

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

Viện Công nghệ thông tin và Truyền thông

Tài liệu đặc tả thiết kế phần mềm

(Software Design Description)

Phiên bản 1

Phân tích và thiết kế hệ thống cho thuê xe đạp EcoBikeRental

Môn: Thiết kế và xây dựng phần mềm

Nhóm 4

Trần Minh Hiếu (20170075)

Nguyễn Trung Hiếu (20170073)

Đinh Văn Hiếu (20170072)

Nguyễn Tuấn Hiệp (20170070)

Hà Nội, ngày 5 tháng 12 năm 2020

Mục lục

Nội dung

Mục lục	2
1 Giới thiệu	5
1.1 Mục đích	5
1.2 Phạm vi	5
1.2.1 Quy trình sử dụng	5
1.2.2 Cách tính tiền thuê xe	6
1.2.3 Các loại xe đạp	6
1.3 Từ điển thuật ngữ	6
1.4 Tham khảo	6
2 Thiết kế kiến trúc	6
2.1 Lựa chọn kiến trúc phần mềm	6
2.2 Thiết kế tổng quan	6
2.3 Thiết kế chi tiết gói	8
2.3.1 Gói dao	8
2.3.2 Gói models	9
2.3.3 Gói payment	10
2.3.4 Gói renting	10
2.3.5 Gói barcode	10
2.3.6 Gói utilities	10
2.3.7 Gói servlets	10
2.4 Biểu đồ tương tác	11
2.4.1 Biểu đồ tương tác cho UC001 – Xem thông tin bãi xe	11
2.4.2 Biểu đồ tương tác cho UC002 - Xem thông tin xe	11
2.4.3 Biểu đồ tương tác cho UC003 – Thuê xe đạp	11
2.4.4 Biểu đồ tương tác cho UC004 - Trả xe đạp	12
3 Thiết kế giao diện	12
3.1 Giao diện trang chủ	12
3.2 Giao diện trang tìm kiếm barcode	13
3.3 Giao diện danh sách bãi xe	14
3.4 Giao diện chi tiết bãi xe	15
3.5 Giao diện chi tiết xe	16

3.6	Giao diện thanh toán	18
3.7	Giao diện trả xe.....	20
3.8	Sơ đồ chuyển màn hình.....	22
4	Thiết kế mô hình dữ liệu.....	22
4.1	Mô hình dữ liệu mức khái niệm.....	22
4.2	Mô hình dữ liệu mức logic.....	23
4.3	Thiết kế chi tiết	24
4.3.1	Docks	24
4.3.2	Bikes	25
4.3.3	User.....	25
4.3.4	Transactions	25
5	Thiết kế lớp.....	26
5.1	Biểu đồ lớp thiết kế.....	26
5.2	Thiết kế lớp chi tiết.....	26
5.2.1	Thiết kế lớp thuộc gói dao	26
5.2.2	Thiết kế lớp thuộc gói model	31
5.2.3	Thiết kế lớp thuộc gói payment	46
5.2.4	Thiết kế lớp thuộc gói servlets	50
5.2.5	Thiết kế lớp thuộc gói barcode.....	56
5.2.6	Thiết kế lớp thuộc gói renting.....	56
5.2.7	Thiết kế lớp thuộc gói utilities	58
6	Design considerations	59
6.1	Design Concept – Coupling và Cohesion	59
6.1.1	Coupling	59
6.1.2	Cohesion	63
6.2	Đánh giá tuân thủ nguyên lý SOLID	68
6.2.1	S – Single-responsibility principle.....	68
6.2.2	O – Open-closed principle	68
6.2.3	L – Liskov substitution principle	70
6.2.4	I – Interface segregation principle	70
6.2.5	D – Dependency inversion Principle	70
6.3	Áp dụng Design Pattern.....	71
6.3.1	Singleton	71
6.3.2	Strategy	71
6.3.3	Abstract Factory.....	73

1 Giới thiệu

1.1 Mục đích

Tài liệu này đưa ra mô tả chi tiết về thiết kế hệ thống và giao diện của phần mềm quản lý cho thuê xe EcoBike Rental. Các nhà phát triển phần mềm có thể thực hiện cài đặt và kiểm thử phần mềm dựa theo các mô tả trong tài liệu này.

1.2 Phạm vi

EcoBikeRental là một hệ thống đa nền tảng, hoạt động 24/7, cho phép người dùng là cư dân/khách vãng lai tại khu đô thị Ecopark nhằm quản lý dịch vụ cho thuê xe đạp theo giờ với nhiều bãi để xe để thuê/trả xe tự động trong khu đô thị. Mỗi khi trả xe, khách đưa xe vào ổ khoá tại các trạm để xe.

1.2.1 Quy trình sử dụng

Trước tiên, khách hàng cần phải tạo tài khoản trên ứng dụng EcoBikeRental, xác thực thông tin, thiết lập quyền truy cập của ứng dụng, và thiết lập phương thức thanh toán để trả phí thuê xe (bằng cách liên kết với liên ngân hàng hoặc ví điện tử).

Khi các bước khởi động hoàn tất và ứng dụng khởi chạy, vị trí hiện tại của khách hàng cùng với vị trí của các bãi để xe ở gần sẽ được hiện lên trên bản đồ (số lượng bãi để xe có thể thay đổi khi kích thước bản đồ thay đổi). Khách hàng có thể nhấn chọn một bãi xe trên bản đồ hoặc sử dụng tính năng tìm kiếm tên/địa chỉ để xem thông tin chi tiết về bãi xe đó, bao gồm:

- Tên của bãi xe
- Địa chỉ bãi xe
- Diện tích bãi
- Số xe hiện tại đang có
- Vị trí trống của từng loại xe ở bãi xe
- Khoảng cách và thời gian đi bộ từ vị trí của khách hàng tới bãi xe này.
- Thông tin chi tiết về từng xe trong bãi.
- Riêng loại xe đạp điện có thêm thông tin về pin của motor điện và thời gian tối đa tương ứng có thể sử dụng được xe để khách hàng xem xét khi mượn.

Để có thể thuê một xe, khách hàng cần sử dụng ứng dụng EcoBikeRental để quét mã vạch trên ổ khóa. Lúc này, thông tin của xe sẽ hiện lên và khách hàng sẽ được yêu cầu chọn một phương thức thanh toán để thực hiện giao dịch. Khách hàng cần phải đặt cọc trước số tiền bằng 40% giá trị của xe. Sau khi xác nhận giao dịch, hệ thống sẽ tự động trừ tiền cọc trong thẻ/tài khoản của khách hàng và lưu lại giao dịch, khóa sẽ được tự động mở và khách hàng có thể lấy xe ra sử dụng.

Trong thời gian thuê xe, khách hàng luôn có thể sử dụng ứng dụng để xem thông tin về xe đang thuê, bao gồm: loại xe, thời gian thuê tính tới hiện tại, số tiền cần trả, và tình trạng xe (ví dụ: lượng pin hiện tại của xe đạp điện).

Khi muốn trả xe, khách hàng đưa xe vào vị trí trống bất kỳ trong bãi bất kỳ (thông thường là bãi xe gần nhất dựa vào vị trí thực tế) và đóng khoá xe lại. Lúc này, hệ thống sẽ tự động trả lại tiền cọc xe và trừ đi số tiền phải trả tương ứng với thời gian thuê xe; đồng thời, lưu lại giao dịch thuê xe.

Thời gian đáp ứng cho mọi giao dịch không được phép quá 1 giây.

Mỗi khi thực hiện một giao dịch, khách hàng cần cung cấp thông tin thẻ (card info, bao gồm tên chủ thẻ - cardholder name, mã thẻ - card number, ngân hàng phát hành - issuing bank, ngày hết hạn - expiration date, và mã bảo mật - security code) và nội dung giao dịch (transaction description). App sẽ hiển thị, đồng thời lưu lại thông tin giao dịch vào hệ thống. Sau đó hệ thống sẽ gửi email chứa thông tin giao dịch tới hộp thư điện tử của khách hàng.

1.2.2 Cách tính tiền thuê xe

Khách hàng sẽ được miễn phí thuê xe nếu trả xe trong vòng 10 phút, kể cả tại các điểm trả khác nhau. Nếu khách hàng dùng xe hơn 10 phút, phí thuê xe được tính lũy tiến theo thời gian thuê như sau:

- Giá khởi điểm cho 30 phút đầu là 10.000 đồng.
- Cứ mỗi 15 phút tiếp theo, khách sẽ phải trả thêm 3.000 đồng.

Ví dụ, khách thuê 1 tiếng 10 phút cần trả $10.000 + 3 \times 3.000 = 19.000$ đồng.

1.2.3 Các loại xe đạp

Có 3 loại xe đạp:

- Xe đạp đơn thường chỉ có 1 yên/bàn đạp và 1 ghế ngồi phía sau.
- Xe đạp đơn điện giống xe đạp đơn thường nhưng có motor điện giúp đạp xe nhanh hơn.
- Xe đạp đôi thường có 2 yên/bàn đạp và 1 ghế ngồi phía sau.

Với xe đạp đơn điện và xe đạp đôi thường, khách cần trả đắt hơn cách tính giá phía trên 1,5 lần.

1.3 Từ điển thuật ngữ

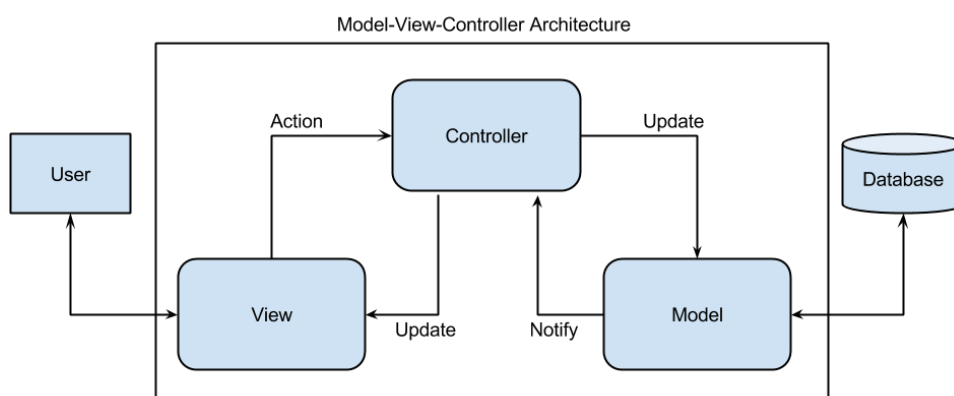
1.4 Tham khảo

Tài liệu SRS

2 Thiết kế kiến trúc

2.1 Lựa chọn kiến trúc phần mềm

Hệ thống sử dụng kiến trúc Model-View-Controller (MVC).



Kiến trúc MVC gồm 3 thành phần chính: Model, View, Controller. Trong đó, thành phần Model đảm nhiệm các chức năng liên quan đến dữ liệu, gồm có biểu diễn dữ liệu thông qua các cấu trúc dữ liệu, truy vấn, cập nhật dữ liệu. Thành phần View có trách nhiệm biểu diễn dữ liệu cũng như các logic trên giao diện người dùng. View không nhận dữ liệu trực tiếp từ Model mà thông qua Controller. Controller thực hiện các logic nghiệp vụ, các request từ người dùng, xử lý dữ liệu thông qua Model, và tương tác với View để hiển thị kết quả.

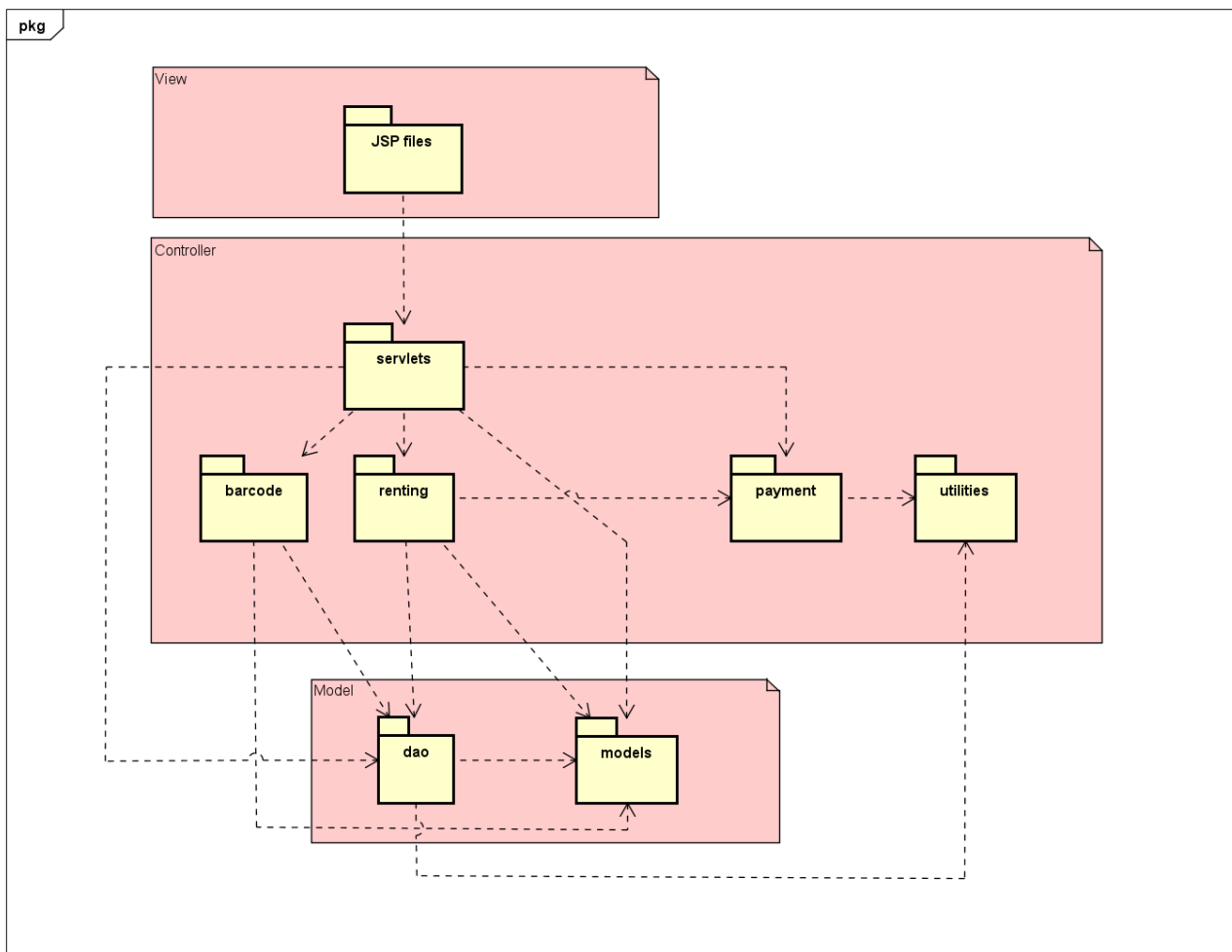
2.2 Thiết kế tổng quan

Hệ thống gồm các gói như hình vẽ bên dưới.

Thành phần Model gồm 2 gói **models** và **dao**. Gói **models** gồm các lớp ứng với các đối tượng nghiệp vụ của phần mềm, ví dụ như Bike, Dock, User. Gói **dao** gồm các lớp có chức năng thực hiện kết nối với cơ sở dữ liệu, truy vấn và cập nhật dữ liệu cho các đối tượng ứng với các lớp trong **models**.

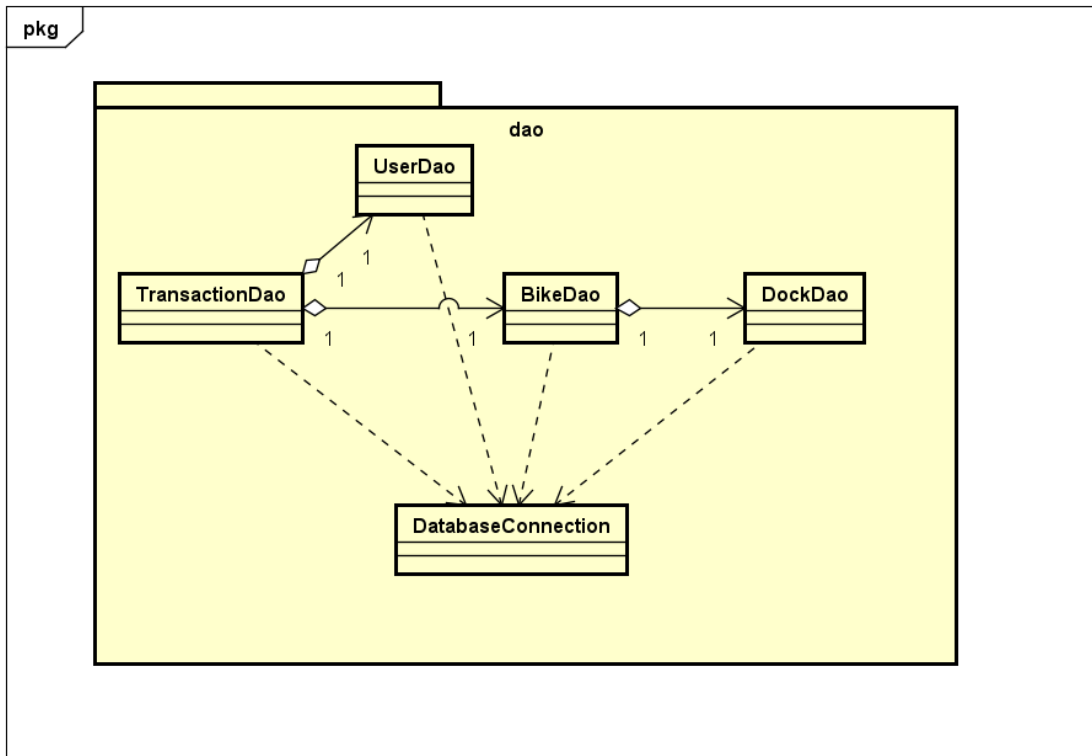
Thành phần view gồm có các file jsp. Các file jsp quy định giao diện đồ hoạ người dùng.

Thành phần controller gồm có các gói **servlets**, **barcode**, **payment**, **renting** và **utilities**. Gói **servlets** là cầu nối giữa view và các lớp điều khiển nghiệp vụ, thực hiện việc xử lý request từ giao diện thông qua các lớp nghiệp vụ, và trả về kết quả trên giao diện. Gói nghiệp vụ **renting** quản lý các nghiệp vụ chính liên quan thuê xe, ví dụ như thuê xe, trả xe. Gói **payment** có chức năng thực hiện thanh toán. Gói **barcode** thực hiện việc truy vấn thông tin xe từ barcode. Gói **utilities** phục vụ một số chức năng hệ thống.



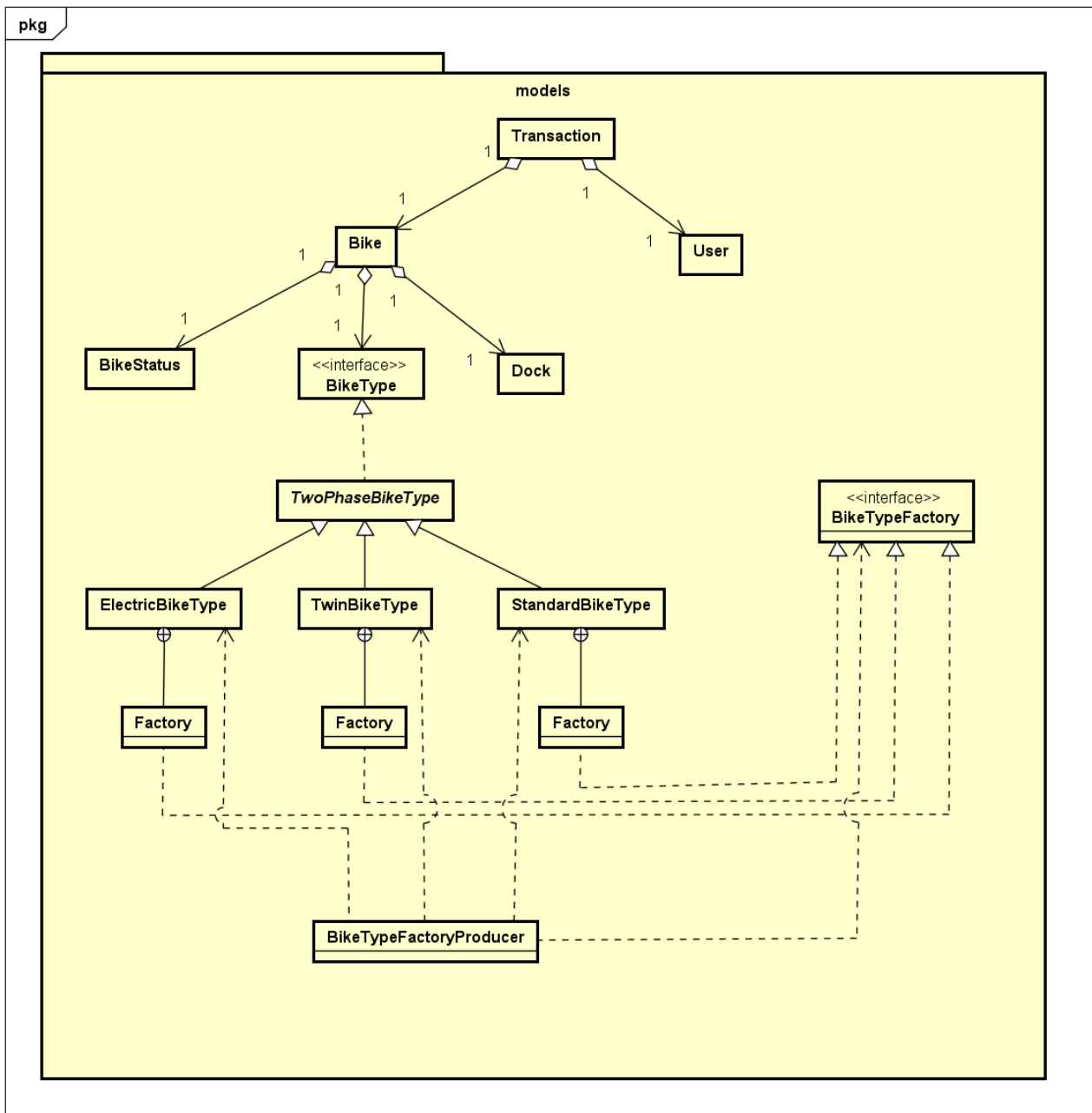
2.3 Thiết kế chi tiết gói

2.3.1 Gói dao



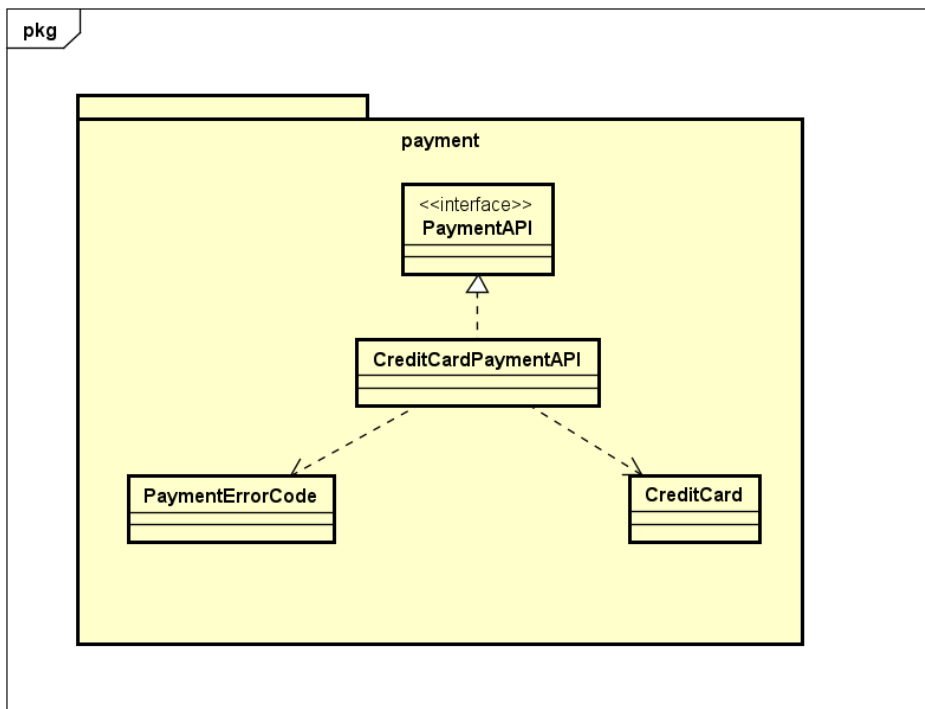
Gói **dao** gồm 5 lớp trong đó **DatabaseConnection** thiết lập kết nối với cơ sở dữ liệu, các lớp còn lại giúp truy cập, và cập nhật dữ liệu ứng với các đối tượng nghiệp vụ: xe, trạm, giao dịch, người dùng.

2.3.2 Gói models



Gói **models** gồm các lớp đại diện cho các đối tượng nghiệp vụ. Trong đó **BikeType** được thiết kế theo Strategy pattern cùng với AbstractFactory pattern để thuận tiện cho việc mở rộng thêm các loại xe trong tương lai.

2.3.3 Gói payment



Gói **payment** gồm các lớp chịu trách nhiệm thực hiện các giao dịch thanh toán giữa người dùng với phần mềm, thông qua nhiều phương thức thanh toán khác nhau.

2.3.4 Gói renting

Gồm lớp **RentingController** có chức năng thực hiện các thủ tục thuê xe và trả xe

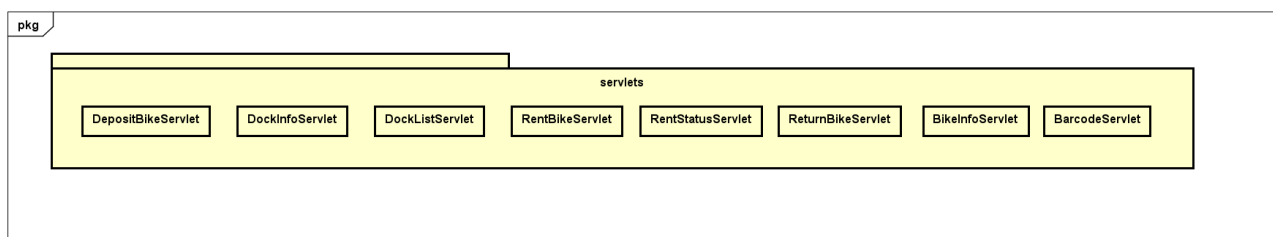
2.3.5 Gói barcode

Gồm có lớp **BarcodeController** có chức năng xác định bike mã barcode

2.3.6 Gói utilities

Gồm lớp **Config** chứa các thông tin cấu hình về môi trường phục vụ việc kết nối với cơ sở dữ liệu và API thanh toán

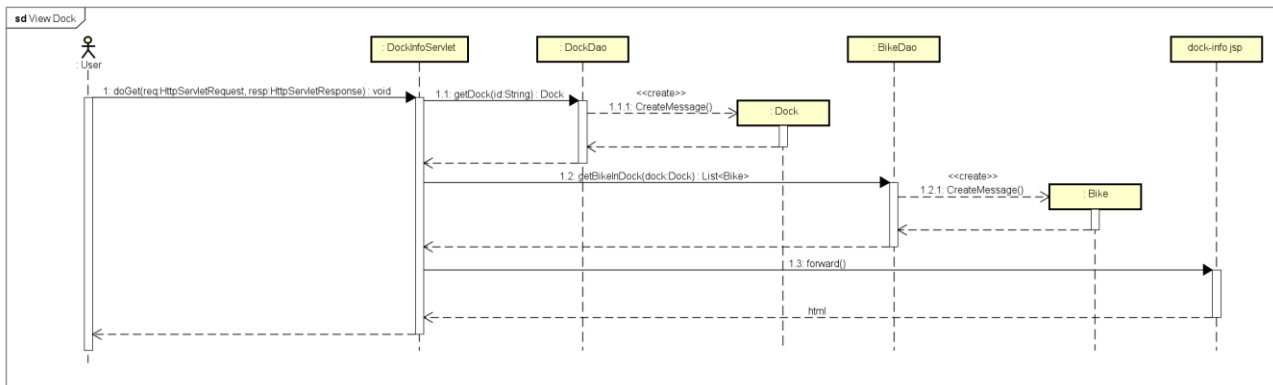
2.3.7 Gói servlets



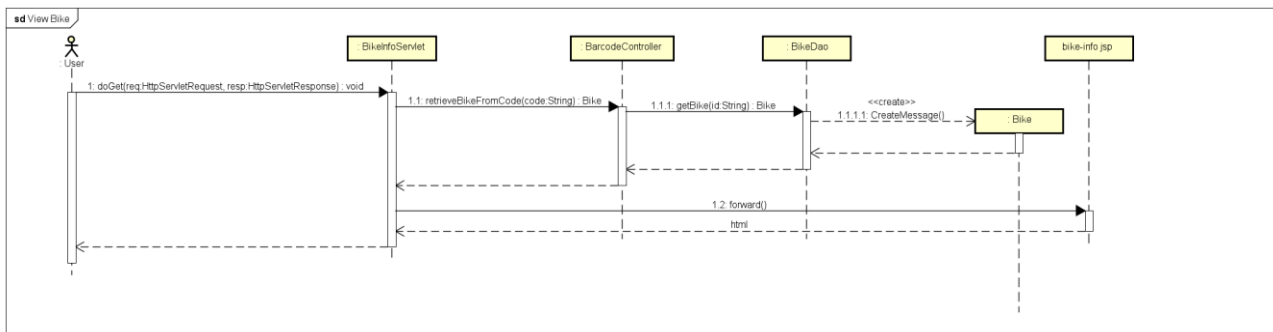
Gồm các lớp **Servlet** quản lý điều khiển các màn hình giao diện tương ứng, có chức năng xử lý request và nhận kết quả từ các lớp quản lý nghiệp vụ liên quan.

2.4 Biểu đồ tương tác

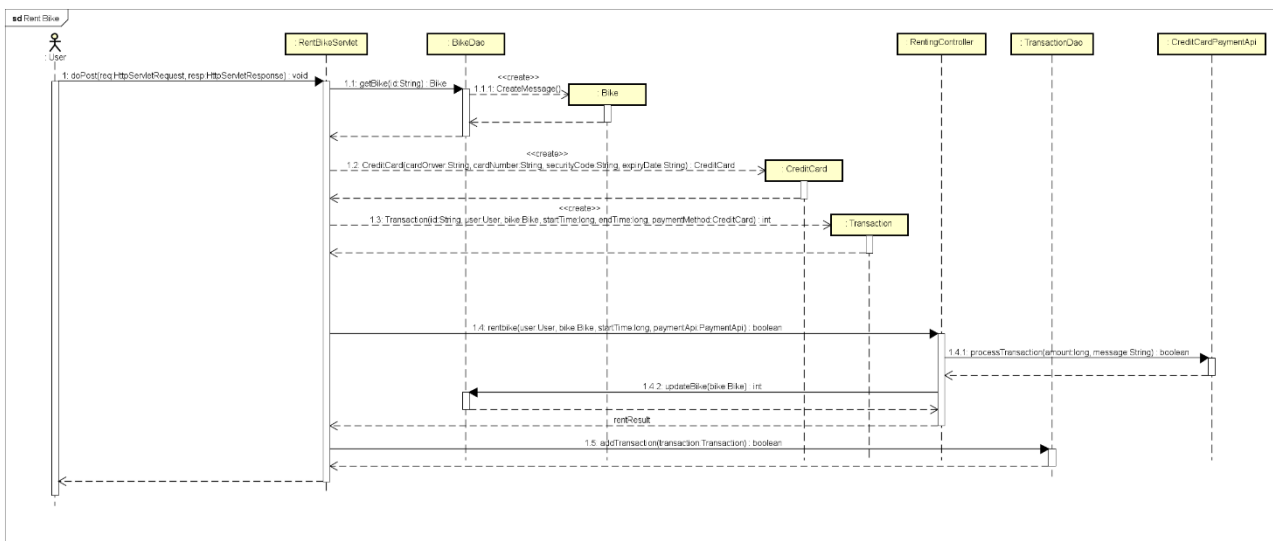
2.4.1 Biểu đồ tương tác cho UC001 – Xem thông tin bãi xe



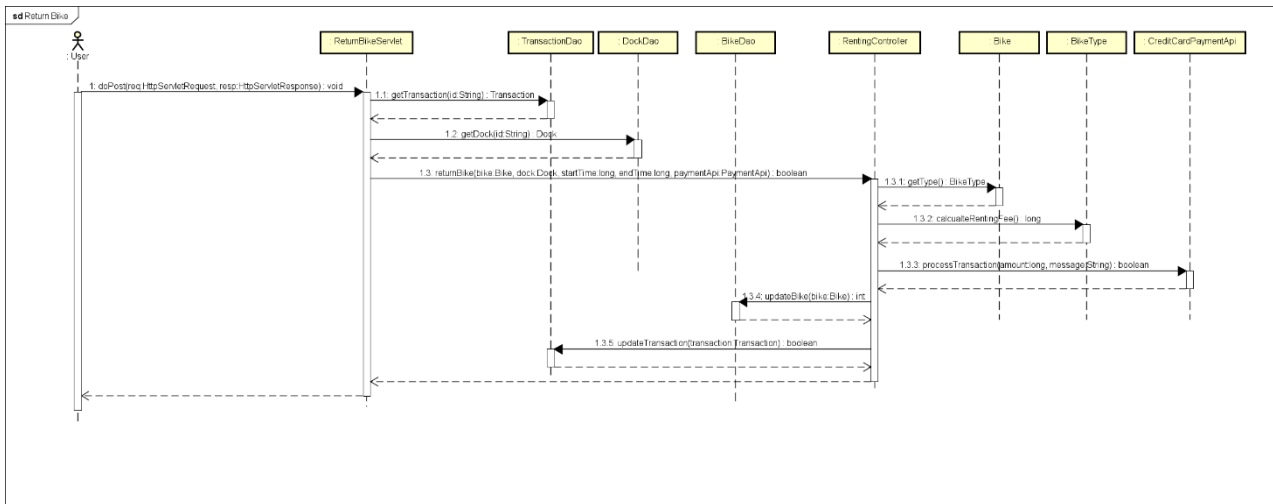
2.4.2 Biểu đồ tương tác cho UC002 - Xem thông tin xe



2.4.3 Biểu đồ tương tác cho UC003 – Thuê xe đạp

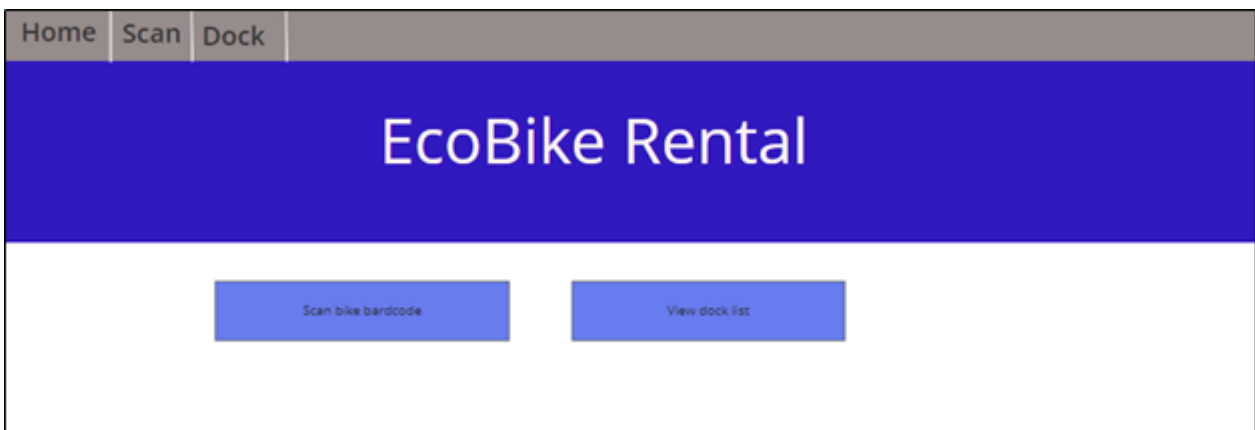


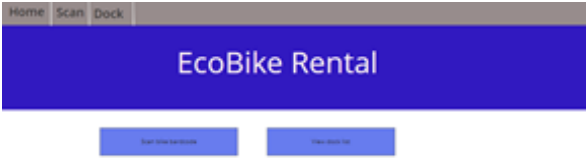
2.4.4 Biểu đồ tương tác cho UC004 - Trả xe đạp



3 Thiết kế giao diện

3.1 Giao diện trang chủ



Màn hình trang chủ			
	Điều khiển	Thao tác	Chức năng
	Nút tìm xe theo barcode	Bấm	Hiển thị giao diện tìm kiếm barcode
	Nút xem bãi xe	Bấm	Hiển thị giao diện danh sách bãi xe

--	--	--	--

3.2 Giao diện trang tìm kiếm barcode

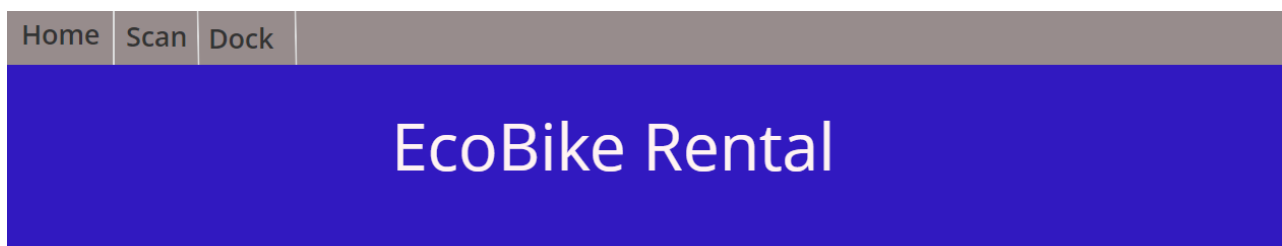
Home	Scan	Dock
------	------	------

EcoBike Rental

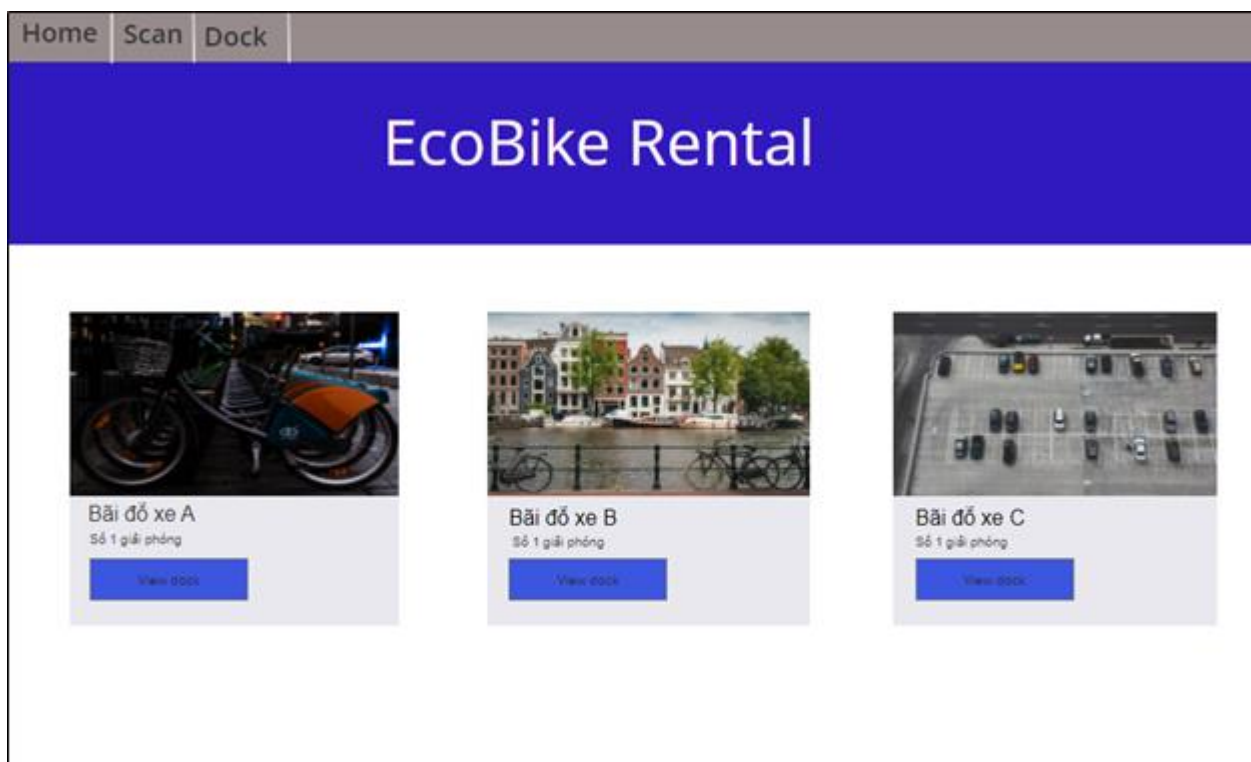
Get Bike from Barcode

<div> <div> <div> <div>Home</div> <div>Scan</div> <div>Dock</div> </div> <div style="background-color: #000080; color: white; text-align: center; padding: 10px;"> <div>EcoBike Rental</div> </div> <div> <div>Get Bike from Barcode</div> <div> <div>Bike Code</div> <div></div> <div>Get bike</div> </div> </div> </div> </div>

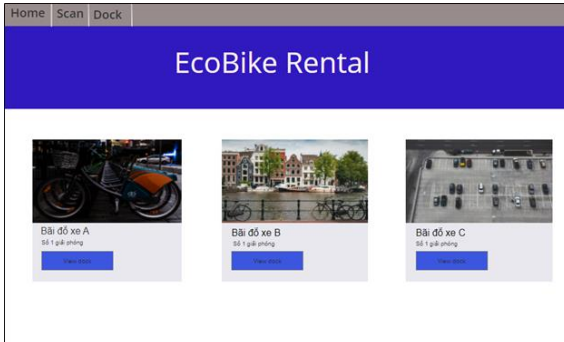
Thông báo thất bại :



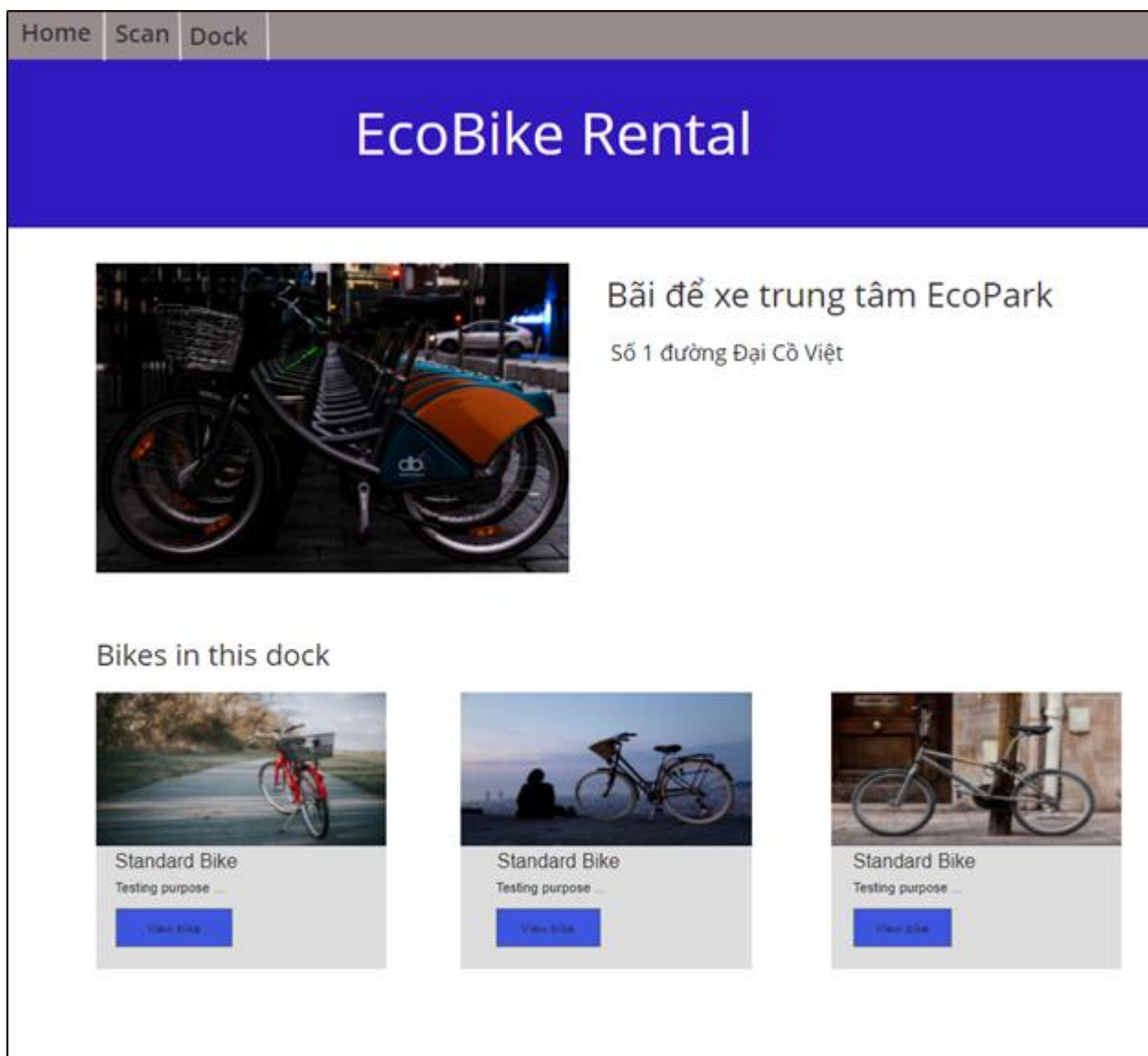
3.3 Giao diện danh sách bãi xe

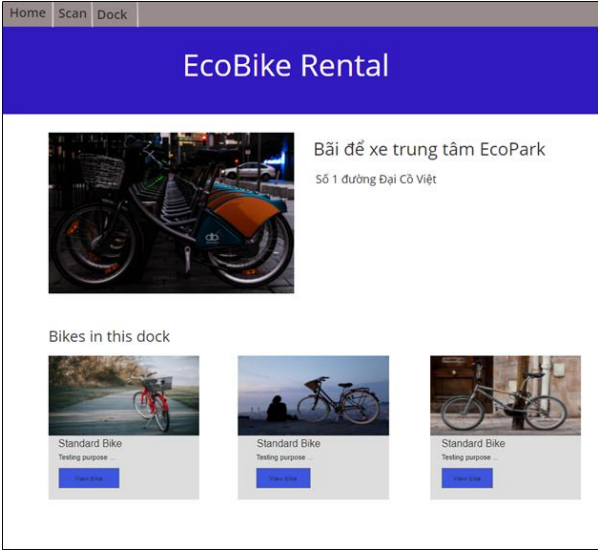


Hiện thị danh sách các bãi xe hiện có			
	Điều khiển	Thao tác	Chức năng
	Vùng hiển thị từng bãi gửi xe	Khởi tạo	+ Hiện thị ảnh bãi xe + Hiện thị tên bãi xe

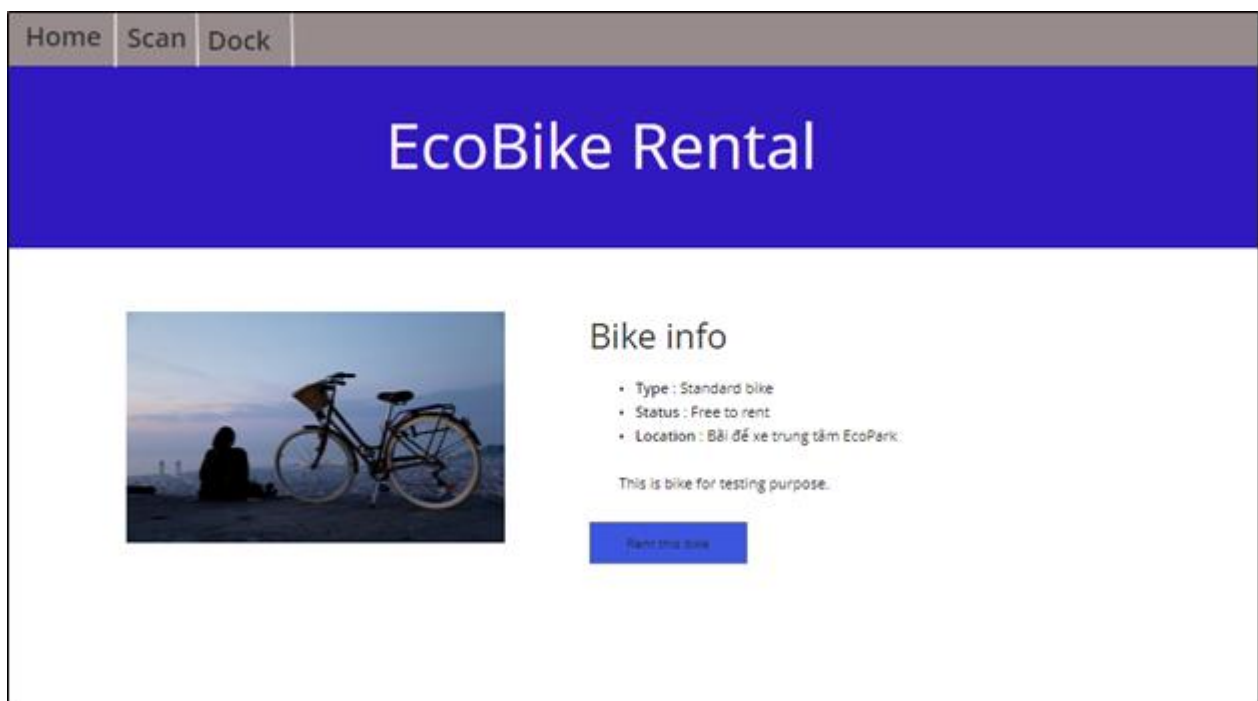
			+ Hiện thị địa chỉ bãi xe
	Nút xem bãi xe	Bấm	Hiện thị màn hình xem bãi xe

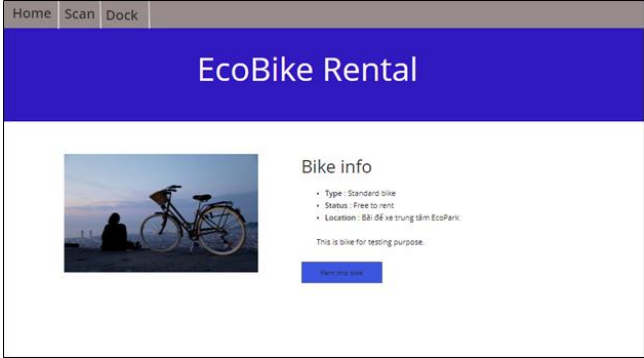
3.4 Giao diện chi tiết bãi xe



Hiển thị chi tiết về bãi xe			
	Điều khiển	Thao tác	Chức năng
	Vùng hiển thị thông tin bãi xe	Khởi tạo	+ Hiển thị ảnh bãi xe + Hiển thị tên bãi xe + Hiển thị địa chỉ bãi xe
	Vùng hiển thị các xe trong bãi	Khởi tạo	+ Hiển thị danh sách các bãi xe + Mỗi thành phần bao gồm ảnh, tên xe, mô tả
	Nút xem chi tiết xe	Bấm	Hiển thị màn hình xem chi tiết xe

3.5 Giao diện chi tiết xe




Hiển thị chi tiết xe			
	Điều khiển	Thao tác	Chức năng
	Vùng hiển thị thông tin xe	Khởi tạo	Hiển thị thông tin xe về loại xe, trạng thái, địa chỉ
	Nút thuê xe	Bấm	+ Hiển thị giao diện thuê xe

3.6 Giao diện thanh toán

[Home](#) [Scan](#) [Dock](#)

EcoBike Rental



Deposit bike

- Deposit : 400000 VND
- Renting fee :
 - + First 30 minutes : 10000 VND
 - + Every 15 minutes afterwards : 3000 VND

This different between the deposit and the eventual renting fee will be returned to the user's account renting the bike.

Payment information

Name on card

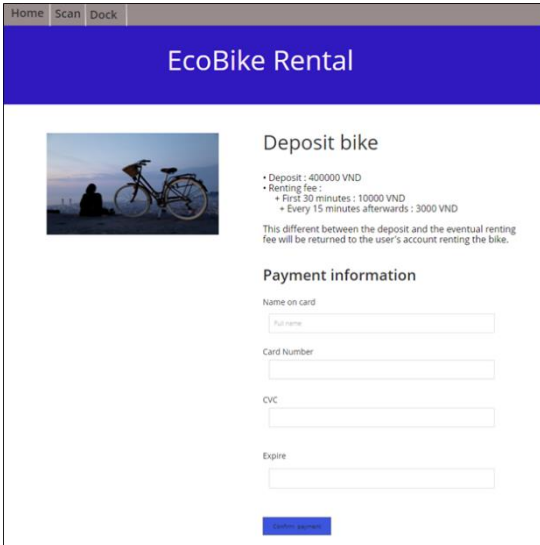
Card Number

CVC

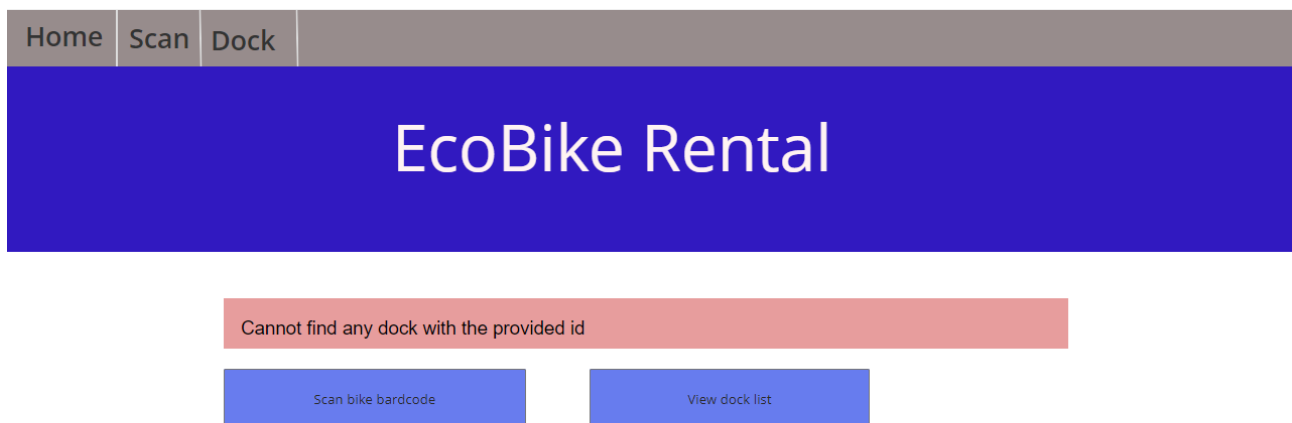
Expire

[Confirm payment](#)

Hiện thị thông tin giá cả và form thanh toán			
	Điều khiển	Thao tác	Chức năng
	Vùng hiển thị giá	Khởi tạo	Hiện thị thông tin giá thuê xe

	Vùng điền thông tin người thuê	Nhập thông tin	+ Các vùng để người dùng nhập thông tin về tên, thẻ, ...
	Nút thuê xe	Bấm	Xác nhận thuê xe, chuyển sang màn hình trả xe nếu thành công, không thì hiển thị thông báo thất bại

Thông báo thất bại :




3.7 Giao diện trả xe

Home

Scan

Dock

EcoBike Rental



Renting bike

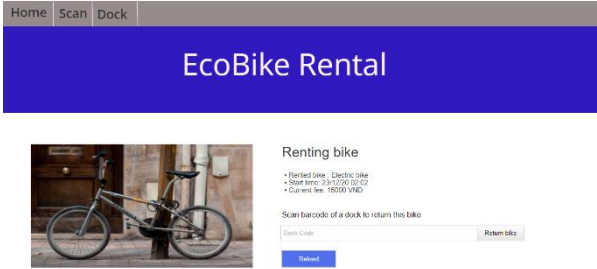
- Rented bike : Electric bike
- Start time: 23/12/20 02:02
- Current fee: 15000 VND

Scan barcode of a dock to return this bike

Dock Code

Return bike

Reload

Hiển thị thông tin giá cả và form thanh toán			
	Điều khiển	Thao tác	Chức năng
	Vùng hiển thị thông tin thuê xe	Khởi tạo	Hiển thị thông tin thuê xe bao gồm tên xe, thời gian bắt đầu, giá hiện tại
	Vùng nhập dock code để trả xe	Nhập	+ Các vùng để người dùng nhập thông tin về tên, thẻ, ...
	Nút trả xe	Bấm	Xác nhận trả xe vào dock, trở về giao diện màn hình nếu tìm thấy dock

Thông báo trả thành công :

Bike returned successfully

Scan bike bardcode

View dock list

Thông báo lỗi :

Cannot find any dock with the provided id

Scan bike bardcode

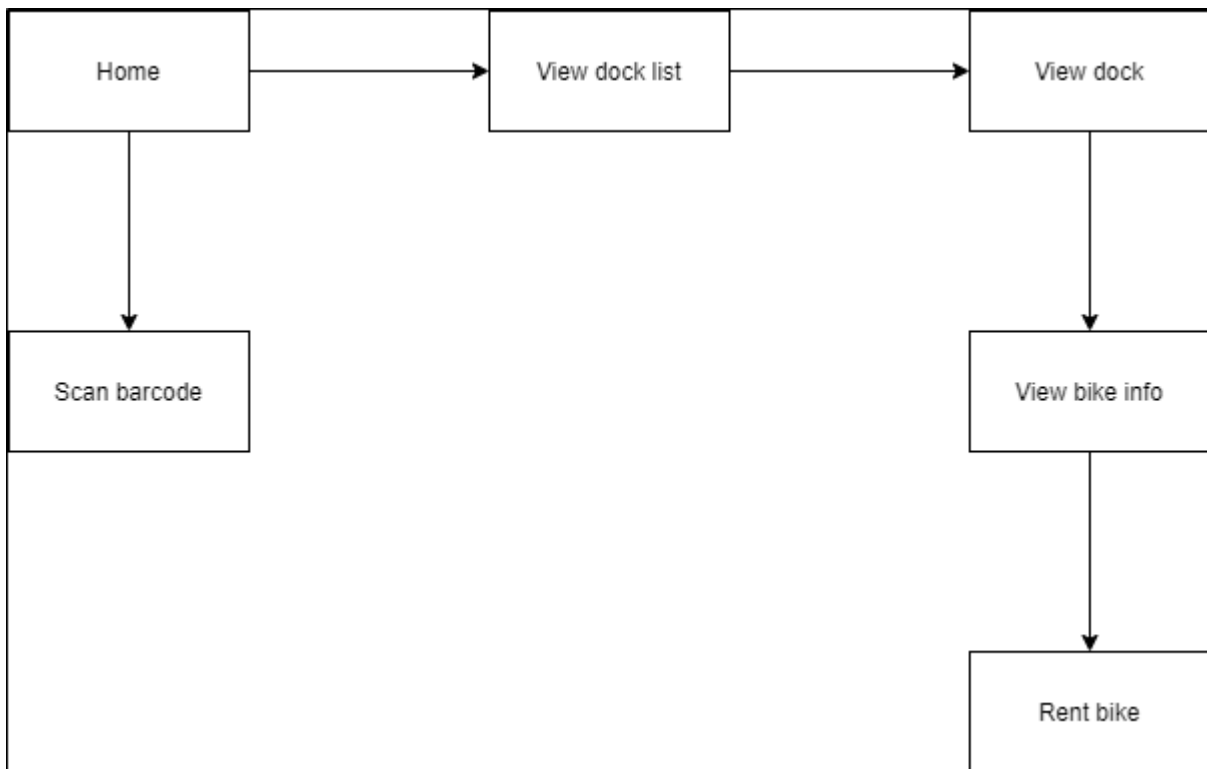
View dock list

Transaction already ended

Scan bike bardcode

View dock list

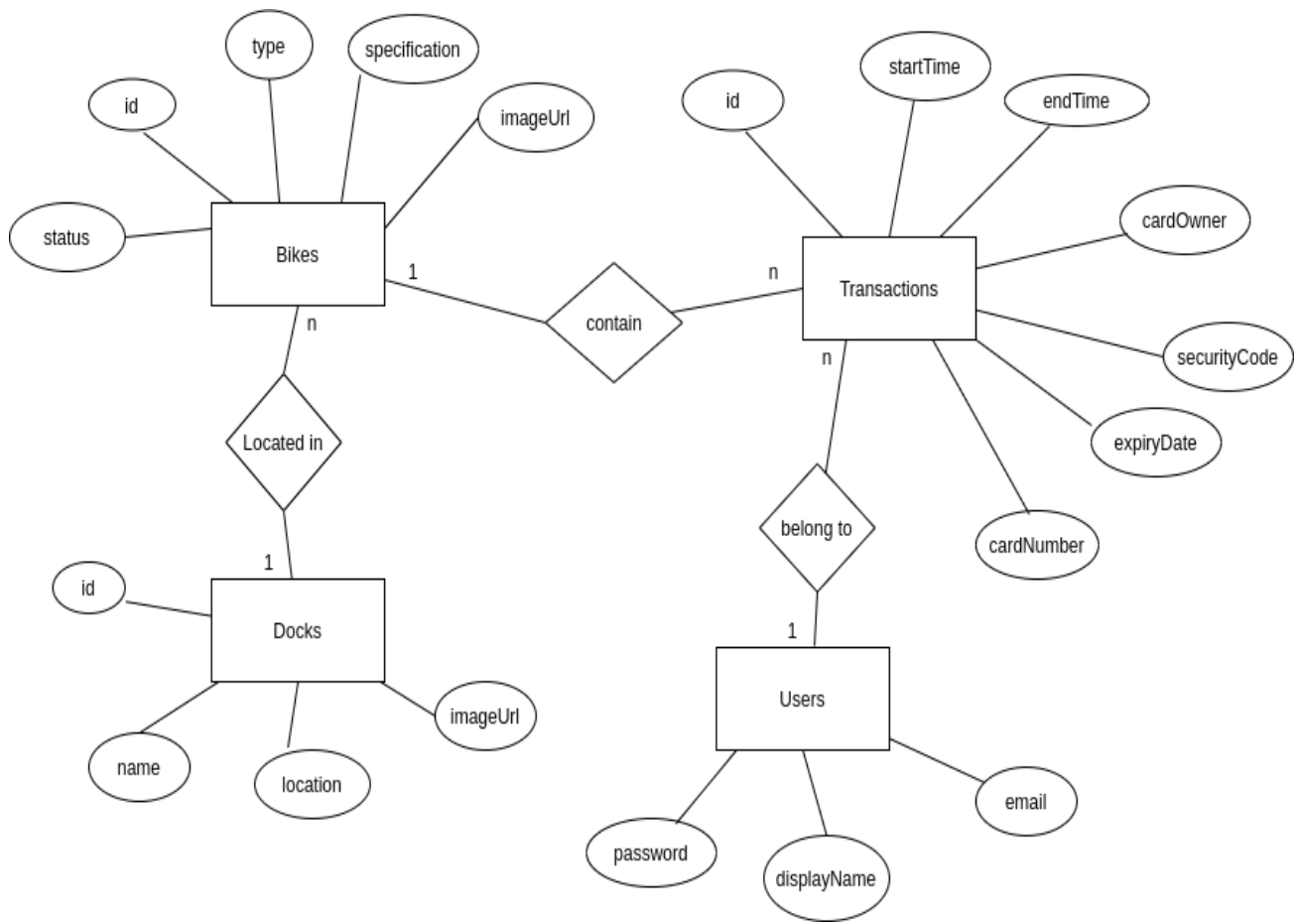
3.8 Sơ đồ chuyển màn hình



4 Thiết kế mô hình dữ liệu

4.1 Mô hình dữ liệu mức khái niệm

Sơ đồ thực thể liên kết:



4.2 Mô hình dữ liệu mức logic

Bước 1: Trích xuất entity

Sau khi trích xuất các entity nhóm quyết định có 4 bảng trong CSDL quan hệ là Docks, Bikes, Users và Transactions.

Bảng Docks chứa các thông tin về bãi xe như tên, vị trí, ảnh bãi xe

Bảng Bikes chứa các thông tin về xe như loại xe, mô tả chi tiết, trạng thái xe, vị trí và ảnh đại diện

Bảng Users chứa các thông tin về người dùng như email, tên đăng nhập và mật khẩu

Bảng Transactions chứa các thông tin về giao dịch như người thực hiện, xe nào, thời gian mượn, thời gian trả, thông tin về thẻ của khách hàng

Bước 2: Thiết lập khóa chính

Các bảng Docks, Bikes và Transactions có thêm trường ID là giá trị duy nhất

Bảng Users sử dụng email làm khóa chính

Bước 3: Thiết lập các mối quan hệ giữa các bảng

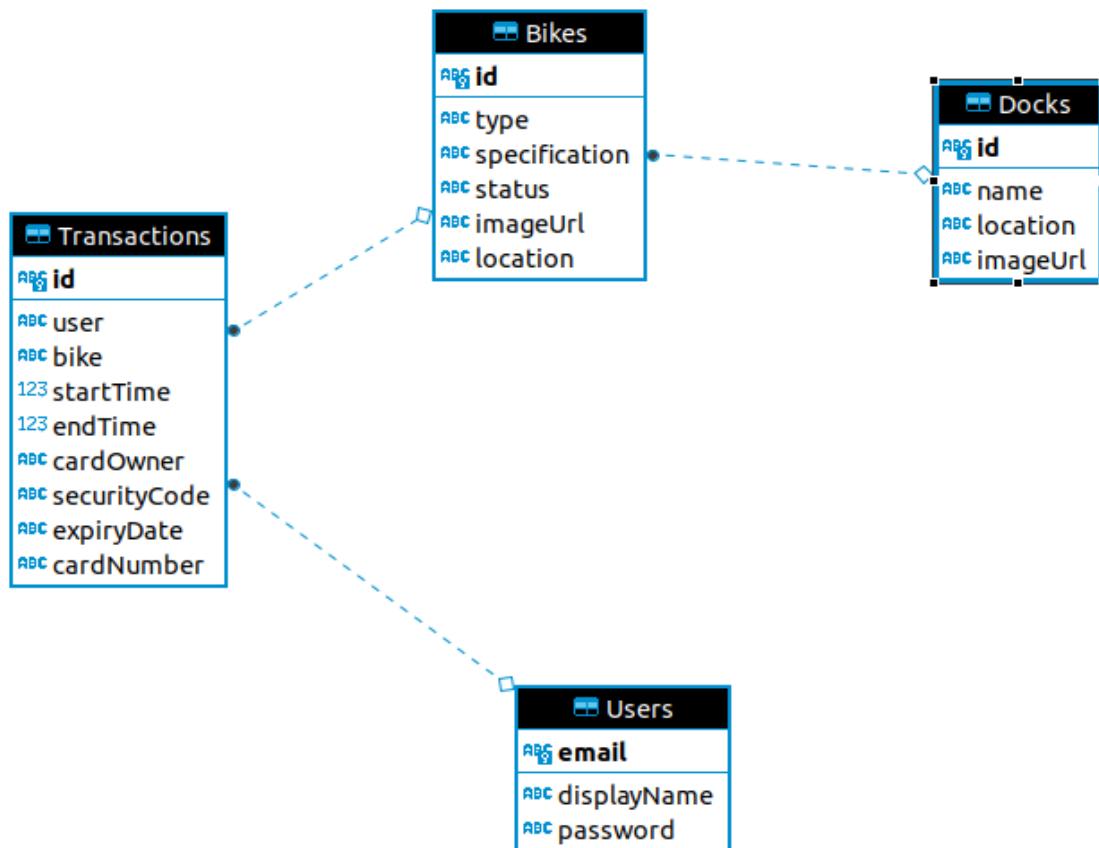
Bảng Bikes có trường location thể hiện vị trí xe nằm ở bãi nào nên nó sẽ là khóa ngoại liên kết đến bảng Docks (1 xe tại mỗi thời điểm chỉ có thể nằm ở 1 bãi xe hoặc không bãi nào)

Bảng Transactions chứa thông tin người sở hữu và xe nên xe có 2 khóa ngoài liên kết đến Bikes và Users. Mỗi Transaction liên kết đến 1 Bike và 1 User

Bước 3: Chuẩn hóa về dạng chuẩn 3

Kiểm tra tính lặp, tính dư thừa và các quan hệ giữa các bảng để chuẩn hóa. Kết quả cuối cùng được thể hiện như hình dưới.

Nhóm sử dụng PostgreSQL để lưu trữ dữ liệu vì đây không chỉ là cơ sở dữ liệu quan hệ mà còn là quan hệ hướng đối tượng, hỗ trợ rất tốt trong việc tự định nghĩa các đối tượng và các hành vi của chúng bao gồm các kiểu dữ liệu, các hàm, các thao tác, các tên miền và các chỉ mục.



4.3 Thiết kế chi tiết

4.3.1 Docks

Tên cột	Kiểu dữ liệu	Khóa chính	Khóa ngoại	Duy nhất	Ràng buộc	Mô tả
id	text	X		X	Not null	id bãi xe
name	text				Not null	Tên bãi xe
imageURL	text					Link ảnh
location	text				Not null	Nằm ở đâu trên bản đồ

4.3.2 Bikes

Tên cột	Kiểu dữ liệu	Khóa chính	Khóa ngoại	Duy nhất	Ràng buộc	Mô tả
id	text	X		X	Not null	id xe
type	text				Not null	Kiểu xe (Xe đạp, xe đạp điện)
specification	text					Mô tả về xe
status	text				Not null	Trạng thái xe (Đang mượn hay không)
imageUrl	text					Link ảnh
location	text		x		Not null	Nằm ở bãi xe nào

4.3.3 User

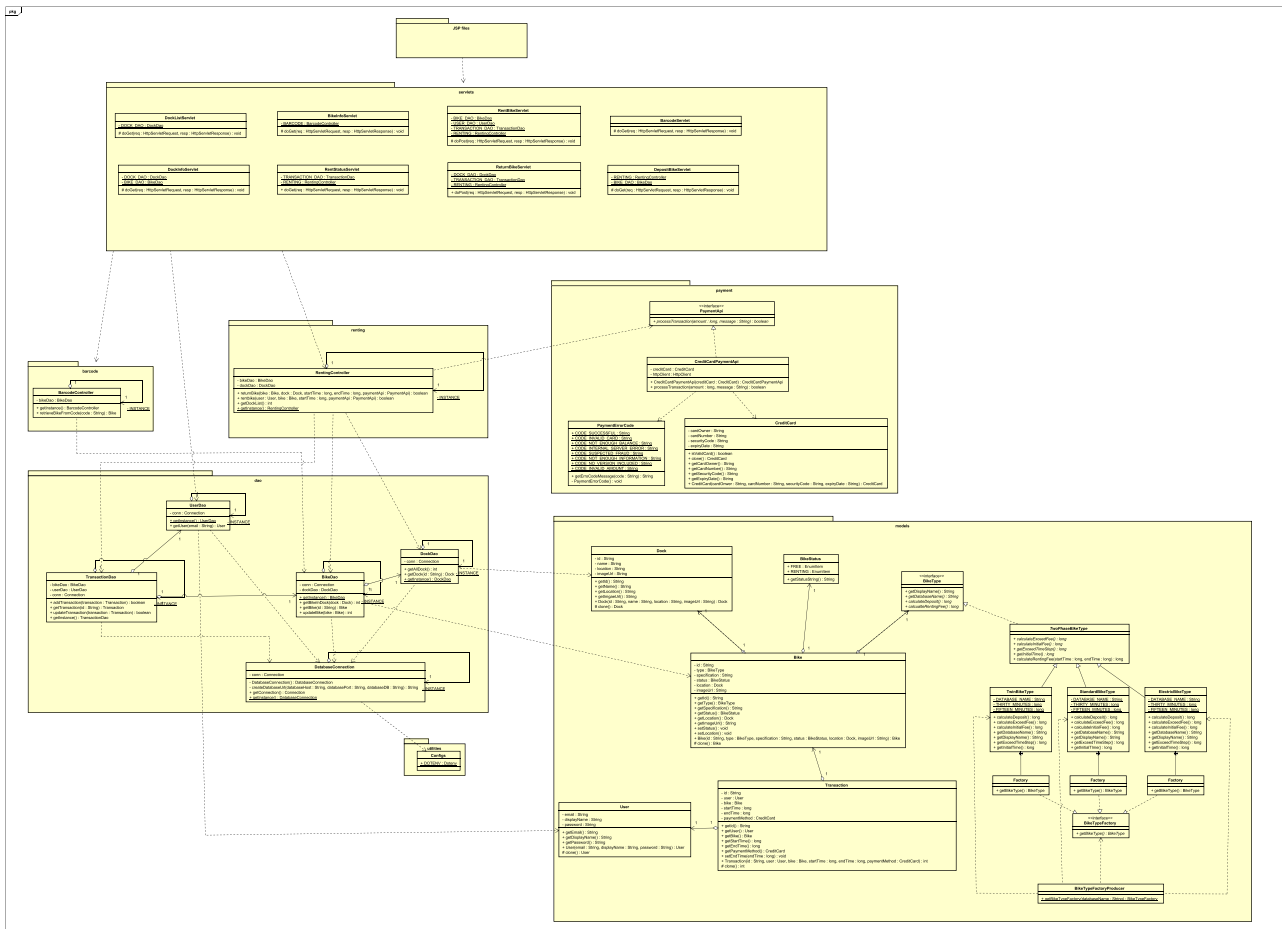
Tên cột	Kiểu dữ liệu	Khóa chính	Khóa ngoại	Duy nhất	Ràng buộc	Mô tả
email	text	X		X	Not null	Email người dùng
displayName	text				Not null	Tên hiển thị khi đăng nhập
password	text				Not null	Mật khẩu

4.3.4 Transactions

Tên cột	Kiểu dữ liệu	Khóa chính	Khóa ngoại	Duy nhất	Ràng buộc	Mô tả
id	text	X		X	Not null	id giao dịch
user	text		X		Not null	Người sở hữu
bike	text		X		Not null	Xe mượn
startTime	bigint				Not null	Thời điểm bắt đầu mượn xe
endTime	bigint				Not null	Thời điểm trả xe
cardOwner	text				Not null	Thông tin thẻ người sở hữu
securityCode	text				Not null	Mã bảo mật
expireDate	text				Not null	Hạn dùng thẻ

5 Thiết kế lớp

5.1 Biểu đồ lớp thiết kế

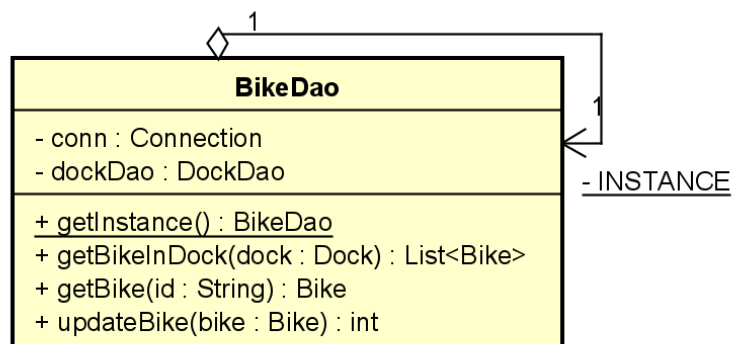


5.2 Thiết kế lớp chi tiết

5.2.1 Thiết kế lớp thuộc gói dao

5.2.1.1 Lớp BikeDao

Lớp BikeDao có chức năng đọc, ghi dữ liệu cho object thuộc lớp Bike từ cơ sở dữ liệu



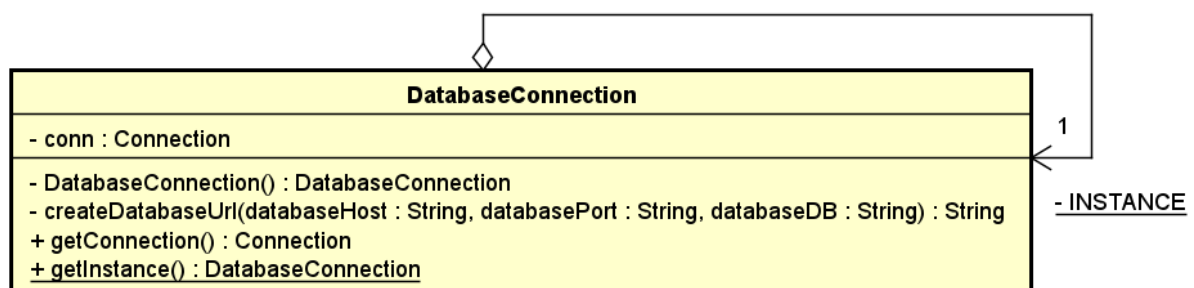
Attribute:

#	Tên	Kiểu dữ liệu	Mô tả
1	INSTANCE	BikeDao	Tham chiếu đến instance duy nhất của class (singleton)
2	conn	Connection	Kết nối với cơ sở dữ liệu
3	dockDao	DockDao	Đối tượng giúp đọc ghi dữ liệu bãi xe từ cơ sở dữ liệu

Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	getInstance	không	BikeDao	Lấy 1 đối tượng của lớp	Không
2	getBike	<ul style="list-style-type: none"> id (String): id của xe 	Bike	Lấy thông tin xe theo id từ cơ sở dữ liệu	SQLException
3	getBikeInDock	<ul style="list-style-type: none"> dock (Dock): đối tượng bãi xe 	List<Bike>	Lấy thông tin tất cả xe trong bãi xe	SQLException
4	updateBike	<ul style="list-style-type: none"> bike (Bike): đối tượng xe 	int	Ghi thông tin xe vào cơ sở dữ liệu	SQLException

5.2.1.2 Lớp DatabaseConnection



Lớp có trách nhiệm kết nối với cơ sở dữ liệu, và được thiết kế dạng Singleton

Attribute:

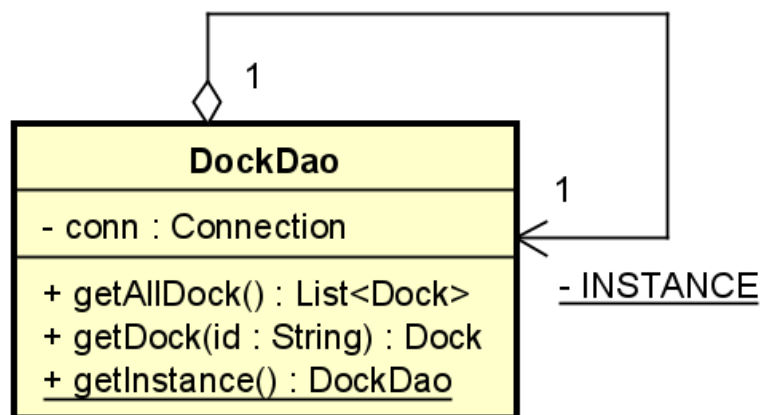
#	Tên	Kiểu dữ liệu	Mô tả
1	INSTANCE	DatabaseConnection	Tham chiếu đến instance duy nhất của class (singleton)
2	conn	Connection	Kết nối với cơ sở dữ liệu

Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	getInstance	không	DatabaseConnection	Lấy 1 đối tượng của lớp	Không
2	createDatabaseUrl	<ul style="list-style-type: none"> DatabaseHost (String): tên host server của cơ sở dữ liệu DatabasePort (String): số port của csdl DatabaseDB: tên csdl 	String	Tạo đường dẫn giúp kết nối csdl	Không
3	getConnection	không	Connection	Lấy 1 đối tượng Connection	không

5.2.1.3 Lớp DockDao

Lớp DockDao có chức năng đọc, ghi dữ liệu cho object thuộc lớp Dock từ cơ sở dữ liệu



Attribute:

#	Tên	Kiểu dữ liệu	Mô tả
1	INSTANCE	DockDao	Tham chiếu đến instance duy nhất của class (singleton)

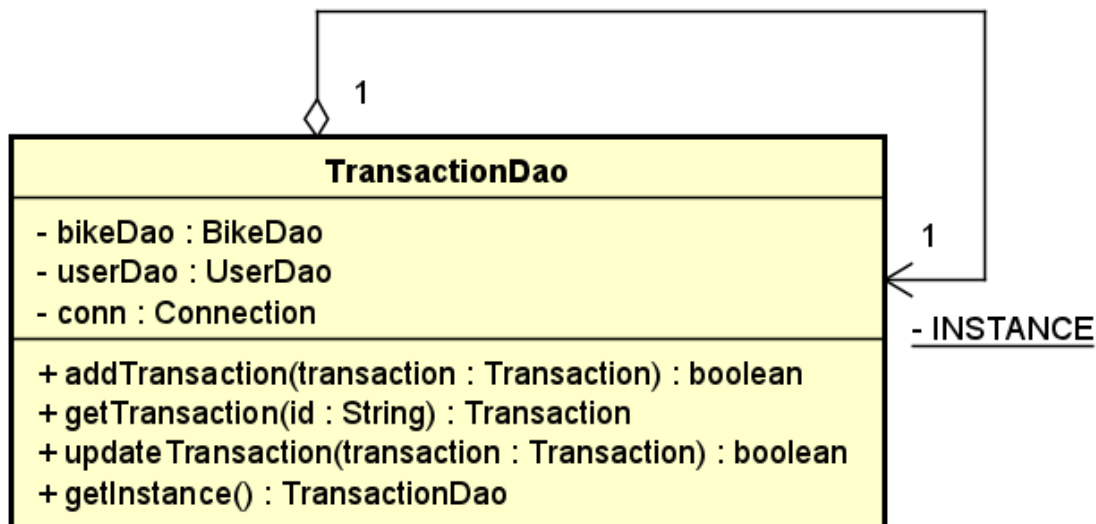
2	conn	Connection	Kết nối với cơ sở dữ liệu
---	------	------------	---------------------------

Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	getInstance	không	BikeDao	Lấy 1 đối tượng của lớp	Không
2	getDock	<ul style="list-style-type: none"> id (String): id của dock 	Dock	Lấy thông tin bãi xe theo id từ cơ sở dữ liệu	SQLException
3	getAllDock	không	List<Dock>	Lấy thông tin tất cả bãi xe	SQLException

5.2.1.4 Lớp TransactionDao

Lớp TransactionDao có chức năng đọc, ghi dữ liệu cho object thuộc lớp Transaction từ cơ sở dữ liệu



Attribute:

#	Tên	Kiểu dữ liệu	Mô tả
1	INSTANCE	BikeDao	Tham chiếu đến instance duy nhất của class (singleton)
2	conn	Connection	Kết nối với cơ sở dữ liệu

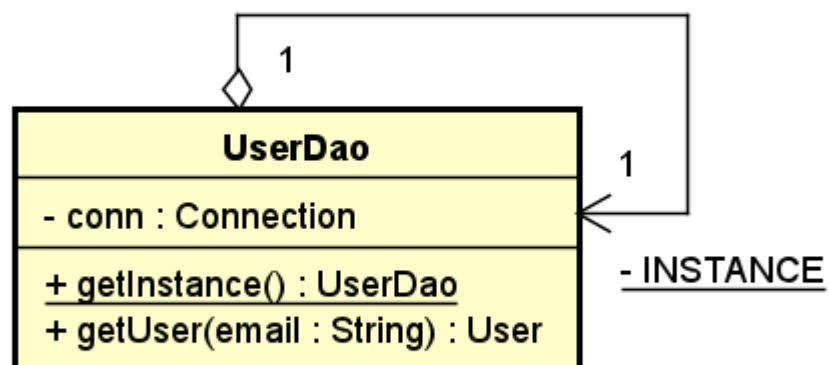
3	userDao	UserDao	Đối tượng giúp đọc ghi dữ liệu người dùng ứng với giao dịch
4	bikeDao	BikeDao	Đối tượng giúp đọc ghi dữ liệu xe ứng với giao dịch

Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	getInstance	không	TransactionDao	Lấy 1 đối tượng của lớp	Không
2	getTransaction	<ul style="list-style-type: none"> id (String): id của giao dịch 	Transaction	Lấy thông tin giao dịch theo id từ cơ sở dữ liệu	SQLException
3	addTransaction	<ul style="list-style-type: none"> transaction (Transaction): đối tượng giao dịch 	boolean	Thêm giao dịch vào csdl	SQLException
4	updateTransaction	<ul style="list-style-type: none"> transaction (Transaction): đối tượng giao dịch 	int	cập nhật csdl	SQLException

5.2.1.5 Lớp UserDao

Lớp UserDao có chức năng đọc, ghi dữ liệu cho object thuộc lớp User từ cơ sở dữ liệu



Attribute:

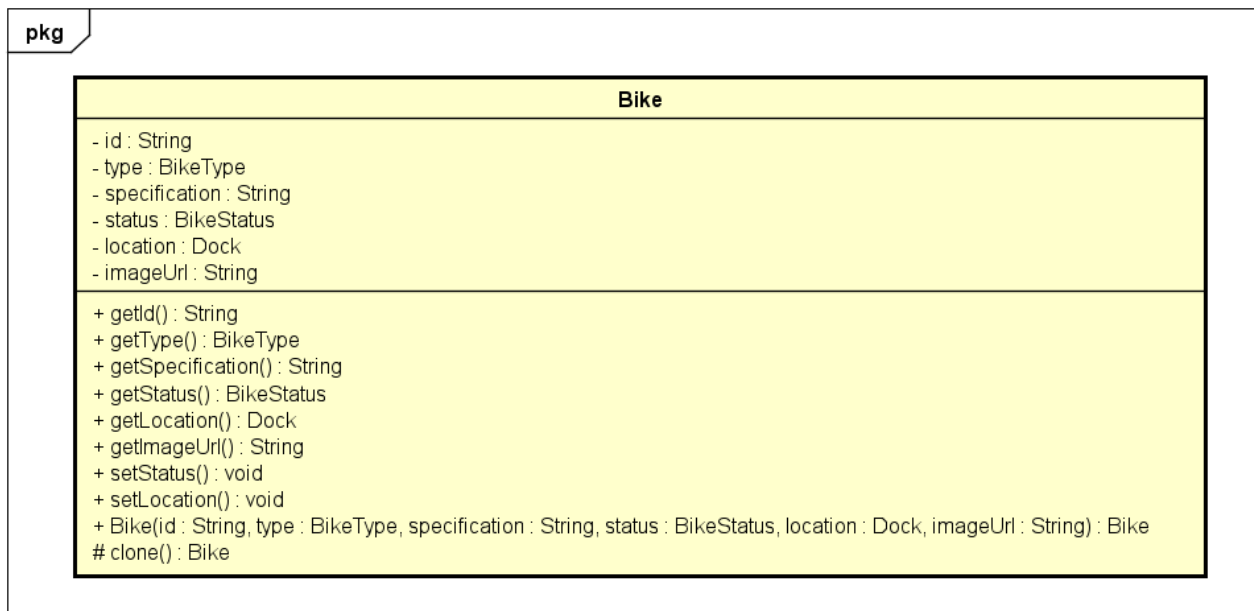
#	Tên	Kiểu dữ liệu	Mô tả
1	INSTANCE	UserDao	Tham chiếu đến instance duy nhất của class (singleton)
2	conn	Connection	Kết nối với cơ sở dữ liệu

Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	getInstance	không	UserDao	Lấy 1 đối tượng của lớp	Không
2	getUser	<ul style="list-style-type: none"> email (String): email người dùng 	User	Lấy thông tin người dùng theo email từ cơ sở dữ liệu	SQLException

5.2.2 Thiết kế lớp thuộc gói model

5.2.2.1 Lớp Dock



Attribute:

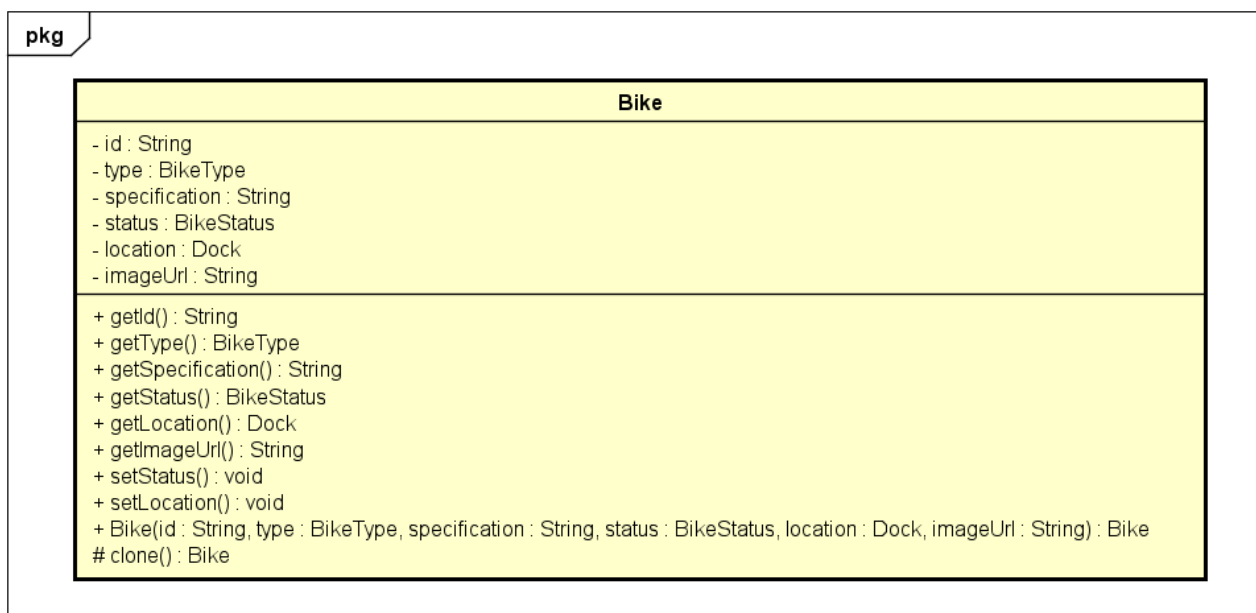
#	Tên	Kiểu dữ liệu	Giá trị mặc định	Mô tả
1	id	String	Null	Id of the dock
2	name	String	Null	Name of the dock

3	location	String	Null	Location of the dock
4	imageUrl	String	Null	Image of the dock

Method:

#	Tên	Kiểu dữ liệu trả về	Tham số	Mô tả	Exception
1	getId	String	No	Get the Dock's id	No
2	getName	String	No	Get the Dock's name	No
3	getLocation	String	No	Get the Dock's location	No
4	getImageUrl	String	No	Get the Dock's image	No
5	Dock	Dock	id: String, name:String, location:String, imageUrl: String	Constructor a Dock	No
6	clone	Dock	No	Clone a Dock	No

5.2.2.2 Lớp Bike



Attribute:

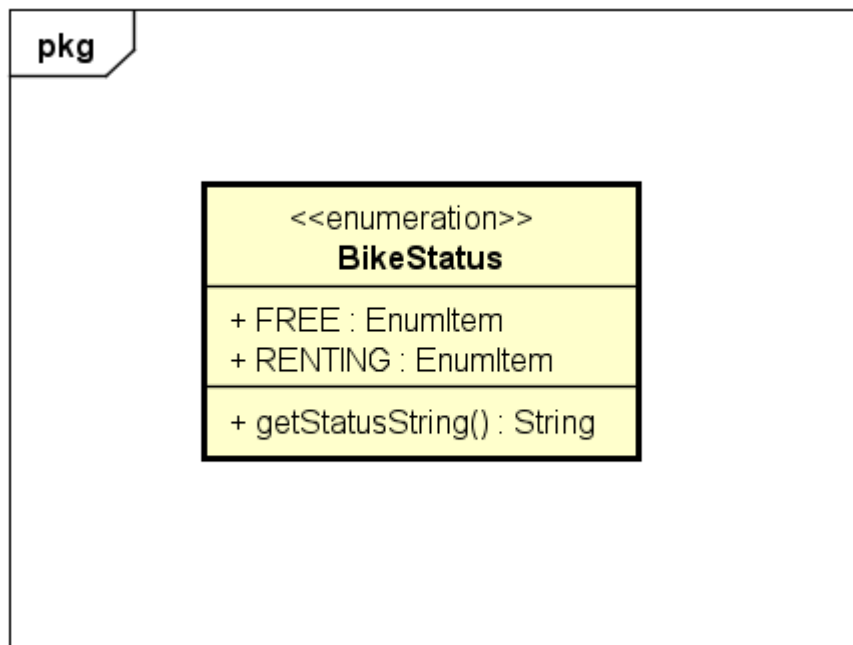
#	Tên	Kiểu dữ liệu	Giá trị mặc định	Mô tả
1	id	String	Null	Id of the bike
2	type	BikeType	Null	Type of the dock: standard, Twin or Electric

3	specification	String	Null	Specification of the bike, is various unstructured information of the bike likes: brand, special condition, promotion, ...
4	status	BikeStatus	Null	Status of the bike, whether this bike is being rented or not
5	location	Dock	Null	the Dock this bike is being docked at
6	imageUrl	String	Null	Image of the bike

Method:

#	Tên	Kiểu dữ liệu trả về	Tham số	Mô tả	Exception
1	getId	String	No	Get the bike's id	No
2	getType	BikeType	No	Get the bike's type	No
3	getSpecification	String	No	Get the bike's specification	No
4	getStatus	BikeStatus	No	Get the bike's status	No
5	getLocation	Dock	No	Get the bike's location	No
6	getImageUrl	String	No	Get the bike's image	No
7	setStatus	void	status: BikeStatus	Set the bike's status	No
8	setLocation	void	location: Dock	Set the bike's location	No
9	Bike	Bike	id: String, type: BikeType, specification: String, status: BikeStatus, location: Dock, imageUrl: String	Constructor a Bike	No
9	clone	Bike	No	Clone a Bike	No

5.2.2.3 Lớp BikeStatus



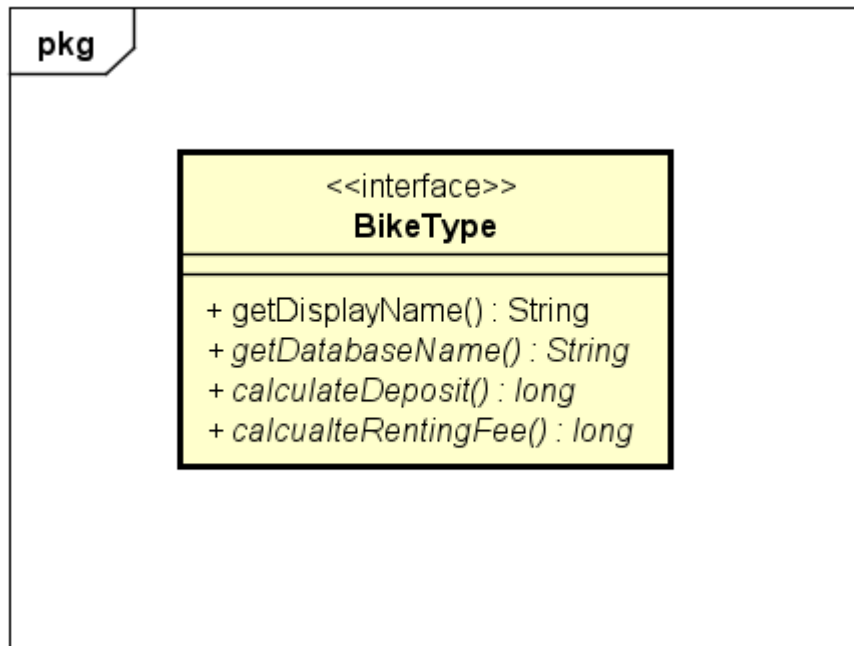
Attribute:

#	Tên	Kiểu dữ liệu	Giá trị mặc định	Mô tả
1	FREE	Enum	FREE	Free to rent
2	RENTING	Enum	RENTING	Beign rented

Method:

#	Tên	Kiểu dữ liệu trả về	Tham số	Mô tả	Exception
1	getStatusString	String	No	Get status of this bike	No

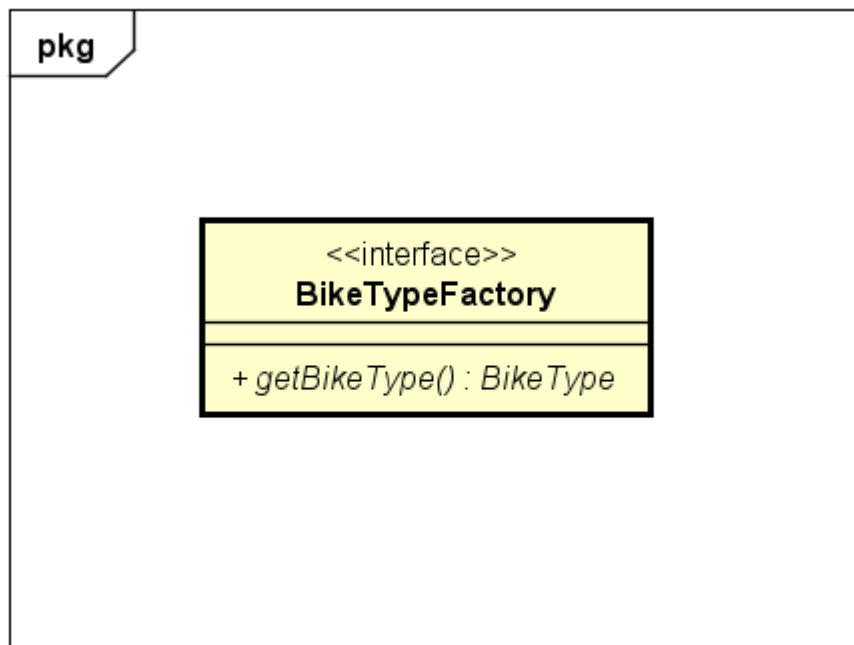
5.2.2.4 Lớp BikeType



Method:

#	Tên	Kiểu dữ liệu trả về	Tham số	Mô tả	Exception
1	getDisplayName	String	No	Return the human-readable name of this bike type, used to display in the UI	No
2	getDatabaseName	String	No	Return a non-whitespace safe ASCII string, used to determine the bike type in the database	No
3	calculateDeposit	long	No	Return the amount of money needed to be deposited before the user can start renting bike	No
4	calculateRentingFee	long	<ul style="list-style-type: none"> StartTime: long – the start time of the rent EndTime: long – the end time of the rent 	Return the amount of fee needed to rent the bike for a given time period	No

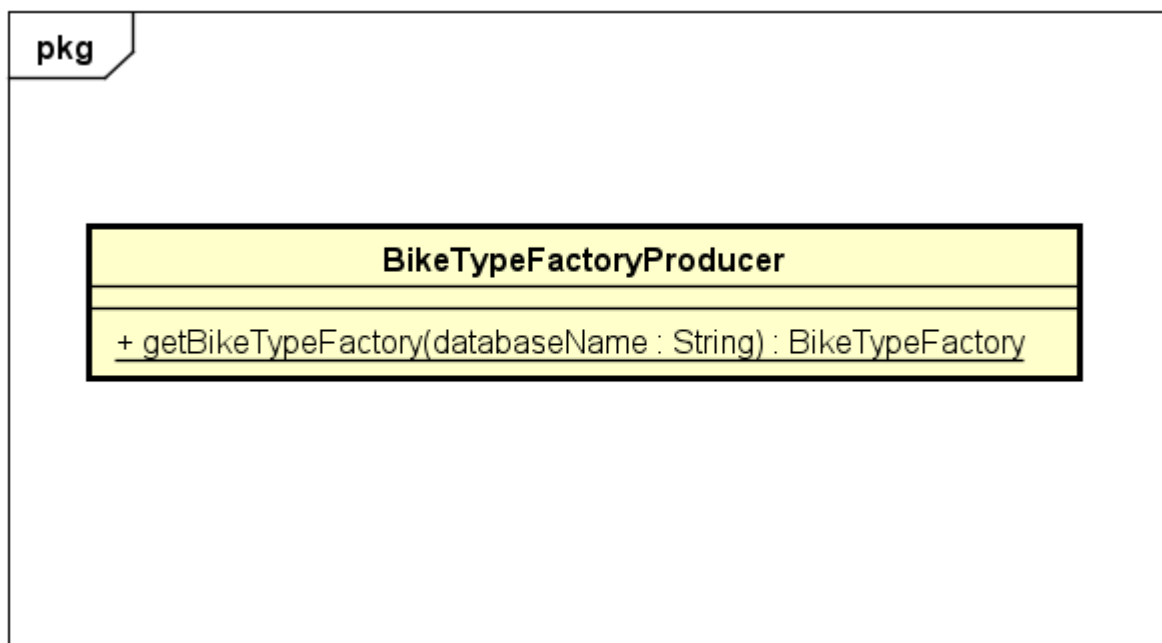
5.2.2.5 Lớp BikeTypeFactory



Method:

#	Tên	Kiểu dữ liệu trả về	Tham số	Mô tả	Exception
1	getBikeType	BikeType	No	Construct a BikeType object	No

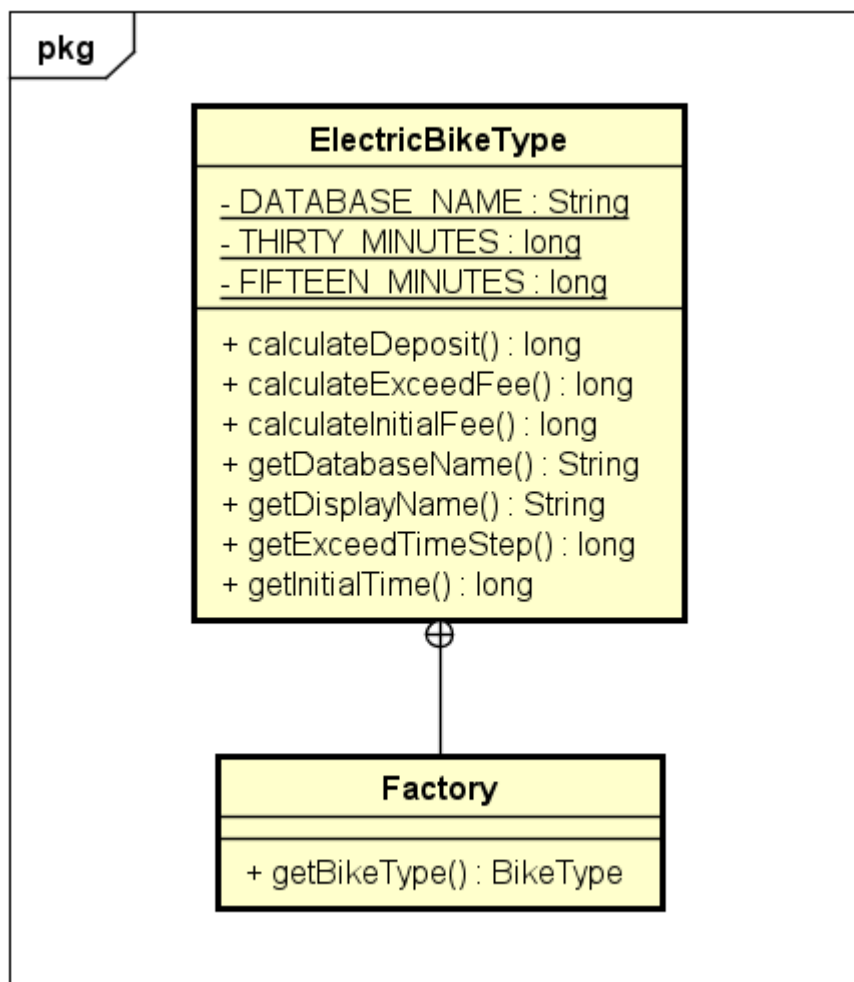
5.2.2.6 Lớp BikeTypeFactoryProducer



Method:

#	Tên	Kiểu dữ liệu trả về	Tham số	Mô tả	Exception
1	getBikeTypeFactory	BikeTypeFactory	DatabaseName: String - the database name of the BikeType class	Get a BikeTypeFactory from the BikeType's database name, used to generate new BikeType objects	No

5.2.2.7 Lớp ElectricBikeType



Attribute:

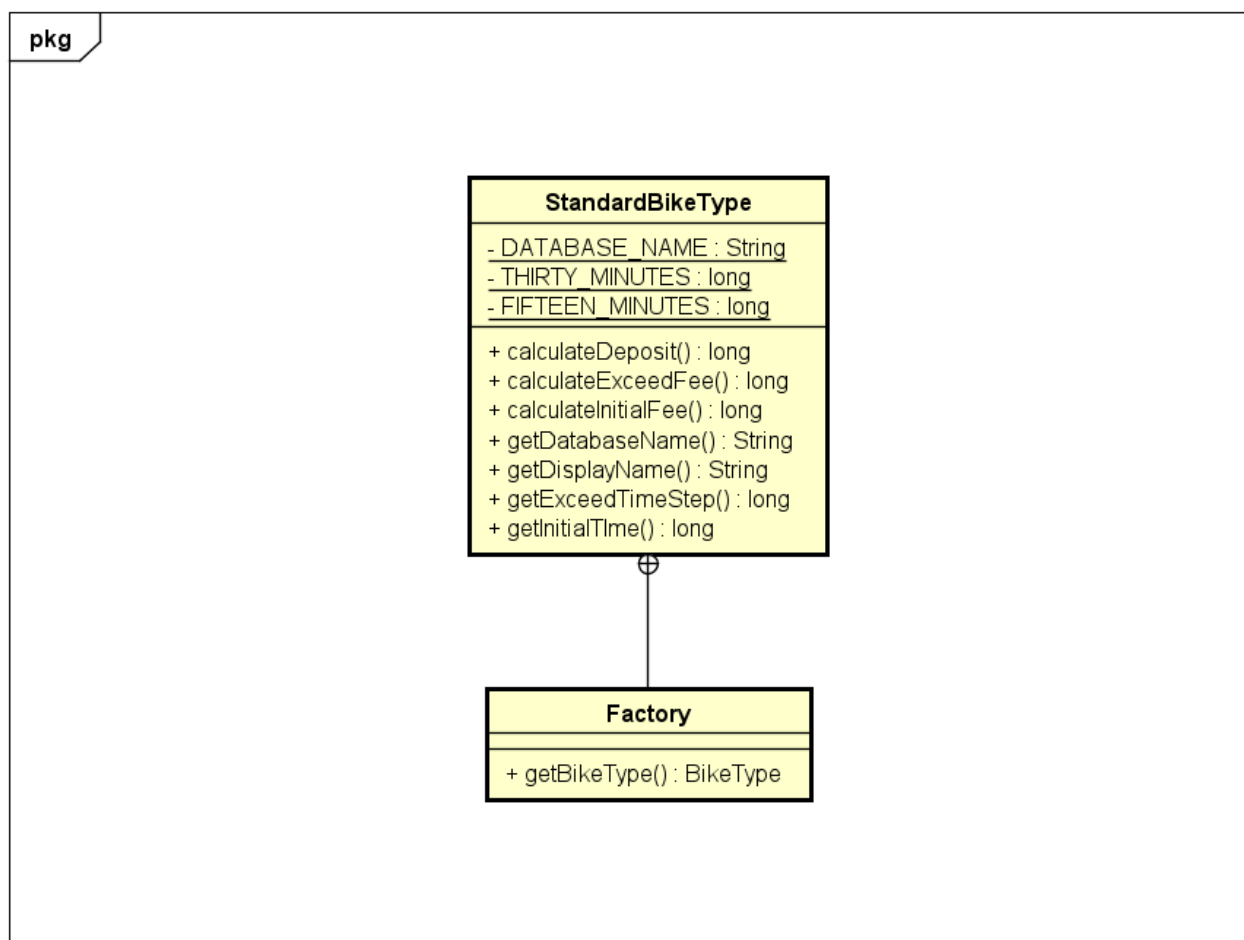
#	Tên	Kiểu dữ liệu	Giá trị mặc định	Mô tả
1	DATABASENAME	String	ELECTRIC	Name of this byte type in database

2	THIRTY_MINUTES	long	30 * 60 * 1000	Constant value of thirty minutes in seconds
3	FIFTEEN_MINUTES	long	15 * 60 * 1000	Constant value of fifteen minutes in seconds

Method:

#	Tên	Kiểu dữ liệu trả về	Tham số	Mô tả	Exception
1	getDisplayName	String	No	Return the human-readable name of this bike type, used to display in the UI	No
2	getDatabaseName	String	No	Return a non-whitespace safe ASCII string, used to determine the bike type in the database	No
3	calculateDeposit	long	No	Return the amount of money needed to be deposited before the user can start renting bike	No
4	getInitialTime	long	No	Get the length of the initial starting phrase	No
5	getExceedTimeStep	long	No	Get the length of each time step after the renting fee will be increased	No
6	calculateInitialTime	long	No	Get the initial starting fee in the first phrase	No
7	calculateExceedFee	long	No	Get the amount of fee increasing after each time step of the second phrase	No

5.2.2.8 Lớp StandradBikeType



Attribute:

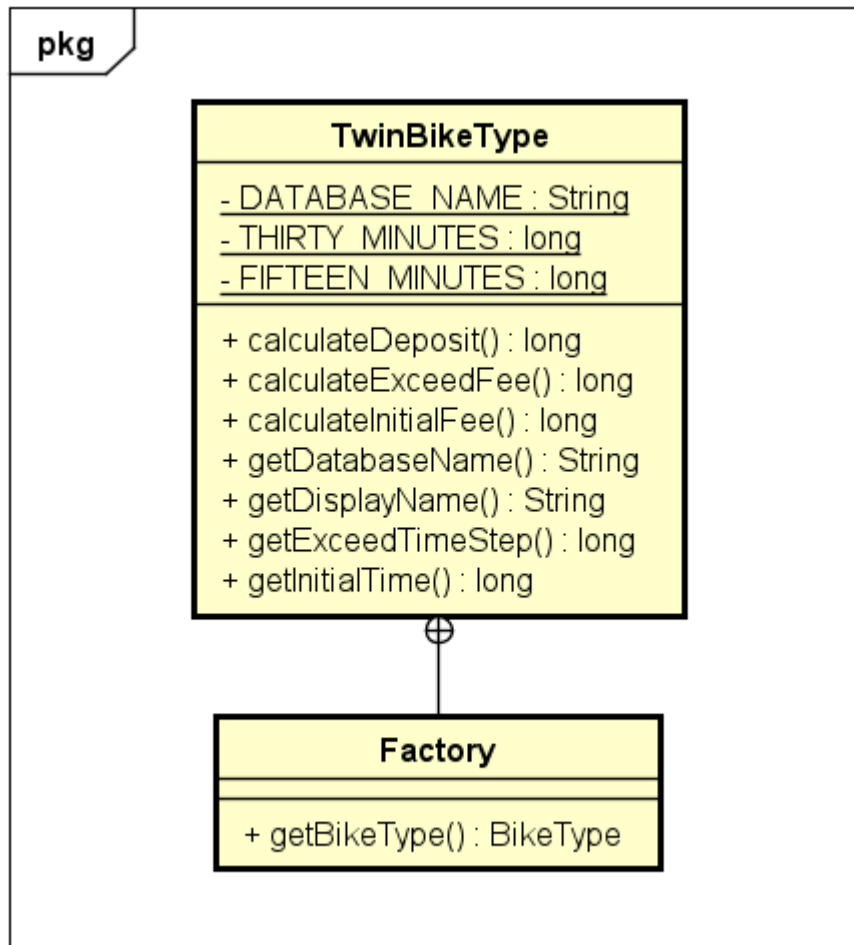
#	Tên	Kiểu dữ liệu	Giá trị mặc định	Mô tả
1	DATABASENAME	String	STANDARD	Name of this byte type in database
2	THIRTY_MINUTES	long	30 * 60 * 1000	Constant value of thirty minutes in seconds
3	FIFTEEN_MINUTES	long	15 * 60 * 1000	Constant value of fifteen minutes in seconds

Method:

#	Tên	Kiểu dữ liệu trả về	Tham số	Mô tả	Exception
1	getDisplayname	String	No	Return the human-readable name of this bike type, used to display in the UI	No

2	getDatabaseName	String	No	Return a non-whitespace safe ASCII string, used to determine the bike type in the database	No
3	calculateDeposit	long	No	Return the amount of money needed to be deposited before the user can start renting bike	No
4	getInitialTime	long	No	Get the length of the initial starting phrase	No
5	getExceedTimeStep	long	No	Get the length of each time step after the renting fee will be increased	No
6	calculateInitialTime	long	No	Get the initial starting fee in the first phrase	No
7	calculateExceedFee	long	No	Get the amount of fee increasing after each time step of the second phrase	No

5.2.2.9 Lớp TwinBikeType



Attribute:

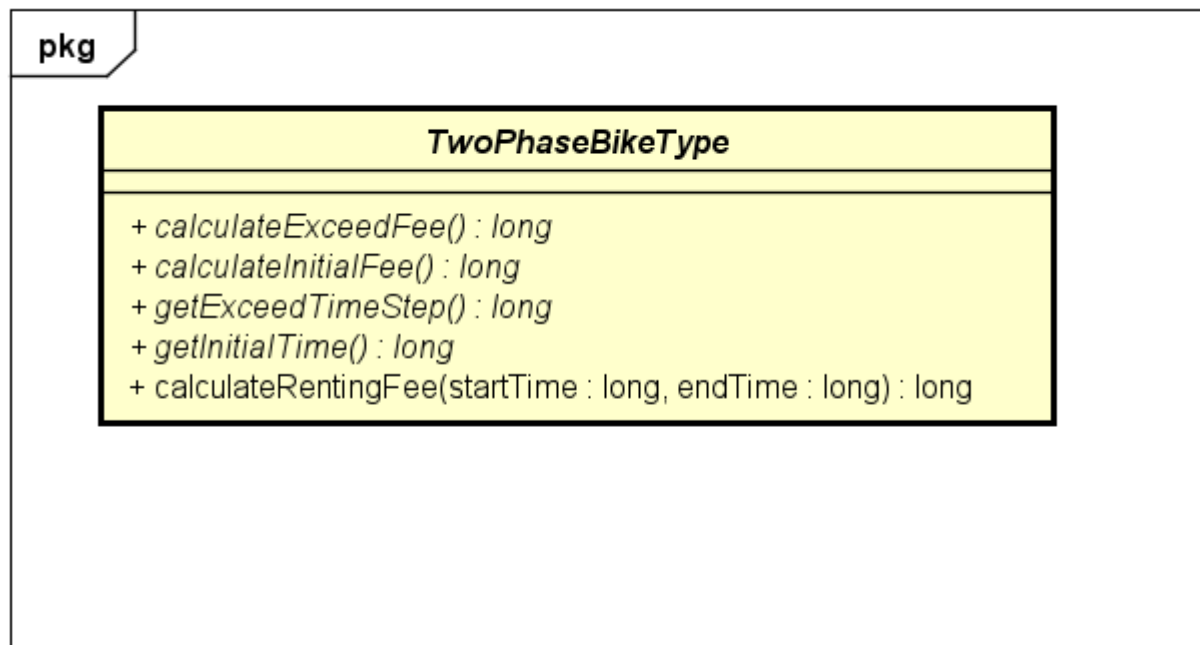
#	Tên	Kiểu dữ liệu	Giá trị mặc định	Mô tả
1	DATABASENAME	String	TWIN	Name of this byte type in database
2	THIRTY_MINUTES	long	30 * 60 * 1000	Constant value of thirty minutes in seconds
3	FIFTEEN_MINUTES	long	15 * 60 * 1000	Constant value of fifteen minutes in seconds

Method:

#	Tên	Kiểu dữ liệu trả về	Tham số	Mô tả	Exception
1	getDisplayName	String	No	Return the human-readable name of this bike type, used to display in the UI	No

2	getDatabaseName	String	No	Return a non-whitespace safe ASCII string, used to determine the bike type in the database	No
3	calculateDeposit	long	No	Return the amount of money needed to be deposited before the user can start renting bike	No
4	getInitialTime	long	No	Get the length of the initial starting phrase	No
5	getExceedTimeStep	long	No	Get the length of each time step after the renting fee will be increased	No
6	calculateInitialTime	long	No	Get the initial starting fee in the first phrase	No
7	calculateExceedFee	long	No	Get the amount of fee increasing after each time step of the second phrase	No

5.2.2.10 Lớp TwoPhaseBikeType

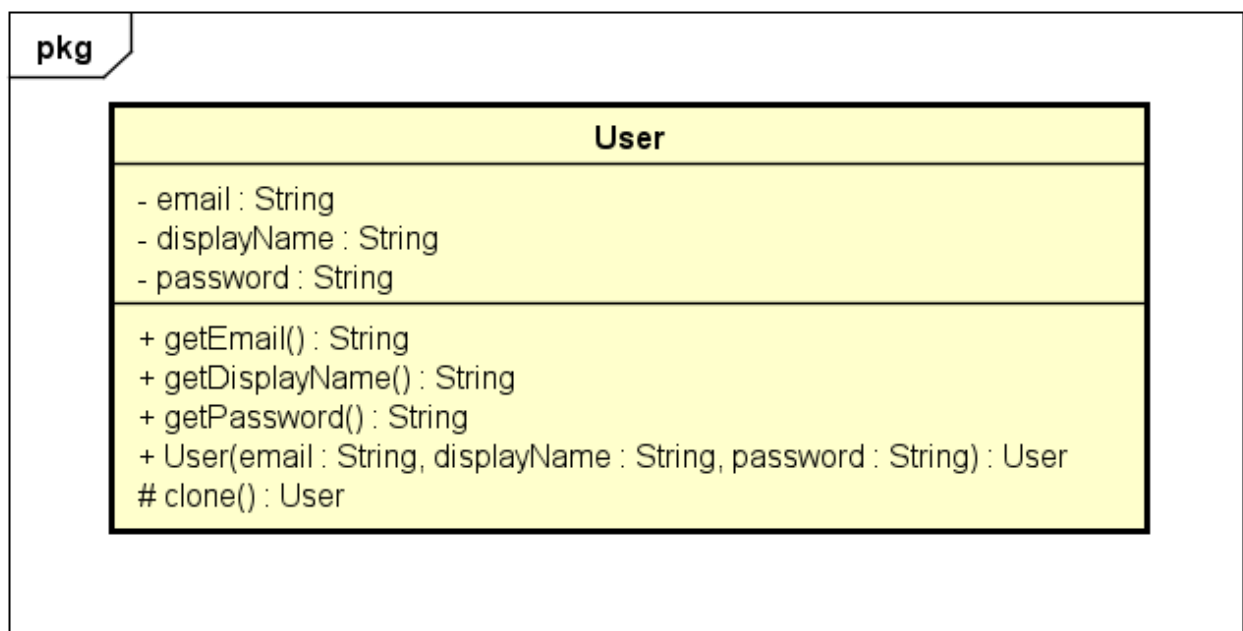


Method:

#	Tên	Kiểu dữ liệu trả về	Tham số	Mô tả	Exception
---	-----	---------------------	---------	-------	-----------

1	getInitialTime	long	No	Get the length of the initial starting phrase	No
2	getExceedTimeStep	long	No	Get the length of each time step after the renting fee will be increased	No
3	calculateInitialTime	long	No	Get the initial starting fee in the first phrase	No
4	calculateExceedFee	long	No	Get the amount of fee increasing after each time step of the second phrase	No
5	calculateRentingFee	long	<ul style="list-style-type: none"> • StartTime: long – the start time of the rent • EndTime: long – the end time of the rent 	Return the amount of fee needed to rent the bike for a given time period	No

5.2.2.11 Lớp User



Attribute:

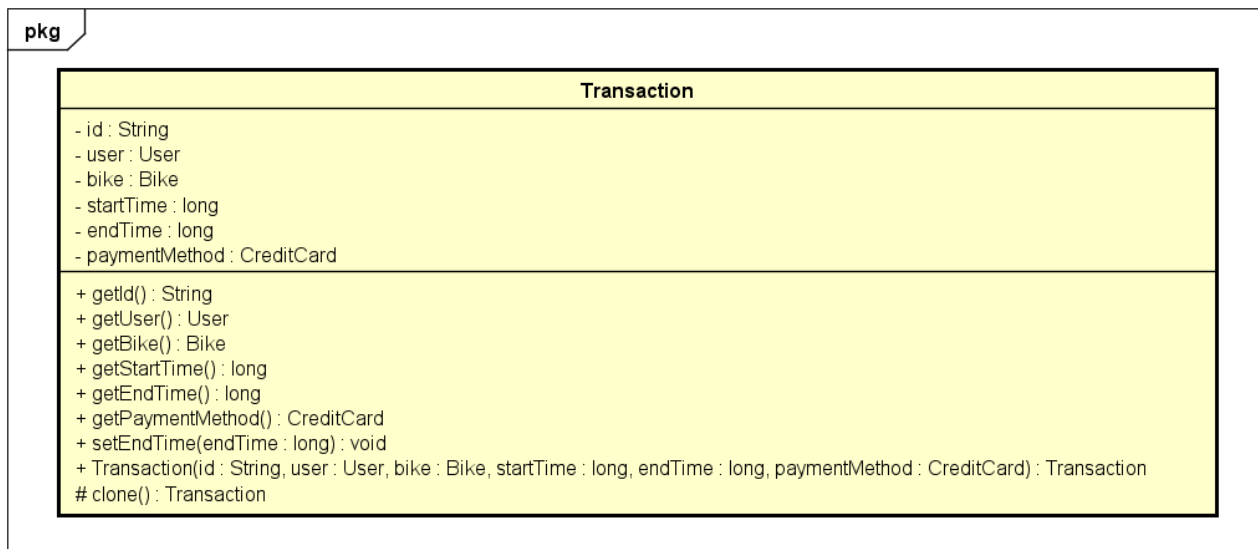
#	Tên	Kiểu dữ liệu	Giá trị mặc định	Mô tả
1	email	String	Null	Email of user

2	displayName	String	Null	Display name of user
3	password	String	Null	Password of user

Method:

#	Tên	Kiểu dữ liệu trả về	Tham số	Mô tả	Exception
1	getEmail	String	No	Get the User's email	No
2	getDisplayName	String	No	Get the User's displayName	No
3	getPassword	String	No	Get the User's password	No
4	User	User	email: String, displayName: String, password: String	Constructor a User	No
5	clone	User	No	Clone a User	No

5.2.2.12 Lớp Transaction



Attribute:

#	Tên	Kiểu dữ liệu	Giá trị mặc định	Mô tả
1	id	String	Null	Id of the transaction
2	user	User	Null	User making this transaction

3	bike	Bike	Null	The bike being rented in this transaction
4	startTime	long	Null	The time the rent started, in Epoch milisecond
5	endTime	long	Null	The time the rent ended, in Epoch milisecond
6	paymentMethod	CreditCard	Null	Information of the card used for this transaction

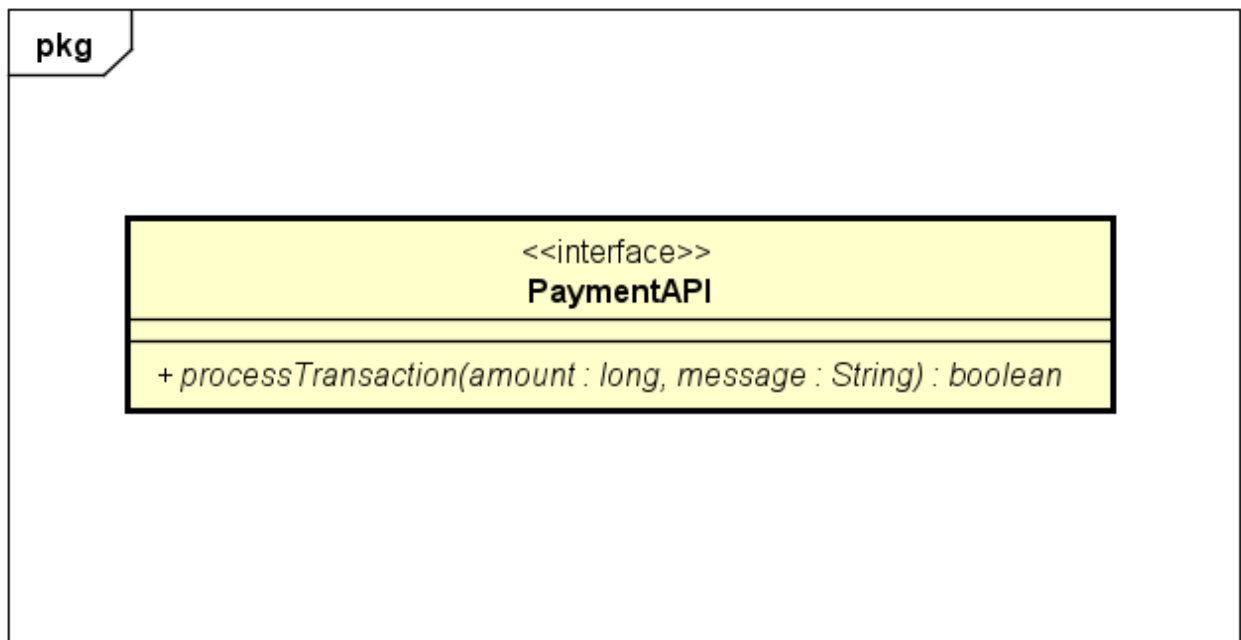
Method:

#	Tên	Kiểu dữ liệu trả về	Tham số	Mô tả	Exception
1	getId	String	No	Get the id of this transaction	No
2	getUser	User	No	Get the user making this transaction	No
3	getBike	Bike	No	Get the bike being rented in this transaction	No
4	getStartTime	long	No	Get the start time rent	No
5	getEndTime	long	No	Get the end time rent	No
6	getPaymentMethod	CreditCard	No	Get the information of the card used for this transaction	No
7	formatTime	String	time: long	Convert time format from float to String	No
8	getStartTimeString	String	No	Get the end time the rent started as a human-readable string	No
9	getEndTimeString	String	No	Get the end time the rent ended as a human-readable string	No
10	setEndTime	void	endTime: long	Set the time the rent end tin Epoch milisecond	No
11	Transaction	Transaction	id: String, user: User, bike: Bike,	Contructor a Transaction	No

			startTime: long, endTime: long, paymentMethod: Creditcard		
12	clone	Transaction	No	Clone a Transaction	No

5.2.3 Thiết kế lớp thuộc gói payment

5.2.3.1 Interface PaymentApi

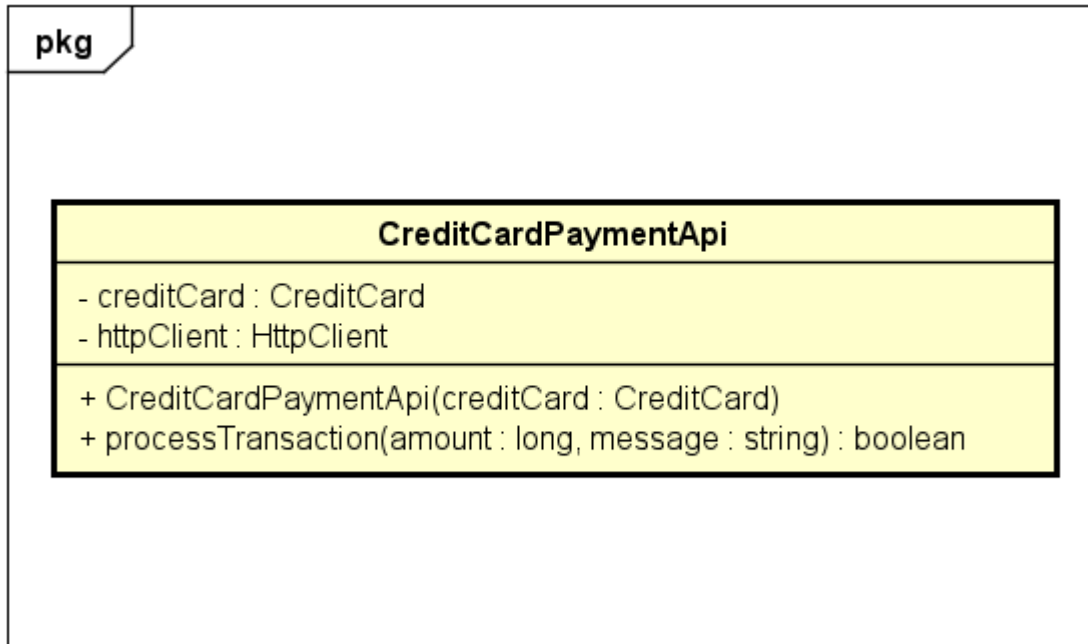


PaymentApi là interface chung cho mọi phương thức thanh toán có thể được hỗ trợ bởi phần mềm. Chương trình sẽ thao tác qua giao diện này để thực hiện thanh toán.

Method

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	processTransaction	<ul style="list-style-type: none"> amount (long): Số tiền thực hiện thanh toán. Đặt giá trị dương nếu thực hiện thanh toán trừ tiền khỏi tài khoản người dùng, và giá trị âm nếu thực hiện thanh toán hoàn tiền cho người dùng. Message (string): Thông điệp gửi kèm giao dịch. 	boolean	Thực hiện một giao dịch thanh toán chuyển tiền và trả về true nếu giao dịch thanh toán thành công	Không

5.2.3.2 Lớp CreditCardPaymentApi



CreditCardPaymentApi là một lớp cài đặt **PaymentApi**, thực hiện việc thanh toán với thẻ tín dụng. Một đối tượng CreditCardPaymentApi sau khi được khởi tạo sẽ đảm nhận giao dịch với một thẻ duy nhất, và chương trình có thể giao dịch với thẻ tín dụng đó qua đối tượng này.

Attribute

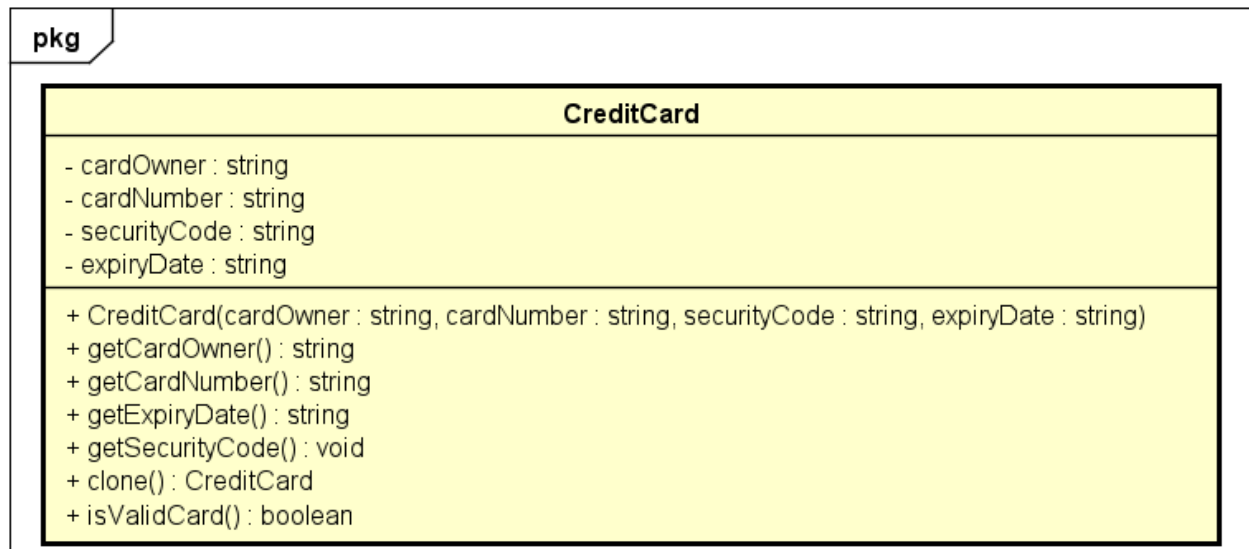
#	Tên	Kiểu dữ liệu trả về	Giá trị mặc định	Mô tả
1	creditCard	CreditCard	null	Thẻ tín dụng mà chương trình có thể thực hiện giao dịch thông qua đối tượng này.
2	httpClient	HttpClient	null	Đối tượng thực hiện các giao dịch HTTP tới API thanh toán.

Method

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	CreditCardPaymentApi	<ul style="list-style-type: none">creditCard (CreditCard): Thẻ tín dụng được sử dụng để thực hiện thanh toán.	Hàm khởi tạo	Khởi tạo một đối tượng CreditCardPaymentApi với một thẻ tín dụng xác định. Chương trình có thể thực hiện giao dịch với thẻ tín dụng này thông qua đối tượng này.	Không
2	processTransaction	<ul style="list-style-type: none">amount (long): Số tiền thực hiện thanh toán. Đặt giá trị dương nếu thực hiện thanh toán trừ	boolean	Thực hiện một giao dịch thanh toán chuyển tiền và trả về true nếu	Không

		tiền khởi tài khoản người dùng, và giá trị âm nếu thực hiện thanh toán hoàn tiền cho người dùng. <ul style="list-style-type: none"> Message (string): Thông điệp gửi kèm giao dịch. 		giao dịch thanh toán thành công	
--	--	---	--	---------------------------------	--

5.2.3.3 Lớp CreditCard



CreditCard là lớp lưu trữ thông tin về thẻ tín dụng, và được sử dụng bởi **CreditCardPaymentApi** để thực hiện giao dịch.

Attribute

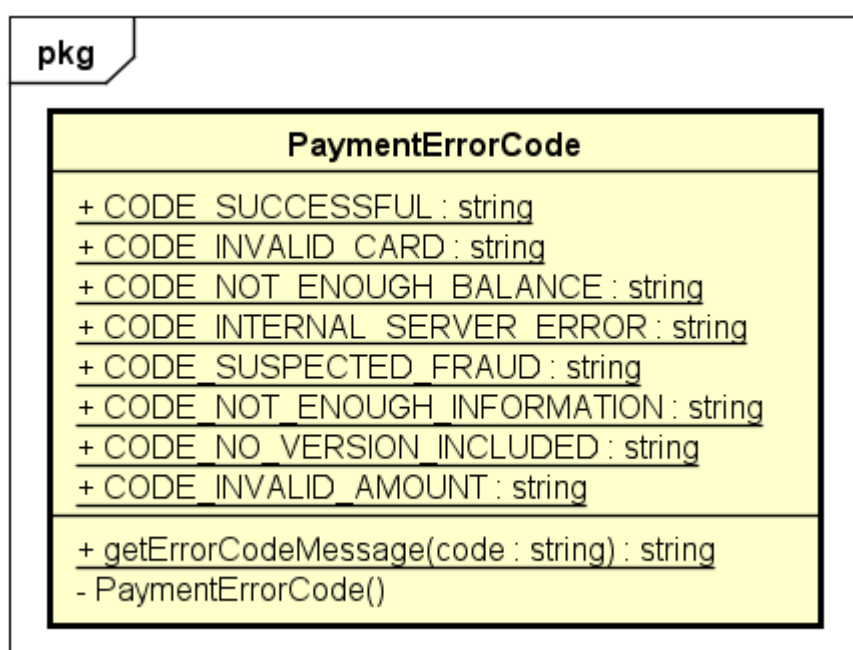
#	Tên	Kiểu dữ liệu trả về	Giá trị mặc định	Mô tả
1	cardOwner	string	null	Tên chủ sở hữu được ghi trên thẻ.
2	cardNumber	String	null	Mã số thẻ.
3	securityCode	string	null	Mã an toàn trên thẻ.
4	expiryDate	string	null	Ngày hết hạn của thẻ.

Method

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	CreditCard	<ul style="list-style-type: none"> cardOwner (string) cardNumber (string) securityCode (string) expiryDate (string) 	Hàm khởi tạo	Khởi tạo một đối tượng CreditCard.	Không
2	getCardOwner	Không	string	Lấy tên chủ sở hữu được ghi trên thẻ.	Không
3	getCardNumber	Không	string	Lấy mã số thẻ.	Không

4	getSecurityCode	Không	string	Lấy mã an toàn trên thẻ.	Không
5	getExpiryDate	Không	string	Lấy ngày hết hạn của thẻ.	Không
6	clone	Không	CreditCard	Tạo một đối tượng CreditCard độc lập với cùng thông tin với đối tượng này.	Không
7	isValidCard	Không	boolean	Trả về true nếu thông tin thẻ trong đối tượng này là hợp lệ.	Không

5.2.3.4 Lớp PaymentErrorCode



PaymentErrorCode là một lớp hỗ trợ để lưu lại các trạng thái thanh toán được trả về bởi Payment API.

Attribute

#	Tên	Kiểu dữ liệu trả về	Giá trị mặc định	Mô tả
1	CODE_SUCCESSFUL	string	"00"	Giá trị Payment API trả về khi thanh toán thành công.
2	CODE_INVALID_CARD	string	"01"	Giá trị Payment API trả về khi thanh toán không thành công do thông tin thẻ không hợp lệ.
3	CODE_NOT_ENOUGH_BALANCE	string	"02"	Giá trị Payment API trả về khi thanh toán không thành công do thẻ không đủ số dư.

4	CODE_INTERNAL_SERVER_ERROR	string	"03"	Giá trị Payment API trả về khi thanh toán không thành công do lỗi từ nội bộ server Payment API.
5	CODE_SUSPECTED_FRAUD	string	"04"	Giá trị Payment API trả về khi thanh toán không thành công do nghi ngờ gian lận.
6	CODE_NOT_ENOUGH_INFORMATION	string	"05"	Giá trị Payment API trả về khi thanh toán không thành công do yêu cầu gửi thiếu thông tin.
7	CODE_NO_VERSION_INCLUDED	string	"06"	Giá trị Payment API trả về khi thanh toán không thành công do thiếu thông tin về phiên bản API được sử dụng.
8	CODE_INVALID_AMOUNT	string	"07"	Giá trị Payment API trả về khi thanh toán không thành công do giá trị được gửi tới không hợp lệ.

Method

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	getErrorMessage	<ul style="list-style-type: none"> code (string): Một trong các giá trị trạng thái thanh toán. 	string	Trả về một string con người có thể đọc được mô tả ý nghĩa của giá trị trạng thái.	Không

5.2.4 Thiết kế lớp thuộc gói servlets

5.2.4.1 Lớp BarcodeServlet

Lớp BarcodeServlet có chức năng thực hiện HTTP GET request của người dùng từ giao diện Barcode

BarcodeServlet
doGet(req : HttpServletRequest, resp : HttpServletResponse) : void

Attribute:

Không

Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	doGet	<ul style="list-style-type: none"> req (HttpServletRequest): đối tượng request 	string	Trả về một string con người có thể đọc được mô tả ý nghĩa của giá trị trạng thái.	Không

		<ul style="list-style-type: none"> • resp (HttpServletRequest): đối tượng resp 			
--	--	---	--	--	--

5.2.4.2 Lớp BikeInfoServlet

Lớp BikeInfoServlet có chức năng thực hiện HTTP GET request của người dùng từ giao diện để hiển thị thông tin xe

BikeInfoServlet
- <u>BARCODE : BarcodeController</u>
doGet(req : HttpServletRequest, resp : HttpServletResponse) : void

Attribute:

#	Tên	Kiểu dữ liệu	Mô tả
1	BARCODE	BarcodeController	Tham chiếu đến object Barcodecontroller

Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	doGet	<ul style="list-style-type: none"> • req (HttpServletRequest): đối tượng request • resp (HttpServletResponse): đối tượng resp 	void	Tìm kiếm xe từ code do người dùng cung cấp và chuyển đến giao diện BikeInfo, nếu không có kết quả tìm kiếm thì trở lại giao diện barcode	ServletException, IOException

5.2.4.3 Lớp DepositBikeServlet

Lớp DepositBikeServlet có chức năng thực hiện HTTP GET request của người dùng từ trên giao diện đặt cọc

DepositBikeServlet
- RENTING : RentingController
doGet(req : HttpServletRequest, resp : HttpServletResponse) : void

Attribute:

#	Tên	Kiểu dữ liệu	Mô tả
1	RENTING	RentingController	Tham chiếu đến object RentingController

Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	doGet	<ul style="list-style-type: none"> req (HttpServletRequest): đối tượng request resp (HttpServletResponse): đối tượng resp 	void	Thực hiện kiểm tra trạng thái xe, nếu xe chưa được thuê, sẽ tính toán các chi phí và chuyển đến giao diện đặt cọc.	ServletException, IOException

5.2.4.4 Lớp DockListServlet

Lớp DocklistServlet có chức năng thực hiện HTTP GET request của người dùng để hiển thị danh sách trạm xe

DockListServlet
- RENTING : RentingController
doGet(req : HttpServletRequest, resp : HttpServletResponse) : void

Attribute:

#	Tên	Kiểu dữ liệu	Mô tả
1	RENTING	RentingController	Tham chiếu đến object RentingController

Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
---	-----	---------	---------------------	------------------	-----------

1	doGet	<ul style="list-style-type: none"> req (HttpServletRequest): đối tượng request resp (HttpServletResponse): đối tượng resp 	void	Truy vấn thông tin các trạm xe và chuyển đến giao diện danh sách trạm	ServletException, IOException
---	-------	---	------	---	-------------------------------

5.2.4.5 Lớp DockInfoServlet

Lớp DockInfoServlet có chức năng thực hiện HTTP GET request của người dùng để hiển thị thông tin trạm xe

DockInfoServlet
- <u>DOCK_DAO : DockDao</u>
- <u>BIKE_DAO : BikeDao</u>
doGet(req : HttpServletRequest, resp : HttpServletResponse) : void

Attribute:

#	Tên	Kiểu dữ liệu	Mô tả
1	DOCK_DAO	DockDao	Tham chiếu đến object DockDao
	BIKE_DAO	BikeDao	Tham chiếu đến object BikeDao

Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	doGet	<ul style="list-style-type: none"> req (HttpServletRequest): đối tượng request resp (HttpServletResponse): đối tượng resp 	void	Thực hiện truy vấn thông tin trạm xe và truy vấn thông tin tất cả các xe có trong trạm và chuyển đến giao diện thông tin trạm xe để hiển thị kết quả	ServletException, IOException

5.2.4.6 Lớp RentBikeServlet

Lớp RentBikeServlet có chức năng thực hiện HTTP POST request của người dùng để thuê xe

RentBikeServlet
- BIKE_DAO : BikeDao - USER_DAO : UserDao - TRANSACTION_DAO : TransactionDao - RENTING : RentingController
doPost(req : HttpServletRequest, resp : HttpServletResponse) : void

Attribute:

#	Tên	Kiểu dữ liệu	Mô tả
1	USER_DAO	UserDao	Tham chiếu đến object UserDao
2	BIKE_DAO	BikeDao	Tham chiếu đến object BikeDao
3	TRANSACTION_DAO	TransactionDao	Tham chiếu đến object TransactionDao
4	RENTING	RentingController	Tham chiếu đến object RentingController

Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	doPost	<ul style="list-style-type: none"> req (HttpServletRequest): đối tượng request resp (HttpServletResponse): đối tượng resp 	void	Xử lý yêu cầu thuê xe	ServletException, IOException

5.2.4.7 Lớp RentStatusServlet

Lớp RentStatusServlet có chức năng thực hiện HTTP GETrequest của người dùng để xem trạng thái xe đang thuê

RentStatusServlet
- TRANSACTION_DAO : TransactionDao - RENTING : RentingController
+ doGet(req : HttpServletRequest, resp : HttpServletResponse) : void

Attribute:

#	Tên	Kiểu dữ liệu	Mô tả
1	TRANSACTION_DAO	TransactionDao	Tham chiếu đến object TransactionDao

Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	doGet	<ul style="list-style-type: none"> req (HttpServletRequest): đối tượng request resp (HttpServletResponse): đối tượng resp 	void	Xử lý yêu cầu xem trạng thái xe, cho biết thông tin xe, chi phí thuê xe cho đến thời điểm xem	ServletException, IOException

5.2.4.8 Lớp ReturnBikeServlet

Lớp RentBikeServlet có chức năng thực hiện HTTP POST request của người dùng để thuê xe

ReturnBikeServlet
- DOCK_DAO : DockDao - TRANSACTION_DAO : TransactionDao - RENTING : RentingController
+ doPost(req : HttpServletRequest, resp : HttpServletResponse) : void

Attribute:

#	Tên	Kiểu dữ liệu	Mô tả
1	DOCK_DAO	DockDao	Tham chiếu đến object DockDao
2	TRANSACTION_DAO	TransactionDao	Tham chiếu đến object TransactionDao
3	RENTING	RentingController	Tham chiếu đến object RentingController

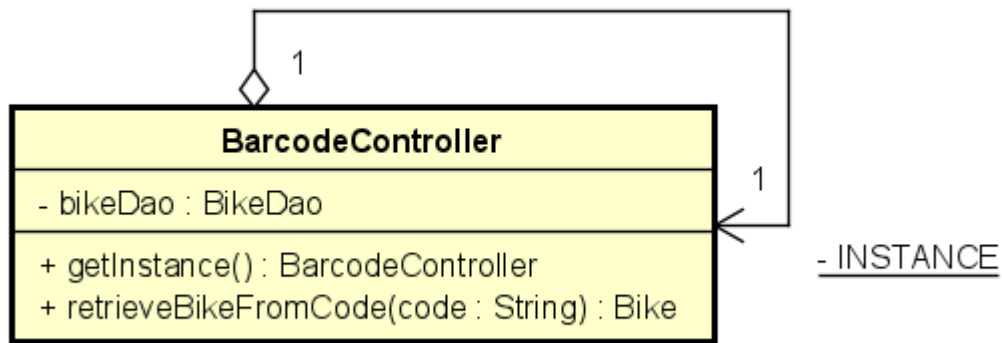
Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	doPost	<ul style="list-style-type: none"> req (HttpServletRequest): đối tượng request resp (HttpServletResponse): đối tượng resp 	void	Xử lý yêu cầu trả xe	ServletException, IOException

5.2.5 Thiết kế lớp thuộc gói barcode

5.2.5.1 Lớp BarcodeController

Lớp BarcodeController có chức năng tìm xe từ mã barcode



Attribute:

#	Tên	Kiểu dữ liệu	Mô tả
1	INSTANCE	BarcodeController	Tham chiếu đến instance duy nhất của class (singleton)
2	bikeDao	BikeDao	Giúp truy vấn xe từ csdl

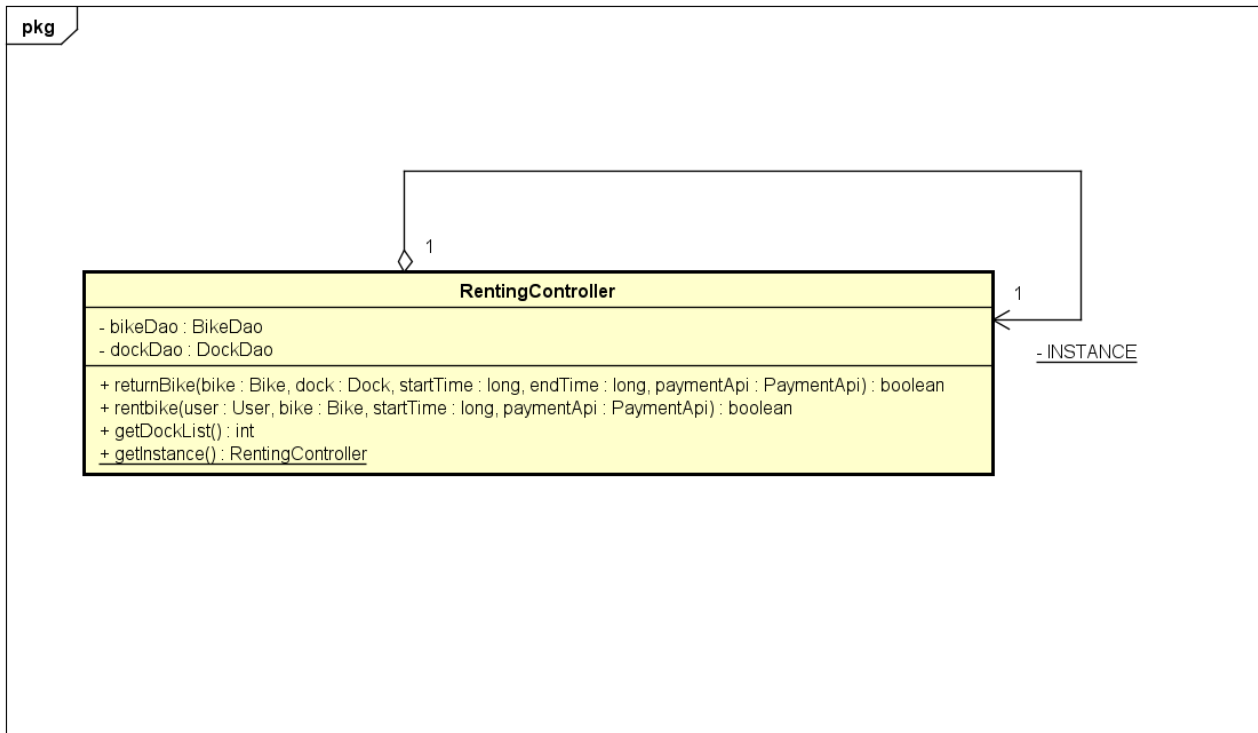
Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	getInstance	không	BarcodeController	Lấy 1 đối tượng của lớp	Không
2	retrieveBikeFromCode	<ul style="list-style-type: none">code (String): barcode	Bike	Lấy thông tin xe dùng theo barcode từ cơ sở dữ liệu	không

5.2.6 Thiết kế lớp thuộc gói renting

5.2.6.1 Lớp RentingController

Lớp RentingController có chức năng thực hiện các nghiệp vụ liên quan đến việc thuê trả xe



Attribute:

#	Tên	Kiểu dữ liệu	Mô tả
1	INSTANCE	RentingController	Tham chiếu đến instance duy nhất của class (singleton)
2	bikeDao	BikeDao	Giúp truy vấn xe từ csdl
3	dockDao	DockDao	Giúp truy vấn bãi xe từ csdl

Method:

#	Tên	Tham số	Kiểu dữ liệu trả về	Mô tả (mục đích)	Exception
1	getInstance	không	RentingController	Lấy 1 đối tượng của lớp	Không
2	getDockList	không	List<Dock>	Lấy thông tin tất cả các bãi xe	không
3	rentBike	<ul style="list-style-type: none"> user (User): người dùng bike (Bike): xe được thuê startTime (long): thời gian bắt đầu thuê xe paymentApi (PaymentApi): 	boolean	Cho phép thuê xe, thanh toán tiền đặt cọc	

		phương thức giao dịch			
4	returnBike	<ul style="list-style-type: none"> bike (Bike): xe được thuê dock (Dock): nơi trả xe startTime (long): thời gian bắt đầu thuê xe endTime (long): thời gian trả xe paymentApi (PaymentApi): phương thức giao dịch 	boolean	Trả xe và trả loại tiền đặt cọc (sau khi đã trừ tiền thuê xe)	không

5.2.7 Thiết kế lớp thuộc gói utilities

5.2.7.1 Lớp Configs

Lớp Configs có chức năng cung cấp các thông tin cấu hình cho hệ thống

Configs
+ DOTENV : Dotenv

Attribute:

#	Tên	Kiểu dữ liệu	Mô tả
1	DOTENV	Dotenv	Chứa các thông tin liên quan đến các biến môi trường

Method:

không

6 Design considerations

6.1 Design Concept – Coupling và Cohesion

Coupling là sự liên kết giữa các đối tượng trong các module khác nhau, còn Cohesion là sự liên kết giữa các đối tượng trong cùng một module. Khi thiết kế dự án, ta hướng tới việc tối giản Coupling và tối đa Cohesion.

Báo cáo sau đây sẽ liệt kê ra các cấp độ Coupling và Cohesion, các vị trí tuân thủ/vi phạm các cấp độ này trong mã nguồn của dự án, và các phương hướng cải tiến/khắc phục.

6.1.1 Coupling

6.1.1.1 Content coupling

Hai module được gọi là content coupling với nhau nếu nội dung cài đặt bên trong của một module này có thể bị thay đổi bởi một module khác. Đây là cấp độ coupling xấu nhất, vì điều này có nghĩa là nội dung trong một module có thể bị sửa mà không hề được báo trước, và khi làm việc với một module thì ta phải quan tâm đến tất cả các module có content coupling với nó.

Rất may mắn là trong thiết kế của dự án không xuất hiện content coupling.

6.1.1.2 Common coupling

Hai module được gọi là common coupling với nhau nếu chúng cùng thao tác chung trên một trường dữ liệu toàn cục nằm ngoài cả hai. Ví dụ có thể kể đến các biến toàn cục của ngôn ngữ C++ - hai function cùng chỉnh sửa trên giá trị của biến này sẽ được coi là có common coupling.

Trong ngôn ngữ Java, tất cả các dữ liệu như biến và hàm đều thuộc về class. Mọi trường hợp hai class có chia sẻ chung một trường dữ liệu sẽ rơi vào cấp độ content coupling. Vì đó mà ta chứng minh được dự án hiện tại không tồn tại Common coupling.

6.1.1.3 Control coupling

Hai module được gọi là control coupling với nhau nếu một module phải truyền vào thông tin về cách hoạt động cho module còn lại. Điều này là không tốt vì đòi hỏi người dùng sẽ phải biết nội dung cài đặt bên trong của module được gọi.

Một ví dụ điển hình cho control coupling là khi ta truyền vào một giá trị để lựa chọn phương pháp thanh toán cho sản phẩm. Nếu ta sử dụng control structure như if-else, sẽ rất dễ sinh ra những đoạn code như sau:

```
public void processTransaction(String method, long amount, String message) {  
    if (method == "CreditCard") {  
        ...  
    } else if (method == "OnlineWallet") {  
        ...  
    } else ...  
}
```

Để tránh gặp phải trường hợp này, nhóm đã sử dụng Strategy Pattern để trừu tượng hoá việc thanh toán thành các interface mà thông qua đó chương trình có thể làm việc.

```
public interface PaymentApi {  
    /**
```

```

    * Process a transaction, and return whether the transaction succeeded. The user
    * should only be charged if the transaction succeeded.
    *
    * @param amount The amount of money to be transacted. If this value is
    *               positive, the amount of money is moved from
    *               <code>paymentMethod</code> to the app's account. Otherwise,
    *               the amount of money is refunded to <code>paymentMethod</code>.
    * @param message The transaction message.
    * @return <code>true</code> if the transaction is success, <code>false</code>
    *         otherwise.
    */
    public boolean processTransaction(long amount, String message);
}

```

Một ví dụ khác – trong mã nguồn của chương trình tại commit [5f7196a](#) trở về trước, hệ thống sử dụng enum `BikeType` để biểu diễn loại xe đạp. Khi thực hiện tính giá tiền thuê xe, hàm tính giá tiền sẽ dựa vào loại xe để trả về giá trị hợp lý:

```

public long calculateDeposit(Bike bike) {
    switch (bike.getType()) {
        case STANDARD:
            return 400000;
        case TWIN:
            return 550000;
        case ELECTRIC:
            return 700000;
        default:
            return 0;
    }
}

```

Cài đặt này vi phạm control coupling - khi chúng ta bổ sung một loại xe mới trong tương lai, chúng ta cũng cần phải biết được cài đặt bên trong của các hàm tính tiền thuê xe không hỗ trợ các loại xe mới này. Để sửa chữa vấn đề này, nhóm cũng áp dụng Strategy Pattern để trừu tượng hoá các loại xe, cùng với các phương thức tính toán giá tiền của loại xe đó.

```

public interface BikeType {
    /**
     * Return the human-readable name of this bike type, which will be used to
     * display in the UI.
     *

```

```

    * @return
    */
    String getDisplayName();

    /**
     * Return a non-whitespace safe ASCII string, which will be used to determine
     * the bike type in the database. This should be distinct among different bike
     * types.
     *
     * @return
     */
    String getDatabaseName();

    /**
     * Return the amount of money needed to be deposited before the user can start
     * renting the bike.
     *
     * @return The amount of deposit fee needed to be paid, in Vietnamese Dong.
     */
    long calculateDeposit();

    /**
     * Return the amount of fee needed to rent the bike for a given time period.
     *
     * @param startTime The start time of the rent, in milliseconds from Epoch.
     * @param endTime   The end time of the rent, in milliseconds from Epoch.
     * @return The amount of renting fee needed to be paid, in Vietnamese Dong.
     */
    long calculateRentingFee(long startTime, long endTime);
}

```

Vấn đề control coupling vẫn chưa được khắc phục 100% - trong cài đặt của chương trình vẫn sử dụng một lớp `BikeTypeFactoryProducer`, dùng để lấy một đối tượng `BikeTypeFactory` để sản sinh ra chính xác đối tượng `BikeType` từ tên loại xe ở trong database. Cài đặt này có thể chấp nhận được, vì chúng ta luôn cần phải cài đặt cơ chế sản sinh loại xe từ các giá trị đã được quy định và biết trước trong database.

```
public class BikeTypeFactoryProducer {
```

```

private BikeTypeFactoryProducer() {
}

/**
 * Get a BikeTypeFactory from the BikeType's database name, which can be used to
 * generate new BikeType objects.
 *
 * @param databaseName The database name of the BikeType class.
 * @return The appropriate BikeType, or null if an invalid name was passed in.
 */
public static BikeTypeFactory getBikeTypeFactory(String databaseName) {
    switch (databaseName) {
        case StandardBikeType.DATABASE_NAME:
            return new StandardBikeType.Factory();
        case TwinBikeType.DATABASE_NAME:
            return new TwinBikeType.Factory();
        case ElectricBikeType.DATABASE_NAME:
            return new ElectricBikeType.Factory();
        default:
            return null;
    }
}
}

```

6.1.1.4 Stamp coupling

Hai module được coi là stamp coupling với nhau nếu một module truyền vào nhiều dữ liệu cho module kia hơn cần thiết. Đây là mức coupling chấp nhận được, nhưng nên cân nhắc thêm vì ta muốn các module đạt được sự độc lập lớn nhất có thể với nhau.

Quay lại vấn đề cài đặt loại xe trong phần trước – các hàm tính tiền thuê xe nhận vào toàn bộ thông tin về xe đạp, nhưng chỉ sử dụng loại của xe vào công thức tính toán (cùng với thời điểm bắt đầu và kết thúc việc thuê xe):

```

public long calculateRentingFee(Bike bike, long startTime, long endTime) {
    long fee = calculateInitialFee(bike);
    long rentDuration = endTime - startTime;
    if (rentDuration > THIRTY_MINUTES) {
        long extraCount = (rentDuration - THIRTY_MINUTES + FIFTEEN_MINUTES - 1) /
FIFTEEN_MINUTES;
        fee += extraCount * calculateExceedFee(bike);
    }
}

```

```

    }

    return fee;
}

```

Cài đặt việc tính tiền thuê xe về sau không sử dụng hàm ngoài để tính toán tiền thuê xe nữa, thay vào đó việc thuê xe được quyết định bởi interface BikeType – loại xe. Do đó vấn đề này cũng được khắc phục.

6.1.1.5 Data coupling

Hai module được coi là data coupling với nhau nếu một module chỉ truyền vào các dữ liệu cơ bản cho module kia hoạt động. Đây là mức coupling tốt nhất có thể đạt được – hai module hoàn toàn độc lập với nhau, và ta hoàn toàn có thể thay thế một module bằng module khác tuân thủ theo API đã đề ra, mà không cần phải quan tâm về cài đặt bên trong của chúng.

Thiết kế của dự án hiện tại đạt mức Data Coupling – việc thao tác của các lớp trong project được thực hiện thông qua việc truyền các dữ liệu dạng nguyên thủy và các đối tượng nghiệp vụ trong của dự án và không xảy ra dư thừa. Ta hoàn toàn có thể thay thế các module điều khiển trang web để cài đặt trên các nền tảng khác mà không phải lo về việc xảy ra xung đột.

6.1.2 Cohesion

6.1.2.1 Coincidental cohesion

Một module được coi là có coincidental cohesion nếu như các thành phần của module đó không liên quan hoặc liên quan tới nhau một cách rất ngẫu nhiên. Ta có thể dễ dàng đánh giá lớp Config của chương trình vào dạng cohesion này – đây là một lớp tổng hợp các chức năng hỗ trợ trong quá trình chạy chương trình, và giữa các chức năng này không có sự liên quan gì khác.

```

public class Configs {

    private Configs() {

    }

    public static final Dotenv DOTENV =
Dotenv.configure().directory("./.env").ignoreIfMalformed().ignoreIfMissing()

        .load();

}

```

6.1.2.2 Logical cohesion

Một module được coi là có logical cohesion nếu như các thành phần của module đó có liên quan đến nhau về mặt logic, nhưng không phải về mặt chức năng.

Ta có thể đánh giá lớp RentingController là một lớp có logical cohesion – lớp này quản lý các thao tác thuê và trả xe, tuy nhiên lại xuất hiện hàm getDockList() để lấy danh sách các bến đỗ xe đạp. Hàm này có liên quan tới các chức năng khác trong lớp – nó cung cấp một giao diện độc lập với database và không throw exception cho các lớp khác lấy thông tin về bãi đỗ xe để thuê – tuy nhiên nó lại không đóng góp gì về mặt chức năng cho các phương thức còn lại.

```

public class RentingController {

    ...

    public List<Dock> getDockList() {

        try {

```

```

        return dockDao.getAllDock();
    } catch (Exception e) {
        e.printStackTrace();
        return new ArrayList<>();
    }
}

```

6.1.2.3 Temporal cohesion

Một module được coi là có temporal cohesion nếu như các thành phần của nó có liên quan đến nhau về mặt thời gian – ví dụ chúng cùng được sử dụng vào một thời điểm. Thông thường ta có thể thấy mức độ cohesion này ở trong các hàm khởi tạo hoặc dọn dẹp chương trình – một loạt các hệ thống khác nhau của chương trình sẽ được sử dụng trong cùng một điểm.

Vì dự án được cài đặt dưới dạng web app chạy trong Jetty container, do đó các thao tác khởi tạo và clean up đã được thực thi bởi Jetty. Bên trong nội bộ cài đặt của chương trình không vi phạm vào temporal cohesion.

6.1.2.4 Procedural cohesion

Một module được coi là có procedural cohesion nếu như các thành phần của nó có liên quan đến nhau theo thứ tự chứ không phải về mặt chức năng.

Ta có thể thấy mức độ cohesion này ở trong các lớp Servlet – các lớp Servlet phụ trách việc tiếp nhận request của người dùng, verify các request này tuân thủ đúng format được định ra và xử lý chúng. Các hàm verify và xử lý request được cài đặt trong lớp Servlet bởi vì chúng là các bước trong quá trình xử lý request, và không liên quan tới chức năng nhận và trả lời HTTP Request của Servlet.

```

@WebServlet("/barcode")
public class BarcodeServlet extends HttpServlet {

    private static final long serialVersionUID = -426860543039798673L;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {

        String code = req.getParameter("code");
        if (code == null) {
            req.getRequestDispatcher("/WEB-INF/barcode.jsp").forward(req, resp);
            return;
        }
        code = code.trim();
        req.setAttribute("code", code);
        req.getRequestDispatcher("/WEB-INF/barcode.jsp").forward(req, resp);
    }
}

```



```
}
```

6.1.2.5 Communicational cohesion

Một module được gọi là có communicational cohesion nếu các thành phần của nó cùng thao tác trên một dữ liệu.

Trong dự án, các lớp Data Access Object có mức độ cohesion này – các thành phần của chúng đều có chức năng thao tác trên các bảng của cơ sở dữ liệu để đọc/ghi/cập nhật các giá trị trong chương trình.

```
public class BikeDao {  
  
    /**  
     * Get a bike from its id.  
     *  
     * @param id  
     * @return  
     * @throws SQLException  
     */  
    public Bike getBike(String id) throws SQLException {  
        ...  
    }  
  
    /**  
     * Get all bike located inside a dock.  
     *  
     * @param dock  
     * @return  
     * @throws SQLException  
     */  
    public List<Bike> getBikeInDock(Dock dock) throws SQLException {  
        ...  
    }  
  
    /**  
     * Update information of a bike.  
     *  
     * @param bike  
     * @return  
     * @throws SQLException  
     */  
}
```

```

    public int updateBike(Bike bike) throws SQLException {
        ...
    }
}

```

6.1.2.6 Sequential cohesion

Một module được gọi là có sequential cohesion nếu các thành phần của nó phải được sử dụng theo một thứ tự xác định. Trong dự án không xuất hiện trường hợp nào rơi vào cấp độ cohesion này.

6.1.2.7 Functional cohesion

Một module được gọi là có functional cohesion nếu các thành phần của nó có liên quan tới nhau về mặt chức năng và cùng đi tới mục đích chung của module đó.

Một ví dụ cho functional cohesion trong dự án là lớp `CreditCardPaymentApi` – các hàm trong lớp này đều phục vụ một mục đích chung duy nhất là tạo dựng một HTTP request để gửi tới Payment API, và xử lý response để xác nhận việc thanh toán thành công.

```

public class CreditCardPaymentApi implements PaymentApi {
    public CreditCardPaymentApi(CreditCard creditCard) {
        ...
    }

    public boolean processTransaction(long amount, String message) {
        ...
    }

    private String makeRequestBody(CreditCard paymentMethod, long amount, String
message) {
        ...
    }

    private JsonObject getTransactionJson(CreditCard paymentMethod, long amount, String
message) {
        ...
    }

    private String getTodayString() {
        ...
    }
}

```

```
private JsonObject getTransactionWithSecret(JsonObject transactionJson) {  
    ...  
}  
  
private String hash(String s) {  
    ...  
}  
  
private String getTransactionCode(HttpResponse response) {  
    ...  
}  
}
```

6.2 Đánh giá tuân thủ nguyên lý SOLID

SOLID là năm nguyên lý cơ bản của thiết kế hướng đối tượng (First five object-oriented design), đề xuất bởi Robert C. Martin. Năm nguyên lý này kết hợp với nhau giúp lập trình viên có thể thiết kế phần mềm dễ bảo trì và mở rộng.

Báo cáo sau đây liệt kê các nguyên lý của SOLID và đánh giá mức độ tuân thủ các nguyên lý này của mã nguồn trong dự án Capstone Project.

6.2.1 S – Single-responsibility principle

Nguyên lý này đề ra rằng mỗi lớp chỉ nên đảm nhận một chức năng duy nhất.

Trong mã nguồn tại commit [edd1365](#), lớp `RentingController` phụ trách các thao tác liên quan tới việc thuê và trả xe. Tuy nhiên, mã nguồn của lớp này cũng bao gồm hàm `sendConfirmationEmail()` để gửi email xác nhận giao dịch thành công tới người dùng:

```
/**
 * Sent a confirmation email to the user, with the information of a transaction.
 *
 * @param transaction
 */
public void sendConfirmationEmail(Transaction transaction) {
    logger.debug("Emailed about transaction {} to email address {}",
transaction.getBike().getId(),
    transaction.getUser().getEmail());
}
```

Đề tuân thủ tiêu chí S, tính năng này đã được loại bỏ khỏi lớp này trong commit kế tiếp.

6.2.2 O – Open-closed principle

Nguyên lý này đề ra rằng các lớp chỉ nên khuyến khích việc mở rộng, và không khuyến khích việc chỉnh sửa. Mở rộng thêm các tính năng mới ở lớp mới không nên phụ thuộc vào việc chỉnh sửa lớp đã có.

Dự án `EcoBike` có nhiều loại xe đạp cho thuê khác nhau – xe đạp cơ bản, xe đạp đôi và xe đạp điện. Các loại xe đạp này có mức tiền đặt cọc và giá cho thuê khác nhau, và giá tiền này được tính toán trong lớp `RentingController`:

```
/**
 * Calculate the amount of fee necessary, including initial fee and exceed fee,
 * to rent a bike from <code>startTime</code> to <code>endTime</code>.
 *
 * @param bike
 * @param startTime
 * @param endTime
 * @return
 */
public long calculateRentingFee(Bike bike, long startTime, long endTime) {
```

```

    long fee = calculateInitialFee(bike);

    long rentDuration = endTime - startTime;

    if (rentDuration > THIRTY_MINUTES) {
        long extraCount = (rentDuration - THIRTY_MINUTES + FIFTEEN_MINUTES - 1) /
FIFTEEN_MINUTES;

        fee += extraCount * calculateExceedFee(bike);
    }

    return fee;
}

```

Giả sử trong tương lai, ta cần mở rộng thêm nhiều loại xe mới. Khi đó ta cũng sẽ phải thay bổ sung thêm công thức tính giá tiền xe ở trong lớp `RentingController`. Điều này vi phạm nguyên tắc O.

Nhóm sửa chữa vấn đề này bằng cách sử dụng các lớp mở rộng từ lớp trừu tượng cơ bản `BikeType` để biểu diễn các loại xe khác nhau. Các loại xe này override các phương thức tính tiền đặt cọc và thuê xe, do đó các công thức này sẽ gắn liền chặt chẽ với loại xe, và bổ sung loại xe mới cũng sẽ yêu cầu override lại công thức. Cài đặt `BikeType` được chỉ ra trong phần `Control Coupling` ở trên.

Để phần còn lại của chương trình có thể tương tác với xe đạp mà không cần quan tâm loại xe đang tương tác là gì, nhóm sử dụng `Abstract Factory pattern` và trả lại reference tới interface cơ bản `BikeType`:

```

/**
 * An abstract Factory interface to retrieve a BikeType object. Implements the
 * Factory method pattern.
 */
public interface BikeTypeFactory {
    /**
     * Construct a BikeType object.
     *
     * @return The constructed object.
     */
    BikeType getBikeType();
}

```

Với mỗi loại xe, ta cần cài đặt một lớp Factory mở rộng từ `BikeTypeFactory` để sản sinh các đối tượng thuộc `BikeType` đó:

```

public class StandardBikeType extends TwoPhraseBikeType {
    ...

    public static class Factory implements BikeTypeFactory {
        @Override
        public BikeType getBikeType() {
            return new StandardBikeType();
        }
    }
}

```

```

    }
}
}

```

6.2.3 L – Liskov substitution principle

Nguyên lý này đề xuất rằng mọi lớp thừa kế đều có thể thay thế vào vị trí của các lớp cơ sở. Nhóm không tìm thấy điểm nào vi phạm nguyên lý này trong mã nguồn của chương trình.

6.2.4 I – Interface segregation principle

Nguyên lý này đề xuất rằng mọi lớp thừa kế đều không nên bị bắt buộc phải cài đặt hay phụ thuộc vào một interface mà nó không dùng tới. Nhóm không tìm thấy điểm nào vi phạm nguyên lý này trong mã nguồn của chương trình.

6.2.5 D – Dependency inversion Principle

Nguyên lý này đề xuất rằng các đối tượng bậc cao không nên phụ thuộc vào cài đặt cụ thể của đối tượng bậc thấp, mà chúng nên phụ thuộc vào nhau thông qua các lớp trừu tượng.

Trong mã nguồn tại commit [b34b0a3](#), chương trình giao tiếp với Interbank API thông qua lớp PaymentController. Vì lớp này cài đặt trực tiếp các phương thức để gọi HTTP request tới Interbank API, ta không thể mở rộng chương trình để tích hợp thêm các phương thức thanh toán khác mà không sửa đổi trong lớp cơ sở. Điều này vi phạm vào nguyên lý D.

Để sửa chữa vấn đề này, commit kế tiếp [edd1365](#) thay đổi việc liên lạc với các API thanh toán. Chương trình sẽ phải giao tiếp thông qua interface PaymentAPI:

```

public interface PaymentApi {

    /**
     * Process a transaction, and return whether the transaction succeeded. The user
     * should only be charged if the transaction succeeded.
     *
     * @param amount The amount of money to be transacted. If this value is
     *                positive, the amount of money is moved from
     *                <code>paymentMethod</code> to the app's account. Otherwise,
     *                the amount of money is refunded to <code>paymentMethod</code>.
     * @param message The transaction message.
     * @return <code>true</code> if the transaction is success, <code>false</code>
     *         otherwise.
     */
    public boolean processTransaction(long amount, String message);
}

```

PaymentController của commit trước sẽ được viết lại thành một lớp cài đặt PaymentApi, đại diện cho phương thức thanh toán sử dụng thẻ ngân hàng. Bất cứ một phương pháp thanh toán nào khác đều có thể cài đặt giao diện này để thay thế vào phương pháp thanh toán mặc định ban đầu.

6.3 Áp dụng Design Pattern

Design pattern là các giải pháp chung, tái tạo lại được cho các bài toán thường gặp trong việc lập trình. Dự án có áp dụng ba design pattern là Singleton, Strategy và AbstractFactory.

6.3.1 Singleton

Singleton là một design pattern thuộc nhóm creational, được thiết kế để duy trì duy nhất một đối tượng của lớp cài đặt Singleton tồn tại trong toàn bộ chương trình. Có nhiều lý do để sử dụng Singleton, ví dụ như khi ta chỉ cần truy cập tới một đối tượng duy nhất của một lớp có quá trình khởi tạo phức tạp, hoặc khi ta muốn kiểm soát chặt hơn tới các đối tượng toàn cục trong chương trình.

Dự án sử dụng PostgreSQL làm database backend, do đó kết nối tới database được duy trì trong một lớp Singleton:

```
public class DatabaseConnection {

    private final Connection conn;

    private DatabaseConnection() {
        ...
    }

    private static final DatabaseConnection INSTANCE = new DatabaseConnection();

    public static final synchronized DatabaseConnection getInstance() {
        return INSTANCE;
    }

    public Connection getConnection() {
        return conn;
    }
}
```

Các lớp dịch vụ như Data Access Object và RentingController cũng sử dụng Singleton pattern, vì ta không có nhu cầu khởi tạo hay tùy chỉnh nhiều đối tượng của các lớp này.

6.3.2 Strategy

Strategy là một design pattern thuộc nhóm behavioral, cho phép ta định nghĩa một lớp các thuật toán giống nhau và phân chia chúng thành các lớp có thể hoán đổi được cho nhau trong lúc chạy.

Dự án áp dụng Strategy pattern vào việc cài đặt phương thức thanh toán – chương trình cần phải hỗ trợ nhiều phương thức thanh toán khác nhau như thẻ tính dụng, ví điện tử, vãn vãn... Để đảm bảo chương trình hoạt động được với bất cứ một phương thức thanh toán nào, ta cài đặt một interface PaymentApi mà tất cả các phương thức thanh toán đều phải tuân thủ:

```

public interface PaymentApi {
    /**
     * Process a transaction, and return whether the transaction succeeded. The user
     * should only be charged if the transaction succeeded.
     *
     * @param amount The amount of money to be transacted. If this value is
     *                positive, the amount of money is moved from
     *                <code>paymentMethod</code> to the app's account. Otherwise,
     *                the amount of money is refunded to <code>paymentMethod</code>.
     * @param message The transaction message.
     * @return <code>true</code> if the transaction is success, <code>false</code>
     *         otherwise.
     */
    public boolean processTransaction(long amount, String message);
}

```

Một khi đã cài đặt đúng theo interface này, ta có thể dễ dàng áp dụng chúng vào trong chương trình để các thành phần khác sử dụng:

```

public boolean returnBike(Bike bike, Dock dock, long startTime, long endTime,
PaymentApi paymentApi) {
    long deposit = calculateDeposit(newBike);
    long fee = calculateRentingFee(bike, startTime, endTime);
    boolean paymentSuccess = paymentApi.processTransaction(fee - deposit,
RETURN_BIKE_MESSAGE);
    if (!paymentSuccess) {
        return false;
    }
    return true;
}

```

Dự án cũng sử dụng Strategy pattern vào việc cài đặt các loại xe. Tên loại xe và các công thức tính tiền thuê xe phụ thuộc vào loại xe, do đó để chúng ta có thể liệt kê chúng interface cơ bản BikeType, và cài đặt cụ thể chúng trong các lớp BikeType cụ thể:

```

public class StandardBikeType extends TwoPhraseBikeType {
    @Override
    public String getDisplayName() {
        return "Standard bike";
    }
}

```



```

@Override
public String getDatabaseName() {
    return DATABASE_NAME;
}

@Override
public long calculateDeposit() {
    return 400000;
}
...
}

```

6.3.3 Abstract Factory

Abstract Factory là một design pattern thuộc nhóm creational, cho phép ta định nghĩa một lớp các Factory gần giống nhau có nhiệm vụ sản sinh các đối tượng gần giống nhau, có thể hoán đổi được cho nhau trong quá trình chạy.

Dự án áp dụng Abstract Factory vào việc cài đặt Factory tạo ra các đối tượng BikeType. Do chúng ta cần phải làm việc với các đối tượng BikeType giống nhau, và chúng ta không thực sự quan tâm đối tượng đó thuộc vào lớp cài đặt cụ thể nào, Abstract Factory có thể được áp dụng để cài đặt cơ chế khởi tạo BikeType đơn giản:

```

/**
 * An abstract Factory interface to retrieve a BikeType object. Implements the
 * Factory method pattern.
 */
public interface BikeTypeFactory {
    /**
     * Construct a BikeType object.
     *
     * @return The constructed object.
     */
    BikeType getBikeType();
}

```