả thông tin gian hàng lẫn khách hàng (như thuê gian hàng, tính doanh thu, ...).
  - *TaiKhoanBUS:* có chức năng xử lý vấn đề về đăng nhập.
  - *HelperBUS (static):* chứa một số phương thức tiện ích.
- **GUI**: gồm các form có chức năng hiển thị dữ liệu về gian hàng, khách hàng, doanh thu, … , đồng thời nhận yêu cầu và dữ liệu từ người dùng như thêm gian hàng, xem thông tin khách hàng, … Do số lượng form khá lớn cũng như đặc thù của layer này nên danh sách form không được liệt kê tại đây.

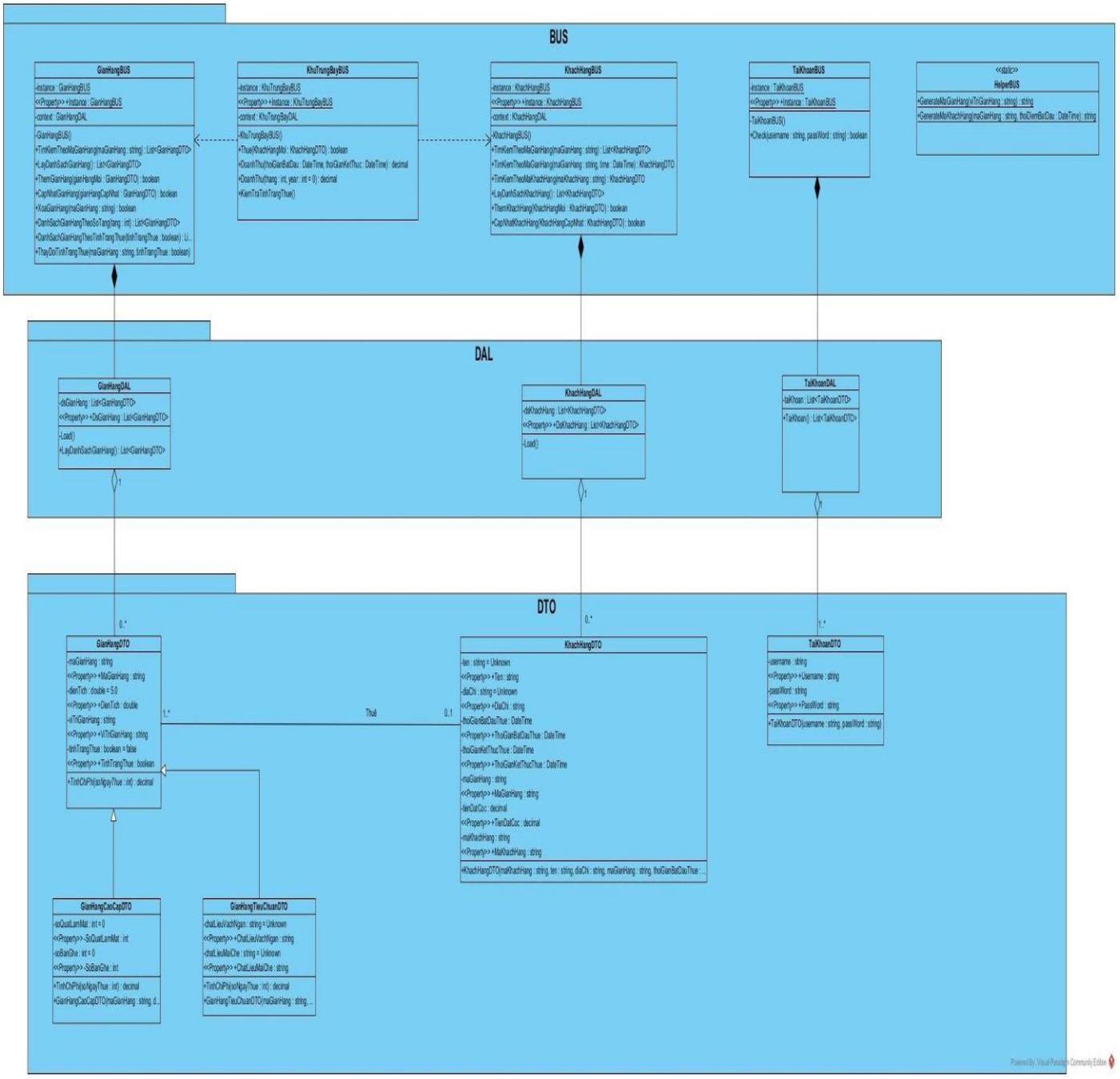
Theo đúng mô hình, các class thuộc DAL sẽ chứa các phương thức thao tác với dữ liệu như: Thêm, Sửa, Xóa dữ liệu, … Tuy nhiên, do project không yêu cầu sử dụng database hay file để lưu trữ và truy vấn dữ liệu, nên nhóm đã điều chỉnh mô hình để phù hợp hơn với project. Cụ thể:

- DAL sẽ trở thành nơi khởi tạo dữ liệu ban đầu (tránh việc mỗi lần chạy chương trình đều phải nhập lại dữ liệu từ đầu) và lưu trữ toàn bộ dữ liệu khi chương trình đang chạy. Các thao tác như thêm, sửa, xóa,… lúc này khá đơn giản (ví dụ: thêm dữ liệu thì gọi phương thức Add của collection List<T>)nên sẽ được thực hiện luôn tại BUS.

- BUS sau khi kiểm tra các vấn đề nghiệp vụ sẽ thực hiện các thao tác về dữ liệu luôn thay vì phải gọi đến DAL, tránh việc phải gọi nhiều phương thức một cách không cần thiết trong project này.

Tóm lại, DAL sẽ là nơi khởi tạo, lưu trữ dữ liệu và BUS sẽ chịu trách nhiệm xử lý và quản lý dữ liệu.

2. Biểu đồ lớp



Trên thực tế, DTO được sử dụng tại cả 3 Layer. Tuy nhiên, để biểu đồ lớp dễ hiểu, dễ nhìn hơn, trên biểu đồ chỉ vẽ mối quan hệ giữa DTO với DAL, do DAL là layer trực tiếp lưu trữ và quản lý dữ liệu. GUI cũng không được thêm vào biểu đồ này.

III. Thiết kế chi tiết

1. DTO

1.1. GianHangDTO

GianHangDTO là lớp trừu tượng, khai báo các thuộc tính chung của gian hàng và các phương thức khởi tạo. Các thành phần của GianHangDTO bao gồm:

- *private string _maGianHang*

  Là thuộc tính sử dụng chuỗi ký tự để lưu trữ mã gian hàng.

- *public string MaGianHang*

  Là property cho _maGianHang (tương tự như getter, setter). Độ dài mã gian hàng gồm 5 ký tự, có dạng: loại gian hàng (TC/CC) + vị trí gian hàng. VD: gian hàng cao cấp có vị trí ở tầng 1, vị trí số 2 sẽ có mã gian hàng CC102.

- *private double _dienTich = 5.0*

  Là thuộc tính diện tích của gian hàng lưu dưới dạng số thực và giá trị khởi tạo là $5m^2$.

- *public double DienTich*

  Là property cho _dienTich (tương tự như getter, setter).

- *private string _viTriGianHang*

  Là thuộc tính vị trí gian hàng lưu dưới dạng chuỗi ký tự.

- *public string ViTriGianHang*

  Là property cho _viTriGianHang (tương tự như getter, setter).

- *private bool _tinhTrangThue = false*

  Là thuộc tính trạng thái thuê của một gian hàng, khởi tạo là chưa được thuê (false).

- *public bool TinhTrangThue*

  Là property cho _tinhTrangThue(tương tự như getter, setter).

- *public GianHangDTO (string maGianHang, double dienTich, string viTriGianHang, bool tinhTrangThue)*

  Là phương thức khởi tạo của lớp GianHangDTO gồm các tham số: mã gian hàng, diện tích, vị trí gian hàng, tính trạng thuê.

- *public abstract decimal TinhChiPhi (int soNgayThue)*

  Là phương thức trừu tượng tính chi phí theo số ngày thuê của gian hàng.

1.2. GianHangTieuChuanDTO

GianHangTieuChuanDTO là class kế thừa từ GianHangDTO, khai báo các thuộc tính của gian hàng tiêu chuẩn và các phương thức khởi tạo. Các thành phần của GianHangTieuChuanDTO bao gồm:

- *private string _chatLieuVachNgan = "Unknown"*

  Là thuộc tính chất liệu vách ngăn của gian hàng tiêu chuẩn, lưu dưới dạng chuỗi ký tự.

- *public string ChatLieuVachNgan*

  Là property cho _chatLieuVachNgan (tương tự như getter, setter).

- *private string _chatLieuMaiChe = "Unknown"*

  Là thuộc tính chất liệu mái che của gian hàng tiêu chuẩn, lưu dưới dạng chuỗi ký tự.

- *public string ChatLieuMaiChe*

  Là property cho _chatLieuMaiChe (tương tự như getter, setter).

- *public GianHangTieuChuanDTO(string maGianHang, double dienTich, string viTriGianHang, bool tinhTrangThue, string chatLieuVachNgan, string*

chatLieuMaiChe) : base(maGianHang, dienTich, viTriGianHang, tinhTrangThue)

Là phương thức khởi tạo của GianHangTieuChuanDTO, sử dụng base để gọi được đến constructor của lớp cha.

- *public override decimal TinhChiPhi (int soNgayThue)*

  Là phương thức tính chi phí thuê gian hàng tiêu chuẩn theo số ngày thuê ghi đè lên phương thức tính chi phí của class GianHangDTO. Chi phí tính theo công thức: 10000 * DienTich * soNgayThue.

1.3. GianHangCaoCapDTO

GianHangCaoCapDTO là class kế thừa từ GianHangDTO, khai báo các thuộc tính của gian hàng cao cấp và các phương thức khởi tạo. Các thành phần của GianHangCaoCapDTO bao gồm:

- *private int _soQuatLamMat = 0*

  Là thuộc tính số quạt làm mát của gian hàng cao cấp, lưu dưới dạng số nguyên với giá trị ban đầu là 0.

- *public int SoQuatLamMat*

  Là property cho _soQuatLamMat (tương tự như getter, setter).

- *private int _soBanGhe = 0*

  Là thuộc tính số bàn ghế của gian hàng cao cấp, lưu dưới dạng số nguyên với giá trị ban đầu là 0.

- *public int SoBanGhe*

  Là property cho _soBanGhe (tương tự như getter, setter).

- *public GianHangCaoCapDTO(string maGianHang, double dienTich, string viTriGianHang, bool tinhTrangThue, int soQuatLamMat, int soBanGhe) : base(maGianHang, dienTich, viTriGianHang, tinhTrangThue)*

Là phương thức khởi tạo của GianHangCaoCapDTO, sử dụng base để gọi được đến constructor của lớp cha.

- *public override decimal TinhChiPhi (int soNgayThue)*

  Là phương thức tính chi phí thuê gian hàng tiêu chuẩn theo số ngày thuê ghi đè lên phương thức tính chi phí của class GianHangDTO. Chi phí tính theo công thức: 120000 * DienTich * soNgayThue + _soQuatLamMat * 50000.

1.4. KhachHangDTO

KhachHangDTO là class khai báo các thuộc tính của khách hàng và các phương thức khởi tạo. Các thành phần của KhachHangDTO bao gồm:

- *private string _maKhachHang*

  Là thuộc tính mã khách hàng của từng khách hàng, lưu dưới dạng chuỗi ký tự.

- *public string MaKhachHang*

  Là property cho _maKhachHang (tương tự như getter, setter).

- *private string _ten = "Unknown"*

  Là thuộc tính tên khách hàng của từng khách hàng, lưu dưới dạng chuỗi ký tự.

- *public string Ten*

  Là property cho _ten (tương tự như getter, setter).

- *private string _diaChi = "Unknown"*

  Là thuộc tính địa chỉ của từng khách hàng, lưu dưới dạng chuỗi ký tự.

- *public string DiaChi*

  Là property cho _diaChi (tương tự như getter, setter).

- *private string _maGianHang*

  Là thuộc tính sử dụng chuỗi ký tự để lưu trữ mã gian hàng tương ứng với gian hàng mà khách hàng thuê.

- *public string MaGianHang*

  Là property cho _maGianHang (tương tự như getter, setter). Độ dài của mã gian hàng là 5 ký tự, tương tự như mã gian hàng của GianHangDTO.

- *private DateTime _thoiGianBatDauThue*

  Sử dụng DateTime để lưu trữ thời gian bắt đầu thuê của một gian hàng.

- *public DateTime ThoiGianBatDauThue*

  Là property cho _thoiGianBatDauThue (tương tự như getter, setter).

- *private DateTime _thoiGianKetThucThue*

  Sử dụng DateTime để lưu trữ thời gian kết thúc thuê của một gian hàng.

- *public DateTime ThoiGianKetThucThue*

  Là property cho _thoiGianKetThucThue (tương tự như getter, setter). Thời điểm kết thúc thuê phải lớn hơn thời gian bắt đầu thuê.

- *private decimal _tienDatCoc*

  Là thuộc tính sử dụng số thập phân để lưu trữ tiền đặt cọc của khách hàng.

- *public decimal TienDatCoc*

  Là property cho _tienDatCoc (tương tự như getter, setter).

- *public KhachHangDTO(string maKhachHang,*

  string ten,

  string diaChi,

  string maGianHang,

  DateTime thoiGianBatDauThue,

  DateTime thoiGianKetThucThue,

  decimal tienDatCoc)

Là phương thức khởi tạo của class KhachHangDTO.

1.5. TaiKhoanDTO

TaiKhoanDTO là class khởi tạo dữ liệu về tài khoản, mật khẩu đăng nhập vào chương trình. Các thành phần của TaiKhoanDTO bao gồm:

- *private string _username*

  Là thuộc tính tài khoản của người dùng lưu dưới dạng chuỗi ký tự.

- *public string Username*

  Là property cho _username (tương tự như getter, setter).

- *private string _passWord*

  Là thuộc tính mật khẩu của người dùng lưu dưới dạng chuỗi ký tự.

- *public string PassWord*

  Là property cho _passWord (tương tự như getter, setter).

- *public TaiKhoanDTO(string username, string passWord)*

  Là phương thức khởi tạo của class TaiKhoanDTO.

2. DAL

2.1. GianHangDAL

GianHangDAL là class chịu trách nhiệm khởi tạo dữ liệu ban đầu về các gian hàng cho chương trình và lưu trữ danh sách gian hàng trong quá trình chạy. Các thành phần của GianHangDAL bao gồm:

- *private _dsGianHang*

  Là thuộc tính sử dụng collection List để lưu trữ toàn bộ danh sách gian hàng, cả gian hàng tiêu chuẩn và gian hàng cao cấp. Bằng việc áp dụng tính đa hình, upcasting và downcasting, ta không cần phải tạo ra 2 danh sách riêng biệt cho Gian hàng tiêu chuẩn và Gian hàng cao cấp mà có thể gộp chung vào một danh

sách nhưng vẫn đảm bảo các thao tác liên quan đến từng loại gian hàng (như tính chi phí thuê) được thực hiện đúng đắn.

- *public List<GianHangDTO> DsGianHang*

  Là property cho _dsGianHang (tương tự như getter, setter). Trong trường hợp gọi đến DsGianHang (gọi getter), nếu _dsGianHang chưa có dữ liệu thì sẽ gọi hàm Load() để khởi tạo dữ liệu trước khi trả về _dsGianHang.

- *private void _Load()*

  Khởi tạo dữ liệu về một số gian hàng cho _dsGianHang, cũng là dữ liệu cho chương trình.

  Phương thức này nên được private để tránh cho việc bên ngoài có thể gọi tới khiến toàn bộ dữ liệu trở lại như ban đầu, tức là các dữ liệu được thêm, sửa, xóa bị mất đi. Phương thức này chỉ được gọi một lần khi truy cập vào *DsGianHang* lần đầu tiên. Nói cách khác, dữ liệu sẽ chỉ được khởi tạo khi nào cần dùng đến (gần giống với *lazy loading*).

- *public List<T> LayDanhSachGianHang<T>()*

  Là phương thức generic, có kiểu generic T được giới hạn là GianHangDTO và các class con của GianHangDTO.

  Phương thức này có chức năng trả về danh sách gian hàng theo kiểu dữ liệu được truyền vào. Nếu T là GianHangDTO, toàn bộ danh sách sẽ được trả về. Nếu T là GianHangTieuChuanDTO (GianHangCaoCapDTO), chỉ danh sách các gian hàng tiêu chuẩn (gian hàng cao cấp) được trả về.

Trong trường hợp cần đến database hay file lưu trữ, GianHangDAL sẽ có vai trò kết nối, truy vấn dữ liệu từ nguồn dữ liệu và trả về kết quả.

2.2. KhachHangDAL

Do cùng thuộc layer DAL nên chức năng của KhachHangDAL tương tự như GianHangDAL. Đó là khởi tạo dữ liệu ban đầu về khách hàng và lưu trữ toàn bộ danh sách

gian hàng trong quá trình chạy chương trình. Các thành phần của class KhachHangDAL là:

- *private List<KhachHangDTO> _dsKhachHang*

  Là thuộc tính sử dụng collection List để lưu trữ toàn bộ danh sách khách hàng đã thuê gian hàng từ trước tới nay.

- *public List<KhachHangDTO> DsKhachHang*

  Là property cho _dsKhachHang (tương tự như getter, setter). Khi DsKhachHang được gọi tới (gọi getter), nếu _dsKhachHang chưa có dữ liệu thì sẽ gọi hàm Load() để khởi tạo dữ liệu trước khi trả về _dsKhachHang.

- *private void _Load()*

  Khởi tạo dữ liệu về một số gian hàng cho _dsKhachHang, cũng là dữ liệu cho chương trình.

  Phương thức này nên được private để tránh cho việc bên ngoài có thể gọi tới khiến toàn bộ dữ liệu trở lại như ban đầu, tức là các dữ liệu được thêm, … bị mất đi, tương tự như phương thức *_Load()* của GianHangDAL.

Do chỉ có một loại khách hàng, nên để lấy danh sách khách hàng, các class khác chỉ cần gọi trực tiếp đến DsKhachHang. Vì vậy, KhachHangDAL không cần có thêm phương thức LayDanhSachKhachHang<T> tương tự như LayDanhSachGianHang<T> của GianHangDAL.

2.3. TaiKhoanDAL

Class này được thêm vào để khởi tạo và lưu trữ thông tin về các tài khoản người dùng (ngươi quản lý), dùng để đăng nhập lúc bắt đầu chương trình.

- *private List<TaiKhoanDTO> _dsTaiKhoan*

  Là thuộc tính kiểu collection List<TaiKhoanDTO> dùng để lưu trữ các tài khoản mà người quản lý dùng để đăng nhập.

- *public List<TaiKhoanDTO> DsTaiKhoan*

Là property cho thuộc tính *_dsTaiKhoan*, chỉ có getter trả về danh sách tài khoản, không có setter.

- *public TaiKhoanDAL()*

  Là phương thức khởi tạo cho class, trong thân phương thức sẽ khởi tạo các dữ liệu về tài khoản người quản lý. Do chương trình chưa có chức năng đăng ký hay thay đổi thông tin người dùng nên những dữ liệu này được fix cứng. Danh sách tài khoản:

  - Username: nh3, password: 123456.
  - Username: nang.lt194339, password: 123456.
  - Username: hung.pm, password: 123456.
  - Username: tien.dd, password: 123456.

3. BUS

3.1. Các thiết kế và thư viện đặc biệt được áp dụng tại BUS

a. Singleton Pattern

Singleton Pattern là một mẫu thiết kế phần mềm, hay hiểu đơn giản, là một cách thức thiết kế class thường xuyên được sử dụng trong mô hình 3 Layers. Pattern này giải quyết vấn đề trong tương tác giữa GUI với BUS, và giữa BUS với DAL.

Thông thường, khi cần sử dụng phương thức của class nào, ta cần có trong tay một object có kiểu class đó và gọi đến phương thức thông qua object này. Layer GUI là layer tương tác trực tiếp với người dùng. Mỗi lần cần xử lý một yêu cầu từ người dùng, thì form mà người dùng đang tương tác (tương ứng với 1 class trong GUI) sẽ cần gọi đến BUS để xử lý yêu cầu đó. Để làm được điều này, form cần:

- Hoặc tạo ra 1 object BUS tương ứng với yêu cầu cần xử lý.
- Hoặc là có thuộc tính kiểu BUS tương ứng.

Với phương án thứ hai, số lượng class thuộc Layer BUS có thể lớn và 1 form khi đó phải khai báo hết các thuộc tính tương ứng với từng class trong Layer BUS mà nó cần. Tuy

nhiên, có những class BUS ít được sử dụng hơn class khác; số lượng form trong GUI cũng không hề nhỏ, nếu mỗi form (tức mỗi class trong GUI) đều có riêng các thuộc tính giống nhau thì sẽ gây lãng phí bộ nhớ.

Với phương án thứ nhất, vấn đề nghiêm trọng hơn. Nếu mỗi lần xử lý lại tạo ra 1 object BUS mới thì kéo theo trong BUS sẽ lại tạo ra 1 object DAL mới tương ứng. Khi đó, dữ liệu được dùng để xử lý là dữ liệu từ ban đầu lúc bắt đầu chạy chương trình, không phải dữ liệu đang được lưu trước đó. Và khi thực hiện xong yêu cầu, thì kết quả cũng sẽ không được lưu cho lần xử lý kế tiếp, vì lần kế tiếp đó sẽ lại tạo ra object BUS và DAL mới, dùng dữ liệu ban đầu.

Có hai giải pháp thường được sử dụng để giải quyết vấn đề này là dùng *static* hoặc *Singleton Pattern*, trong đó giải pháp *Singleton Pattern* thường được ưa chuộng hơn. Với *Singleton Pattern*, chương trình sẽ đảm bảo là với mỗi class thuộc layer BUS, sẽ chỉ có duy nhất 1 object kiểu class đó được tạo ra trong suốt quá trình chạy, dù được gọi từ đâu trong chương trình đi chăng nữa; và object đó sẽ chỉ được tạo ra khi cần dùng đến lần đầu tiên. Cách thông dụng để thực hiện *Singleton Pattern* như sau:

- Phương thức khởi tạo được chỉ định truy cập là *private*. Khi đó, từ bên ngoài sẽ không thể chủ động tạo ra 1 object thuộc kiểu class đó được nữa.
- Khai báo 1 thuộc tính *private static* có kiểu là chính class đó (thường đặt tên là *_instance*).
- Khai báo phương thức static getter cho thuộc tính trên (thường đặt tên là *Instance*). Trong *Instance*, class sẽ kiểm tra xem *_instance* đã được khởi tạo hay chưa, nếu chưa thì khởi tạo cho thuộc tính đó. Cuối cùng là trả về *_instance*.

Bằng cách này, sẽ chỉ có duy nhất một object được tạo ra trong toàn bộ chương trình và được tham chiếu bởi *_instance*. Bên ngoài sẽ gọi đến object này thông qua getter của nó. Ví dụ, để gọi đến phương thức *Them()* của class *QuanLy*, và class này thực hiện *Singleton Pattern* với các quy ước đặt tên như trên, ta viết: *QuanLy.Instance.Them();*

Bên cạnh *Singleton Pattern,* thì *static* cũng là giải pháp thường được sử dụng. Tuy nhiên, so với *static*, thì class thực hiện *Singleton Pattern* mang tính "hướng đối tượng" nhiều hơn, class đó vẫn có thể có các thuộc tính và phương thức non-static, có thể sử dụng kế thừa, đa hình, thực thi giao diện, … Dù vậy, với quy mô bài toán này, thì *static* vẫn là một giải pháp tốt.

Các class trong Layer BUS (ngoại trừ **HelperBUS**) đều thực hiện *Singleton Pattern*.

b. LINQ

LINQ (Language Integrated Query), nằm trong namespace System.Linq, là một bộ thư viện tích hợp sẵn trong C# gồm các phương thức có thể sử dụng để truy vấn dữ liệu từ nhiều nguồn dữ liệu (tập hợp object, XML, SQL Server, …). Bản chất của LINQ tương đối phức tạp nên không được đề cập đến ở đây. Tuy nhiên, cách sử dụng của nó lại rất đơn giản với các phương thức như Where, Select, … gần giống với ngôn ngữ SQL.

Trong project này chỉ sử dụng LINQ để truy vấn (hay *lọc*) dữ liệu từ List<T> theo một hoặc một số điều kiện nào đó, ví dụ như tìm kiếm gian hàng có mã gian hàng người dùng nhập vào. Các trường hợp sử dụng LINQ trong project đều có thể được thay thế bằng vòng lặp *for* hay *foreach*, tuy nhiên cú pháp của LINQ ngắn gọn, đơn giản hơn, giúp cho code dễ đọc, dễ bảo trì hơn. Dù vậy, trong các phương thức generic, việc sử dụng LINQ có phần hơi phức tạp nên được thay thế bằng vòng lặp *foreach* thông thường.

3.2. GianHangBUS

Chức năng:

- Nhận dữ liệu và các yêu cầu liên quan đến gian hàng, xử lý nghiệp vụ trung gian và gọi đến GianHangDAL để thao tác với dữ liệu.
- Cung cấp các chức năng về gian hàng như: Thêm, Sửa, Xóa, Tìm kiếm, …

Các thành phần:

- *private GianHangDAL _context*

Là thuộc tính tham chiếu đến 1 object kiểu GianHangDAL, nắm giữ thông tin về toàn bộ gian hàng. Thuộc tính này được sử dụng khi class thực hiện các chức năng, thao tác về dữ liệu.

- *private GianHangBUS()*

  Phương thức khởi tạo của class, được chỉ định truy cập là private để thực hiện *Singleton Pattern*. Do đó, bên ngoài sẽ không thể trực tiếp khởi tạo ra 1 object kiểu GianHangBUS. Nếu muốn sử dụng GianHangBUS, cần gọi trực tiếp đến tên class.

- *private static GianHangBUS _instance*

  Là thuộc tính tĩnh tham chiếu đến 1 object thuộc chính kiểu GianHangBUS, được sử dụng để thực hiện *Singleton Pattern*.

- *public static GianHangBUS Instance*

  Là property tĩnh cho _instance, chỉ có getter, không có setter và cũng là 1 phần của *Singleton Pattern*. Khi được gọi đến từ bên ngoài thông qua tên class (tức GianHangBUS.Instance), nó sẽ kiểm tra xem _instance đã được khởi tạo hay chưa. Nếu chưa, _instance sẽ được khởi tạo để tham chiếu đến 1 object kiểu GianHangBUS. Cuối cùng, _instance sẽ được trả về.

  Bằng cơ chế như trên, kết hợp với việc constructor của GianHangBUS được chỉ định truy cập là private, chương trình sẽ đảm bảo được là chỉ có duy nhất một object kiểu GianHangBUS được tạo ra trong suốt quá trình chạy. Object đó được tham chiếu bởi _instance. Bên ngoài sẽ không cần phải trực tiếp tạo ra 1 object kiểu GianHangBUS nữa mà sẽ gọi thẳng đến class này luôn.

Các phương thức của class GianHangBUS đã được mô tả chi tiết trong tài liệu API BUS. Tài liệu này được cung cấp cho thành viên đảm nhận layer GUI trong nhóm sử dụng.

3.3. KhachHangBUS

Chức năng:

- Nhận dữ liệu và các yêu cầu liên quan đến khách hàng, xử lý nghiệp vụ trung gian và gọi đến KhachHangDAL để thao tác với dữ liệu.
- Cung cấp các chức năng về khách hàng như: Thêm, Tìm kiếm, …

Các thành phần:

- *private KhachHangDAL _context*

  Là thuộc tính tham chiếu đến 1 object kiểu KhachHangDAL, nắm giữ thông tin về toàn bộ các khách hàng đã thuê. Thuộc tính này được sử dụng khi class thực hiện các chức năng, thao tác về dữ liệu.

- *private KhachHangBUS()*

  Phương thức khởi tạo của class, được chỉ định truy cập là private để thực hiện *Singleton Pattern*, tương tự như class GianHangBUS.

- *private static KhachHangBUS _instance*

  Là thuộc tính tĩnh tham chiếu đến 1 object thuộc chính kiểu GianHangBUS, được sử dụng để thực hiện *Singleton Pattern*.

- *private static Instance*

  Cách thức hoạt động tương tự như Instance của GianHangBUS

Các phương thức của class KhachHangBUS cũng đã được mô tả chi tiết trong tài liệu API BUS.

3.4. KhuTrungBayBUS

Chức năng:

- Nhận dữ liệu và các yêu cầu liên quan đến cả khách hàng và gian hàng, xử lý nghiệp vụ trung gian.
- Cung cấp các chức năng như: Thuê gian hàng, tính doanh thu, …

Các thành phần tương tự như các class BUS khác:

- *private KhuTrungBayBUS()*

  Phương thức khởi tạo của class, được chỉ định truy cập là private để thực hiện *Singleton Pattern*.

- *private static KhuTrungBayBUS _instance*

  Vai trò tương tự như _instance trong các class BUS khác.

- *private static KhuTrungBayBUS Instance*

  Vai trò tương tự như Instance trong các class BUS khác.

Các phương thức của class KhuTrungBayBUS đã được mô tả chi tiết trong tài liệu API BUS.

3.5. HelperBUS

Là static class (class chỉ chứa các thành phần static) với 2 phương thức static là:

- *public static string GenerateMaGianHang<T>(string viTriGianHang)*

  Sinh ra mã gian hàng từ vị trí và loại gian hàng theo đúng quy cách.

- *public static string GenerateMaKhachHang(string maGianHang, DateTime thoiDiemBatDau)*

  Sinh ra mã khách hàng từ mã gian hàng được thuê và thời điểm bắt đầu thuê.

Class này được GUI gọi tới mỗi khi tạo ra 1 object GianHangDTO (hoặc class con) hoặc KhachHangDTO để truyền cho BUS

3.6. TaiKhoanBUS

Class này có chức năng xử lý vấn đề về tài khoản, đăng nhập.

Các thành phần tương tự như các class BUS khác:

- *private TaiKhoanBUS()*

  Phương thức khởi tạo của class, được chỉ định truy cập là private để thực hiển Singleton Pattern.

- *private static TaiKhoanBUS _instance*

  Vai trò tương tự như _instance trong các class BUS khác.

- *private static TaiKhoanBUS Instance*

  Vai trò tương tự như Instance trong các class BUS khác.

- *public bool Check(string username, string password)*

  Phương thức kiểm tra xem username và password người dùng nhập vào có khớp với username và password của tài khoản nào trong hệ thống hay không. Nếu có, trả về *true*; ngược lại, trả về *false*.

Hiện tại, do chưa có các chức năng như: đăng xuất, quên mật khẩu, đăng ký, … nên **TaiKhoanBUS** chỉ có 1 phương thức là *Check(string, string)* để kiểm tra username và password người dùng nhập vào khi đăng nhập có hợp lệ hay không. Class này cũng chỉ được gọi ở phần đầu chạy chương trình, khi người dùng đăng nhập. Chính vì thế mà TaiKhoanDAL cũng chỉ được gọi đến ở phần đầu chương trình. Vì lý do này nên TaiKhoanBUS sẽ không chứa thuộc tính *_context* như các class BUS khác do sẽ không thường xuyên được sử dụng. Object kiểu TaiKhoanDAL sẽ chỉ được tạo ra trong phương thức *Check(string, string)* khi cần dữ liệu để kiểm tra đăng nhập mà thôi.

4. GUI

Sử dụng Winforms để xây dựng giao diện. Winforms hay còn gọi là Windows forms là một .NET Framework dùng để xây dựng ứng dụng. Winforms được xây dựng dựa trên thư viện Windows API. Tại sự kiện Microsoft Connect vào ngày 4 tháng 12 năm 2018, Microsoft đã công bố phát hành Windows Forms dưới dạng một dự án mã nguồn mở trên GitHub.

Windows Forms hoàn toàn đơn giản hóa việc lập trình GUI (giao diện đồ họa), hỗ trợ thiết kế giao diện trực quan, đồng thời nhận được sự hỗ trợ rất tốt từ các hãng thứ 3 như Devexpress, Telerik, ... và cộng đồng.

Một cấu trúc Winforms cơ bản bao gồm:

- Một Form là khung hiển thị
- Các control được đặt trong form và được lập trình để đáp ứng sự kiện

Với nhiều ưu điểm nổi trội như: tốc độ xử lý nhanh chóng, bảo mật, … Có thể chạy trên các phiên bản Windows khác nhau.

IV. Đánh giá kết quả

1. Ưu điểm

1.1. Về phương diện người sử dụng

- Giao diện đẹp
- Giao diện được thiết kế hợp lý dễ sử dụng
- Thống kê sử dụng đồ thị quan sát.
- File setup dễ dàng không phải cài thêm cái gói hỗ trợ
- Có thêm tài khoản người quản lý đảm bảo tính bảo mật
- Có các thông báo ở góc màn hình giúp nắm bắt tình trạng của hành động
- Chương trình chạy ổn định ít xảy ra lỗi

1.2. Về phương diện người lập trình

- Hoàn thành các yêu cầu của đề bài.
- Áp dụng các kiến thức về lập trình hướng đối tượng: kế thừa, đa hình, generic, …
- Sử dụng mô hình 3 Layers nên cấu trúc chương trình rõ ràng, phân chia công việc đơn giản.
- Dễ mở rộng thêm các chức năng khác hoặc thêm cơ sở dữ liệu.

2. Hạn chế

2.1. Về phương diện người sử dụng

- Chưa có cơ sở dữ liệu để lưu trữ dữ liệu khi tắt chương trình (Có thể bổ sung trong thời gian tới)
- Chỉ có thể cài đặt trên hệ điều hành Windows
- Chưa có đăng ký thêm tài khoản khách hàng

2.2. Về phương diện người lập trình

- Mô hình có phần hơi cồng kềnh so với yêu cầu của bài toán
- Có thể lựa chọn giải pháp *static* thay cho *Singleton Pattern* với quy mô hiện tại của bài toán.

V. Lời kết

Trên đây là báo cáo bài tập lớn môn Lập trình hướng đối tượng, lớp 124175 của nhóm số 28, bài tập lớn số 19. Do là lần đầu làm báo cáo bài tập lớn, kiến thức về lập trình hướng đối tượng được tiếp xúc chưa lâu, một số kiến thức như Mô hình 3 Layers, Singleton

Pattern, … được tham khảo chủ yếu qua các bài viết trên Internet nên kiến thức chưa sâu và không thể tránh khỏi sai sót. Vì vậy, chúng em rất mong muốn được giảng viên góp ý và chỉnh sửa để có được những sản phẩm chất lượng hơn trong tương lai.

Cuối cùng, chúng em xin chân thành cảm ơn giảng viên Đỗ Thị Ngọc Diệp đã nhiệt tình giúp đỡ chúng em trong quá trình làm bài tập lớn. Các bài giảng của cô cũng vô cùng chi tiết và dễ hiểu. Chúc cô sẽ tiếp tục thành công hơn nữa trong sự nghiệp giảng dạy của mình.