TRƯỜNG ĐẠI HỌC KHOA HỌC VÀ CÔNG NGHỆ HÀ NỘI

Viện Công nghệ Thông tin và Truyền thông

TÀI LIỆU THIẾT KẾ PHẦN MỀM

Software Design Document

Mục lục

[**1. Giới thiệu** 5](#_Toc61826602)

[1.1 Mục tiêu 5](#_Toc61826603)

[1.2 Phạm vi 5](#_Toc61826604)

[1.3 Bảng chú giải thuật ngữ 5](#_Toc61826605)

[1.4 Tài liệu tham khảo 5](#_Toc61826606)

[**2. Mô tả chung** 6](#_Toc61826607)

[2.1 Tổng quan chung 6](#_Toc61826608)

[2.2 Giả định / Ràng buộc / Rủi ro 6](#_Toc61826609)

[2.2.1 Giả định 6](#_Toc61826610)

[2.2.2 Ràng buộc 6](#_Toc61826611)

[2.2.3 Rủi ro 7](#_Toc61826612)

[**3. Kiến trúc hệ thống và thiết kế kiến trúc** 7](#_Toc61826613)

[3.1 Mô hình kiến trúc 7](#_Toc61826614)

[3.2 Biểu đồ tương tác 8](#_Toc61826615)

[3.2.1 Biểu đồ trình tự 8](#_Toc61826616)

[3.2.2 Biểu đồ giao tiếp 10](#_Toc61826617)

[3.3 Biểu đồ lớp phân tích 13](#_Toc61826618)

[**4. Thiết kế chi tiết** 15](#_Toc61826619)

[4.1 Thiết kế giao diện người dùng 15](#_Toc61826620)

[4.1.1 Chuẩn hóa cấu hình màn hình 15](#_Toc61826621)

[4.1.2 Sơ đồ chuyển đổi màn hình 15](#_Toc61826622)

[4.1.3 Đặc tả màn hình 16](#_Toc61826623)

[4.2 Mô hình dữ liệu 22](#_Toc61826624)

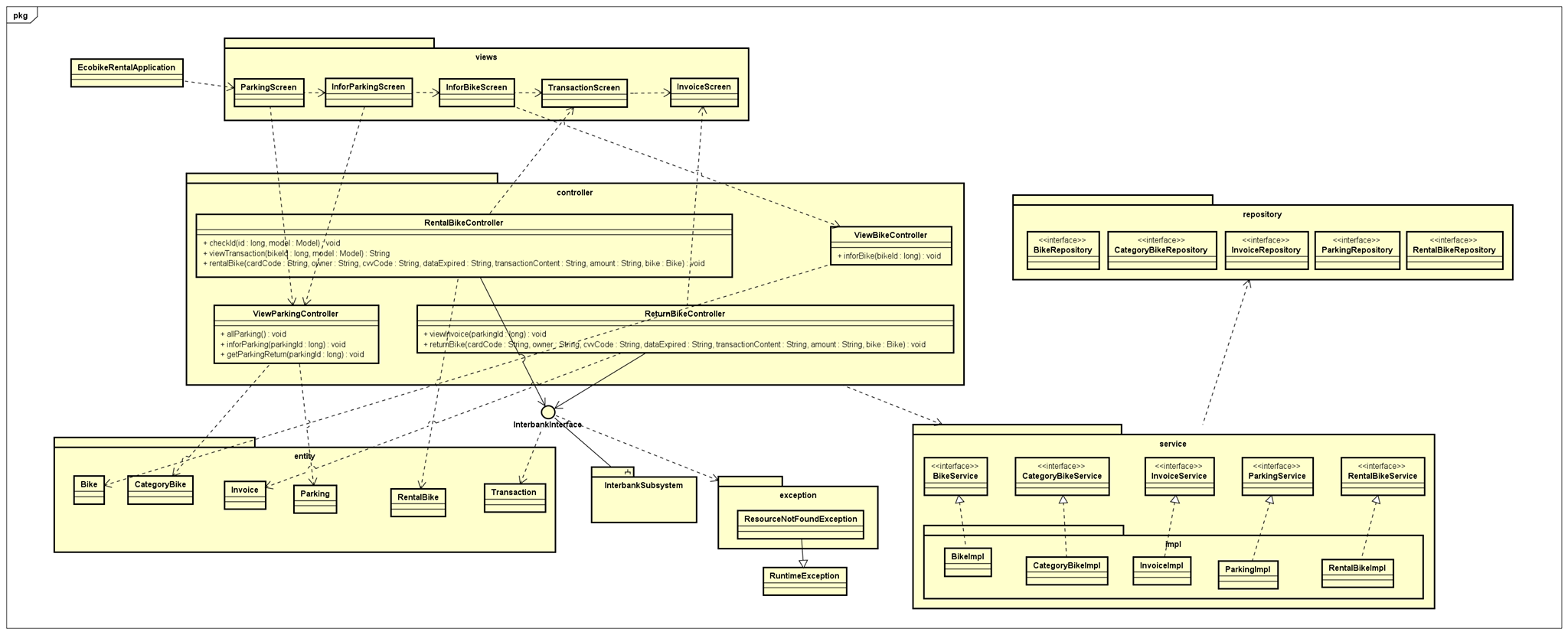
[4.2.1 Mô hình hóa dữ liệu khái niệm 22](#_Toc61826625)

[4.2.2 Thiết kế cơ sở dữ liệu 22](#_Toc61826626)

[4.3 Non-Database Management System Files 23](#_Toc61826627)

[4.4 Thiết kế lớp 23](#_Toc61826628)

[4.4.1 Biểu đồ lớp tổng quát 23](#_Toc61826629)

[ 24](#_Toc61826630)

[4.4.2 Biểu đồ lớp 25](#_Toc61826631)

[**5. Cân nhắc kiến trúc** 29](#_Toc61826632)

[5.1 Mục tiêu và nguyên tắc 29](#_Toc61826633)

[5.1.1 Mục tiêu phần mềm 29](#_Toc61826634)

[5.1.2 Nguyên tắc 29](#_Toc61826635)

[5.2 Chiến lược kiến trúc 29](#_Toc61826636)

[5.3 Coupling and Cohesion 29](#_Toc61826637)

[5.3.1 Coupling 29](#_Toc61826638)

[5.3.2 Conhesion 30](#_Toc61826639)

[5.4 Nguyên tắc thiết kế 30](#_Toc61826640)

# **1. Giới thiệu**

## 1.1 Mục tiêu

Tài liệu này đưa ra mô tả chi tiết cho các chức năng người dùng có thể sử dụng được tại thời gian chạy. Tài liệu mô tả mục đích và các tính năng của hệ thống, các giao diện, ràng buộc của hệ thống cần thực hiện để phản ứng tới các kích thích bên ngoài.

## 1.2 Phạm vi

Trong thực tế, bất kỳ phần mềm nào cũng cần có các tính năng quản lý người dùng, nhóm người dùng, và cần phân quyền sử dụng các chức năng trong hệ thống một cách linh động.

Mục đích của phần mềm nhằm tạo ra phân hệ quản lý người dùng (user), vai trò của người dùng (role) và các chức năng (function) mà người dùng / vai trò người dùng có thể sử dụng tại thời điểm chạy. Người dùng có thể đăng ký để tạo ra tài khoản cho mình, sau đó có thể đăng nhập để sử dụng các chức năng của hệ thống. Người dùng có thể đăng nhập sử dụng tài khoản của hệ thống, hoặc đăng nhập sử dụng tài khoản Facebook. Bất kỳ người dùng nào cũng được cập nhật thông tin cá nhân của mình. Khi người dùng quên mật khẩu, có thể yêu cầu hệ thống cho phép mình thiết lập lại mật khẩu qua liên kết kèm token gửi qua email đã đăng ký.

Trong phạm vi môn học ta không xét các chức năng xác thực người dùng như đăng ký, đăng... nhập mà chỉ quan tâm các chức năng liên quan đến trả xe, thuê xe

## 1.3 Bảng chú giải thuật ngữ

- API (**Application Programming Interface**): là các phương thức, giao thức kết nối với các thư viện và ứng dụng khác. API cung cấp khả năng cung cấp khả năng truy xuất đến một tập các hàm hay dùng.

## 1.4 Tài liệu tham khảo

- “EcobikeRental”-Nguyễn Thị Thu Trang.

- Tài liệu đặc tả yêu cầu phần mềm

# **2. Mô tả chung**

## 2.1 Tổng quan chung

Khi hệ thống khởi chạy, một danh sách các bãi xe hiện lên màn hình thay vì bản đồ. Khách hàng vẫn có thể xem các thông tin về bãi xe, loại xe.

Khi thuê xe, sau khi chọn bãi xe để thuê, khách hàng nhập mã vạch tương ứng của xe muốn thuê và hệ thống sẽ gọi đến một API để chuyển mã vạch thành mã xe trong hệ thống.

Tương tự như vậy, khi trả xe, sau khi chọn bãi xe để trả xe, khách hàng cũng nhập mã vạch tương ứng của xe cần trả thay vì đóng khóa xe.

Ngoài ra, khi hiển thị thông tin về xe, hệ thống cũng cần hiển thị thông tin về lượng pin còn lại của xe. Để đơn giản, chúng ta không cần quan tâm việc thay đổi giá trị của lượng pin này trong quá trình xe được mượn hay được trả tại bãi.

Biểu đồ use case tổng quan này là use case phức hợp của một nhóm các use case. Chi tiết về các use case phức này được đưa ra trong các biểu đồ phân rã ở phần sau.

## 2.2 Giả định / Ràng buộc / Rủi ro

### 2.2.1 Giả định

Trong ứng dụng EcoBikeRental sẽ chỉ quan tâm đến các chức năng liên quan tới thuê xe/ trả xe. Khi thuê xe khách hàng nhập mã vạch tương ứng của xe muốn thuê và hệ thống sẽ gọi đến một API để chuyển mã vạch thành mã xe mã xe trong hệ thống. Để trả xe thì hệ thống có tính năng cho phép khác hàng chọn bãi để trả xe. Khách hàng sẽ sử dụng thẻ tín dụng làm phương thức thanh toán. Để thực hiện các thao tác với thẻ tín dụng, hệ thống sẽ gọi đến một số API được cung cấp sẵn: API trừ tiền( sử dụng để cọc tiền khi thuê xe và thanh toán số tiền của một lần thuê), API cộng tiền( sử dụng để thực hiện hoàn trả lại tiền cọc cho khách hàng), API reset, API xem số dư ( sử dụng để xem số dư trong tài khoản)

### 2.2.2 Ràng buộc

* Yêu cầu về giao diện dễ sử dụng, thân thiện với người dùng. Làm nổi bật các tính năng thuê xe, trả xe, xem bãi, xem thông tin xe, …
* Các chức năng thiết kế sao cho dễ thao tác. Cần có thông báo hướng dẫn lỗi sai để người dùng biết là lỗi gì và biết các sửa lỗi
* Thiết bị cần có internet, bật định vị GPS

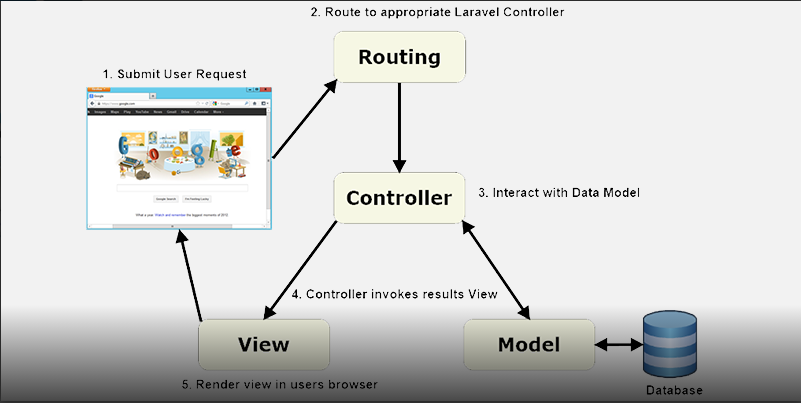
### 2.2.3 Rủi ro

* Các thông báo hướng dẫn lỗi sai không được hiển thị
* Giao diện không dễ sử dụng đối với người dùng

# **3. Kiến trúc hệ thống và thiết kế kiến trúc**

## 3.1 Mô hình kiến trúc

Website kiến trúc theo mô hình MVC model – view – controller



* Model :

- Quản lý dữ liệu của ứng dụng

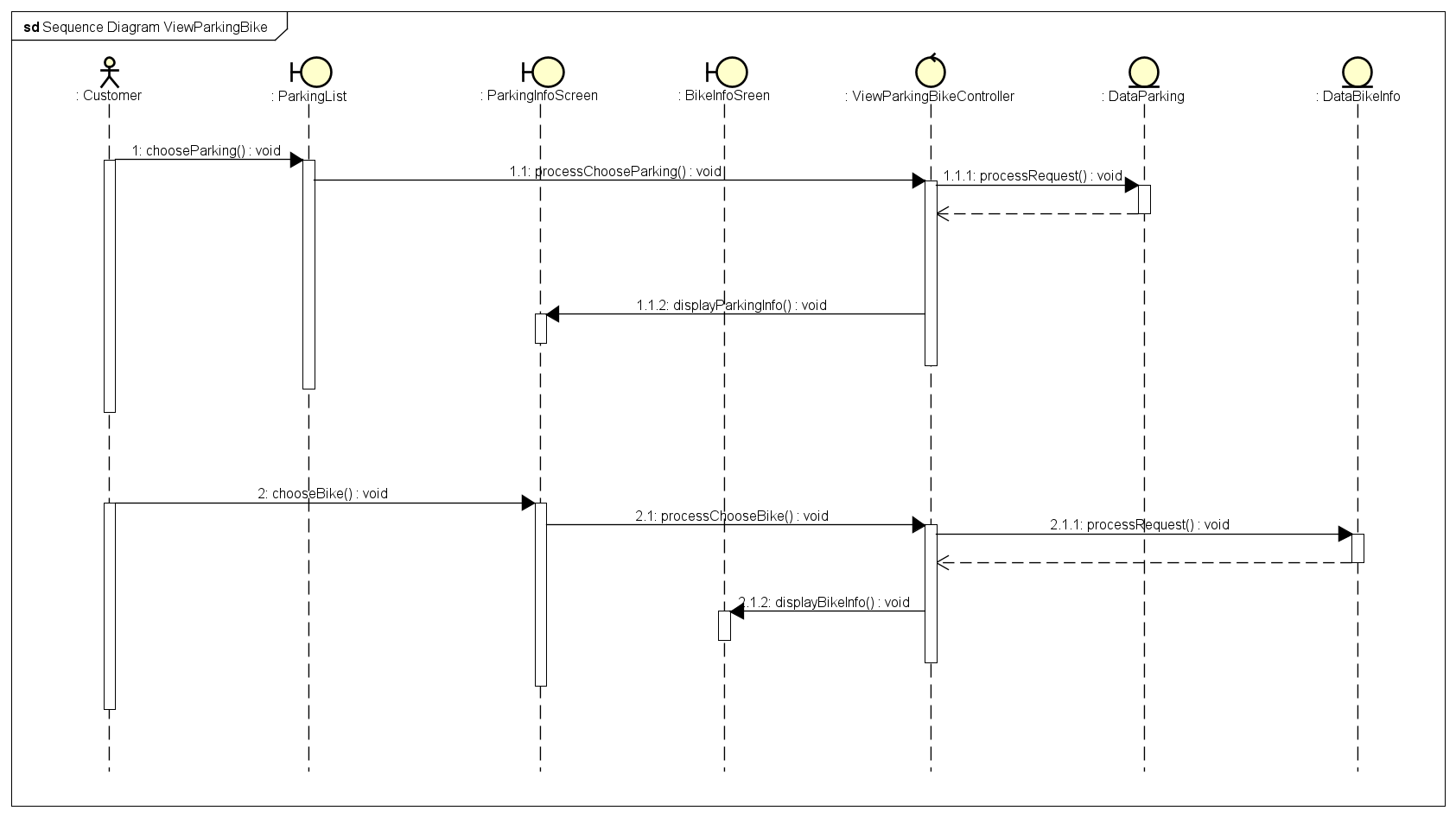
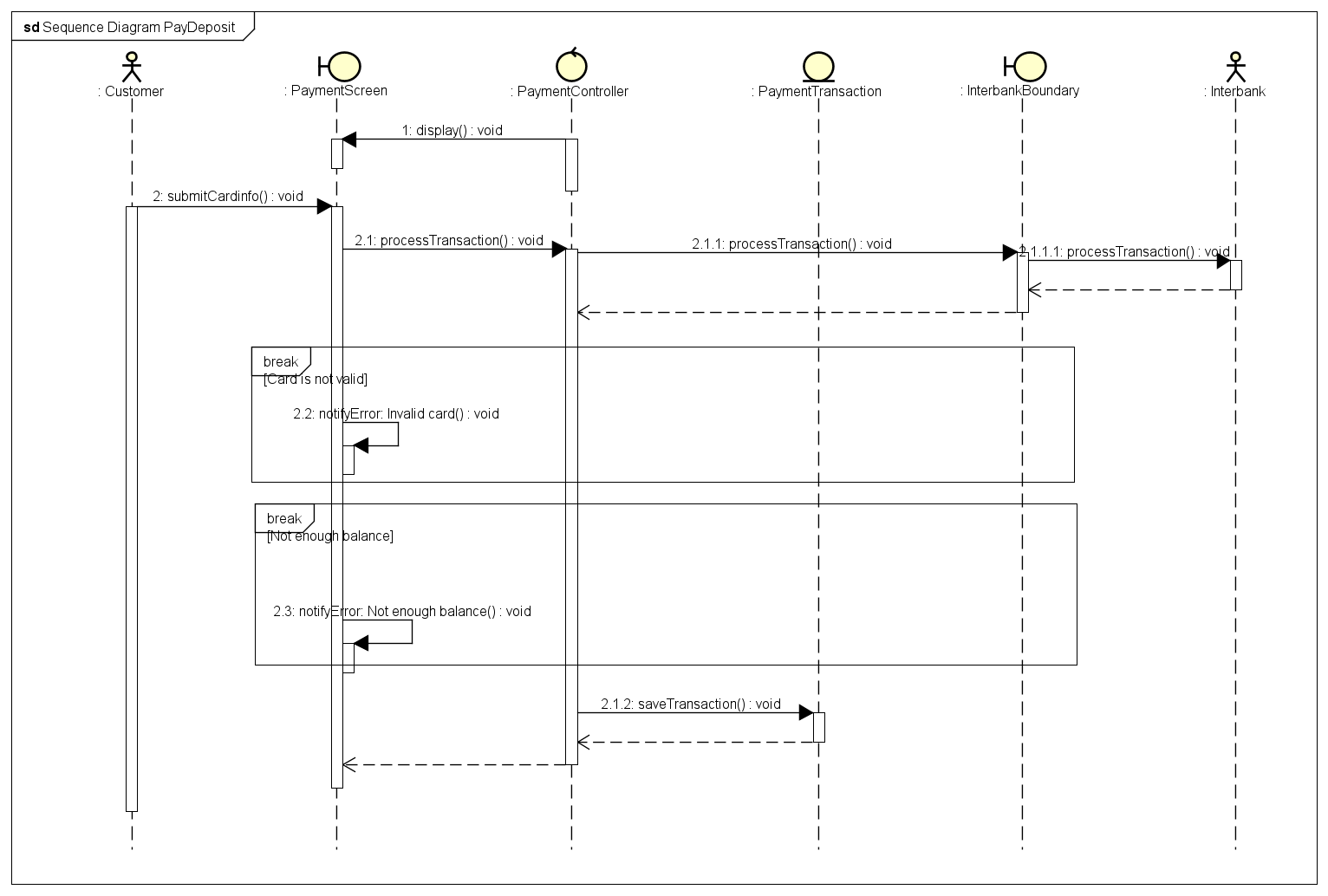
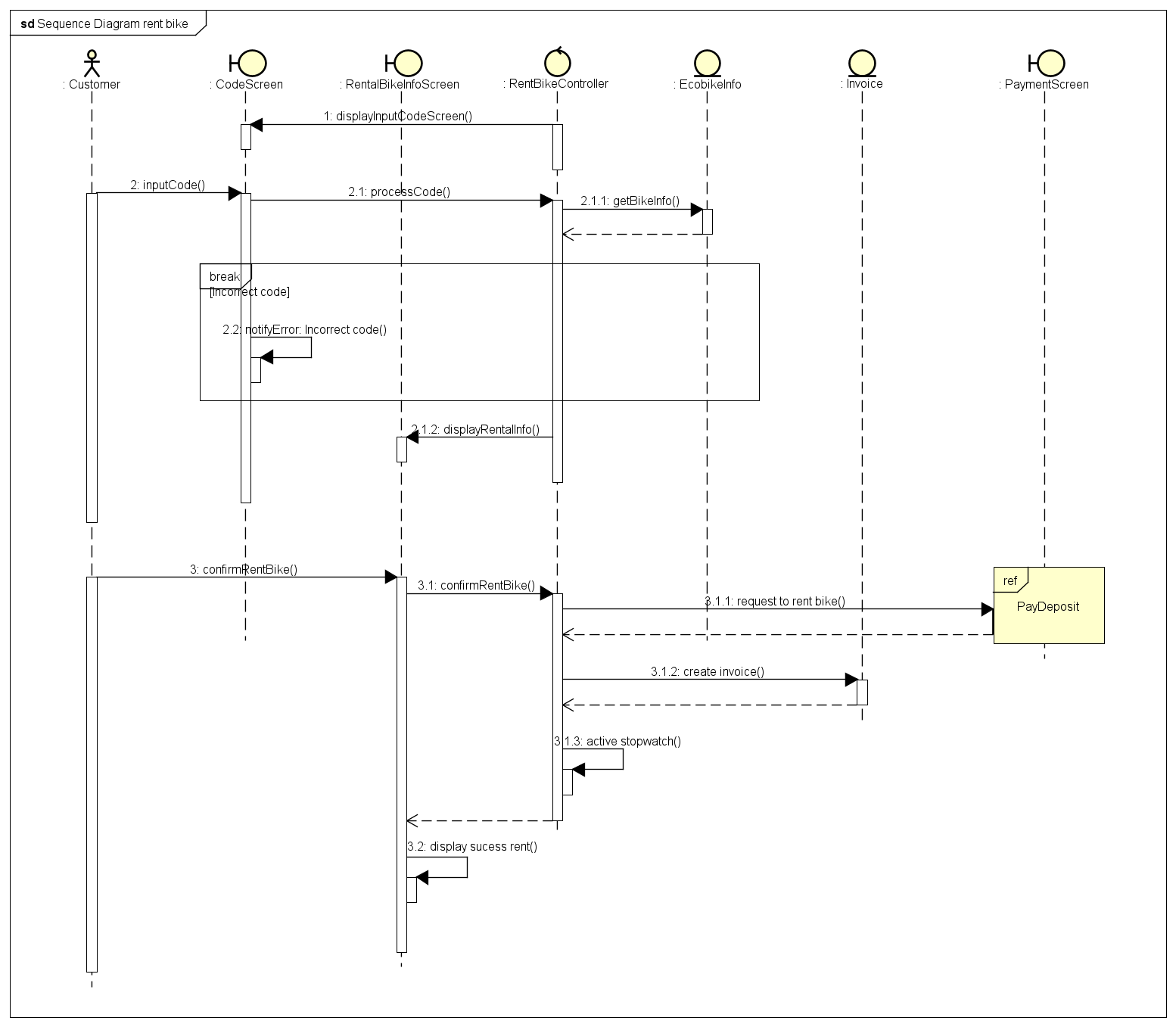
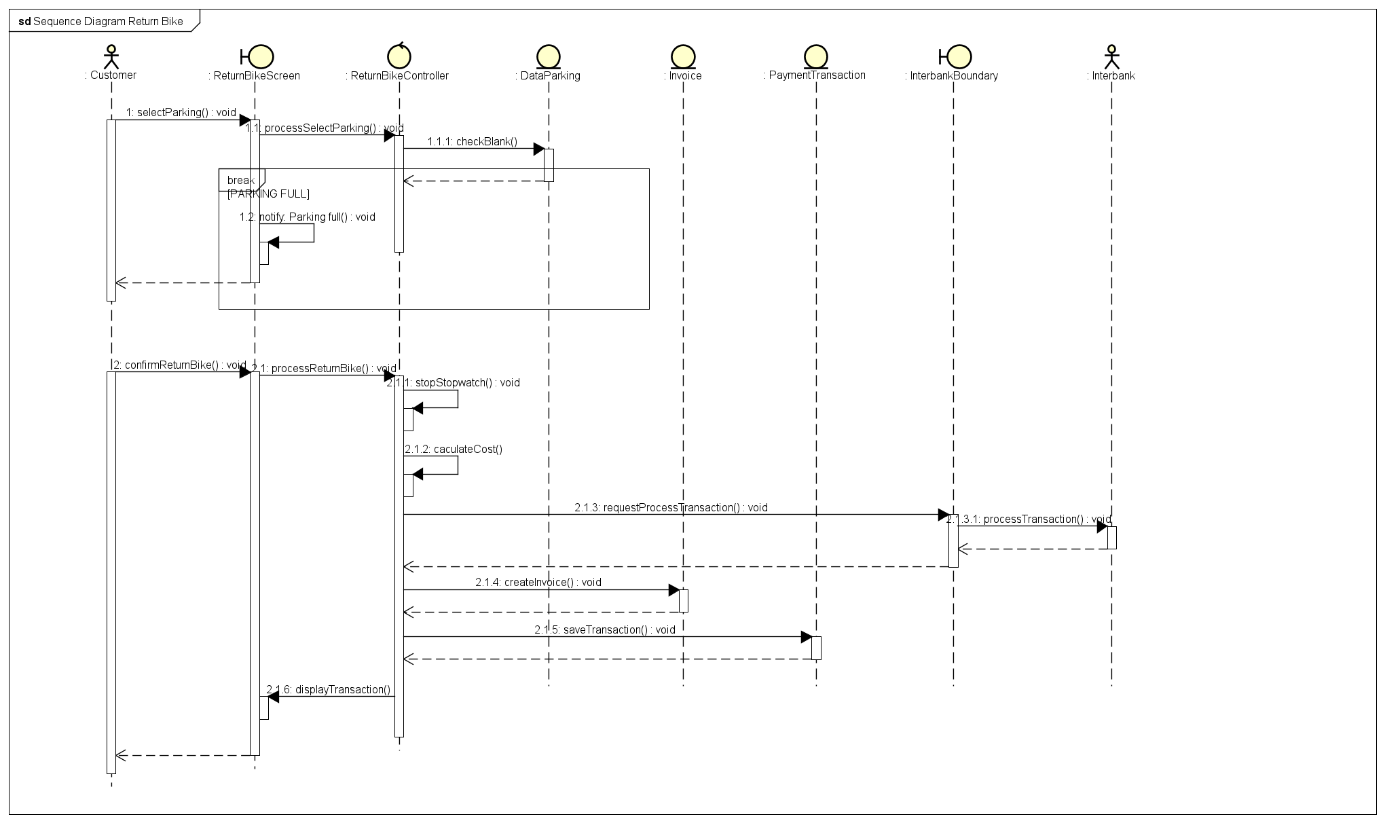
- Biểu diễn, vận chuyển thông tin

- Chứa các đối tượng mô tả dữ liệu

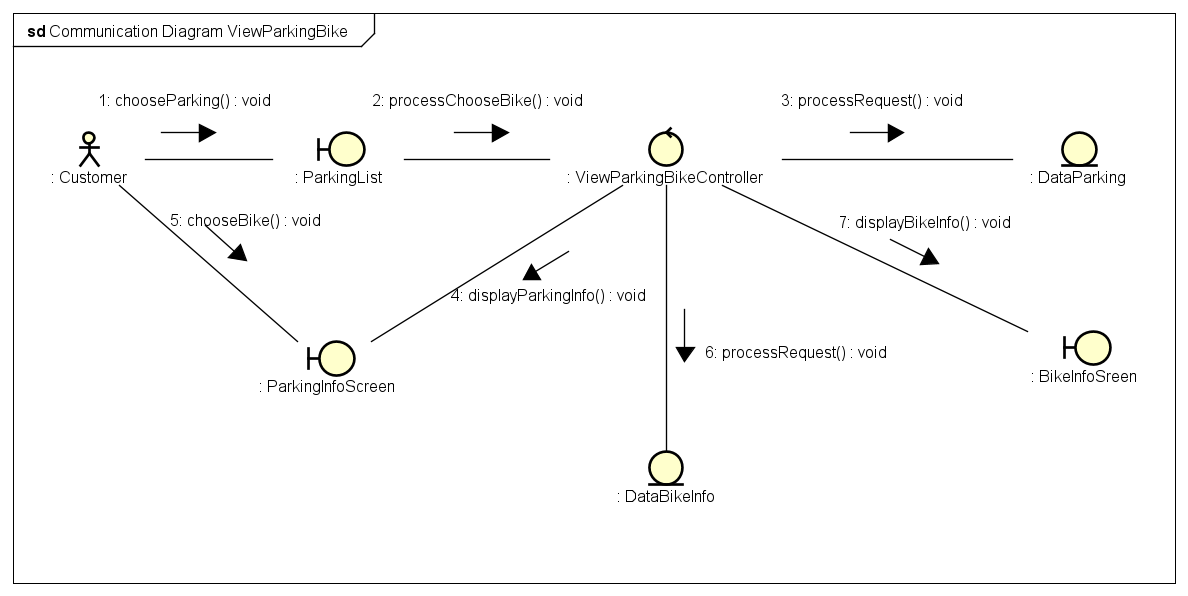
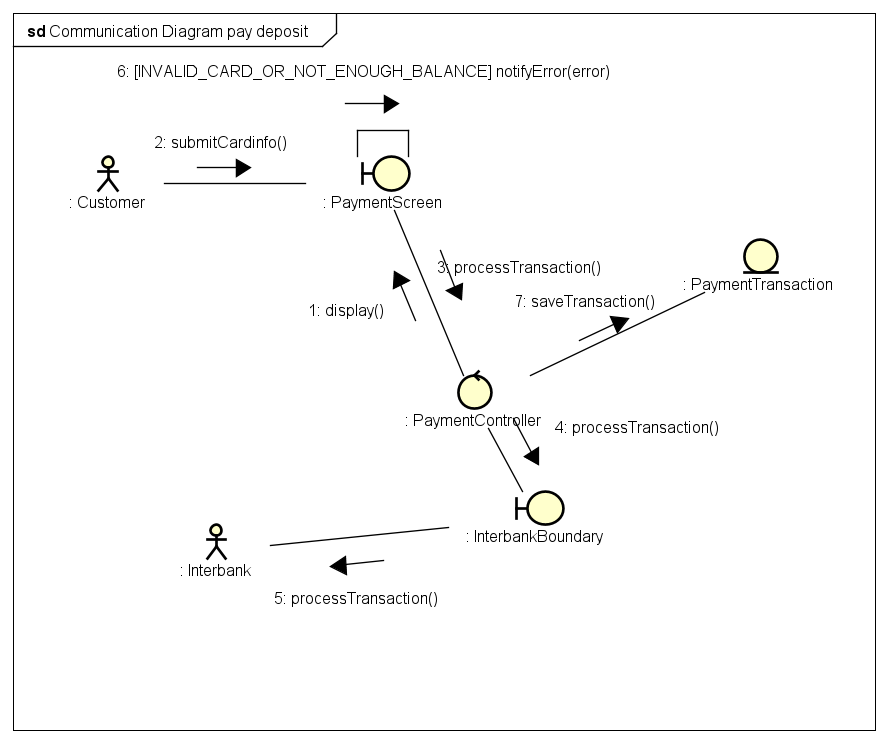
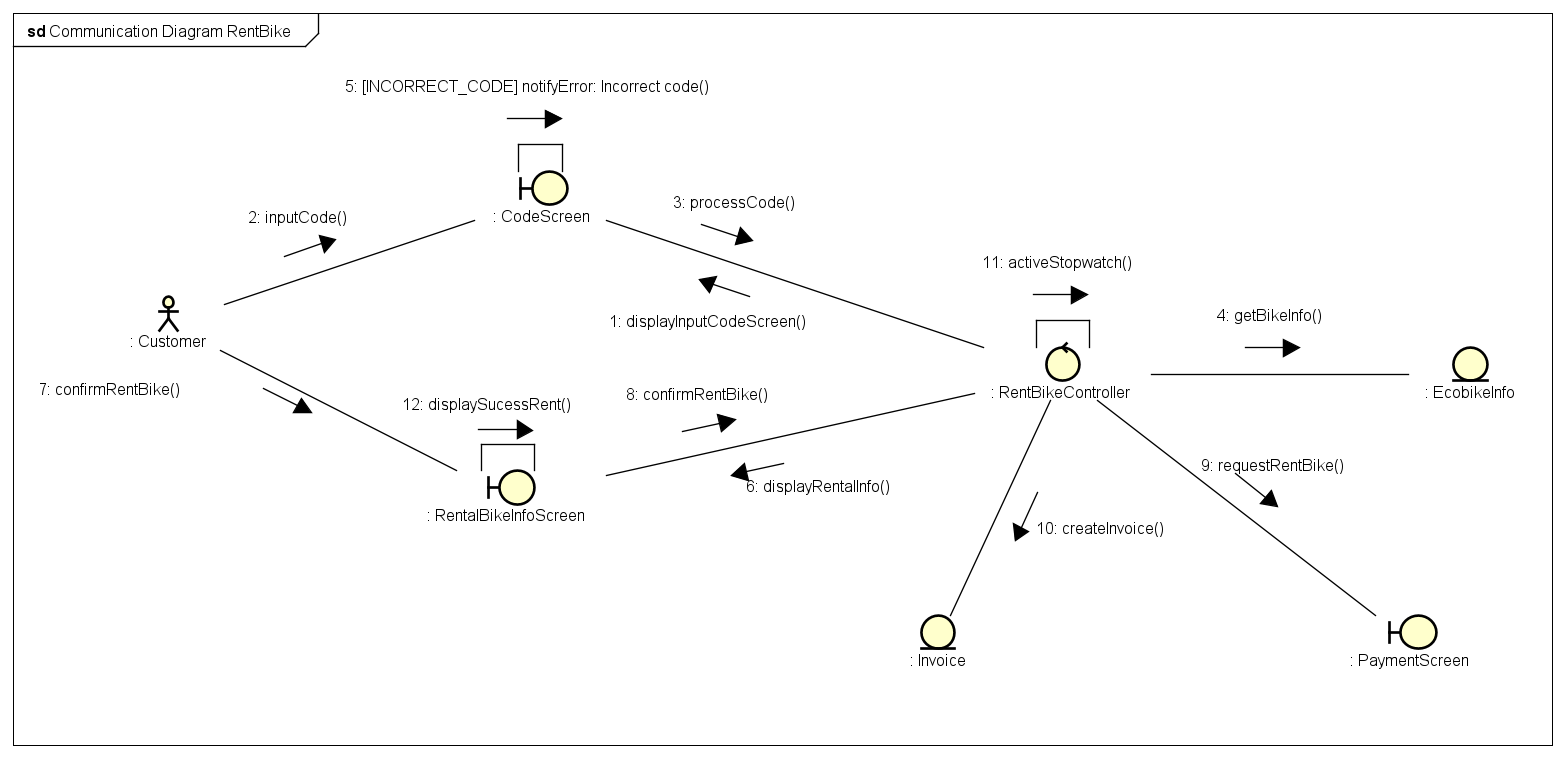
* View:
* Tương tác với user
* Lấy request từ người dùng và chuyển cho Controller xử lý
* Show kết quả của Controller
* View là hệ thống các frame, cửa sổ của ứng dụng; các trang giao diện web: html, jsp, bảng, biểu mẫu…
* Controller
* Định nghĩa các hành vi, hoạt động, xử lý của hệ thống.
* Đối chiếu hành động của người sử dụng từ View. Đồng thời tương tác Model để gọi View và hiển thị thông tin tương ứng cho người dùng.

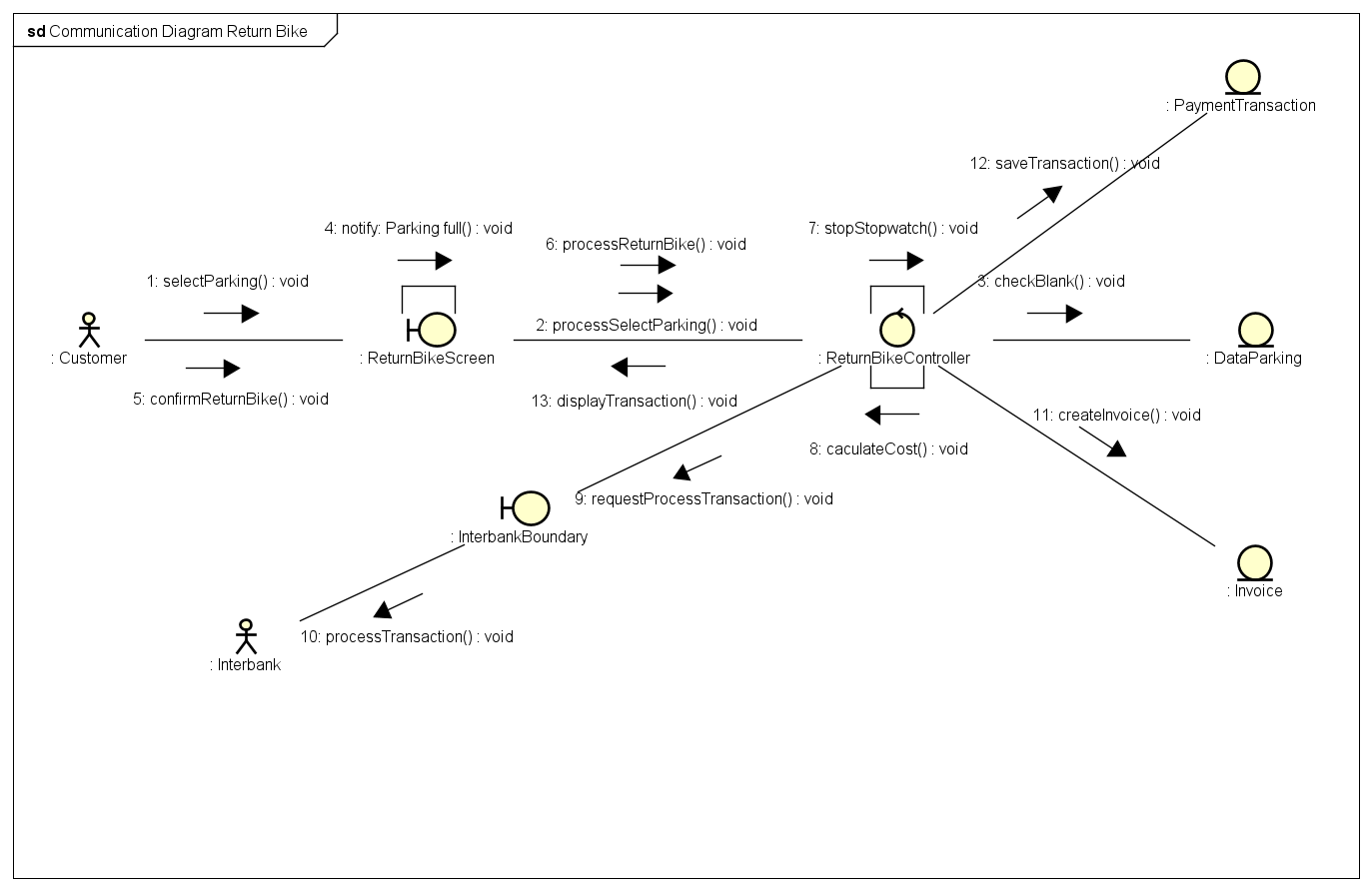
## 3.2 Biểu đồ tương tác

### 3.2.1 Biểu đồ trình tự

* Biểu đồ trình tự xem thông tin xe bãi xe ( choose: chọn, process: quá trình )
* Biểu đồ trình tự thanh toán ( bounnary: ranh giới liên, transaction: giao dịch )
* Biểu đồ trình tự thuê xe
* Biểu đồ trình tự trả xe

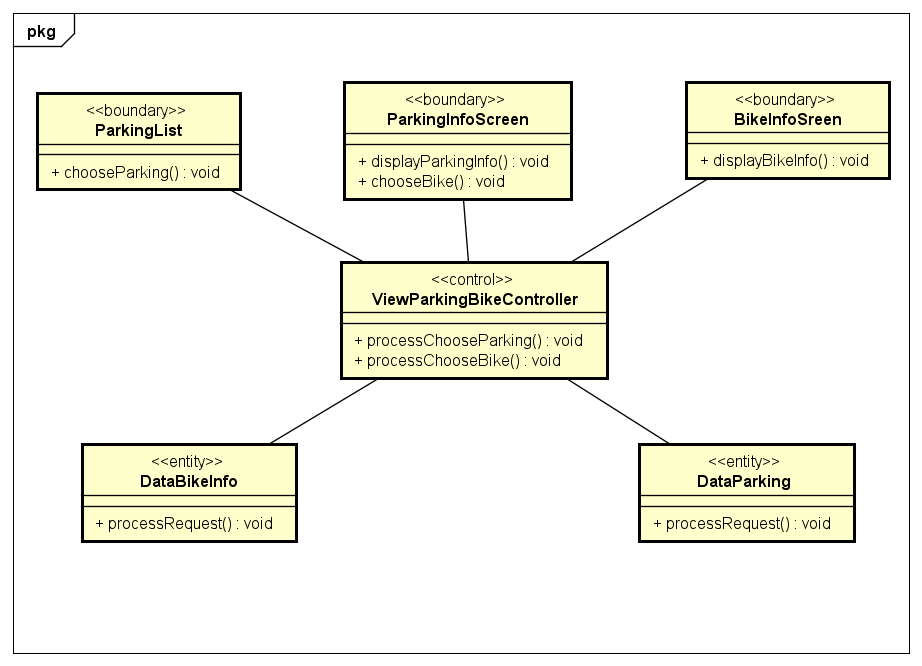
### 3.2.2 Biểu đồ giao tiếp

* Biểu đồ giao tiếp xem thông tin xe bãi xe
* Biểu đồ giao tiếp thanh toán
* Biểu đồ giao tiếp thuê xe
* Biểu đồ giao tiếp trả xe

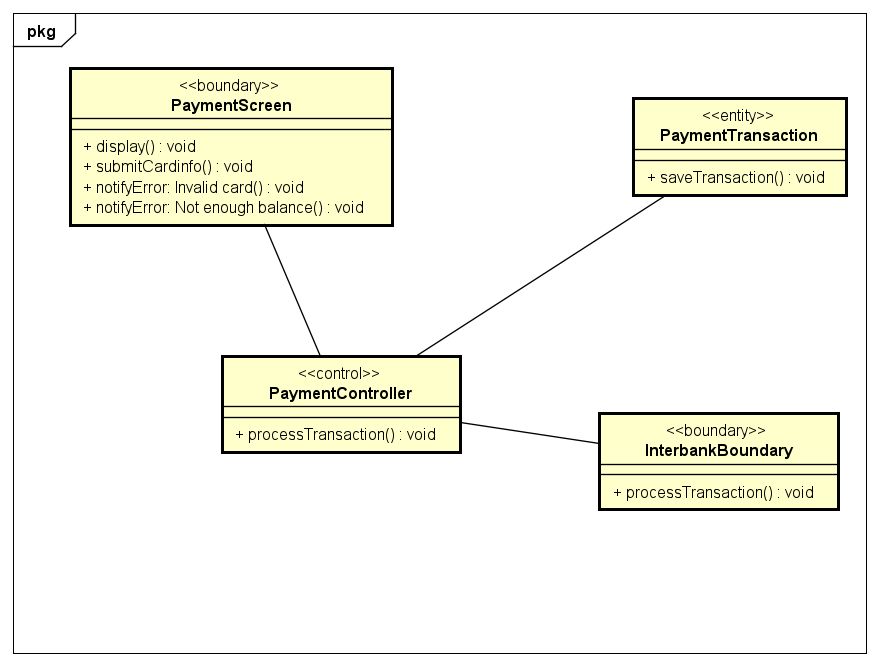


## 3.3 Biểu đồ lớp phân tích

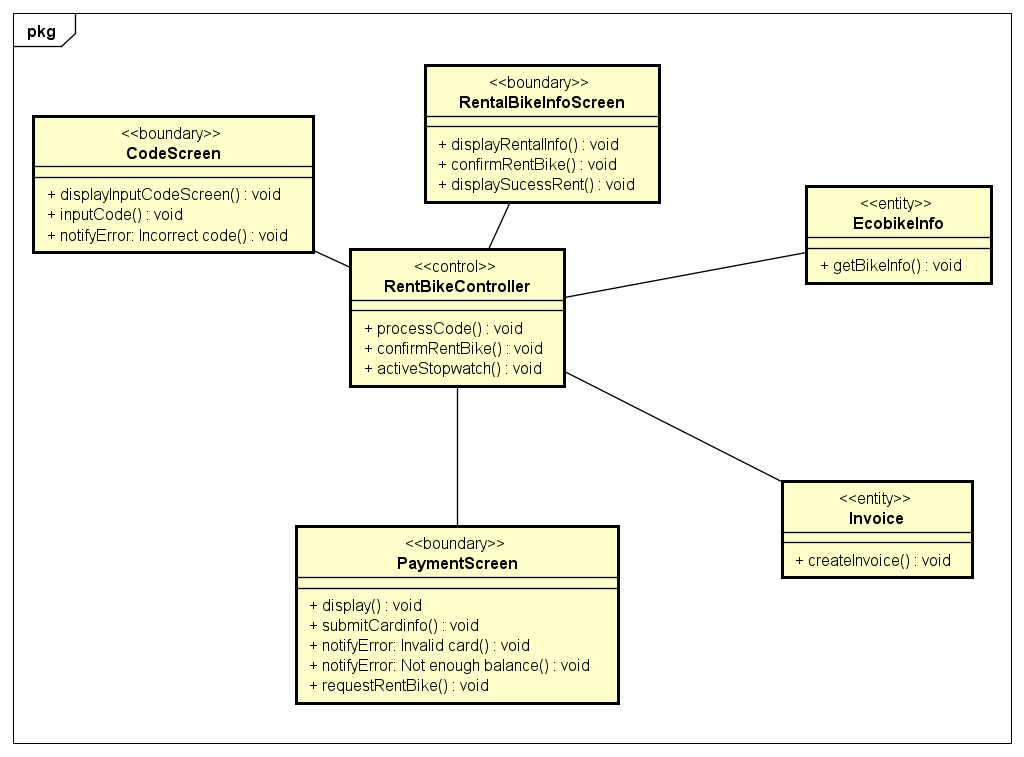
* Biểu đồ lớp phân tích xem thông tin xe bãi xe



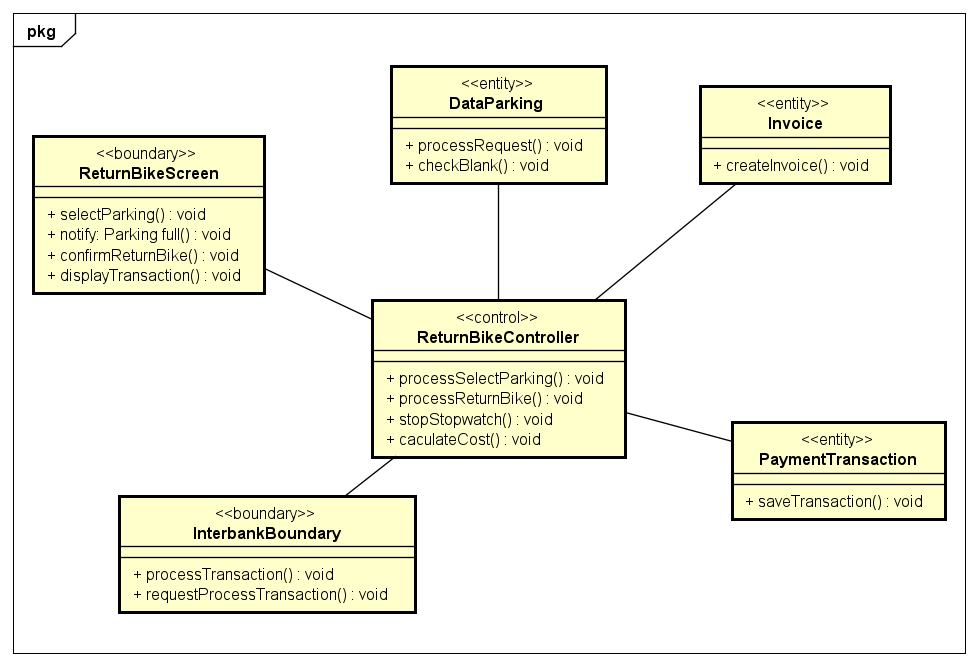
* Biểu đồ lớp phân tích thanh toán



* Biểu đồ lớp phân tích thuê xe



* Biểu đồ lớp phân tích trả xe



# **4. Thiết kế chi tiết**

## 4.1 Thiết kế giao diện người dùng

### 4.1.1 Chuẩn hóa cấu hình màn hình

* Giao diện thiết kế màn hình thân thiện, dễ sử dụng
* Các tính năng nổi bật như tìm xe, thuê xe, trả xe, xem bãi, …
* Chuẩn hóa: sử dụng các chuẩn hóa công nghệ đã tổn tại. Ví dụ: chuẩn giao diện W3C, Microsoft Windows
* Nguyên tắc cấu trúc: tổ chức giao diện người dùng một cách có chủ đich, dựa trên các mô hình rõ ràng, nhất quán và dễ nhân biết đối với người sử dụng
* Nguyên tắc hiển thị: Làm cho tất cả các tùy chọn và yếu tố cần thiết cho một tác vụ nhất định hiển thị cùng lúc mà không làm cho người sử dụng mất tập trung với thông tin không liên quan hoặc dư thừa.
* Nguyên tắc phản hồi: Thiết kế phải thông báo cho người sử dụng về các hành động hoặc giải thích về các thay đổi trạng thái, điều kiện và các lỗi hoặc các trường hợp ngoại lệ có liên quan và được người sử dụng quan tâm. Quá trình này phải thông qua ngôn ngữ rõ ràng, ngắn gọn và quen thuộc với họ.

### 4.1.2 Sơ đồ chuyển đổi màn hình

Diagram

Description automatically generated

### 4.1.3 Đặc tả màn hình

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EcoBikeRental** | | **Ngày tạo** | **Chấp nhận bởi** | **Đã đánh giá bởi** | **Người phụ trách** |
| **Đặc điểm màn hình** | **Màn hình chính: Danh sách bãi xe** | 23/11/2020 | Lâm | Liên | Lan |
|  | | **Điều khiển** | **Hoạt động** | **Chức năng** | |
| Button  Thuê xe | Click | Hiển thị màn hình nhập mã xe muốn thuê. Được hiển thị khi chưa có xe nào được thuê | |
| Khu vực hiển thị vui lòng trả xe | Initial | Được hiển thị khi đã có xe được thuê | |
| Khu vực danh sách bãi đậu xe | Initial | Hiển thị danh sách bãi đậu xe trong bãi | |
| Liên kết Xem bãi | Click | Xem bãi xe được chọn, trả về màn hình thông tin bãi xe | |
| Liên kết  Trả xe | Click | Được hiển thị khi đã có xe được thuê, trả về màn hình hóa đơn | |
| **Đăch tả màn hình** | **Thông tin bãi xe** | 23/11/2020 | Lâm | Liên | Lan |
|  | | **Điều khiển** | **Hoạt động** | **Chức năng** | |
| Khu vực thông tin xe | Initial | Hiển thị thông tin xe | |
| Button  Bãi xe | Click | Hiển thị màn hình danh sách bãi xe | |
| Khu vực thông tin bãi xe | Initial | Hiển thị các thông tin của bãi xe bao gồm: tên bãi, vị trí, số ô chứa trong bãi, số ô trống | |
| Khu vực danh sách xe trong bãi | Initial | Hiển thị danh sách xe trong bãi | |
| Liên kết Xem | Click | Hiển thị màn hình thông tin xe | |
| **Đặc tả màn hình** | **Thông tin xe** | 23/11/2020 | lâm | liên | lan |
|  | | **Điều khiển** | **Hoạt động** | **Chức năng** | |
| Khu vực thông tin xe | Initial | Hiển thị thông tin xe đã chọn | |
| Khu vực hiển thị vui lòng trả xe | Initial | Được hiển thị khi đã có xe được thuê | |
| Button  Đóng | Click | Hiển thị màn hình trước đó | |
| Button  Thuê xe | Click | Thuê xe, trả về màn hình thanh toán cọc xe. Hiển thị khi chưa có xe nào được thuê | |
| **Đặc tả màn hình** | **Nhập mã thuê** | 23/11/2020 | lâm | lâm | liên |
|  | | **Điều khiển** | **Hoạt động** | **Chức năng** | |
| Text Nhập mã xe |  | Nhập mã xe cần thuê | |
| Button Xác nhân | Click | Xác nhận thuê xe, Trả về màn hình thông tin xe vừa nhập | |
| **Đặc điểm màn hình** | **Lỗi mã xe không tồn tại** | 23/11/2020 | lâm | lâm | liên |
|  | | **Điều khiển** | **Hoạt động** | **Chức năng** | |
| Khu vực thông báo | Initial | Thông báo mã xe không tồn tại | |
| **Đặc điểm màn hình** | **Thanh toán cọc xe** | 23/11/2020 | lâm | lâm | liên |
|  | | **Điều khiển** | **Hoạt động** | **Chức năng** | |
| Khu vực các thông tin khách hàng | Initial | Hiển thị các trường thuộc tính thông tin khách hàng cần để thuê xe để nhập | |
| Text-Các thông tin khách hàng cần để thuê xe | Initial | Nhập vào các thông tin cần để thuê xe | |
| Khu vực hiển thị chú ý | Initial | Thông báo về số tiền cần giao dịch | |
| Button Thanh toán | Click | Thanh toán hóa đơn thuê xe qua thẻ tín dụng | |
| Button Hủy | Click | Hủy thuê xe, trả về màn hình danh sách bãi đậu xe | |
| **Đặc tả màn hình** | **Lỗi giao dịch** | 23/11/2020 | Lâm | lâm | liên |
|  | | **Điều khiển** | **Hoạt động** | **Chức năng** | |
| Khu vực hiển thị lỗi | Initial | Thông báo thông tin lỗi giao dịch gồm các lỗi:  -thẻ không hợp lệ  -thẻ không đủ số dư  -Internal Server Error  -Giao dịch bị nghi ngờ gian lận  -Không đủ thông tin giao dịch  -Thiếu thông tin verson  -Amount không hợp lệ  -Amount không đủ | |
| **Đặc điểm màn hình** | **Thanh toán thành công** | 23/11/2020 | lâm | lâm | liên |
|  | | **Điều khiển** | **Hoạt động** | **Chức năng** | |
| Button Close | Click | Xác nhận xe đã được thuê, trả vể màn hình danh sách bãi xe | |
| **Đặc điểm màn hình** | **Lỗi bãi xe đầy** | 23/11/2020 | lâm | lan | linh |
|  | | **Điều khiển** | **Hoạt động** | **Chức năng** | |
| Khu vực hiển thị lỗi | Initial | Được hiển thị khi mà trả xe về bãi đầy từ màn hình danh sách bãi xe | |
| **Đặc điểm màn hình** | **Hóa đơn** | 23/11/2020 | lâm | lan | linh |
|  | | **Điều khiển** | **Hoạt động** | **Chức năng** | |
| Khu vực hiển thị  Hóa đơn thông tin xe đã thuê | Initial | Hiển thị thông tin hóa đơn xe đã thuê | |
| Khu vực các thông tin khách hàng | Initial | Hiển thị các trường thuộc tính thông tin khách hàng cần để thuê xe để nhập | |
| Text-Các thông tin khách hàng cần để trả xe | Intial | Nhập vào các thông tin cần để trả xe | |
| Khu vực chú ý | Initial | Thông báo về số tiền cần giao dịch | |
| Button  Hủy | Click | Trả về màn hình danh sách bãi xe | |
| Button Thanh toán | Click | Thanh toán hóa đơn | |

## 4.2 Mô hình dữ liệu

Mô hình hóa dữ liệu là quá trình tạo ra một mô hình dữ liệu. Khi tạo một mô hình dữ liệu, trước tiên phải xác định dữ liệu, các thuộc tính và mối quan hệ của nó với dữ liệu khác và xác định các ràng buộc hoặc giới hạn đối với dữ liệu

Mô hình dữ liệu xác định dữ liệu, các thuộc tính dữ liệu và các mối quan hệ hoặc liên kết với dữ liệu khác. Mô hình dữ liệu cung cấp một cái nhìn tổng quát, do người dùng định nghĩa về dữ liệu đại diện cho kịch bản và dữ liệu nghiệp vụ.

### 4.2.1 Mô hình hóa dữ liệu khái niệm

Mô hình dữ liệu khái niệm, còn được gọi là mô hình miền (Domain models), thiết lập các khái niệm và ngữ nghĩa cơ bản của một miền nhất định đối với nhiều đối tượng của các bên liên quan.

Các mô hình khái niệm được xây dựng dựa trên Kiến trúc tổng thể bằng cách sử dụng các mô hình Mối quan hệ thực thể hoặc Lớp UML.

### Diagram Description automatically generated4.2.2 Thiết kế cơ sở dữ liệu

Diagram

Description automatically generated

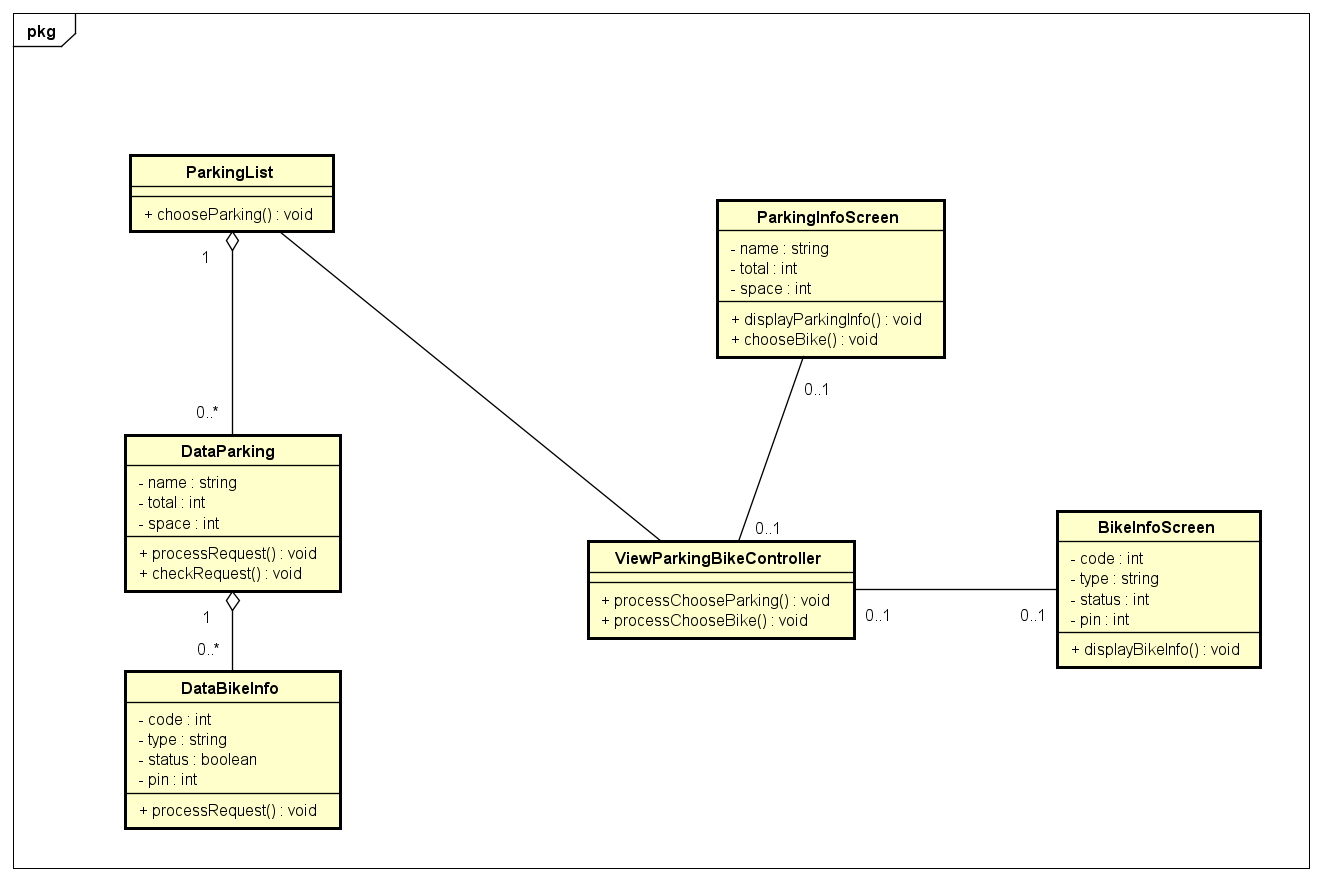
## 4.3 Non-Database Management System Files

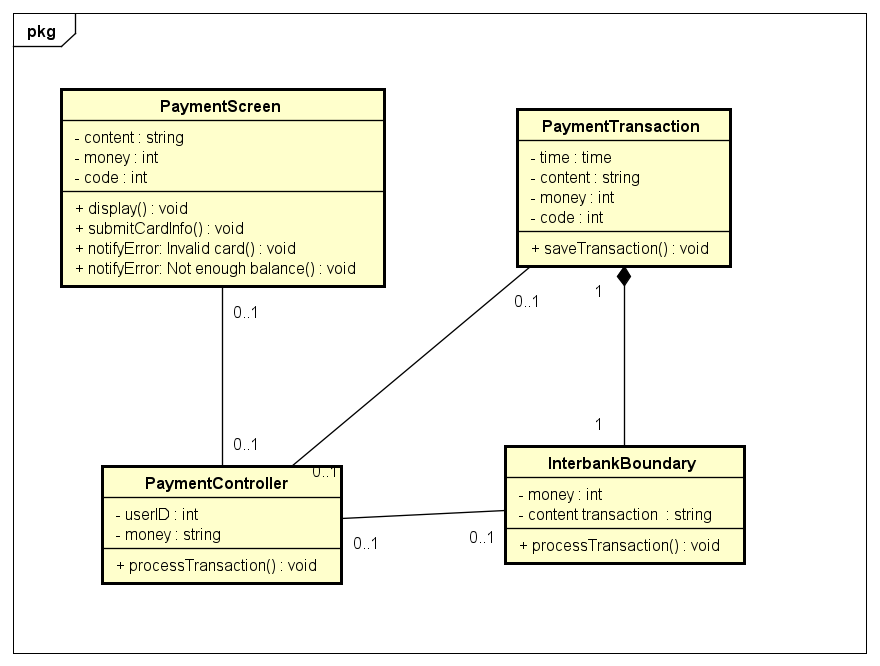
## 4.4 Thiết kế lớp

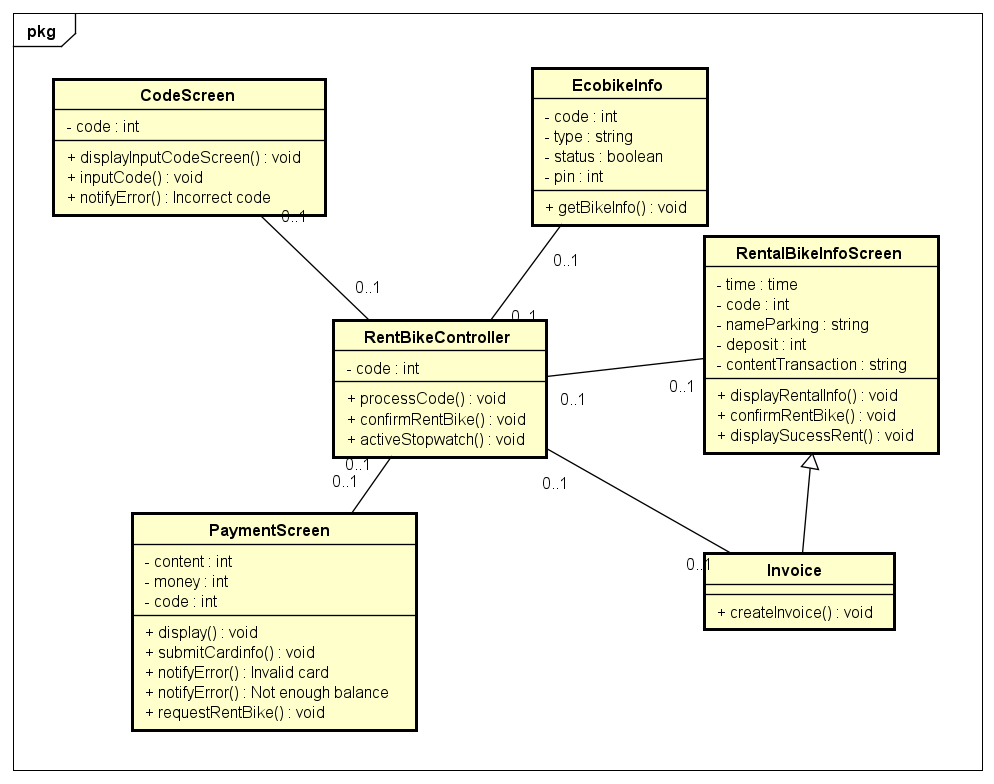
### 4.4.1 Biểu đồ lớp tổng quát

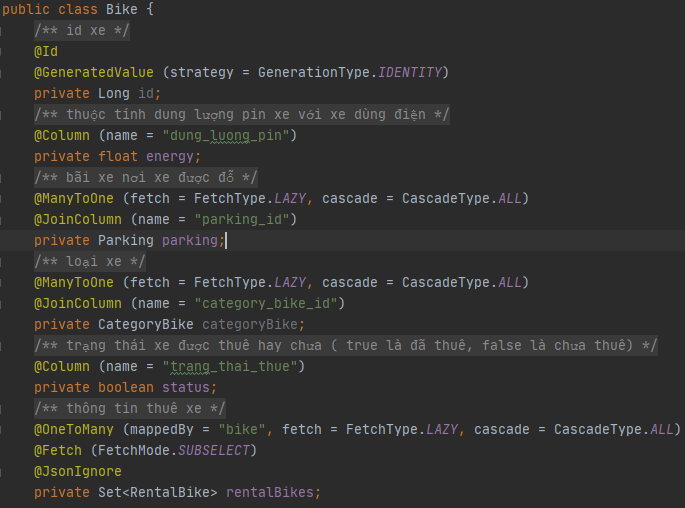
### 

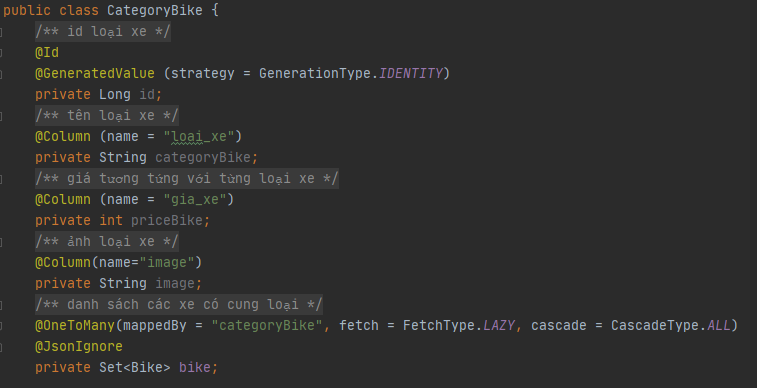
### 4.4.2 Biểu đồ lớp

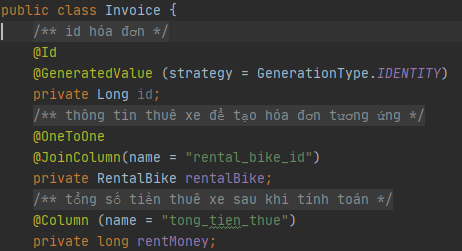
* Xem thông tin bãi xe loại xe
* Thanh toán

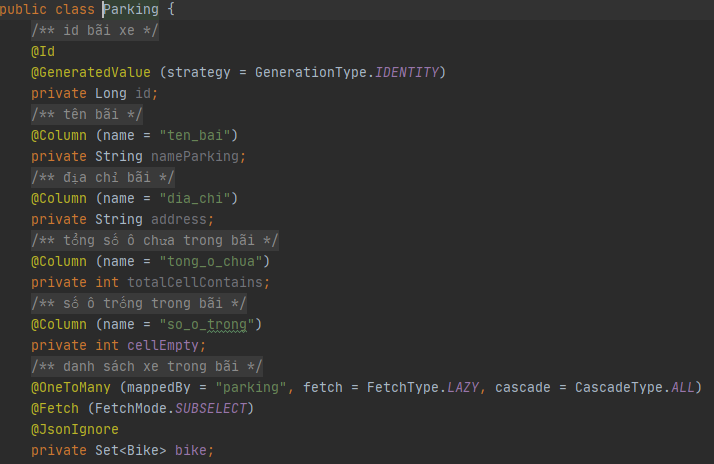


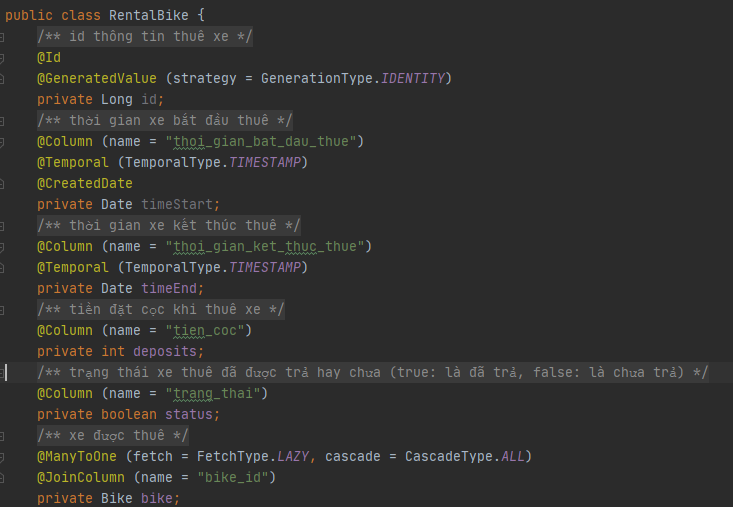
* Thuê xe

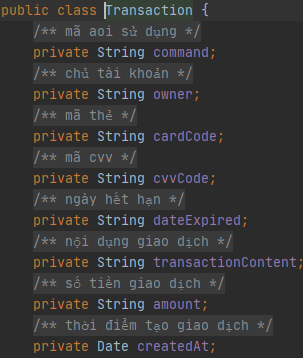
4.4.3 Thiết kế lớp











# **5. Cân nhắc kiến trúc**

## 5.1 Mục tiêu và nguyên tắc

### 5.1.1 Mục tiêu phần mềm

Giao diện dễ sử dụng, tốc độ xử lý dữ liệu nha, các module độc lập nhau, dễ phát triển cũng như sửa chữa, bảo trì hệ thống

### 5.1.2 Nguyên tắc

## 5.2 Chiến lược kiến trúc

- Tái sử dụng các thành phần hiện có để phát triển các thành phần mới

- Xây dựng các mô đun tương đối độc lập để có thể dễ dàng mở rộng phần mềm

- Tối ưu việc sử dụng tài nguyen

- Đồng bộ hóa

## 5.3 Coupling and Cohesion

### 5.3.1 Coupling

Để có một bản thiết kế tốt thì cần phải thiết kế sao cho coupling lỏng để khi

có sự thay đổi ở một module thì ảnh hưởng ít nhất có thể đến các module

khác. Coupling càng lỏng lẻo, càng tốt

- Content coupling: Xảy ra khi 1 module sử dụng code của 1 module khác, điều này sẽ không đúng với những khái niệm thiết kế cơ bản là đóng gói thông tin

- Common coupling: Xảy ra khi các module cùng truy cập vào cùng data global, và điều này có thể dẫn đến việc không thể kiểm soát được lỗi và sẽ xảy ra những lỗi không mong muốn

- Control coupling: là khi 1 module đang control luồng hoạt động của 1 module khác thông qua các tham số đầu vào, và nó có thể dẫn đến 1 vài trường hợp không mong muốn khi các tham số truyền vào không được đúng

- Stamp Coupling: xảy ra khi các module chỉ sử dụng các thành phần cấu trúc dữ liệu được chia sẻ với nhau, và trong trường hợp này, khi thay đổi 1 thuộc tính của 1 module có thể sẽ ảnh hưởng đến các module khác đang phụ thuộc vào nó

- Data coupling: xảy ra khi 2 hoặc nhiều module độc lập với nhau, tương tác, chia sẻ tài nguyên cho nhau và thông thường là qua tham số đầu vào

=>Kết luận: Do quá trình code mỗi bạn thực hiện một phần riêng lẻ nên sự liên kết giữa các module khá lỏng lẻo đạt đến datacoupling.

### 5.3.2 Conhesion

- Coincidental: Sự gắn kết ngẫu nhiên là khi các phần của mô-đun được nhóm lại một cách tùy tiện; mối quan hệ duy nhất giữa các bộ phận là họ đã được nhóm lại với nhau.

- Logical: Sự gắn kết logic là khi các phần của mô-đun được nhóm lại vì chúng được phân loại logic để làm việc tương tự, ngay cả khi chúng khác nhau theo bản chất.

- Temporl: Sự gắn kết thời gian là khi các phần của mô-đun được nhóm lại khi chúng được xử lý – các phần được xử lý tại một thời điểm cụ thể trong quá trình thực hiện chương trình

- Produral: Sự gắn kết theo thủ tục là khi các phần của mô-đun được nhóm lại bởi vì chúng luôn tuân theo một trình tự thực thi nhất định

- Communicational: Sự liên kết giao tiếp là khi các phần của mô-đun được nhóm lại vì chúng hoạt động trên cùng một dữ liệu

- Sequential: Sự liên kết tuần tự là khi các phần của một mô-đun được nhóm lại bởi vì đầu ra từ một phần là đầu vào đến một phần khác giống như một dây chuyền lắp ráp

- Functional: Sự gắn kết chức năng là khi các phần của mô-đun được nhóm lại vì tất cả chúng đóng góp vào một nhiệm vụ duy nhất được định nghĩa rõ ràng của mô-đun

=>Kết luận:

- Hầu hết các class đều đạt mức Functional vì nó đều hướng đến một nhiệm vụ duy nhất.

## 5.4 Nguyên tắc thiết kế

Tuân theo nguyên tắc thiết kế SOLID:

SOLID là gì?

SOLID là viết tắt của 5 chữ cái đầu trong 5 nguyên tắc thiết kế hướng đối tượng. Giúp cho lập trình viên viết ra những đoạn code dễ đọc, dễ hiểu, dễ maintain. Nó được đưa ra bởi Robert C. Martin và Michael Feathers. 5 nguyên tắc đó bao gồm:

**Single responsibility priciple**

Mỗi lớp chỉ nên chịu trách nhiệm về một nhiệm vụ cụ thể nào đó mà thôi.

Nguyên lý đầu tiên ứng với chữ S trong SOLID, có ý nghĩa là một class chỉ nên giữ một trách nhiệm duy nhất. Một class có quá nhiều chức năng sẽ trở nên cồng kềnh và trở nên khó đọc, khó maintain. Mà đối với ngành IT việc requirement thay đổi, cần thêm sửa chức năng là rất bình thường, nên việc code trong sáng, dễ đọc dễ hiểu là rất cần thiết.

Ví dụ: Hình dung rằng nhân viên của một công ty phần mềm cần phải làm 1 trong 3 việc sau đây: lập trình phần mềm (developer), kiểm tra phần mềm (tester), bán phần mềm (salesman). Mỗi nhân viên sẽ có một chức vụ và dựa vào chức vụ sẽ làm công việc tương ứng. Khi đó bạn có nên thiết kế lớp “Employee” với thuộc tính “position” và 3 phương thức developSoftware(), testSoftware() và saleSoftware() không?

class Employee

{

string position;

function developSoftware(){};

function testSoftware(){};

function saleSoftware(){};

}

Câu trả lời là KHÔNG. Thử hình dung nếu có thêm một chức vụ nữa là quản lí nhân sự, ta sẽ phải sửa lại lớp “Employee”, thêm phương thức mới vào sao? Nếu có thêm 10 chức vụ nữa thì sao? Khi đó các đối tượng được tạo ra sẽ dư thừa rất nhiều phương thức: Developer thì đâu cần dùng hàm testSoftware() và saleSoftware() đúng không nào, lỡ may dùng lầm phương thức cũng sẽ gây hậu quả khôn lường.

Áp dụng nguyên tắc Single Responsibility: mỗi lớp 1 trách nhiệm. Ta sẽ tạo 1 lớp trừu tượng là “Employee” có phương thức là working(), từ đây bạn kế thừa ra 3 lớp cụ thể là Developer, Tester và Salesman. Ở mỗi lớp này bạn sẽ implement phương thức working() cụ thể tuy theo nhiệm vụ của từng người. Khi đó chúng ta sẽ bị tình trạng dùng nhầm phương thức nữa.

**Open/Closed principle**

Không được sửa đổi một Class có sẵn, nhưng có thể mở rộng bằng kế thừa.

Nguyên lý thứ 2 ứng với chữ O trong SOLID.

Theo nguyên lý này, mỗi khi ta muốn thêm chức năng cho chương trình, chúng ta nên viết class mới mở rộng class cũ (bằng cách kế thừa hoặc sở hữu class cũ) chứ không nên sửa đổi class cũ. Việc này dẫn đến tình trạng phát sinh nhiều class, nhưng chúng ta sẽ không cần phải test lại các class cũ nữa, mà chỉ tập trung vào test các class mới, nơi chứa các chức năng mới.

Thông thường việc mở rộng thêm chức năng thì phải viết thêm code, vậy để thiết kế ra một module có thể dễ dàng mở rộng nhưng lại hạn chế sửa đổi code ta cần làm gì. Cách giải quyết là tách những phần dễ thay đổi ra khỏi phần khó thay đổi mà vẫn đảm bảo không ảnh hưởng đến phần còn lại.

Đặt vấn đề: Ta cần 1 lớp đảm nhận việc kết nối đến CSDL. Thiết kế ban đầu chỉ có SQL Server và MySQL. Thiết kế ban đầu có dạng như sau:

class ConnectionManager

{

public function doConnection(Object $connection)

{

if($connection instanceof SqlServer) {

//connect with SqlServer

} elseif($connection instanceof MySql) {

//connect with MySql

}

}

Sau đó yêu cầu đặt ra phải kết nối thêm đến Oracle và một vài hệ CSDL khác.

Để thêm chức năng ta phải thêm vào code những khối esleif khác, việc này làm code cồng kềnh và khó quản lý hơn.

*Giải pháp:*

Áp dụng Abstract thiết kế lại các lớp SqlServer, MySql, Oracle…

Các lớp này đều có chung nhiệm vụ tạo kết nối đến csdl tương ứng có thể gọi chung là Connection.

Cách thức kết nối đến csdl thay đổi tùy thuộc vào từng loại kết nối nhưng có thể gọi chung là doConect.

Vậy ta có lớp cơ sở Connection có phương thức doConnect, các lớp cụ thể là SqlServer, MySql, Oracle… kế thừa từ Connection và overwrite lại phương thức doConnect phù hợp với lớp đó.

Thiết kế sau khi làm lại có dạng như sau:

abstract class Connection()

{

public abstract function doConnect();

}

class SqlServer extends Connection

{

public function doConnect()

{

//connect with SqlServer

}

}

class MySql extends Connection

{

public function doConnect()

{

//connect with MySql

}

}

class ConnectionManager

{

public function doConnection(Connection $connection)

{

//something

//.................

//connection

$connection->doConnect();

}

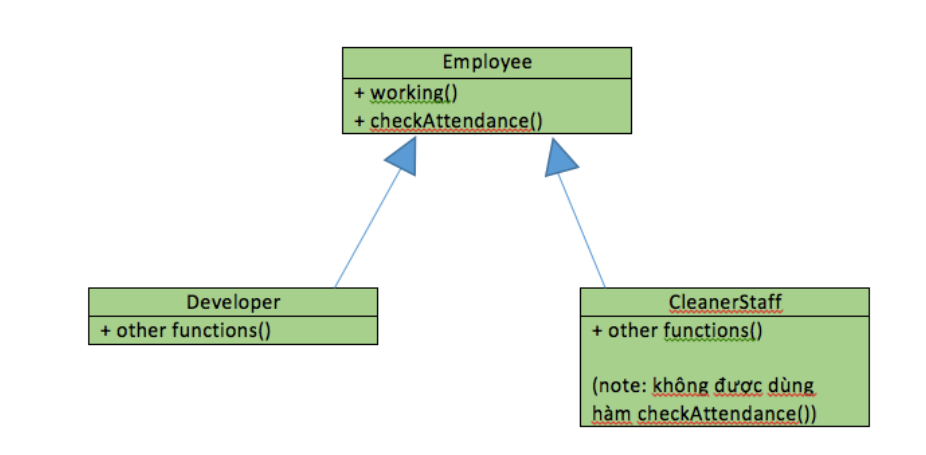
}

Với thiết kế này khi cần kết nối đến 1 loại csdl mới chỉ cần thêm 1 lớp mới kế thừa Connection mà không cần sửa đổi code của lớp ConnectionManager, điều này thỏa mãn 2 điều kiện của nguyên lý OCP.

**Liskov substitution principle**

Các đối tượng (instance) kiểu class con có thể thay thế các đối tượng kiểu class cha mà không gây ra lỗi.

Nguyên tắc thứ 3, ứng với chữ L trong SOLID.



Quay trở lại ví dụ lớp Emloyee trong phần 1, ta giả sử có công ty sẽ điểm danh vào mỗi buổi sáng, và chỉ có các nhân viên thuộc biên chế chính thức mới được phép điểm danh. Ta bổ sung phương thức checkAttendance() vào lớp Employee.

Hình dung có một trường hợp sau: công ty thuê một nhân viên lao công để làm vệ sinh văn phòng, mặc dù là một người làm việc cho công ty nhưng do không được cấp số ID nên không được xem là một nhân viên bình thường, mà chỉ là một nhân viên thời vụ, do đó sẽ không được điểm danh.

Nguyên tắc này nói rằng: Nếu chúng ta tạo ra một lớp CleanerStaff kế thừa từ lớp Employee, và implement hàm working() cho lớp này, thì mọi thứ đều ổn, tuy nhiên lớp mới này cũng lại có hàm checkAttendance() để điểm danh, mà như thế là sai quy định dẫn đến chương trình bị lỗi. Như vậy, thiết kế lớp CleanerStaff kế thừa từ lớp Employee là không được phép.

Có nhiều cách để giải quyết tình huống này ví dụ như tách hàm checkAttendance() ra một interface riêng và chỉ cho các lớp Developer, Tester và Salesman implements interface này.

**Interface segregation principle**

Thay vì dùng 1 interface lớn, ta nên tách thành nhiều interface nhỏ, với nhiều mục đích cụ thể.

Nguyên lý này rất dễ hiểu. Hãy tưởng tượng chúng ta có 1 interface lớn, khoảng 100 methods. Việc implements sẽ rất vất vả vì các class impliment interface này sẽ bắt buộc phải phải thực thi toàn bộ các method của interface. Ngoài ra còn có thể dư thừa vì 1 class không cần dùng hết 100 method. Khi tách interface ra thành nhiều interface nhỏ, gồm các method liên quan tới nhau, việc implement và quản lý sẽ dễ hơn.

Chúng ta có 2 class Dog và Snake implement interface Animal. Nhưng thật vô lý, Dog thì làm sao có thể fly(), cũng như Snake không thể nào run() được? Thay vào đó, chúng ta nên tách thành 3 interface như thế này:

interface Animal {

void eat();

}

interface RunnableAnimal extends Animal {

void run();

}

interface FlyableAnimal extends Animal {

void fly();

}

**Dependency inversion principle**

1.Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ thuộc vào abstraction.

2.Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại (Các class giao tiếp với nhau thông qua interface (abstraction), không phải thông qua implementation.)

Có thể hiểu nguyên lí này như sau: những thành phần trong 1 chương trình chỉ nên phụ thuộc vào những cái trừu tượng (abstraction). Những thành phần trừu tượng không nên phụ thuộc vào các thành phần mang tính cụ thể mà nên ngược lại.

Những cái trừu tượng (abstraction) là những cái ít thay đổi và biến động, nó tập hợp những đặc tính chung nhất của những cái cụ thể. Những cái cụ thể dù khác nhau thế nào đi nữa đều tuân theo các quy tắc chung mà cái trừu tượng đã định ra. Việc phụ thuộc vào cái trừu tượng sẽ giúp chương trình linh động và thích ứng tốt với các sự thay đổi diễn ra liên tục.

Lấy ví dụ về ổ cứng của máy tính, bạn có thể dùng loại ổ cứng thể rắn SSD đời mới để chạy cho nhanh, tuy nhiên cũng có thể dùng ổ đĩa quay HDD thông thường. Nhà sản xuất Mainboard không thể nào biết bạn sẽ dùng ổ SSD hay loại HDD đĩa quay thông thường. Tuy nhiên họ sẽ luôn đảm bảo rằng bạn có thể dùng bất cứ thứ gì bạn muốn, miễn là ổ đĩa cứng đó phải có chuẩn giao tiếp SATA để có thể gắn được vào bo mạch chủ. Ở đây chuẩn giao tiếp SATA chính là interface, còn SSD hay HDD đĩa quay là implementation cụ thể.

Trong khi lập trình cũng vậy, khi áp dụng nguyên lý này, ở những lớp trừu tượng cấp cao, ta thường sử dụng interface nhiều hơn thay vì một kiểu kế thừa cụ thể. Ví dụ, để kết nối tới Database, ta thường thiết kế lớp trừu tượng DataAccess có các phương thức phương thức chung như save(), get(), … Sau đó tùy vào việc sử dụng loại DBMS nào (vd: MySql, MongoDB, …) mà ta kế thừa và implement những phương thức này. Tính chất đa hình của OOP được vận dụng rất nhiều trong nguyên lý này.

**Kiểm thử**

