HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

SOFTWARE DESIGN DOCUMENT

Project: Design and construct Ecobike Rental System

Group 10

Vũ Quang Trường	20194198
Nguyễn Huy Hoàng	20194058
Nguyễn Phúc Tân	20194163

Instructor: Nguyễn Thị Thu Trang

Subject: Software design and construction

Hanoi, 12/2022

Contents

A. Introduction 4
I. Purpose 4
II. Application overview 4
III. Intended audience 4
IV. Abbreviations 4
V. Reference 4
B. High level requirements 5
I. Use case diagram 5
II. Activity diagram 7
C. Use case specialization 14
I. Use case "Check info rented bike" 14
II. Use case "Pause renting" 17
III. Use case "Resume renting" 18
IV. Use case "Check info parking lot" 20
V. Use case "Check info bike in lot" 22
VI. Use case "Rent bike" 25
VII. Use case "Return bike" 28
VIII. Use case "Deposit" 31
D. Other requirements 35
E. Architectural Design 36
I. Analysis Class Diagram 36
II. Communication Diagram 40
III. Sequence Diagram 43
F. Detailed Design 47
I. Class Design 47
1. Class Diagram 47
2. Class Relationship 51
3. Class Description 51
II. Data Modeling 59
1. ERD 59
2. Relational Database Model 60

3. Database Description 60
III. Interface Design 62
1. Subsystem Interface Design 62
2. User Interface Design 66
G. Design Considerations 76
I. Goals and guildlines 76
1. Goals 76
2. Guildlines 76
II. Architectural Strategies 77
III. Coupling and Cohension 77
IV. Design principles 77
1. Single Responsibility Principle 77
2. Open/Closed Principle 77
3. Liskov Substitution Principle 78
4. Interface Segregation Principle 78
5. Dependency Inversion Principle 78
V. Design patterns 79
1. Strategy Pattern 79
H. Conclusion 80

A. Introduction

I. Purpose

Ecopark provides a bike rental service with various parking lots in the area. Customers can rent and return bike at the parking lot. The Ecobike Rental System is created to simulate this process.

This document provides the detailed descriptions for the Ecobike Rental System as well as the functional and non-functional requirements of the system.

II. Application overview

The application is connected to a database which stores the information abount parking lots, bikes, users and rental activities. Users can use this application to rent a bike at any parking lot. During the rental time, they can pause as well as resume the rental process. They can also return the rented bike at any parking lot. The application is also connected to a bank interface to process the rental payment.

III. Intended audience

This document is designed for:

- The system adminstrator
- The users.

IV. Abbreviations

V. Reference

[1] Systems Analysisand Design - Gary B. Shelly, Harry J. Rosenblatt,

Shelly Cashman Series, 2012

[2] Giáo trình phân tích và thiết kế hệ thống thông tin - Trần Đình Quế

[3] Slide bài giảng phân tích và thiết kế hệ thống - Nguyễn Hữu Đức

B. High level requirements

I. Use case diagram

The software consists of two actors: User and Interbank. The role of the Interbank is to handle transactions in the software.

Users can use the software to rent a bike if it is available for renting. During the time of renting, they can choose to manage the rented bike, which are pausing renting, resuming renting, checking its info and return the bike. Users can also check info of any parking lot and any of its bikes.

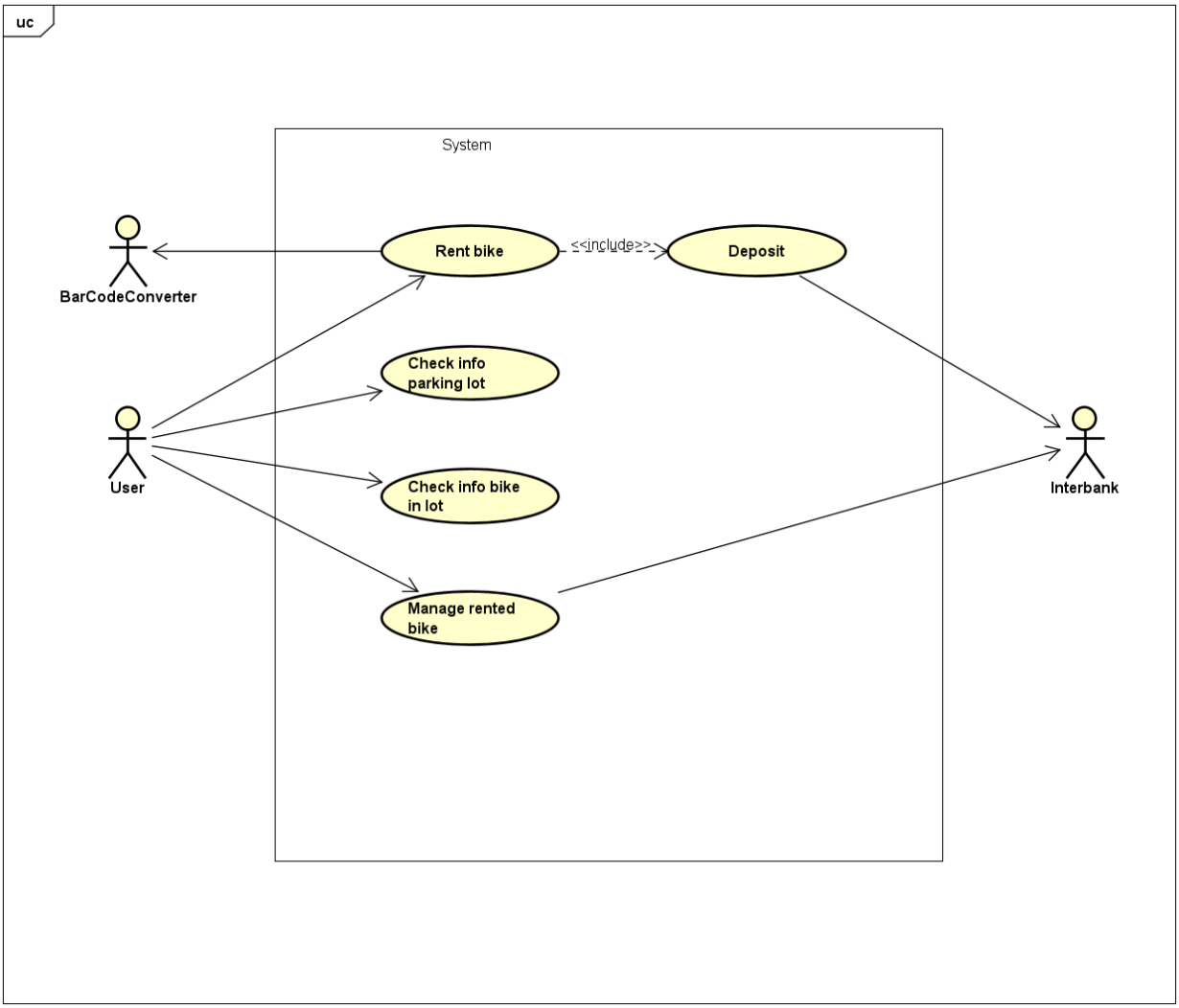


Figure 1: UC Diagram level 1

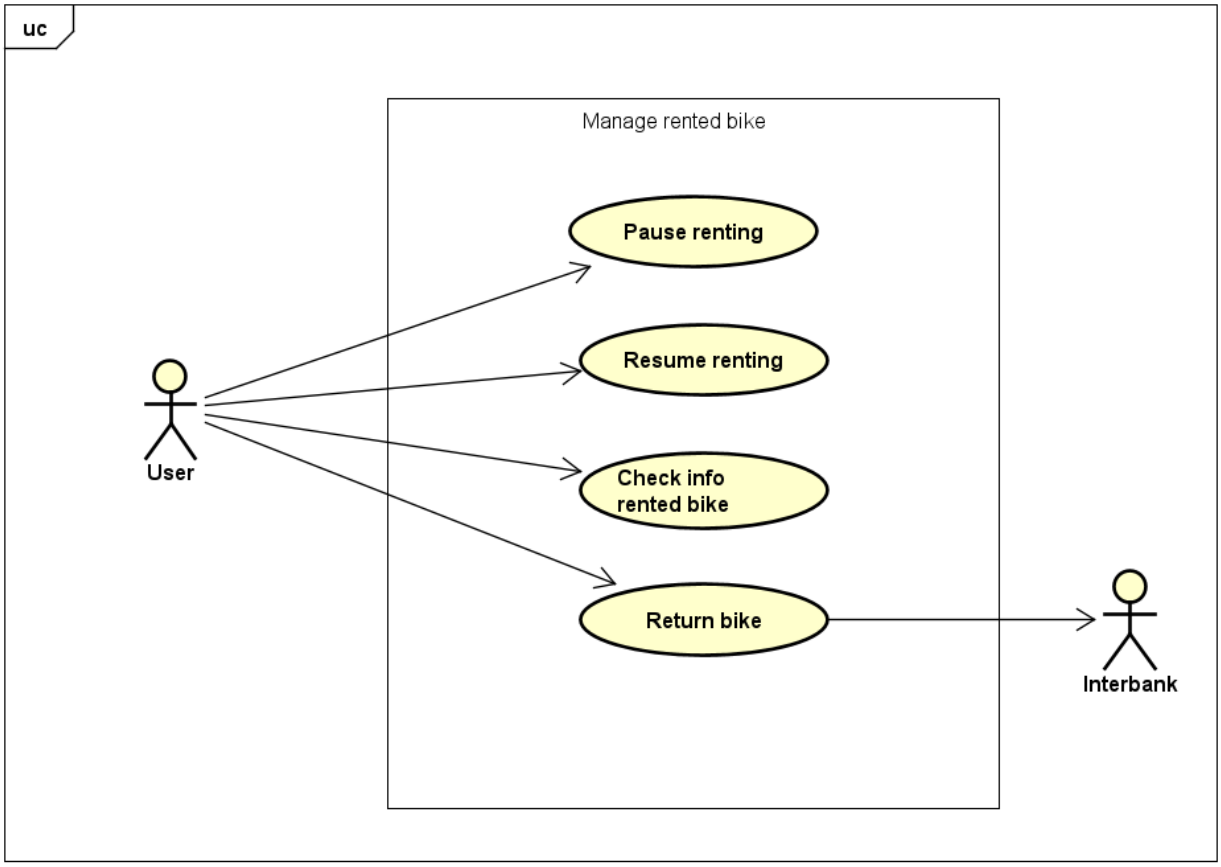


Figure 2: UC Diagram level 2 for UC “Manage rented bike”

II. Activity diagram

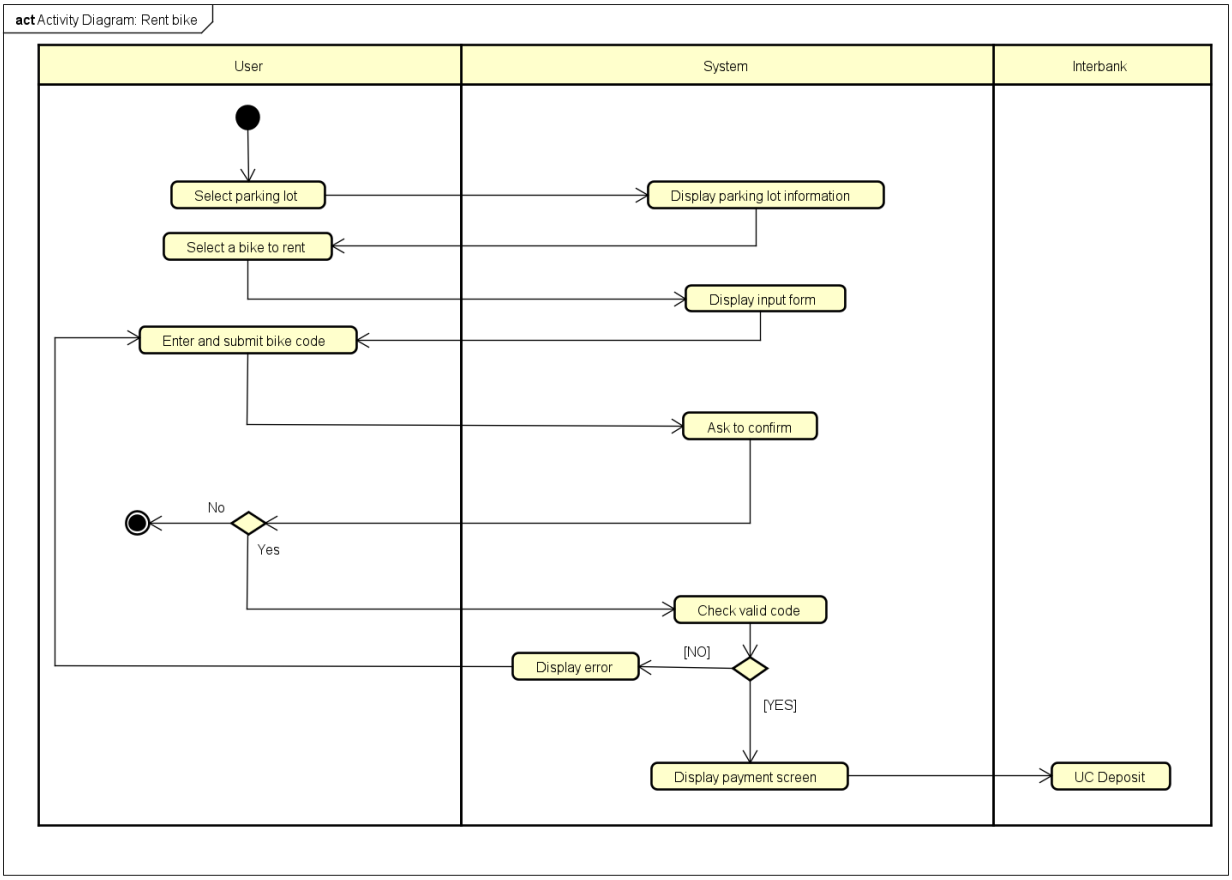


Figure 3: Activity diagram for use case ‘Rent bike’

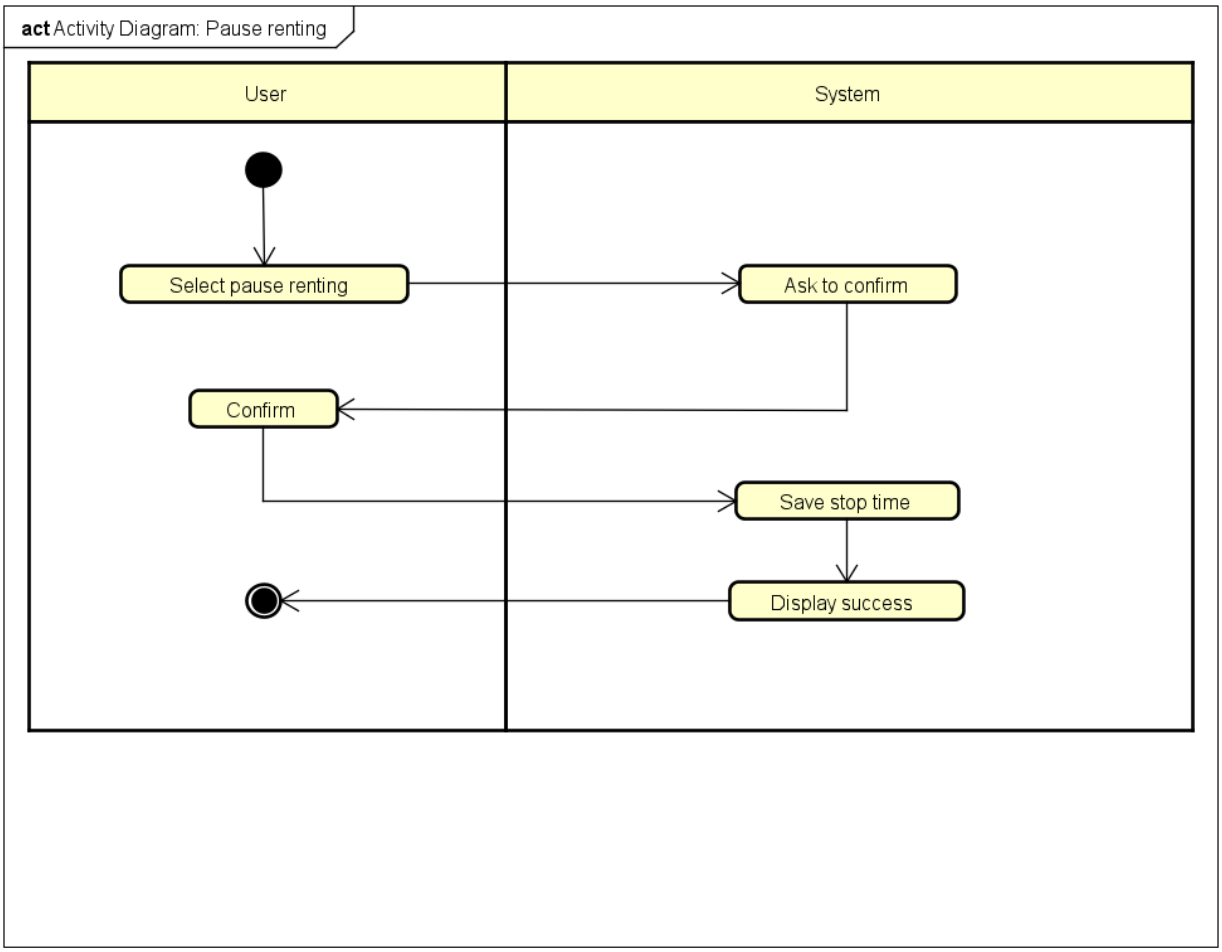


Figure 4: Activity diagram for use case ‘Pause renting’

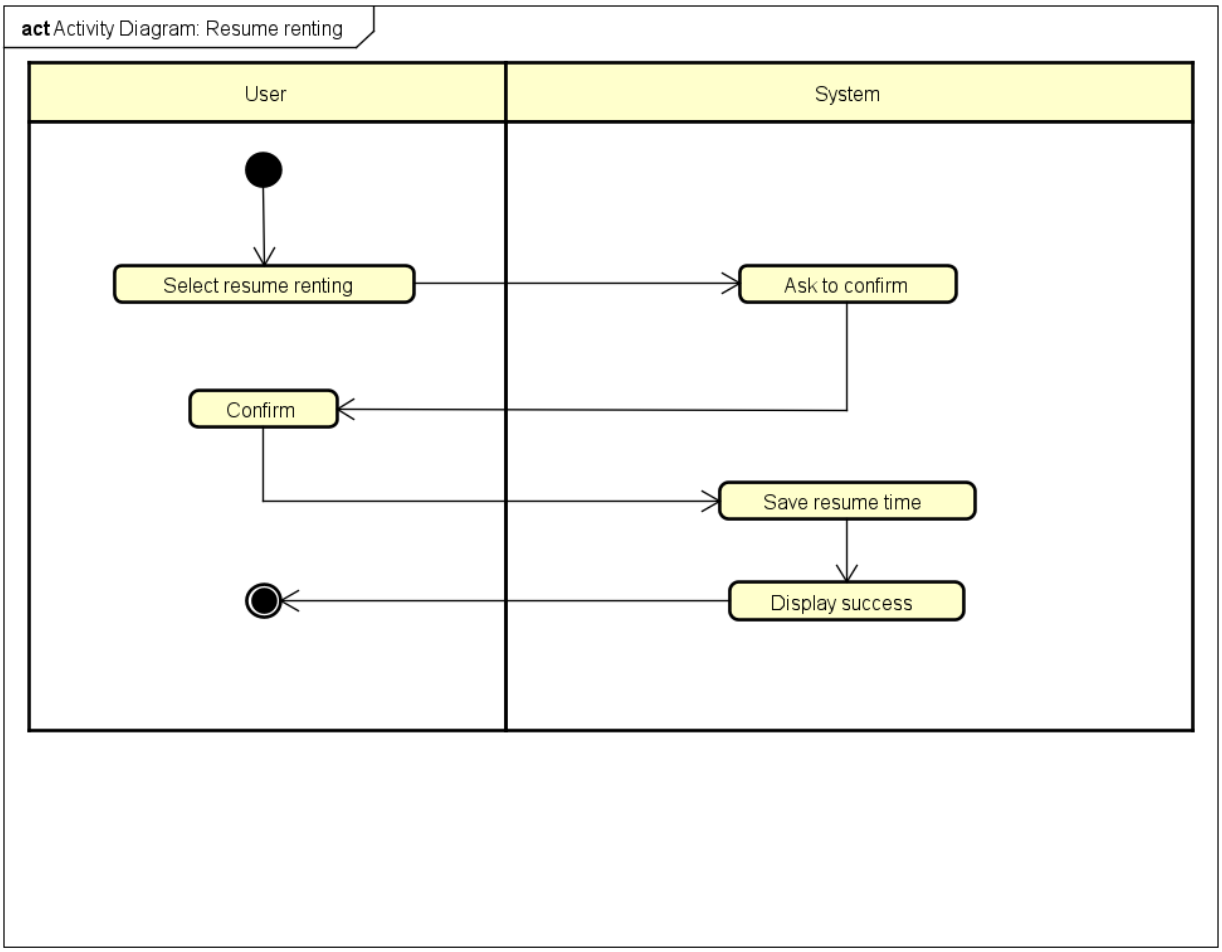


Figure 5: Activity diagram for use case ‘Resume renting’

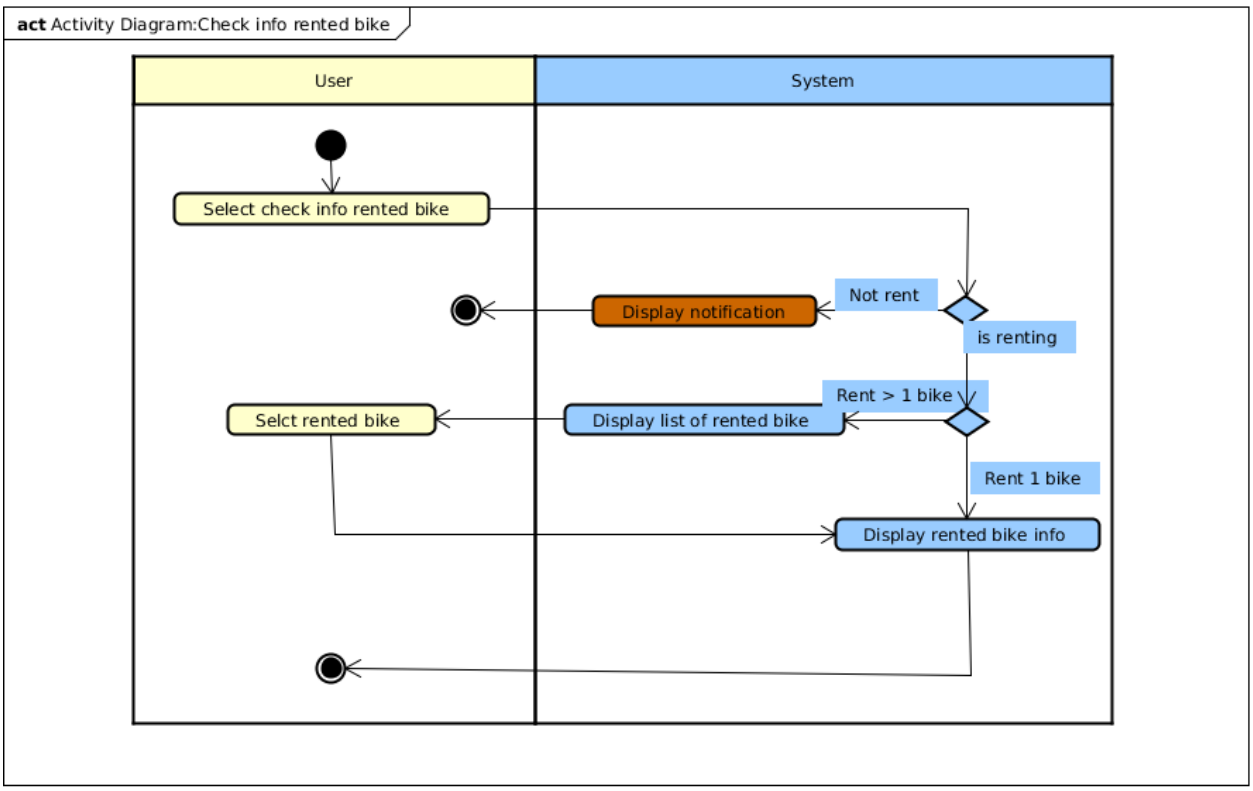


Figure 6: Activity diagram for use case ‘Check info rented bike’

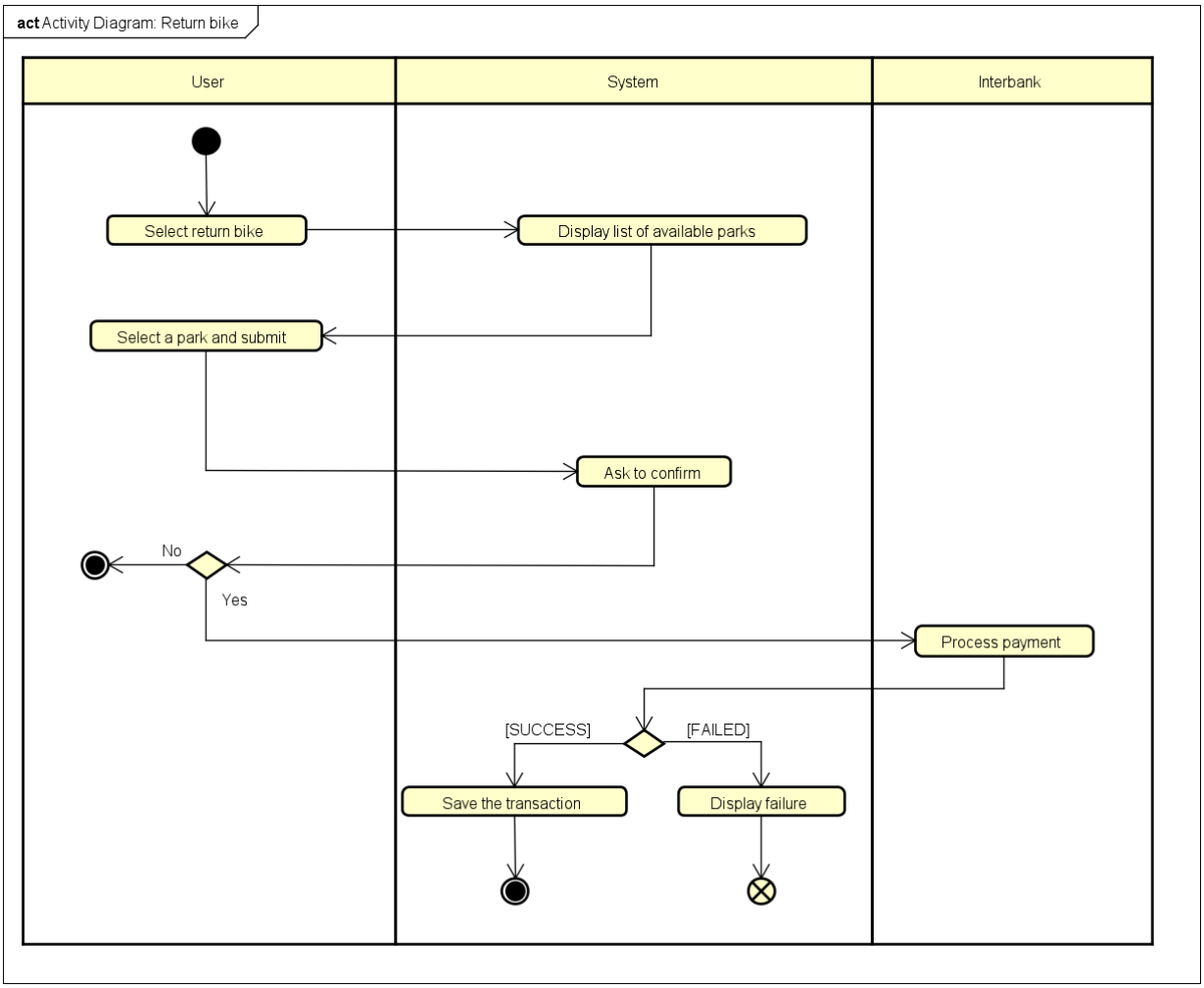


Figure 7: Activity diagram for use case ‘Return bike’

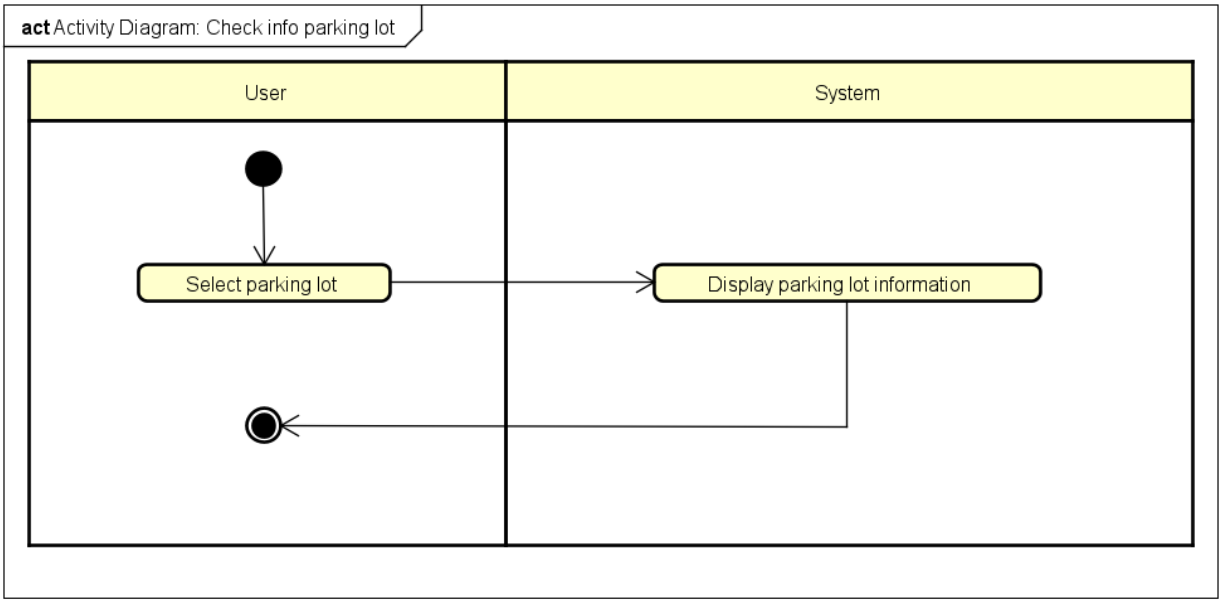


Figure 8: Activity diagram for use case 'Check info parking lot'

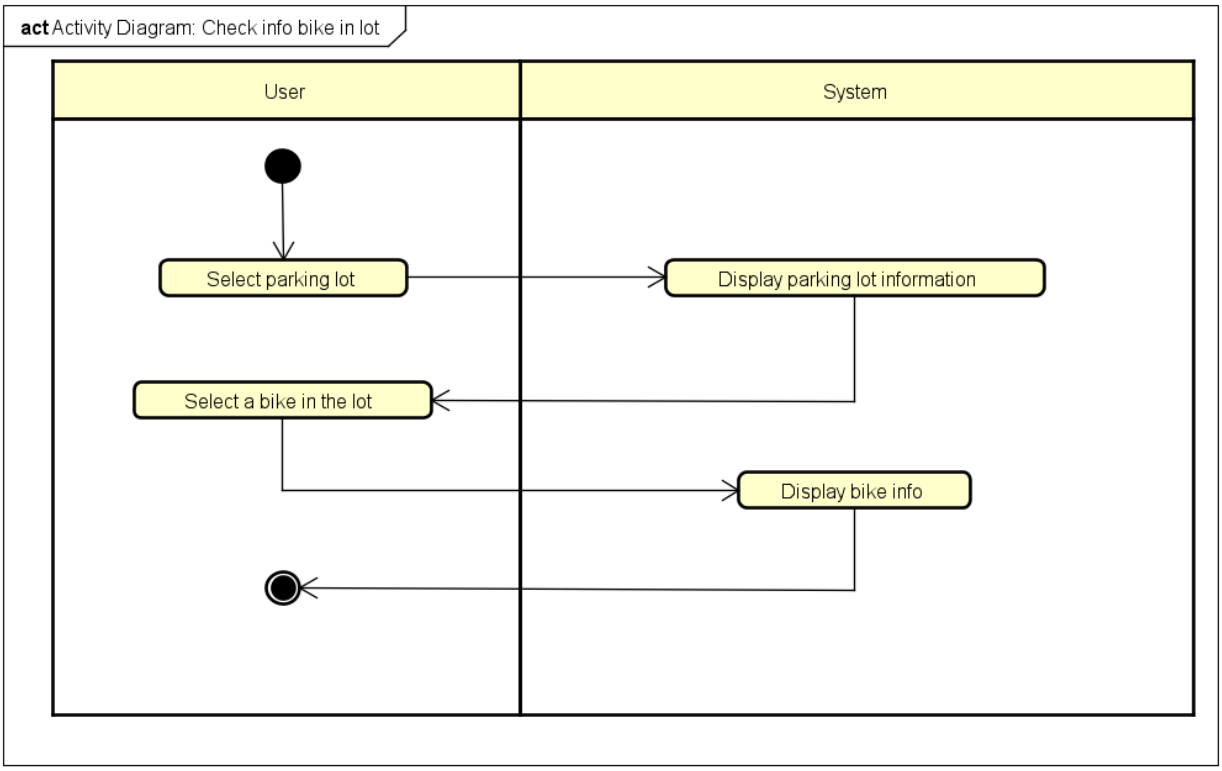


Figure 9: Activity diagram for use case 'Check info bike in lot'

C. Use case specialization

I. Use case “Check info rented bike”

1. Use case code

UC01

2. Brief description

UC “Check info rented bike” describes the interaction between customers

and Ecobike system when the customer wishes to check information of rented bike

3. Actors

3.1. User

4. Preconditions

There is an active network connection to the Internet

5. Basic Flow of Events

Step 1. The customer views rented bike

Step 2. The customer requests to check information of rented bike

Step 3. The Ecobike system checks the information of rented bike

Step 4. The Ecobike system displays the information of rented bike

6. Alternative flows

Table X. Alternative flow of events for UC “Check info rented bike”

No	Location	Condition	Action	Resume location
1	At step 1	If the customer is not renting any bike	● The Ecobike system show notification	At step 1

7. Input data

8. Output data

Table X. Output data of displaying information of rented bike

No	Data fields	Description	Display format	Example
1	ID	ID of bike		B0001
2	Type	Type of bike		Normal bike
3	Name			
4	Country	Country of manufacture		

5	Color	Color of bike		White
6	Date of manufacture			01/01/2022
7	Price	Price of rented bike		1000000 VND
8	Timeline	Display list start time, pause time, resume time in chronological order		
9	Total rented time			
10	Fee	Total fees of rented bike		
11	Battery	Remaining battery of the bike		95%
12	License plate number			MD1-20221

9. Postconditions

II. Use case “Pause renting”

1. Use case code

UC02

2. Brief description

UC “Pause renting” describes the interaction between customers

and Ecobike system when the customer wishes to pause renting

3. Actors

3.1. User

4. Preconditions

There is an active network connection to the Internet

5. Basic Flow of Events

Step 1. The customer views rented bike

Step 2. The customer requests to pause renting

Step 3. The Ecobike system update the information of rented bike

Step 4. The Ecobike system displays success message

6. Alternative flows

Table X. Alternative flow of events for UC “Pause renting”

No	Location	Condition	Action	Resume location

1	At step 1	If the customer is not renting any bike	● The Ecobike system show notification “NO RENTED BIKE”	At step 1
2	At step 3	If system fails to update	● The Ecobike system show notification “FAIL”	At step 2

7. Input data

8. Output data

9. Postconditions

The logs have been updated accordingly.

III. Use case “Resume renting”

1. Use case code

UC03

2. Brief description

UC “Resume renting” describes the interaction between customers

and Ecobike system when the customer wishes to resume renting.

3. Actors

3.1. User

4. Preconditions

There is an active network connection to the Internet and customer is stopping bike rental

5. Basic Flow of Events

Step 1. The customer views rented bike

Step 2. The customer requests to resume renting

Step 3. The Ecobike system update the information of rented bike

Step 4. The Ecobike system displays success message

6. Alternative flows

Table X. Alternative flow of events for UC "Resume renting"

No	Location	Condition	Action	Resume location
1	At step 1	If the customer is not renting a car	● The Ecobike system show notification "NO RENTED BIKE"	At step 1
2	At step 3	If system fails to update	● The Ecobike system show notification "FAIL"	At step 2

7. Input data

8. Output data

9. Postconditions

The logs have been updated accordingly.

IV. Use case “Check info parking lot”

1. Use case code

UC04

2. Brief description

UC “Check info parking lot” describes the interaction between customers

and Ecobike system when the customer wishes to check information of rented bike

3. Actors

3.1. User

4. Preconditions

There is an active network connection to the Internet

5. Basic Flow of Events

Step 1. The customer views the available parking lot.

Step 2. The customer clicks on a parking lot.

Step 3. The Ecobike system checks the information of the lot.

Step 4. The Ecobike system displays the information of the lot.

6. Alternative flows

7. Input data

8. Output data

Table XI. Output data of displaying information of a parking lot

No	Data fields	Description	Display format	Example
1	ID	ID of the parking lot	PL__	PL01
2	Location			1st Giai Phong, Hanoi
3	Name			Giai Phong Parking Lot
4	Distance	Distance from the lot to the customer location		3 km
5	Travelling time	Travelling time (on foot) from the lot to the customer location		20 min

3	Total number of slots		Int	50
4	Number of empty slots		Int	10
5	List of bikes	List of bikes in the parking lot as well as their status (rented or not)		

9. Postconditions

V. Use case “Check info bike in lot”

1. Use case code

UC05

2. Brief description

UC “Check info bike in lot” describes the interaction between customers

and Ecobike system when the customer wishes to check information of a bike in a particular parking lot.

3. Actors

3.1. User

4. Preconditions

There is an active network connection to the Internet

5. Basic Flow of Events

Step 1. The customer views the available parking lot.

Step 2. The customer clicks on a parking lot.

Step 3. The Ecobike system checks the information of the lot.

Step 4. The Ecobike system displays the information of the lot.

Step 5. The customer clicks on a bike in the bike list displayed.

Step 6. The Ecobike system checks the information of the selected bike.

Step 7. The Ecobike system displays the information of the bike.

6. Alternative flows

7. Input data

8. Output data

Table XII. Output data of displaying information of a bike in a parking lot

No	Data fields	Description	Display format	Example

1	ID	ID of bike		B0001
2	Type	Type of bike		Normal bike
3	Name			
4	Country	Country of manufacture		
5	Color	Color of bike		White
6	Date of manufacture			01/01/2022
7	Price	Price of rented bike		1000000 VND
8	Battery	Remaining battery of the bike		95%
9	Maximum usage time	Maximum usage time corresponding to the remaining battery		2 hours
10	License plate number			MD1-20221

11	Slot ID			S01
12	Is rented	Boolean		False

9. Postconditions

VI. Use case “Rent bike”

1. Use case code

UC06

2. Brief description

UC “Rent bike” allows customers to rent a bike.

3. Actors

3.1. User

4. Preconditions

There is an active network connection to the Internet.

5. Basic Flow of Events

Step 1. The customer selects a park.

Step 2. The Ecobike system displays the park information.

Step 3. The customer chooses to rent bike.

Step 4. The Ecobike system displays the input form.

Step 5. The customer enters and submits the barcode of the bike they want to rent.

Step 6. The Ecobike system calls API of barcode converter to convert barcode to bike ID.

Step 7. The Ecobike system ensures that the customer want to rent that bike.

Step 8. User confirms.

Step 9. The Ecobike system display payment screen and call Deposit Use Case.

6. Alternative flows

Table VII. Alternative flow of events for UC "Rent bike"

No	Location	Condition	Action	Resume location
1	At step 6	If the code submitted is invalid or non-existent	● The Ecobike system show notification "BIKE CODE INVALID" or "BIKE CODE NON-EXISTENT"	At step 5
2	At step 8	If the customer click on "NO"	●	At step 5

7. Input data

Table VIII. Input data of rent bike

No	Data fields	Description	Mandatory	Valid Condition	Example
1	Barcode	Barcode of bike	Yes		B0001

8. Output data

Table IX. Output data of rent bike

No	Data fields	Description	Display format	Example
1	Transaction fee	Transaction fee for renting deposit		100,000
2	Bike license plate number			BK23333

3	Battery left			77%

9. Postconditions

VII. Use case “Return bike”

1. Use case code

UC07

2. Brief description

UC “Return bike” allows customers to return a rented bike.

3. Actors

3.1. User

4. Preconditions

There is an active network connection to the Internet and the customer is renting a bike.

5. Basic Flow of Events

Step 1. The customer chooses to return bike.

Step 2. The Ecobike system displays the list of available parking lots.

Step 3. The customer selects a parking lot.

Step 4. The Ecobike system ensures that the customer wants to return the rented bike to the selected parking lot.

Step 5. The Interbank processes payment.

Step 6. The Ecobike system saves the transaction and lock the bike.

6. Alternative flows

Table X. Alternative flow of events for UC "Return bike"

No	Location	Condition	Action	Resume location
1	At step 4	If the customer click on "NO"	●	At step 3
2	At step 5	If transaction fails	● The Ecobike system show reason of failing transaction	At step 3

7. Input data

8. Output data

Table XII. Output data of return bike

No	Data fields	Description	Mandatory	Display format	Example
1	Transaction fee	Renting fee and returning deposit			100,000
2	Transaction ID				0123
3	Card number		Yes		CARD1234
4	Card owner		Yes		Nguyen Phuc Tan
5	Bank		Yes		MB Bank
6	Transaction description		No		Return deposit and charge rental fee
7	Rental start			hh:mm dd/mm/yyyy	08:00 12/12/2022

8	Rental end			hh:mm dd/mm/yyyy	10:00 12/12/2022
9	Bike license plate number				BD-12344

9. Postconditions

VIII. Use case “Deposit”

1. Use case code

UC08

2. Brief description

UC “Deposit” allows customers to deposit for the bike they want to rent.

3. Actors

3.1. User

3.2. Interbank

4. Preconditions

There is an active network connection to the Internet and the customer is renting a bike.

5. Basic Flow of Events

Step 1. The customer enters card info.

Step 2. The Ecobike system validates the info.

Step 3. The Interbank processes the transaction.

Step 4. The Ecobike system saves the transaction info and unlocks the bike.

Step 5. The Ecobike system display success message.

6. Alternative flows

Table XIII. Alternative flow of events for UC "Deposit"

No	Location	Condition	Action	Resume location
1	At step 2	If the info is invalid	● The Ecobike system displays error message	At step 1
2	At step 3	If transaction fails	● The Ecobike system show reason of failing transaction	At step 1

7. Input data

Table XIV. Input for UC "Deposit"

No	Data fields	Description	Mandatory	Valid Condition	Example
1	Card number		Yes		CARD1234
2	Card owner		Yes		Nguyen Phuc Tan
3	Bank		Yes		MB Bank
4	Expire date		Yes	MM/YY	09/23
5	PIN		Yes		090909
6	Transaction description		No		Deposit bike B001

8. Output data

Table XV. Output for UC "Deposit"

No	Data fields	Description	Mandatory	Valid Condition	Example

1	Card number				CARD1234
2	Card owner				Nguyen Phuc Tan
3	Bank				MB Bank
4	Expire date			MM/YY	09/23
6	Transaction description				Deposit bike B001
7	Transaction fee				100,000 $
8	Rental start			hh:mm dd/mm/yyyy	08:00 12/12/2022

9. Postconditions

D. Other requirements

The Ecobike system has to satisfy the use cases that require fast response. The system needs to ensure the security of customer information such as travel schedules, credit cards,... In addition, the software works with transactions related to the economic interests of related parties, so it is necessary to ensure accuracy and reliability as well as consistency across the entire system.

The main properties that the system needs to ensure:

- **Easy to use:** The system, once created, must provide an easy-to-use user interface. Consistent operations, limiting user input steps where possible
- **Response Time:** The maximum response time of the system is 1 second at normal or 2 seconds at peak.
- **Availability:** The system must always satisfy the needs of customers 24/7
- **Simultaneity:** The software can serve 100 users at a time without any change in performance.
- **Fault tolerance:** The system can operate for 200 hours continuously without error. It can be back to normal within 2 hours after the fault
- **Cross-platform compatibility**
- **Security:** The system must ensure the safety of user data and personal transaction information
- **Testability:** The recommendation system must keep a history of resolved orders
- **Maintenance:** Due to the increasing and diverse customer needs, as well as the explosive development of new technologies, the system needs to be easy to maintain and upgrade.
- **Design constraints:**

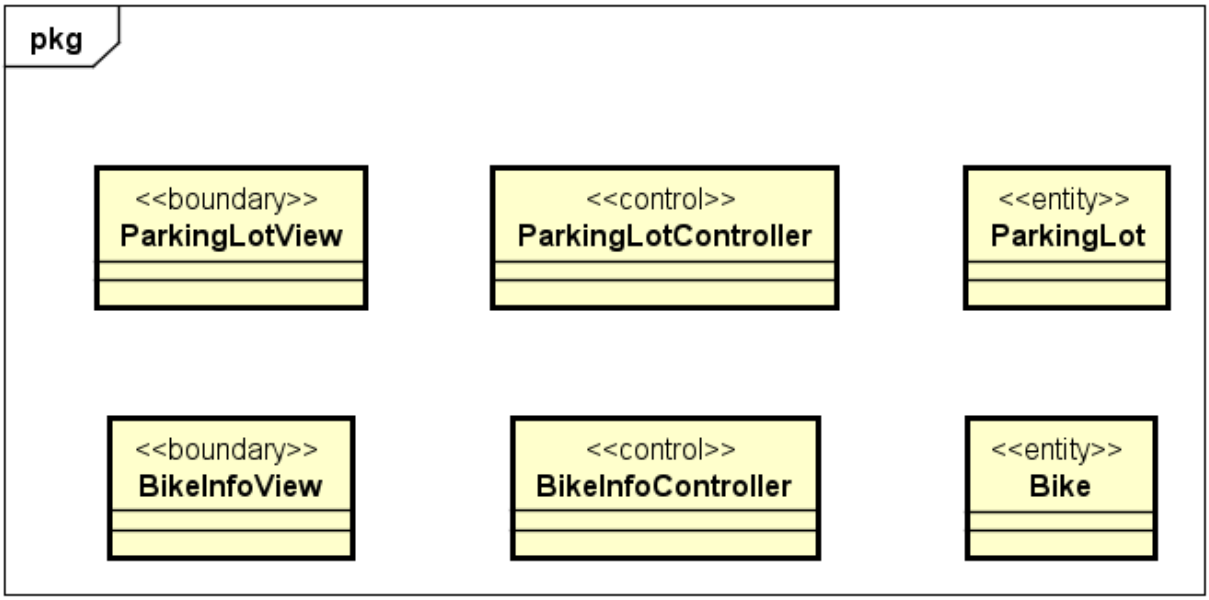
◦ Right-aligned number, displayed thousands separated by dots.

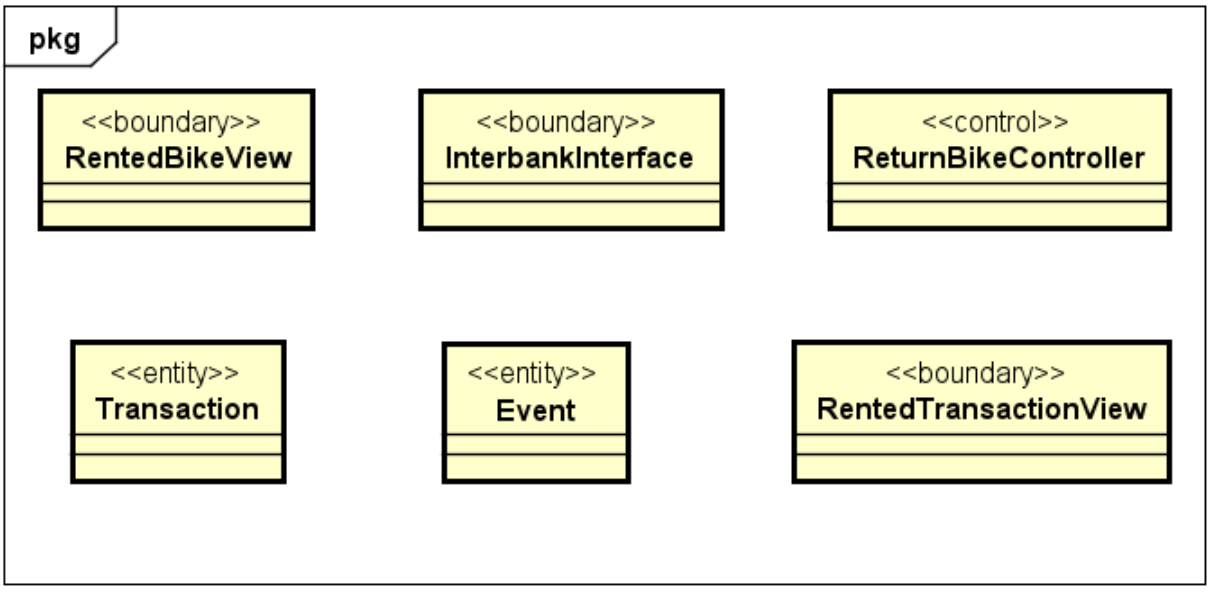
◦ Left aligned letters.

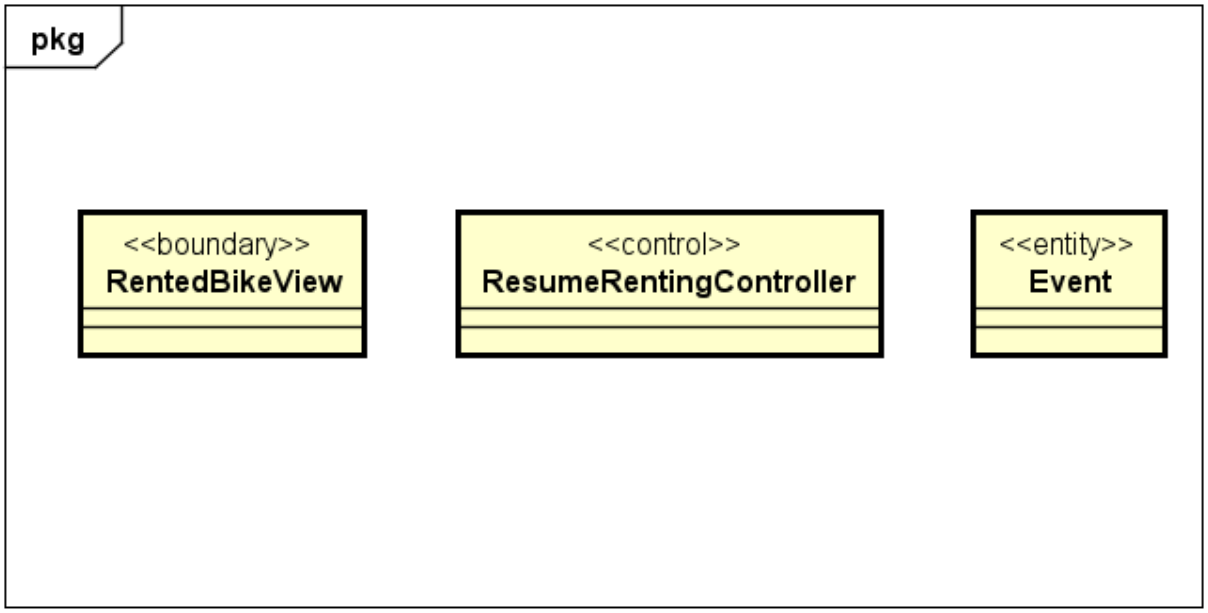
◦ Font: Arial 16, black color.

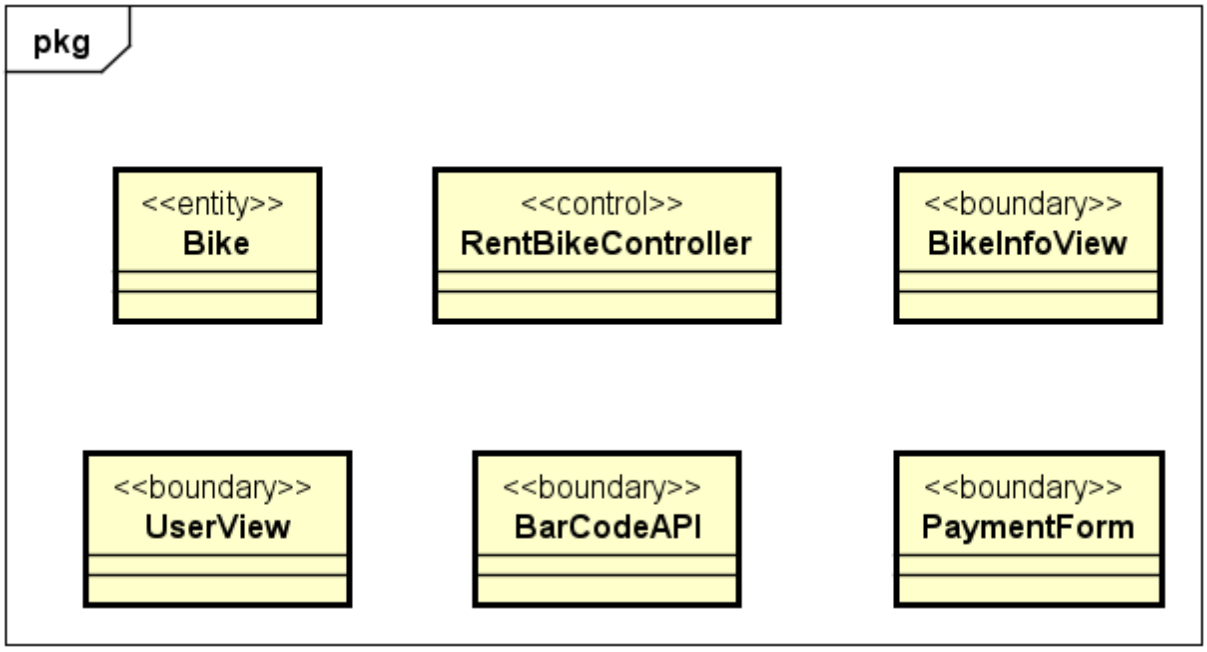
E. Architectural Design

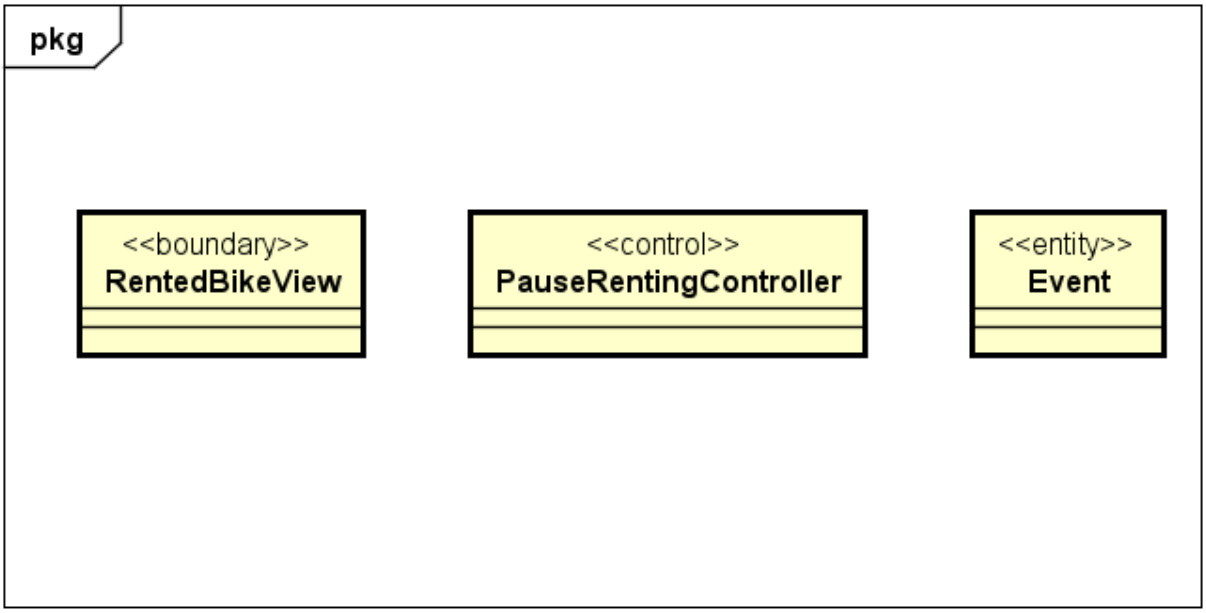
I. Analysis Class Diagram

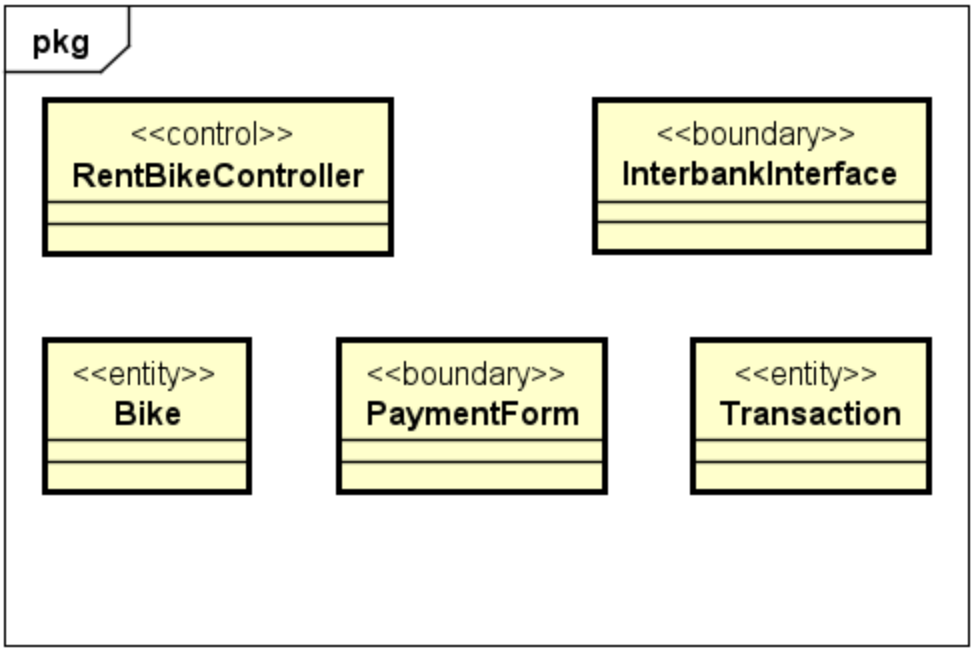


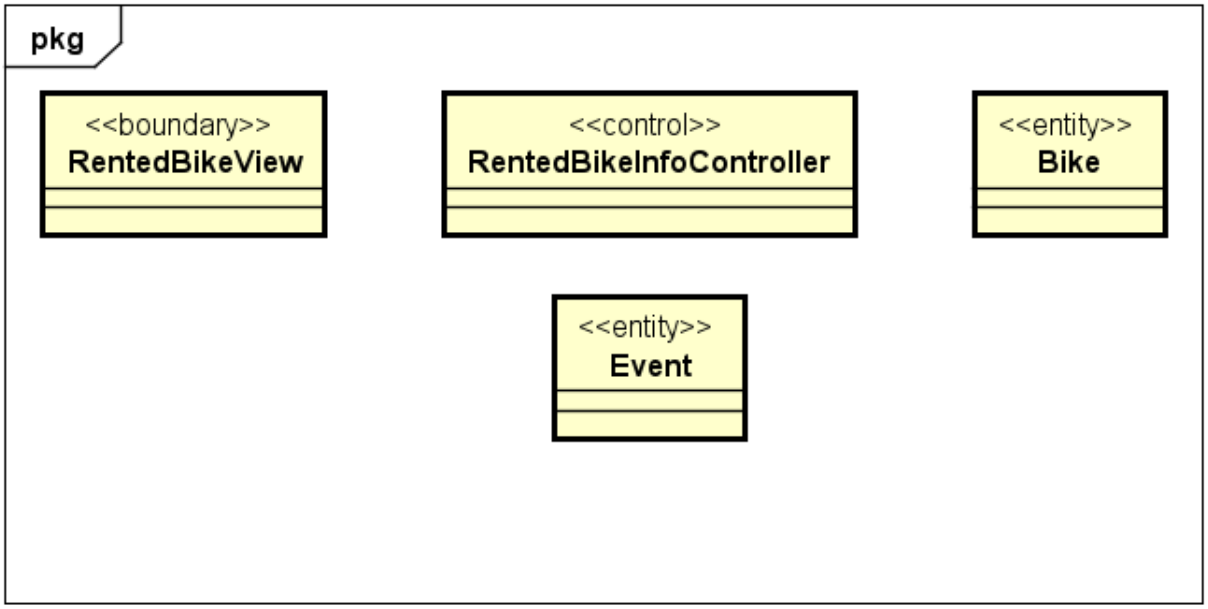


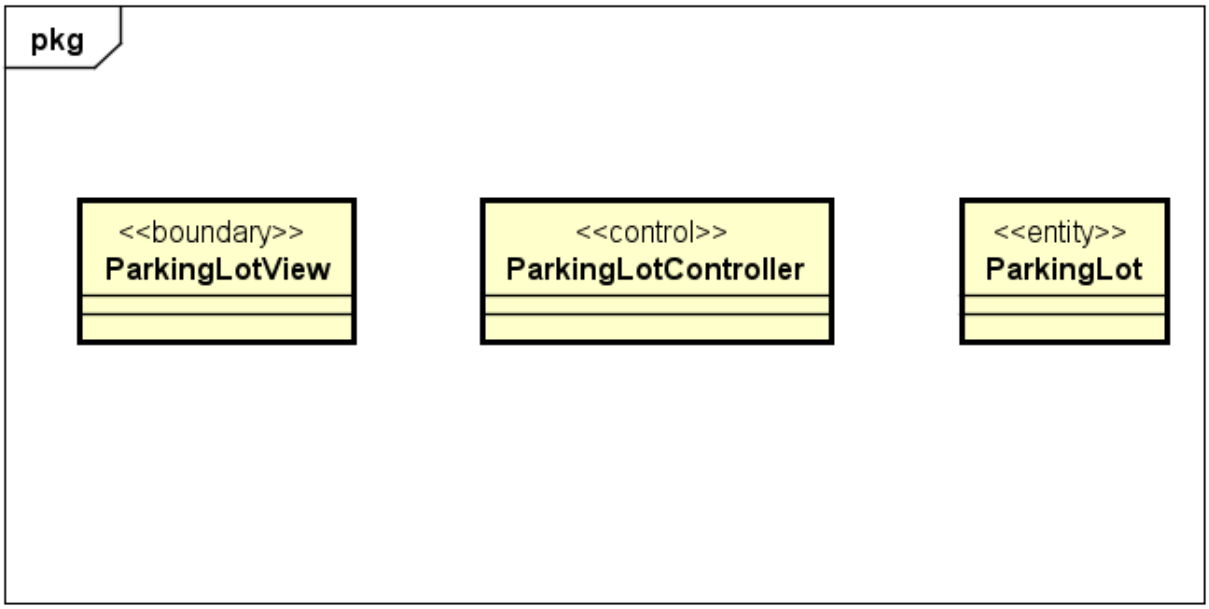

pkg
<<boundary>>
RentedBikeView
<<control>>
ResumeRentingController
<<entity>>
Event


pkg
<<entity>>
Bike
<<control>>
RentBikeController
<<boundary>>
BikeInfoView
<<boundary>>
UserView
<<boundary>>
BarCodeAPI
<<boundary>>
PaymentForm

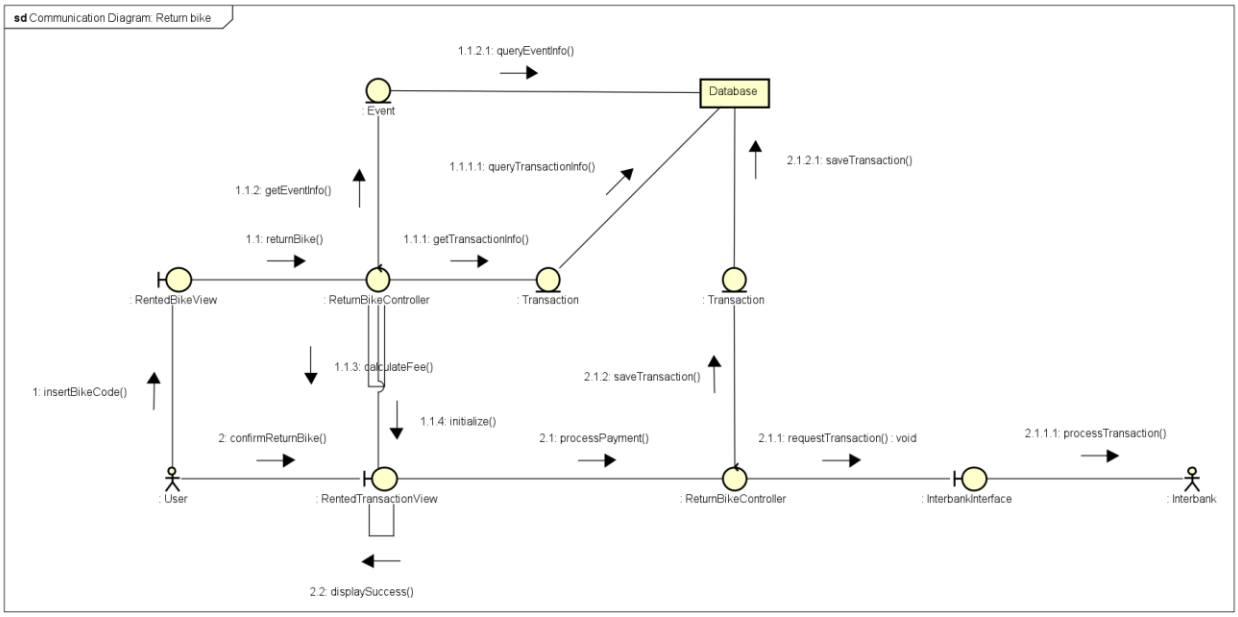

pkg
<<boundary>>
RentedBikeView
<<control>>
PauseRentingController
<<entity>>
Event

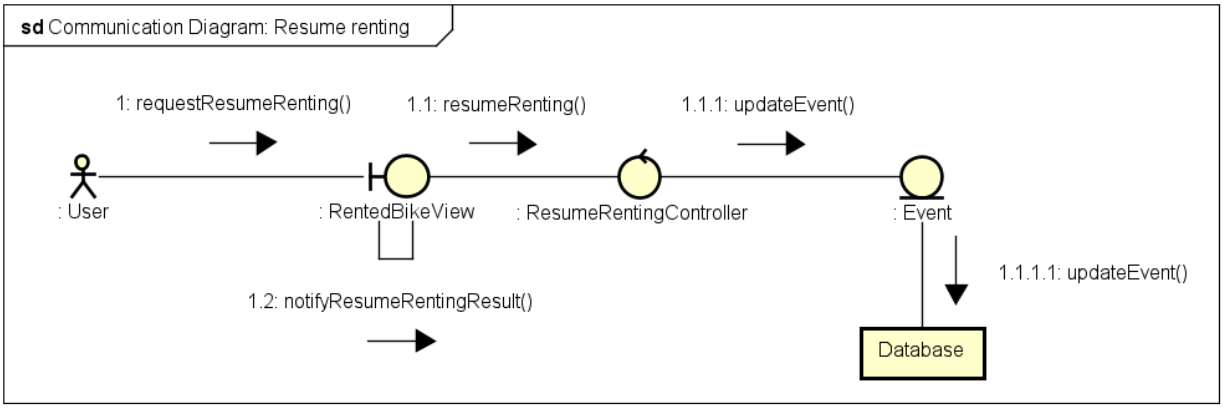

pkg
<<control>>
RentBikeController
<<boundary>>
InterbankInterface
<<entity>>
Bike
<<boundary>>
PaymentForm
<<entity>>
Transaction

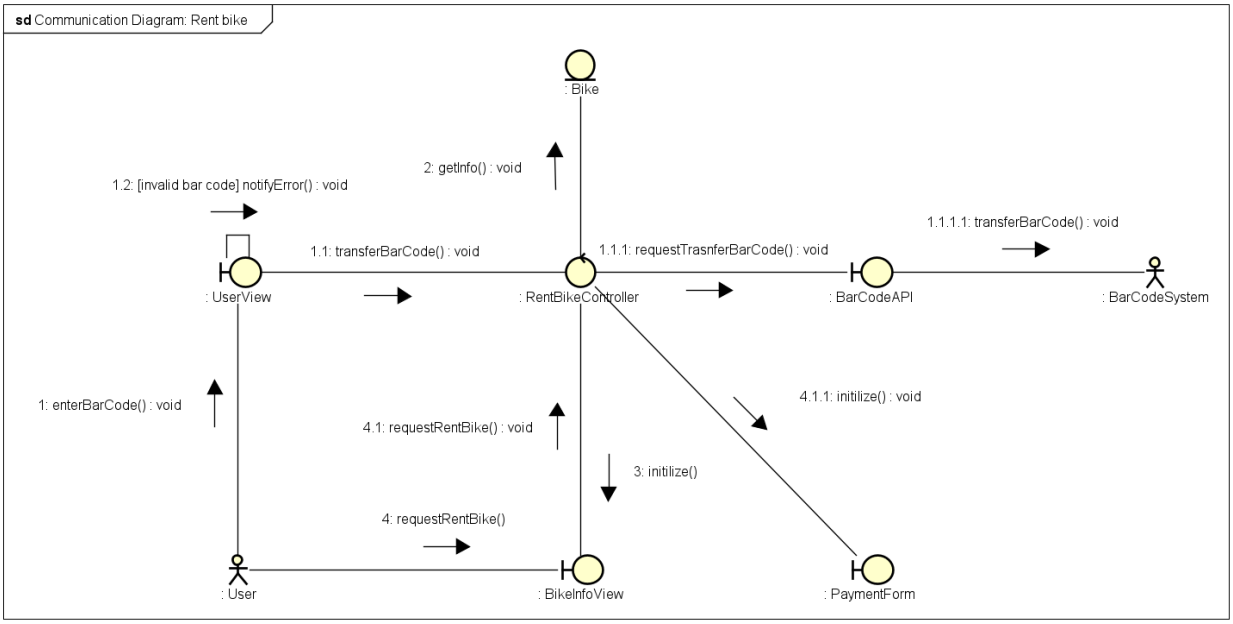

pkg
<<boundary>>
RentedBikeView
<<control>>
RentedBikeInfoController
<<entity>>
Bike
<<entity>>
Event

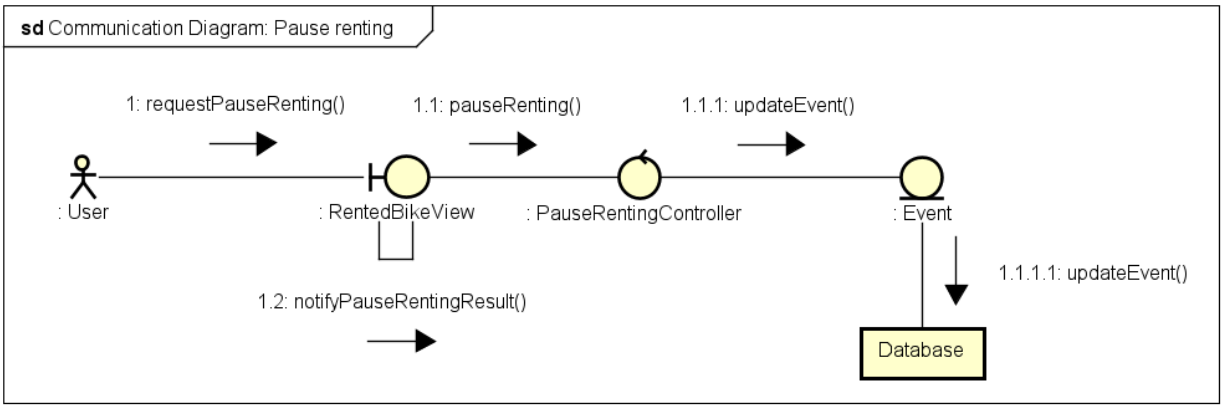

pkg
<<boundary>>
ParkingLotView
<<control>>
ParkingLotController
<<entity>>
ParkingLot

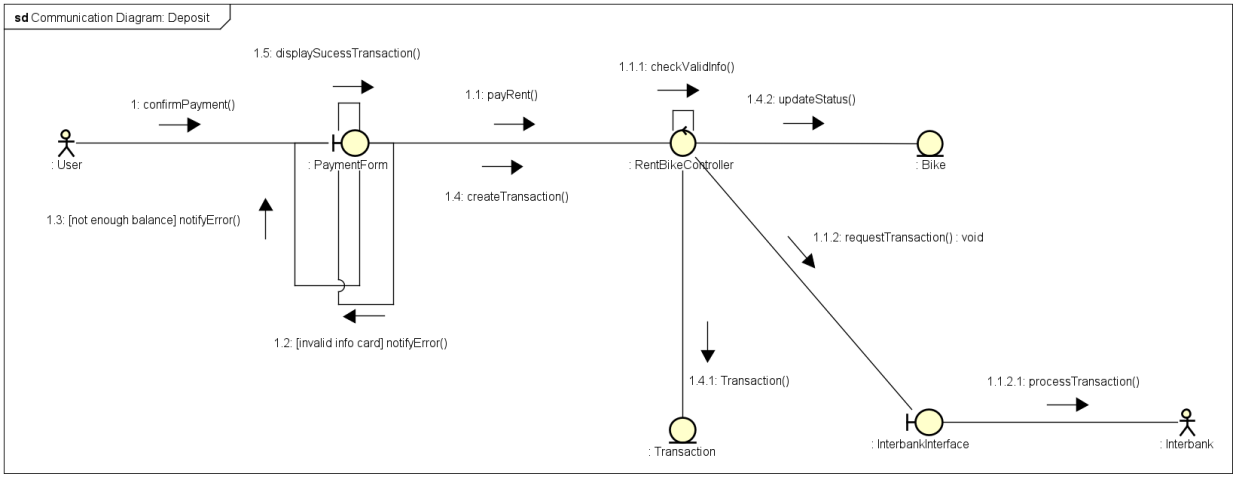
II. Communication Diagram

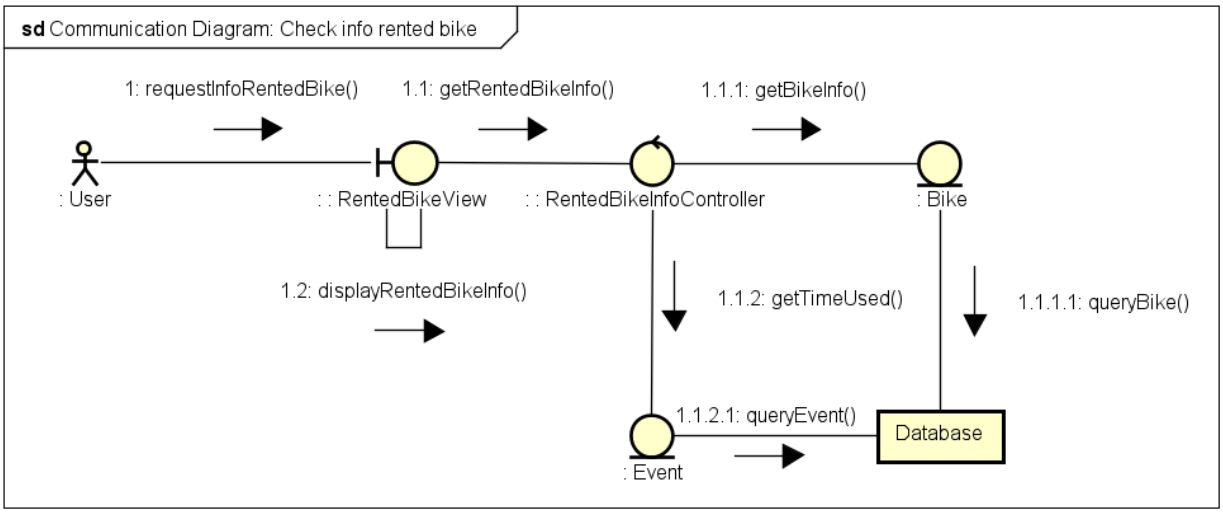


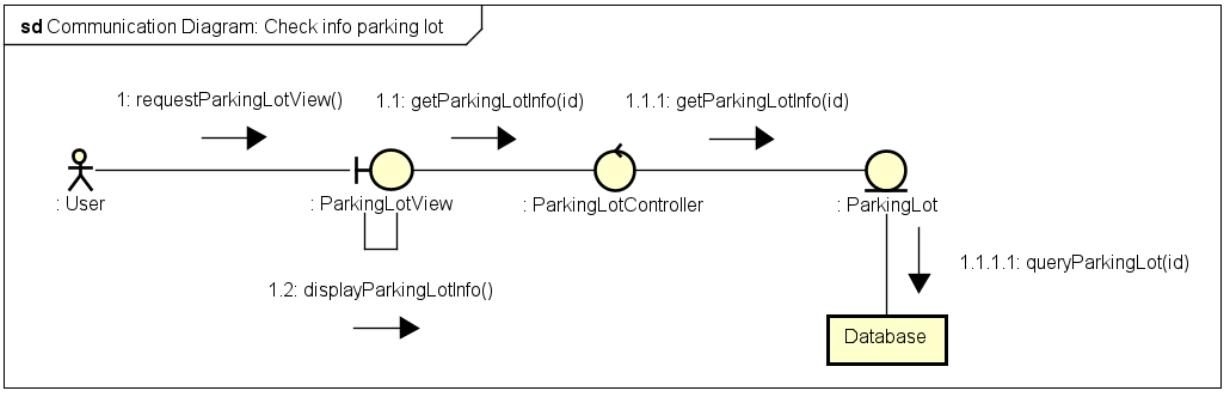


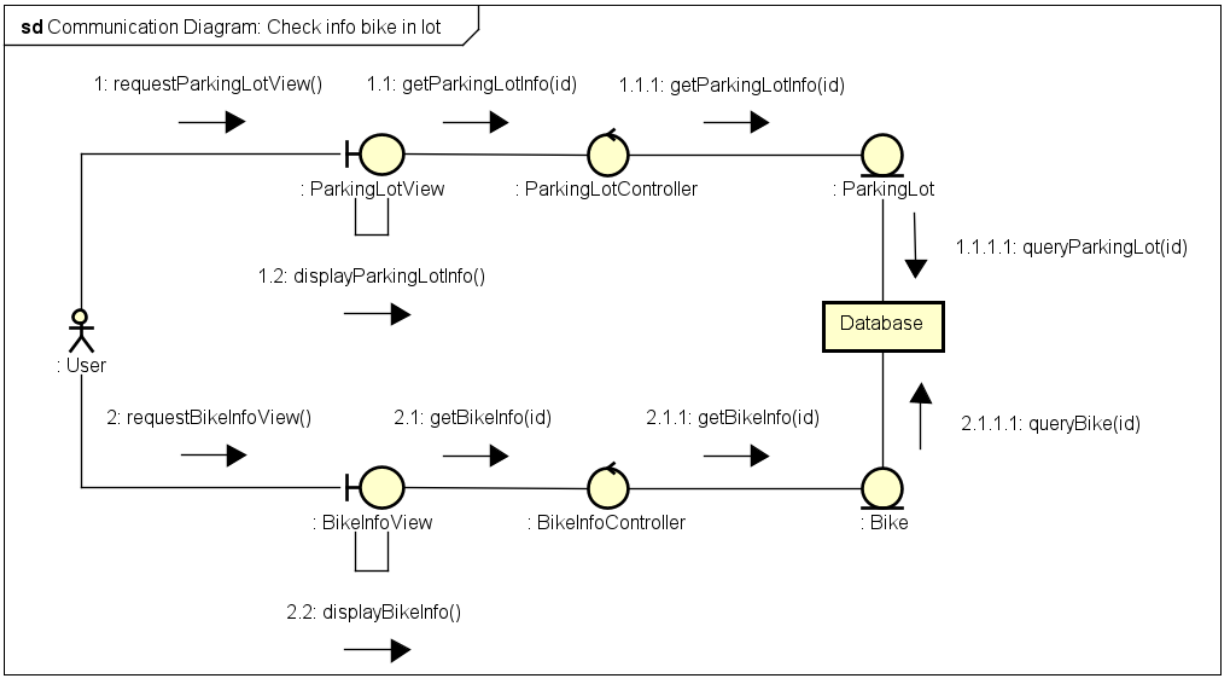

sd Communication Diagram: Rent bike
: Bike
2: getInfo() : void
1.2: [invalid bar code] notifyError() : void
1.1.1.1: transferBarCode() : void
1.1: transferBarCode() : void
1.1.1: requestTrasnferBarCode() : void
: UserView
: RentBikeController
: BarCodeAPI
: BarCodeSystem
1: enterBarCode() : void
4.1.1: initilize() : void
4.1: requestRentBike() : void
3: initilize()
4: requestRentBike()
: User
: BikeInfoView
: PaymentForm


sd Communication Diagram: Pause renting
1: requestPauseRenting()
1.1: pauseRenting()
1.1.1: updateEvent()
: User
: RentedBikeView
: PauseRentingController
: Event
1.1.1.1: updateEvent()
1.2: notifyPauseRentingResult()
Database

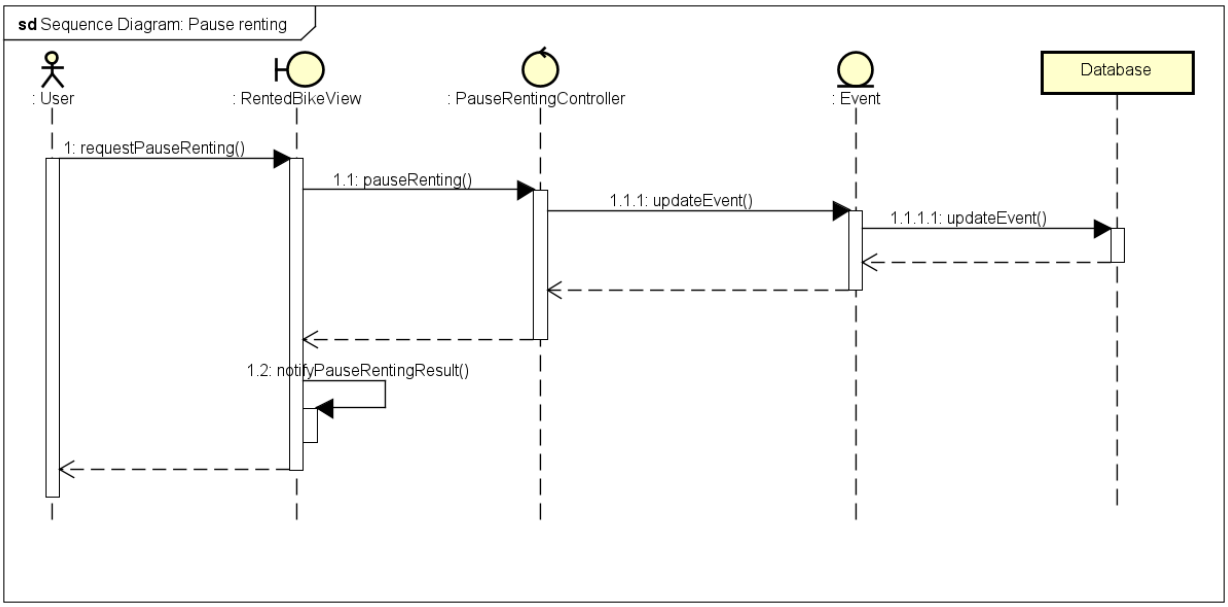

sd Communication Diagram: Deposit
1.5: displaySucessTransaction()
1.1.1: checkValidInfo()
1: confirmPayment()
1.1: payRent()
1.4.2: updateStatus()
: User
: PaymentForm
: RentBikeController
: Bike
1.4: createTransaction()
1.3: [not enough balance] notifyError()
1.1.2: requestTransaction() : void
1.2: [invalid info card] notifyError()
1.4.1: Transaction()
1.1.2.1: processTransaction()
: Transaction
: InterbankInterface
: Interbank

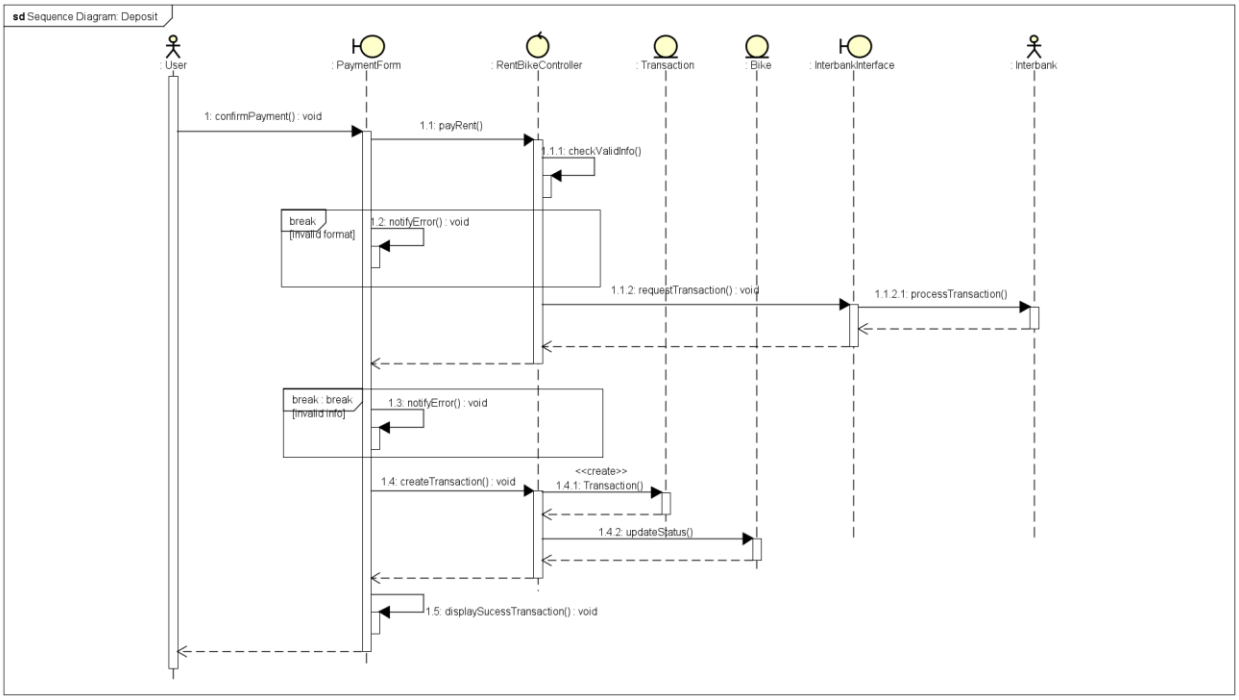

sd Communication Diagram: Check info rented bike
1: requestInfoRentedBike()
1.1: getRentedBikeInfo()
1.1.1: getBikeInfo()
: User
: : RentedBikeView
: : RentedBikeInfoController
: Bike
1.2: displayRentedBikeInfo()
1.1.2: getTimeUsed()
1.1.1.1: queryBike()
1.1.2.1: queryEvent()
Database
: Event

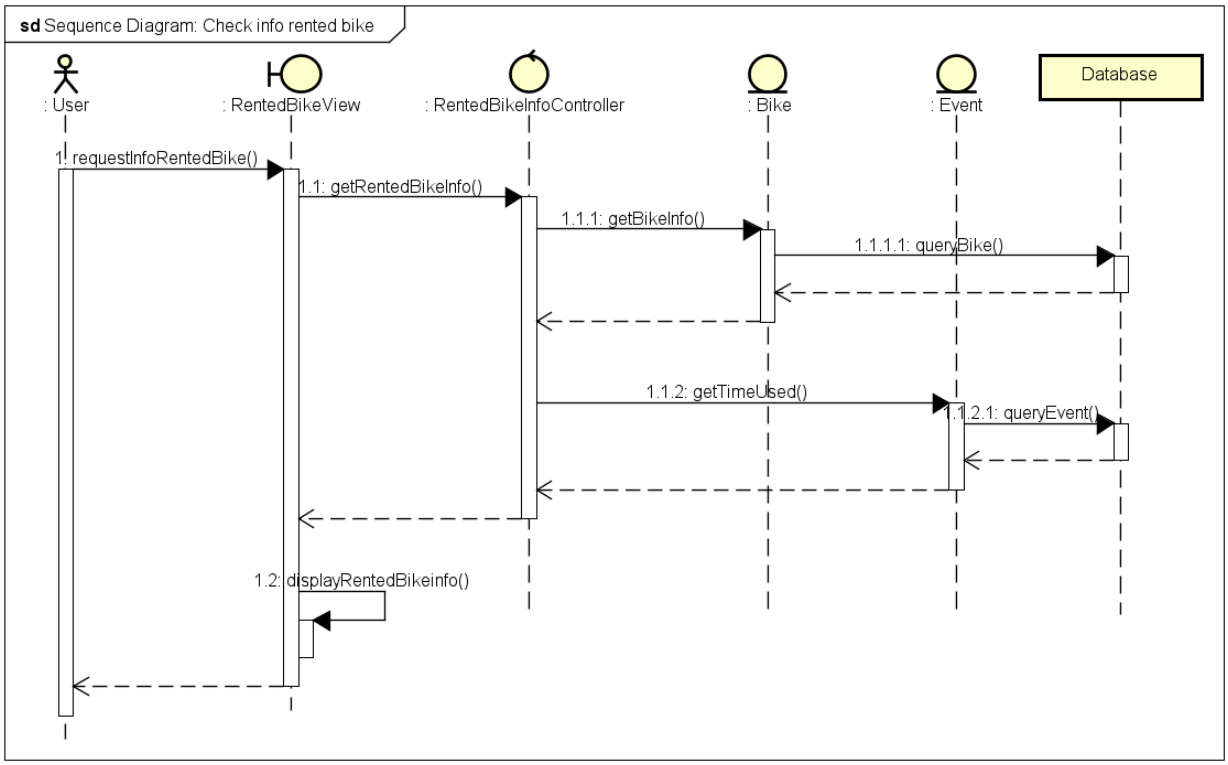

sd Communication Diagram: Check info parking lot
1: requestParkingLotView()
1.1: getParkingLotInfo(id)
1.1.1: getParkingLotInfo(id)
: User
: ParkingLotView
: ParkingLotController
: ParkingLot
1.1.1.1: queryParkingLot(id)
1.2: displayParkingLotInfo()
Database

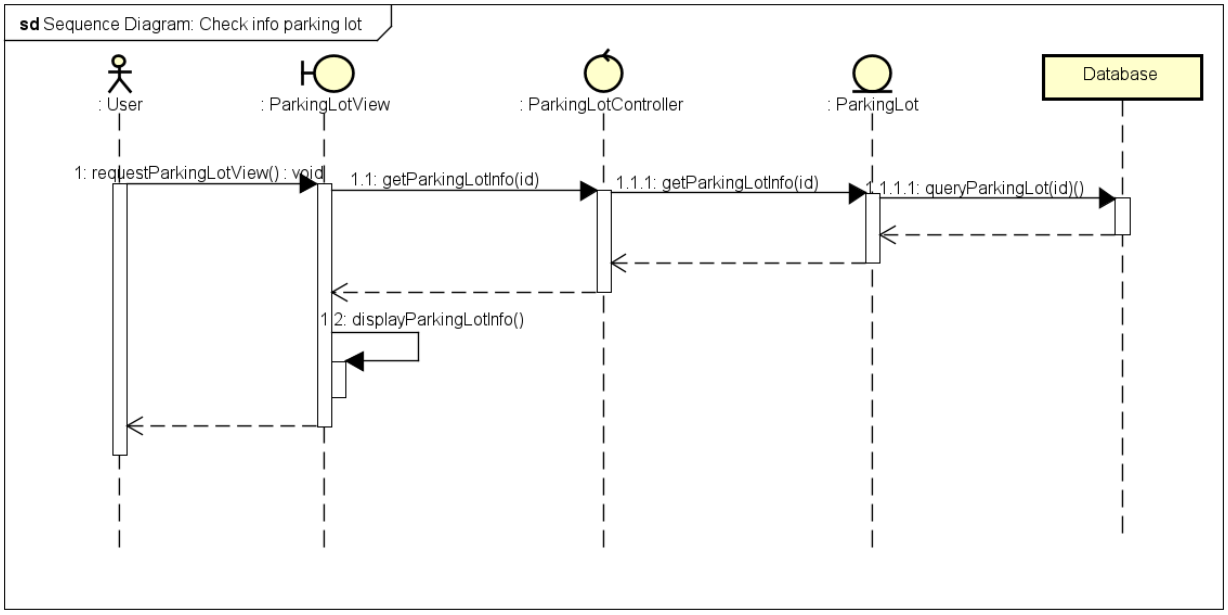

sd Communication Diagram: Check info bike in lot
1: requestParkingLotView()
1.1: getParkingLotInfo(id)
1.1.1: getParkingLotInfo(id)
: ParkingLotView
: ParkingLotController
: ParkingLot
1.1.1.1: queryParkingLot(id)
1.2: displayParkingLotInfo()
Database
: User
2: requestBikeInfoView()
2.1: getBikeInfo(id)
2.1.1: getBikeInfo(id)
2.1.1.1: queryBike(id)
: BikeInfoView
: BikeInfoController
: Bike
2.2: displayBikeInfo()

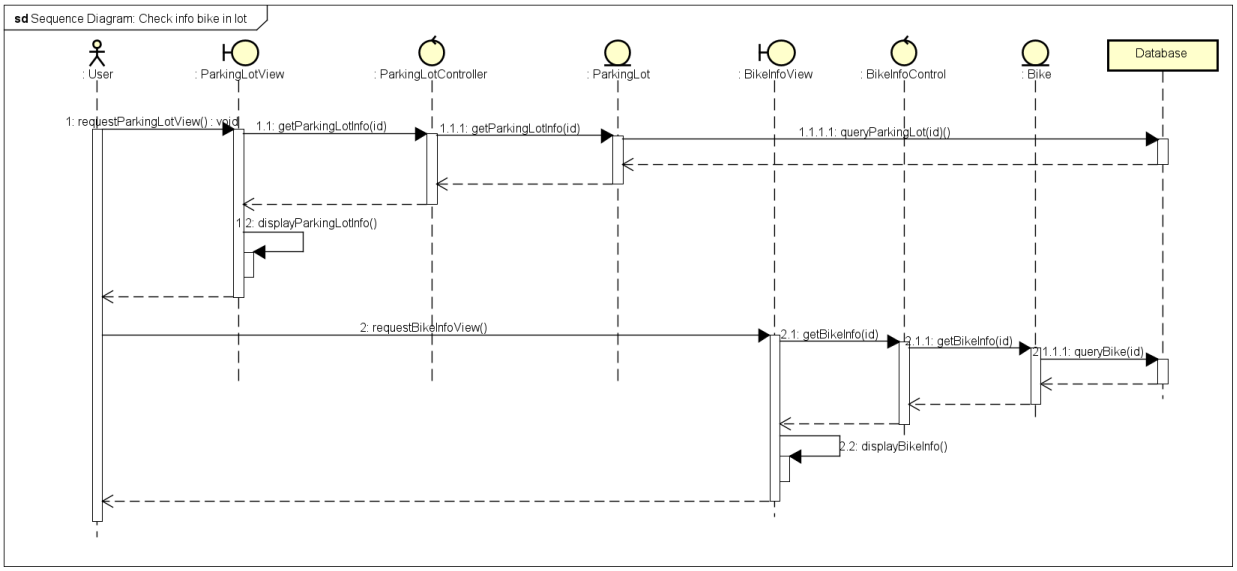
III. Sequence Diagram

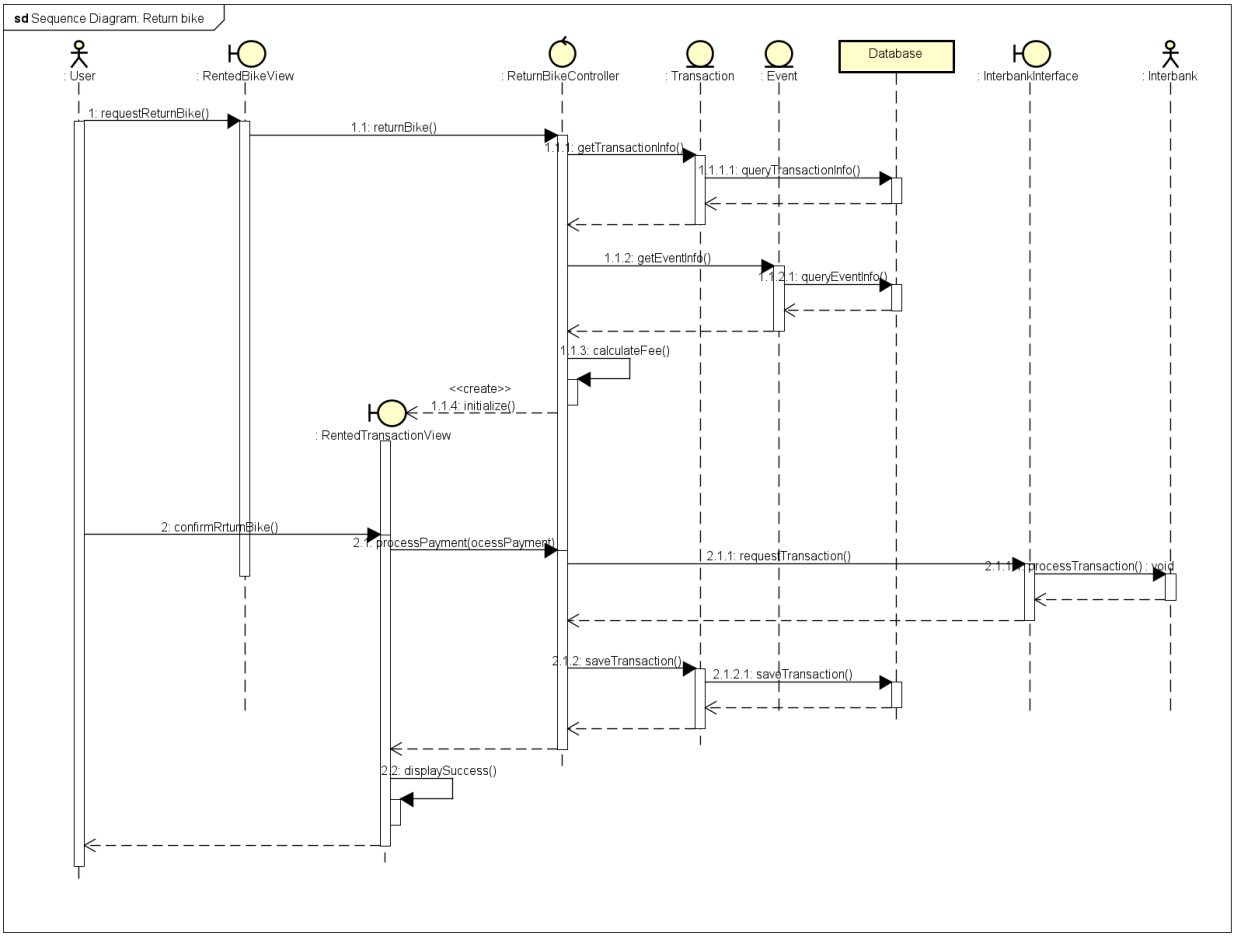


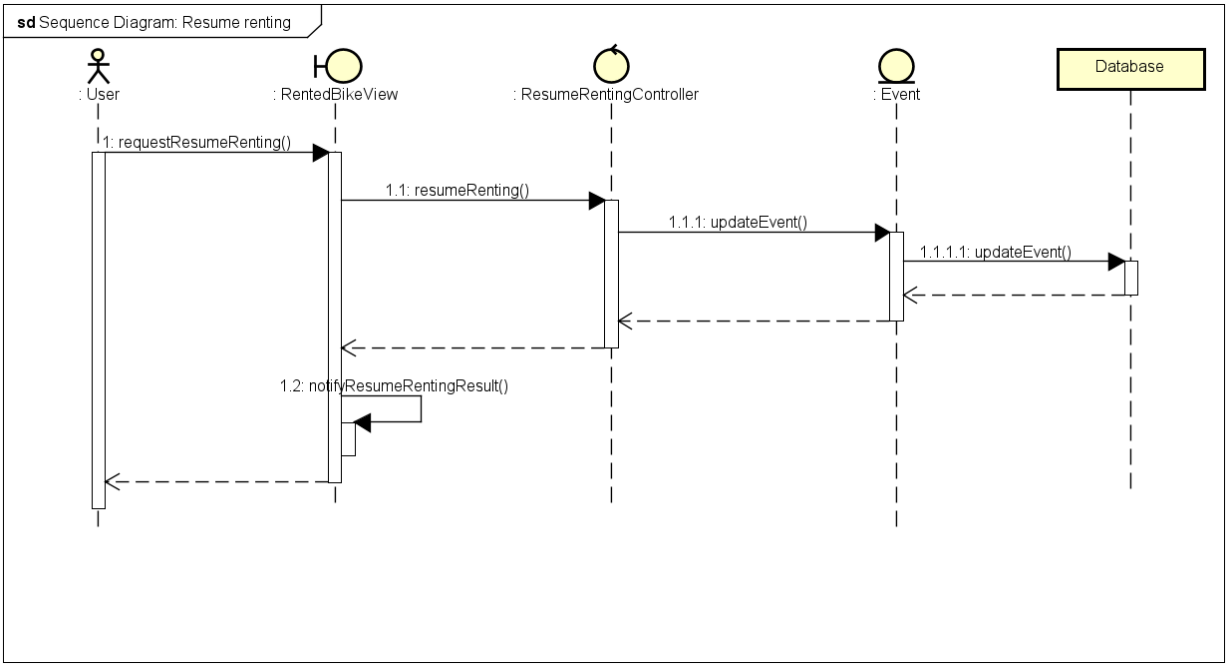


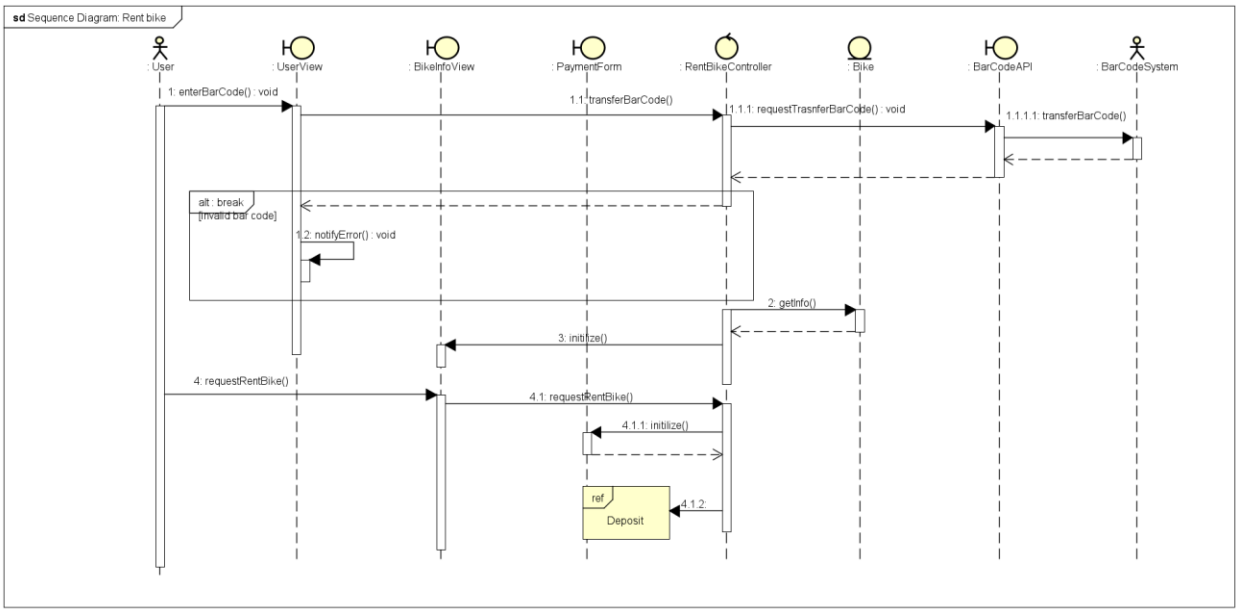

sd Sequence Diagram: Check info rented bike
: User
: RentedBikeView
: RentedBikeInfoController
: Bike
: Event
Database
1: requestInfoRentedBike()
1.1: getRentedBikeInfo()
1.1.1: getBikeInfo()
1.1.1.1: queryBike()
1.1.2: getTimeUsed()
1.1.2.1: queryEvent()
1.2: displayRentedBikeinfo()


sd Sequence Diagram: Check info parking lot
: User
: ParkingLotView
: ParkingLotController
: ParkingLot
Database
1: requestParkingLotView() : void
1.1: getParkingLotInfo(id)
1.1.1: getParkingLotInfo(id)
1.1.1.1: queryParkingLot(id)()
1.2: displayParkingLotInfo()


sd Sequence Diagram: Check info bike in lot
: User
: ParkingLotView
: ParkingLotController
: ParkingLot
: BikeInfoView
: BikeInfoControl
: Bike
Database
1: requestParkingLotView() : void
1.1: getParkingLotInfo(id)
1.1.1: getParkingLotInfo(id)
1.1.1.1: queryParkingLot(id)()
1.2: displayParkingLotInfo()
2: requestBikeInfoView()
2.1: getBikeInfo(id)
2.1.1: getBikeInfo(id)
2.1.1.1: queryBike(id)
2.2: displayBikeInfo()


sd Sequence Diagram: Return bike
: User
: RentedBikeView
: ReturnBikeController
: Transaction
: Event
Database
: InterbankInterface
: Interbank
1: requestReturnBike()
1.1: returnBike()
1.1.1: getTransactionInfo()
1.1.1.1: queryTransactionInfo()
1.1.2: getEventInfo()
1.1.2.1: queryEventInfo()
1.1.3: calculateFee()
<<create>>
1.1.4: initialize()
: RentedTransactionView
2: confirmRtturnBike()
2.1: processPayment(ocessPayment)
2.1.1: requestTransaction()
2.1.1.1: processTransaction() : void
2.1.2: saveTransaction()
2.1.2.1: saveTransaction()
2.2: displaySuccess()

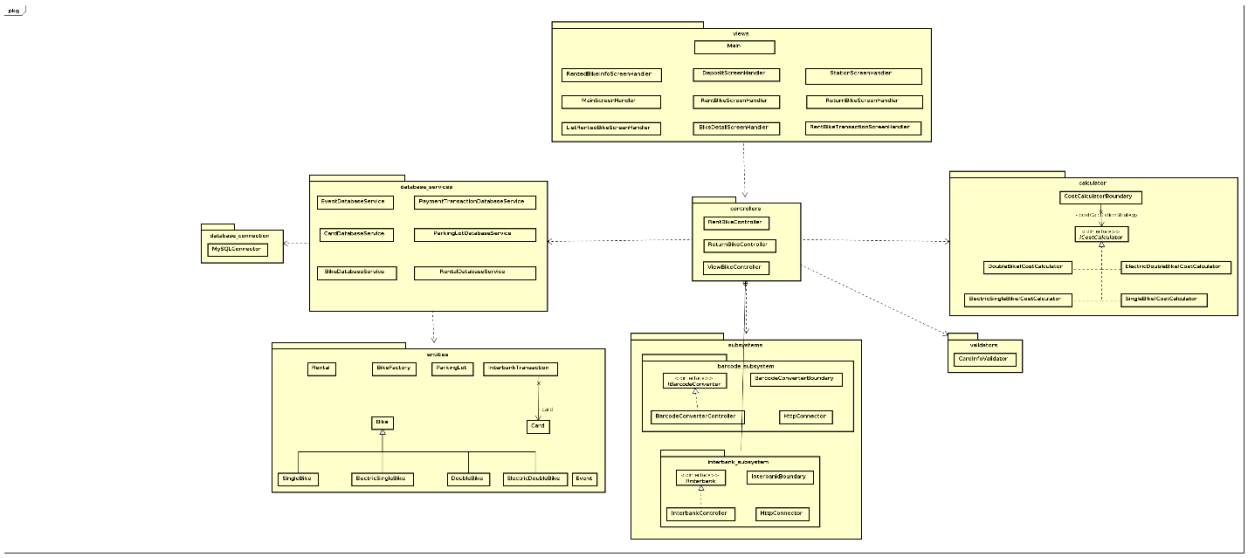

sd Sequence Diagram: Resume renting
: User
: RentedBikeView
: ResumeRentingController
: Event
Database
1: requestResumeRenting()
1.1: resumeRenting()
1.1.1: updateEvent()
1.1.1.1: updateEvent()
1.2: notifyResumeRentingResult()


sd Sequence Diagram: Rent bike
: User
: UserView
: BikeInfoView
: PaymentForm
: RentBikeController
: Bike
: BarCodeAPI
: BarCodeSystem
1: enterBarCode() : void
1.1: transferBarCode()
1.1.1: requestTrasnferBarCode() : void
1.1.1.1: transferBarCode()
alt : break
[invalid bar code]
1.2: notifyError() : void
2: getInfo()
3: initilize()
4: requestRentBike()
4.1: requestRentBike()
4.1.1: initilize()
ref
Deposit
4.1.2:

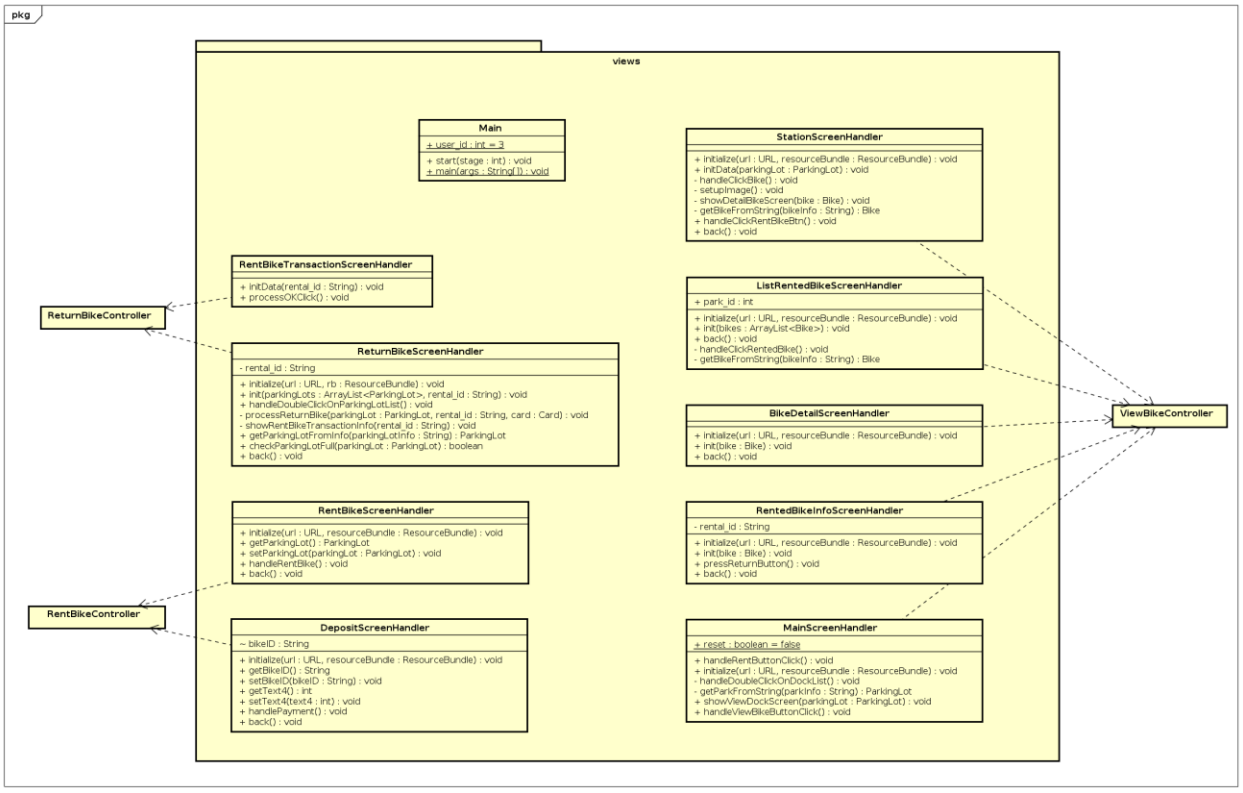
F. Detailed Design

I. Class Design

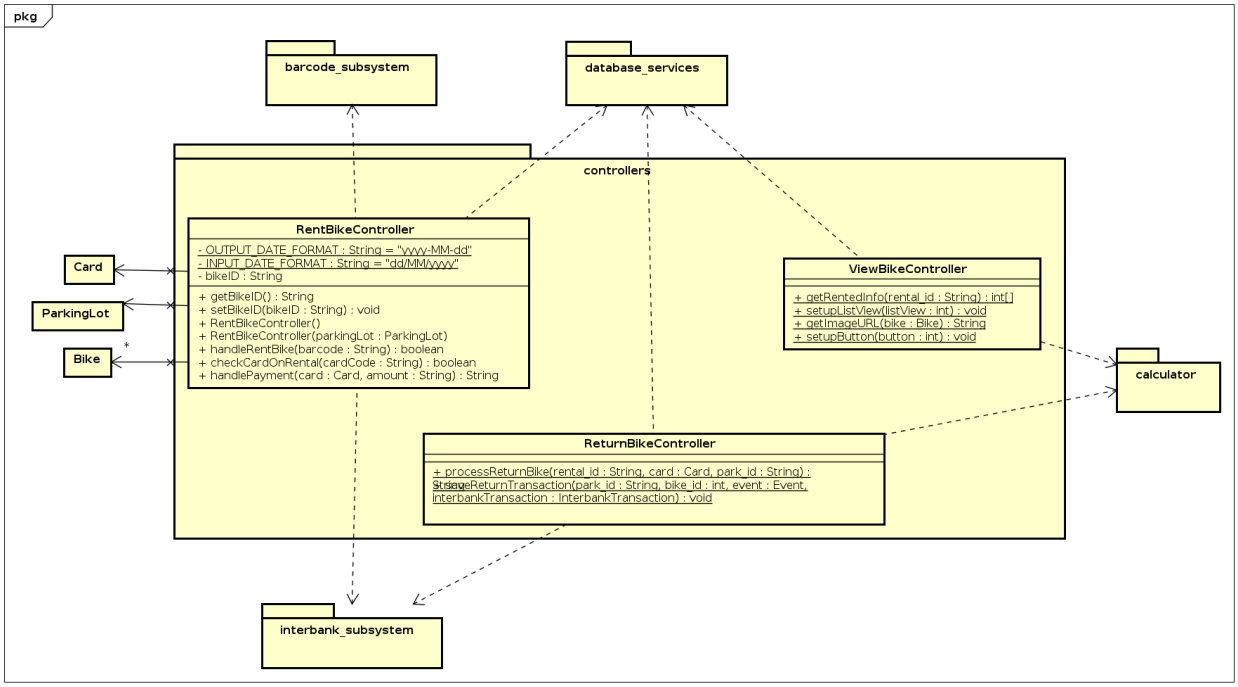
1. Class Diagram



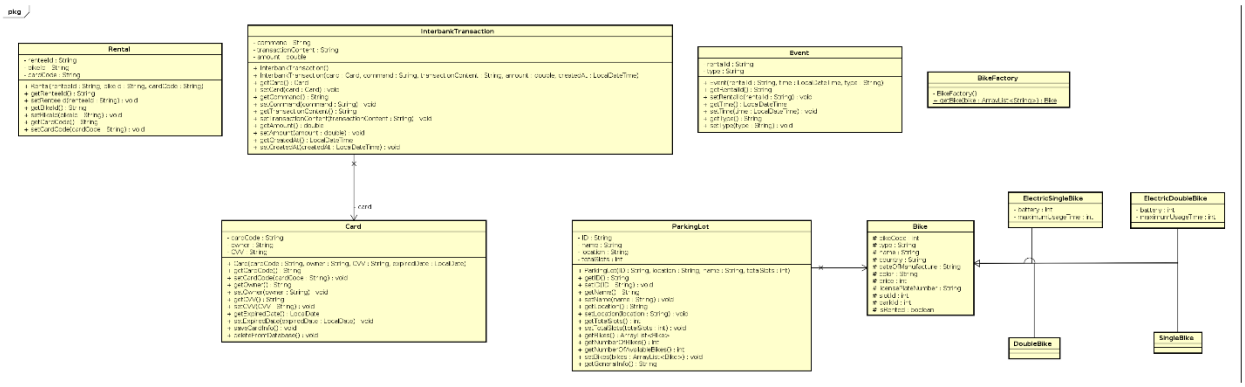
a. Class Diagram for package Views



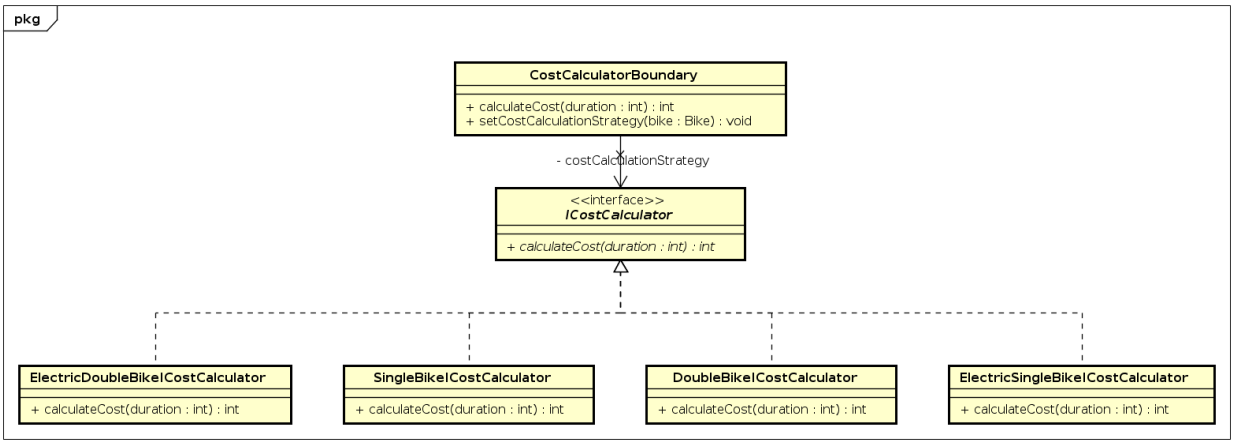
b. Class Diagram for package Controllers



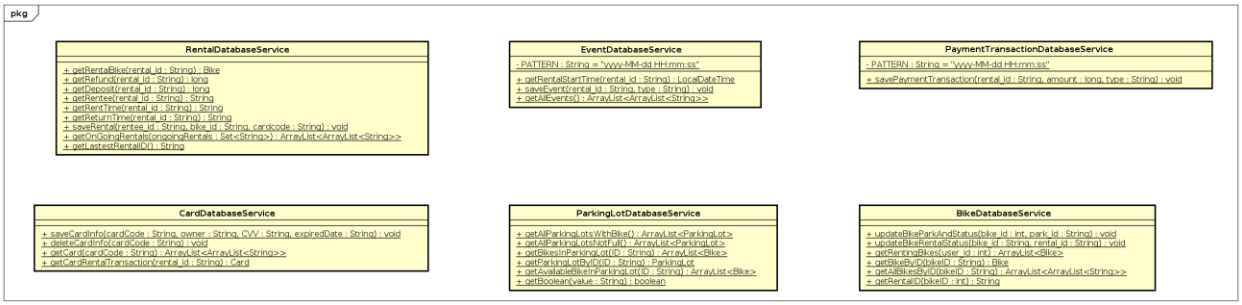
c. Class Diagram for package Entities



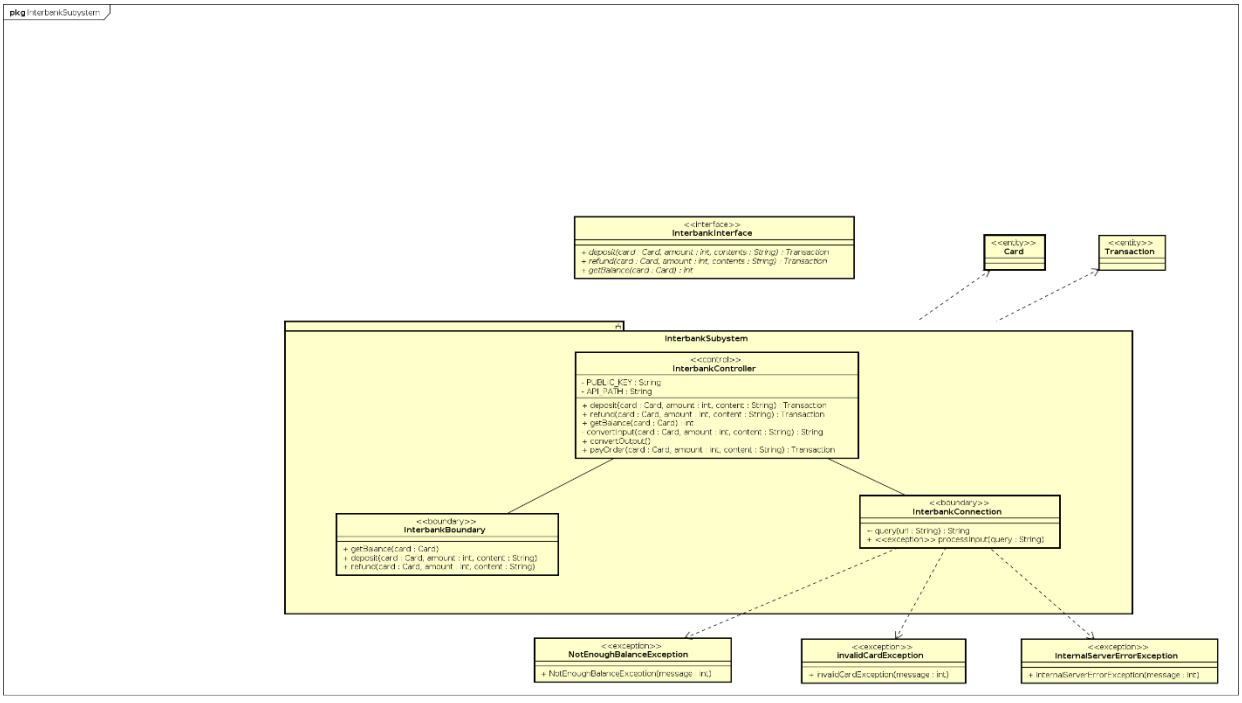
d. Class Diagram for package Caculator



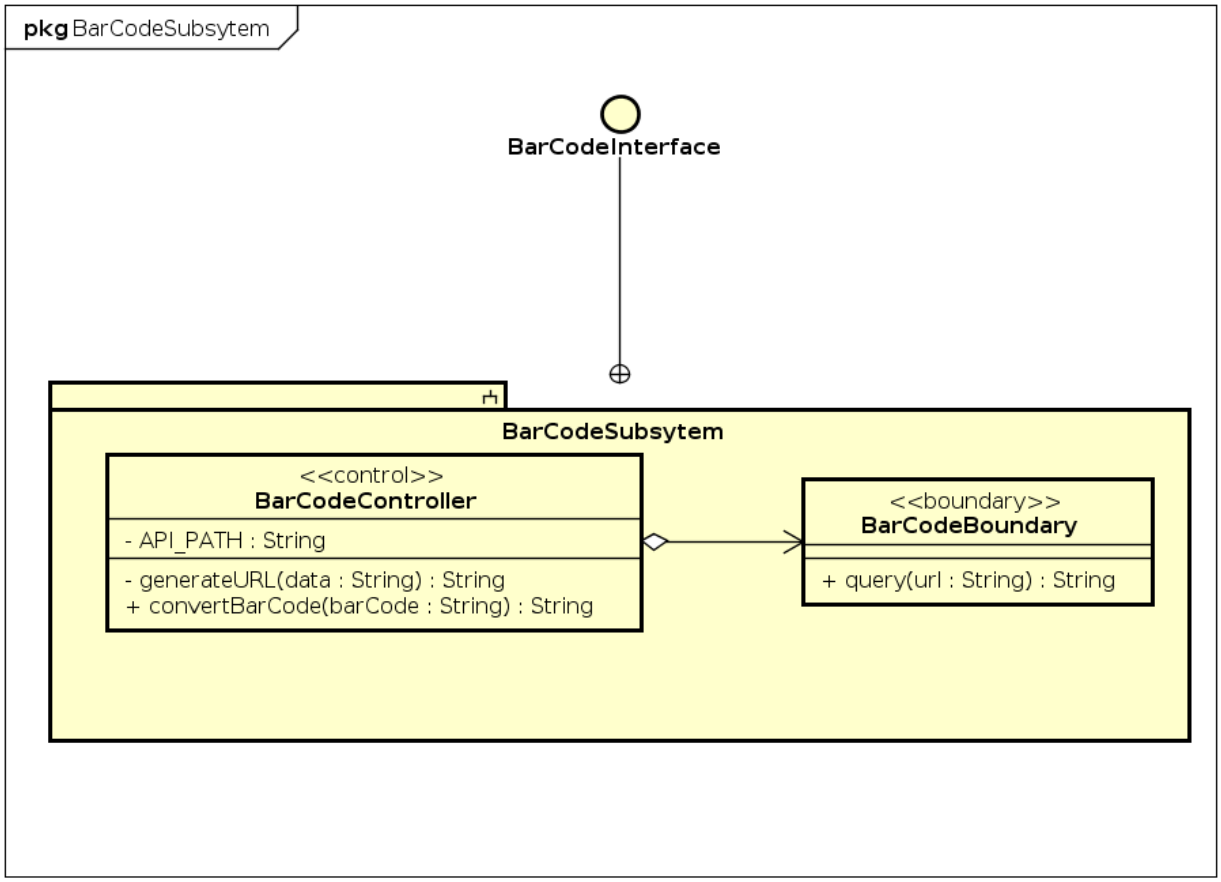
e. Class Diagram for package Database Service



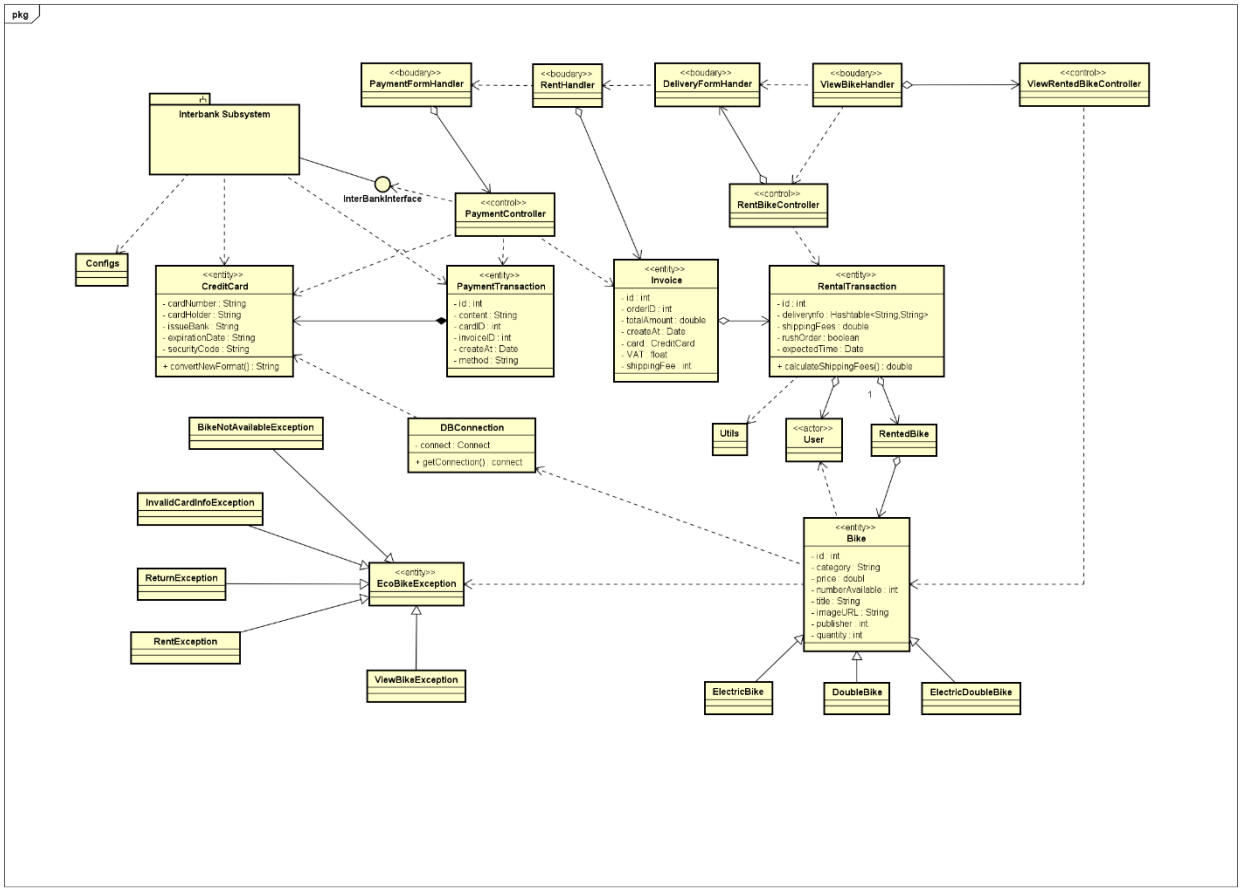
f. Class Diagram for Interbank Subsystem



g. Class Diagram for Barcode Subsystem

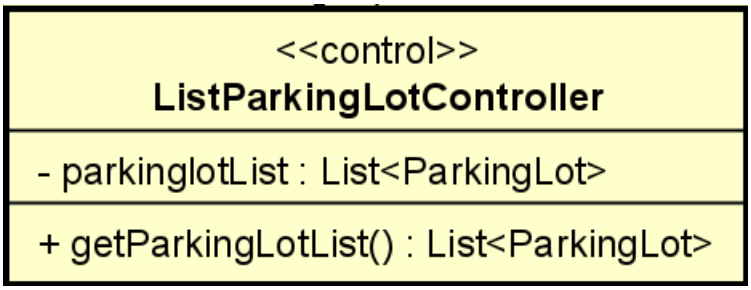


2. Class Relationship



3. Class Description

a. ListParkingLotController



- Attribute

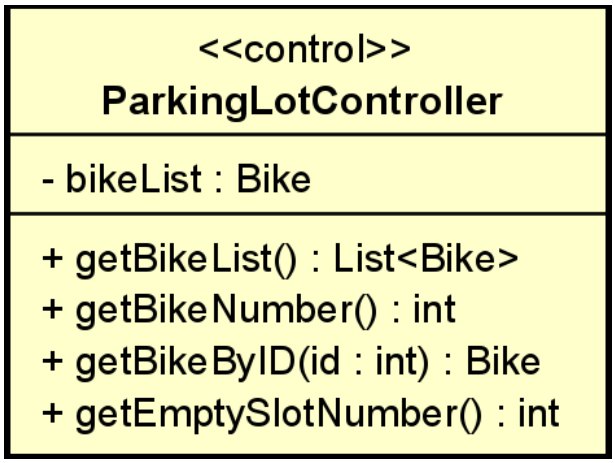
Không

- Operation

No.	Tên	Kiểu dữ liệu	Mô tả
1	getStationList	List<Station>	Trả về danh sách các bãi gửi xe

- ***Parameter:***
  Không
- *Exception:*
  Không
- ***Method***
  Không
- ***State***
  Không

b. ParkingLotController



Attribute

No.	Tên	Kiểu dữ liệu	Giá trị mặc định	Mô tả
1	bikeList	List<Bike>	NULL	Danh sách xe trong bãi

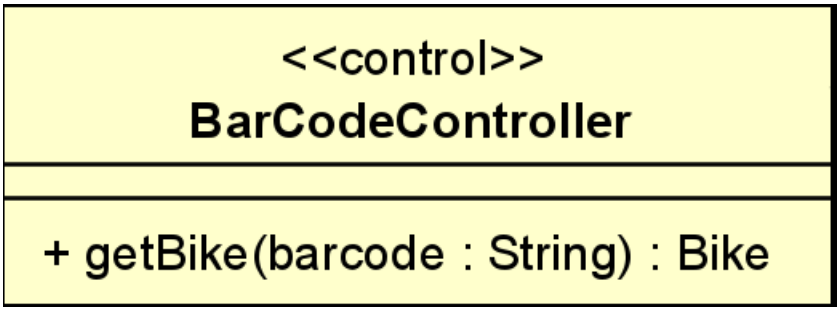
Operation

No.	Tên	Kiểu dữ liệu	Mô tả
2	getBikeList	List<Bike>	Trả về danh sách xe
3	getBikeNumber	int	Số xe trong bãi
4	getBikeByID	Bike	Trả về xe theo tùy chọn (khi người dùng click trong danh sách)
5	getEmptySlotNumber	int	Trả về số slot trống trong bãi

- ***Parameter:***

- index: chỉ số của xe trong danh sách
- barCode: mã barCode nhập vào để tìm xe
- *Exception:*
  Không
- ***Method***
  Không
- ***State***
  Không

c. BarCodeController



- Attribute

Không

- Operation

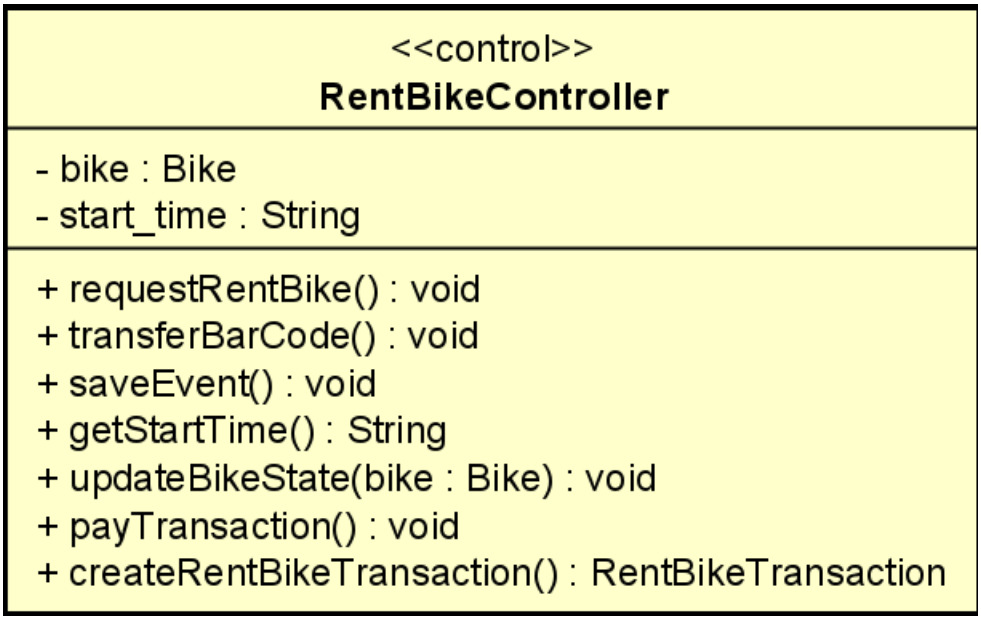
No.	Tên	Kiểu dữ liệu	Mô tả
1	getBarCode	Bike	Trả về thông tin xe được chuyển từ barcode

- ***Parameter:***
  - barCode: mã bar code được truyền vào
- *Exception:*
  - InvalidBarCodeException
- ***Method***

  Không
- ***State***

  Không

d. RentBikeController



Attribute

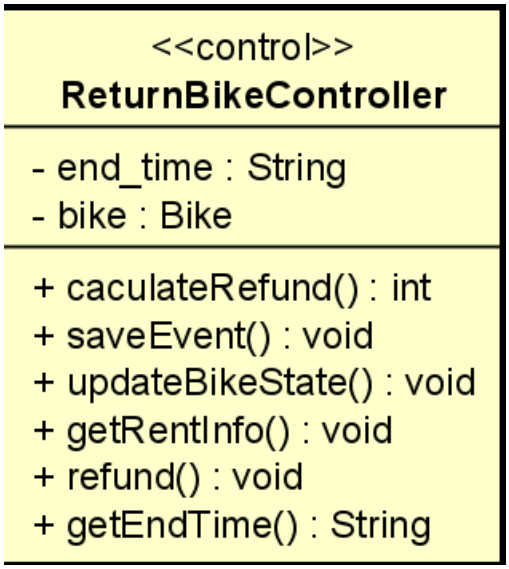
No.	Tên	Kiểu dữ liệu	Giá trị mặc định	Mô tả
1	bike	Bike	NULL	Thông tin về xe được thuê
2	start_time	String	NULL	Thời điểm bắt đầu thuê

Operation

No.	Tên	Kiểu dữ liệu	Mô tả
1	payTransaction	void	Thực hiện việc thanh toán cọc để thuê xe
2	createRentBikeTransaction	RentBikeTransaction	Tạo thông tin chi tiết về việc thuê xe
3	getStartTime	String	Trả về thời gian bắt đầu thuê
4	updateBikeState	void	Cập nhật trạng thái xe thuê vào database
5	transferBarCode	void	Chuyển thông tin barcode thành id xe

- ***Parameter:***
  Không
- *Exception:*
  Không
- ***Method***
  Không
- ***State***
  Không

e. ReturnBikeController



- Attribute

No.	Tên	Kiểu dữ liệu	Mô tả
1	end_time	String	Thời gian trả xe
2	bike	Bike	Xe được trả

Operation

No.	Tên	Kiểu dữ liệu	Mô tả
1	getRentInfo	void	In ra thông tin về xe thuê
2	refund	void	Thực hiện giao dịch trả tiền cọc
3	calculateRefund	int	Tính tiền refund
4	saveEvent	void	Lưu thông tin sự kiện vào database
5	getEndTime	String	Lấy thông tin thời điểm trả xe

Parameter:

Exception:

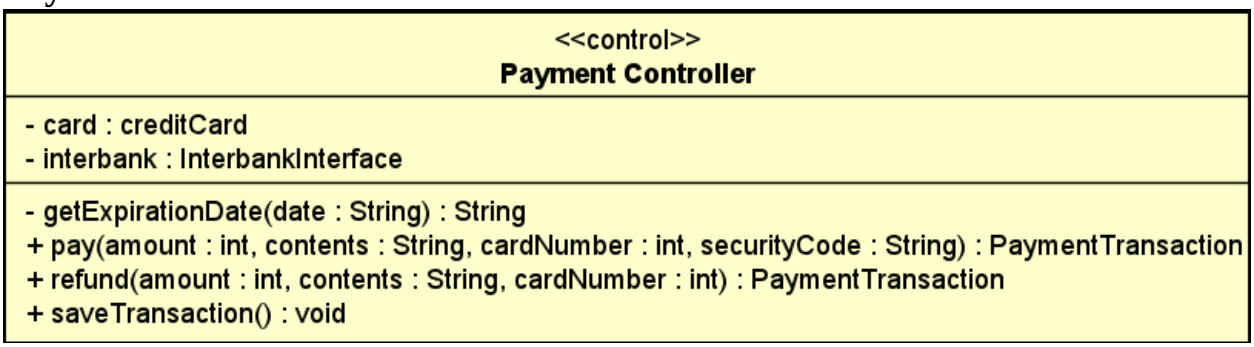
- InvalidCardException: thẻ không hợp lệ

Method

State

Không

f. PaymentController



- Attribute

No.	Tên	Kiểu dữ liệu	Giá trị mặc định	Mô tả
1	card	CreditCard	NULL	Card dùng cho thanh toán
2	interbank	InterbankInterface	NULL	Interbank Subsystem

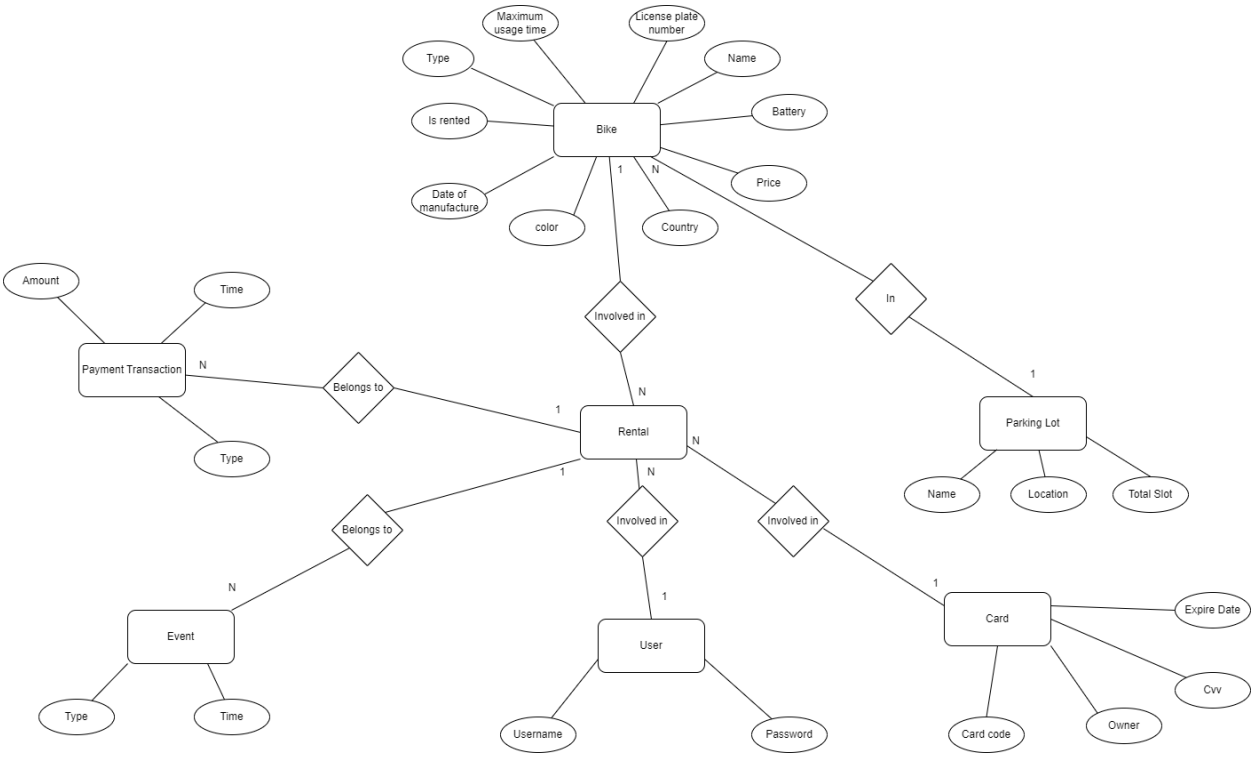
- Operation

No.	Tên	Kiểu dữ liệu	Mô tả
1	pay	Map<String,String>	Thực hiện thanh toán tiền cọc và trả về giao dịch thanh toán
2	refund		Thực hiện hoàn tiền và trả về giao dịch thanh toán
3	saveTransaction	void	Lưu thông tin giao dịch

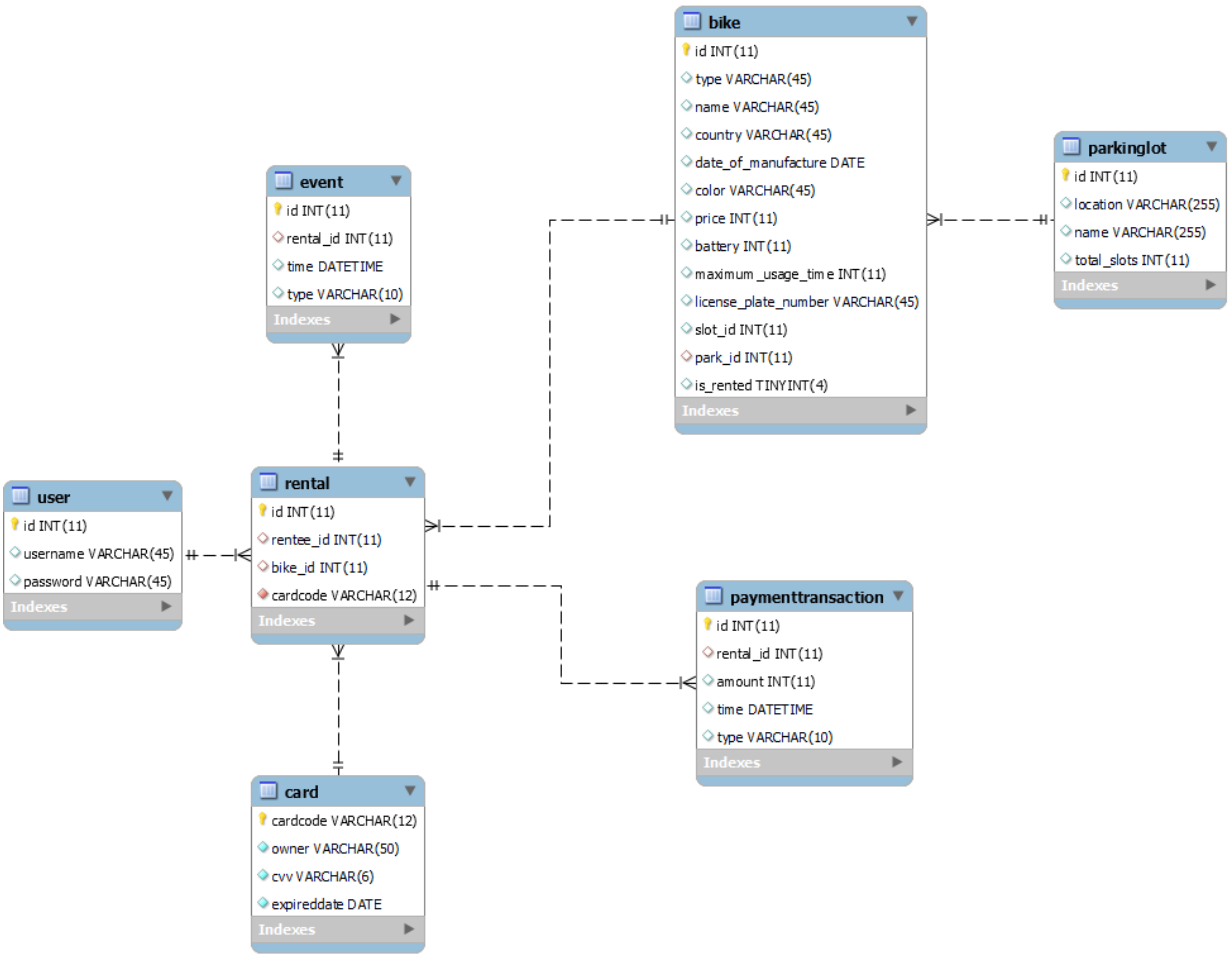
- ***Parameter:***
  - amount – số tiền giao dịch
  - contents – nội dung giao dịch
  - cardNumber – số thẻ
  - cardHolderName – tên chủ sở hữu
  - expirationDate – ngày hết hạn theo định dạng "mm/yy"
  - securityCode - mã bảo mật
- *Exception:*
  Không
- ***Method***
  getExpirationDate: Chuyển dữ liệu ngày từ định dạng “mm/yy” sang “mmyy”
- ***State***
  Không

II. Data Modeling

1. ERD



2. Relational Database Model



3. Database Description

a) Bike:

Name	Type	Meaning
id	Int(11)	Bike code
type	Varchar(45)	
name	Varchar(45)	
country	Varchar(45)	
date_of_manufacture	Datetime	
color	Varchar(45)	
price	Int(11)	
battery	Int(11)	
maximum_usage_time	Int(11)	
license_place_number	Varchar(45)	
slot_id	Int(11)	
park_id	Int(11)	
is_rented	Tinyint(4)	Bike is currently rented or not

b) Card:

Name	Type	Meaning
cardcode	Varchar(12)	
owner	Varchar(50)	
cvv	Varchar(6)	Security code
expireddate	date	

c) Event:

Name	Type	Meaning
id	Int(11)	Id of the event
rental_id	Int(11)	Id of the correspoding rental
time	Datetime	
type	Varchar(10)	Type of event (start, pause, resume, end)

d) ParkingLot:

Name	Type	Meaning
id	Int(11)	
location	Varchar(255)	
name	Varchar(255)	
total_slots	Int(11)	

e) PaymentTransaction:

Name	Type	Meaning
id	Int(11)	Id of the payment transaction
rental_id	Int(11)	Id of the correspoding rental
amount	Int(11)	
time	Datetime	Type of transaction (start, pause, resume, end)
type	Varchar(10)	

f) Rental:

Name	Type	Meaning
id	Int(11)	Id of the event
Rentee_id	Int(11)	Id of the correspoding rentee
Bike_id	Int(11)	Id of the rented bike
cardcode	Varchar(12)	Cardcode of the card used in the rental

g) User:

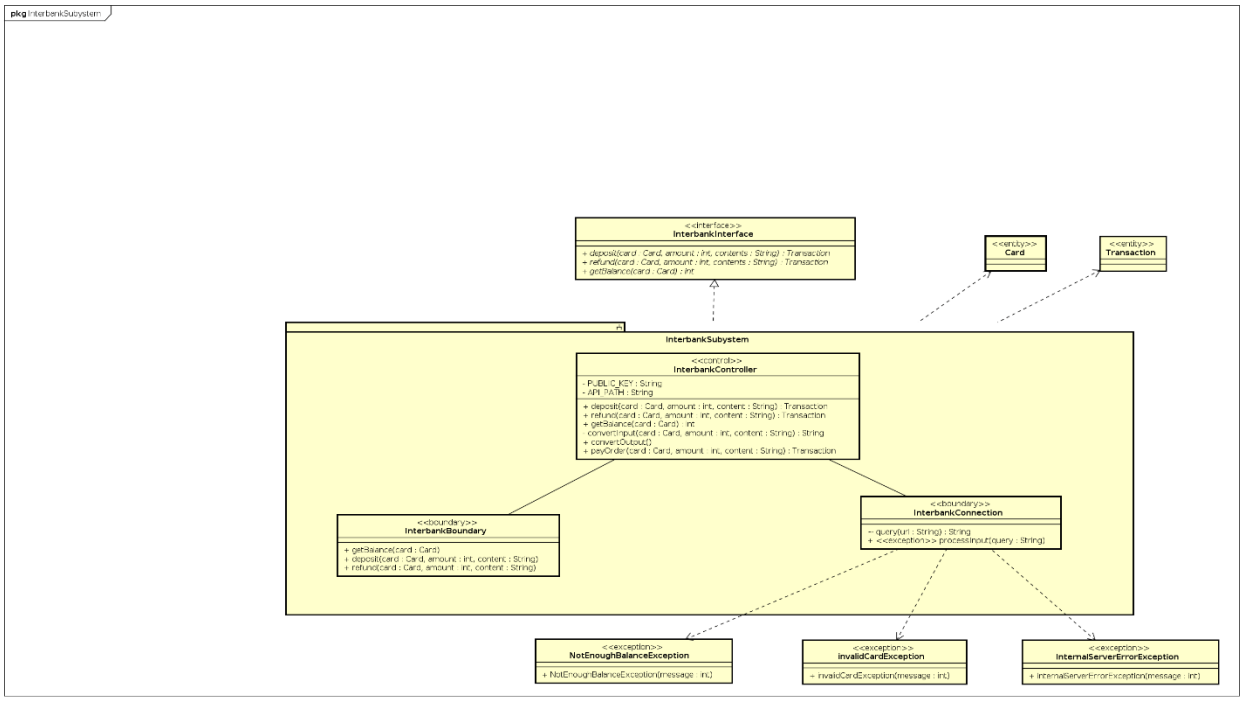
Name	Type	Meaning
id	Int(11)	
username	Varchar(45)	
password	Varchar(45)	

III. Interface Design

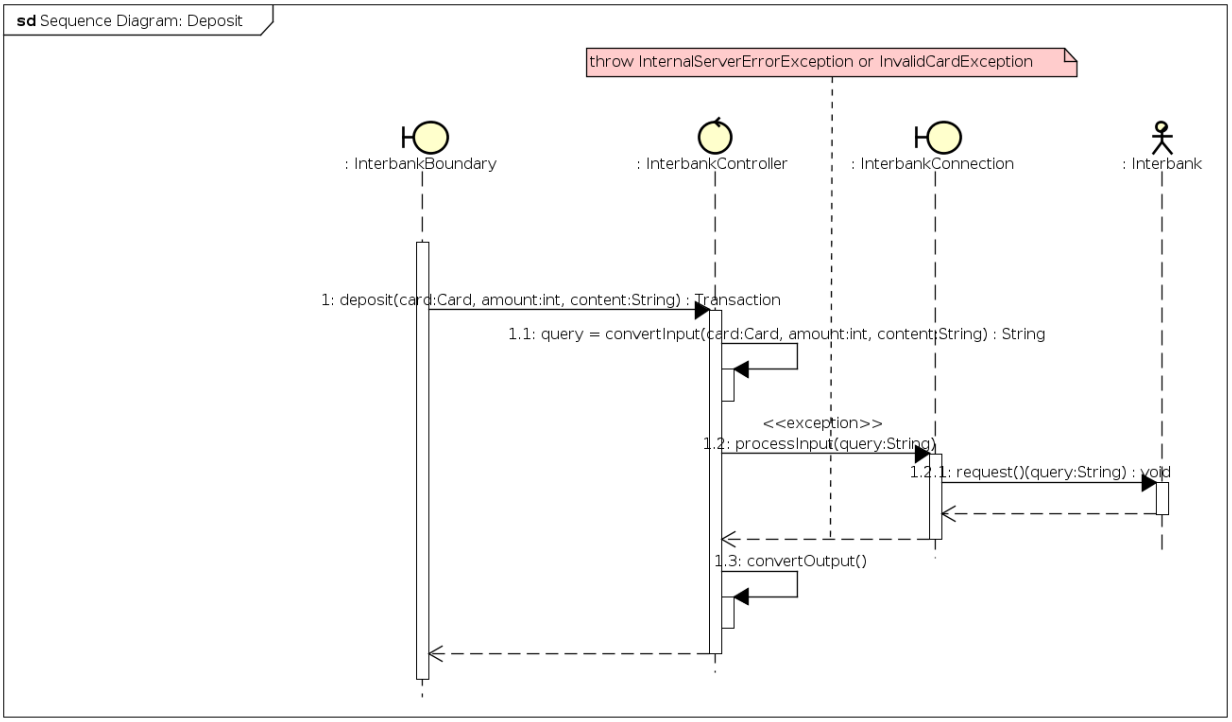
1. Subsystem Interface Design

a. Interbank Subsystem

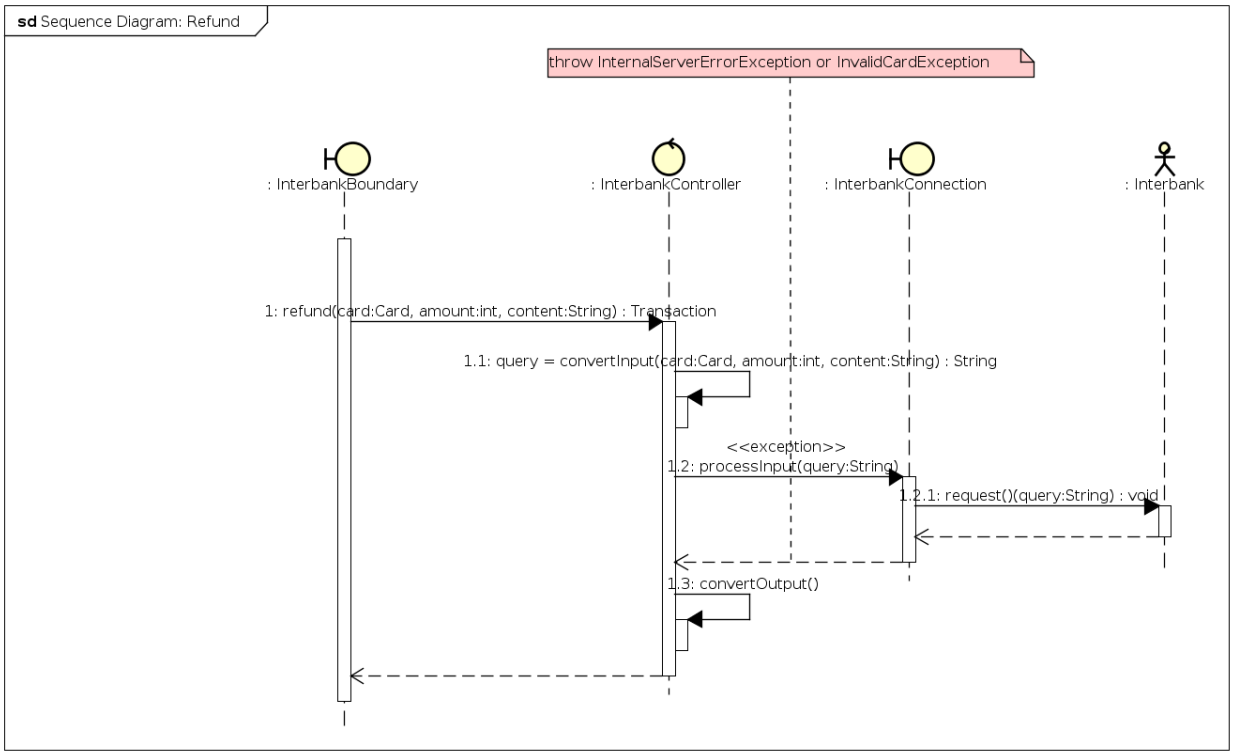
The interbank subsystem has 3 functions: deposit, refund and check balance.



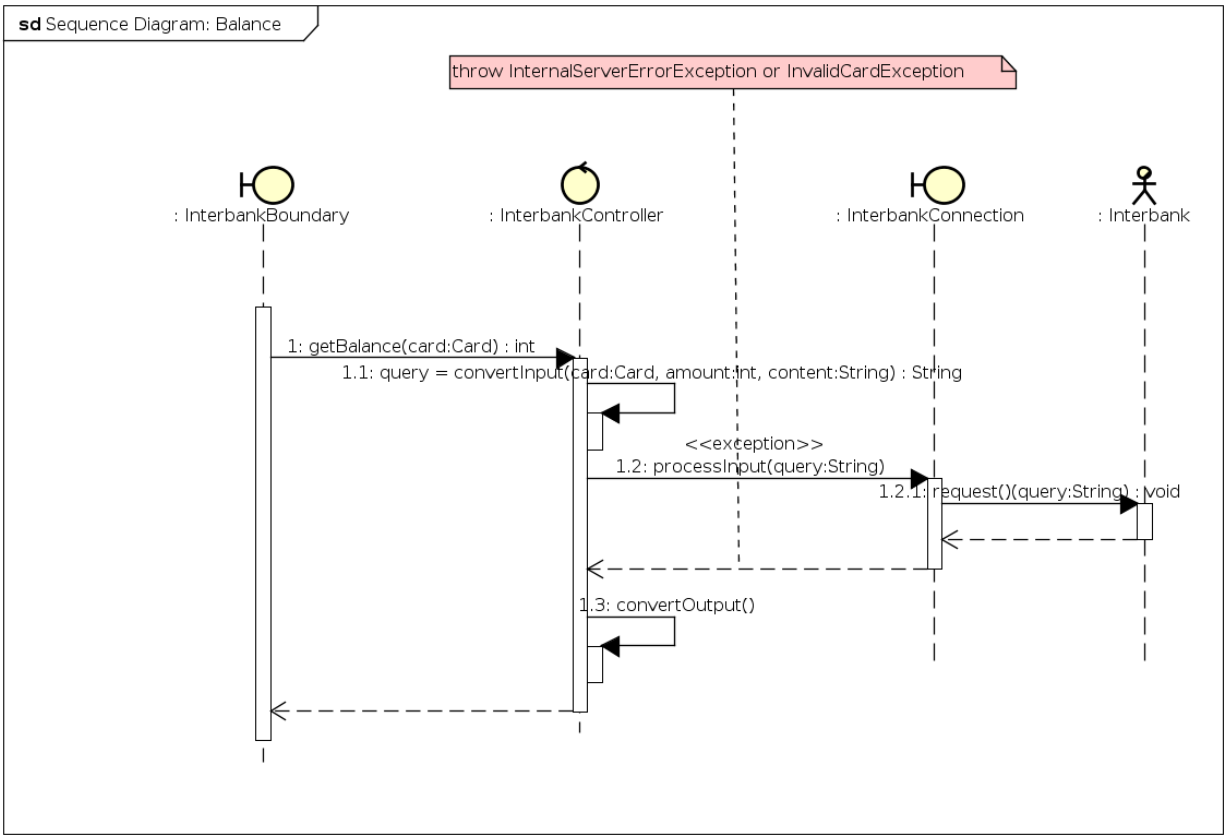
- Deposit:



- Refund:

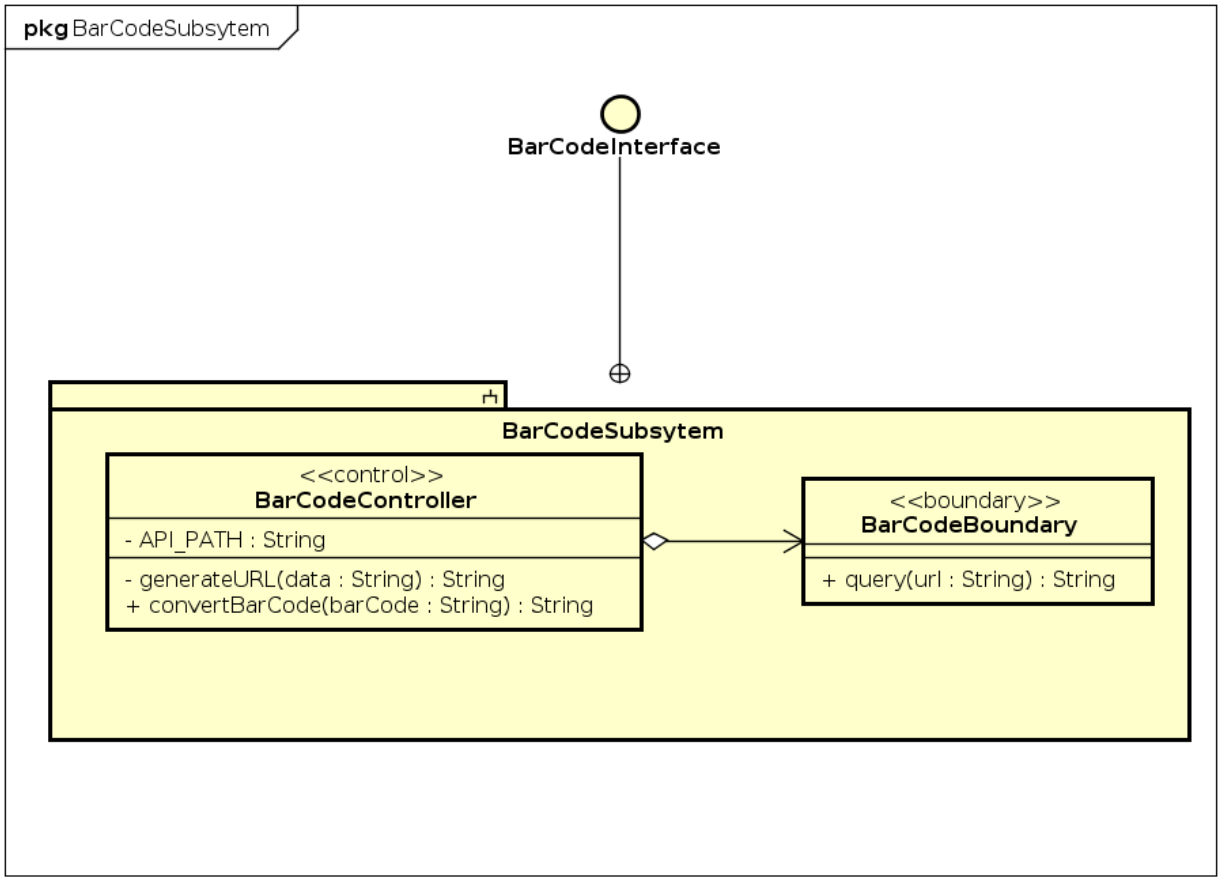


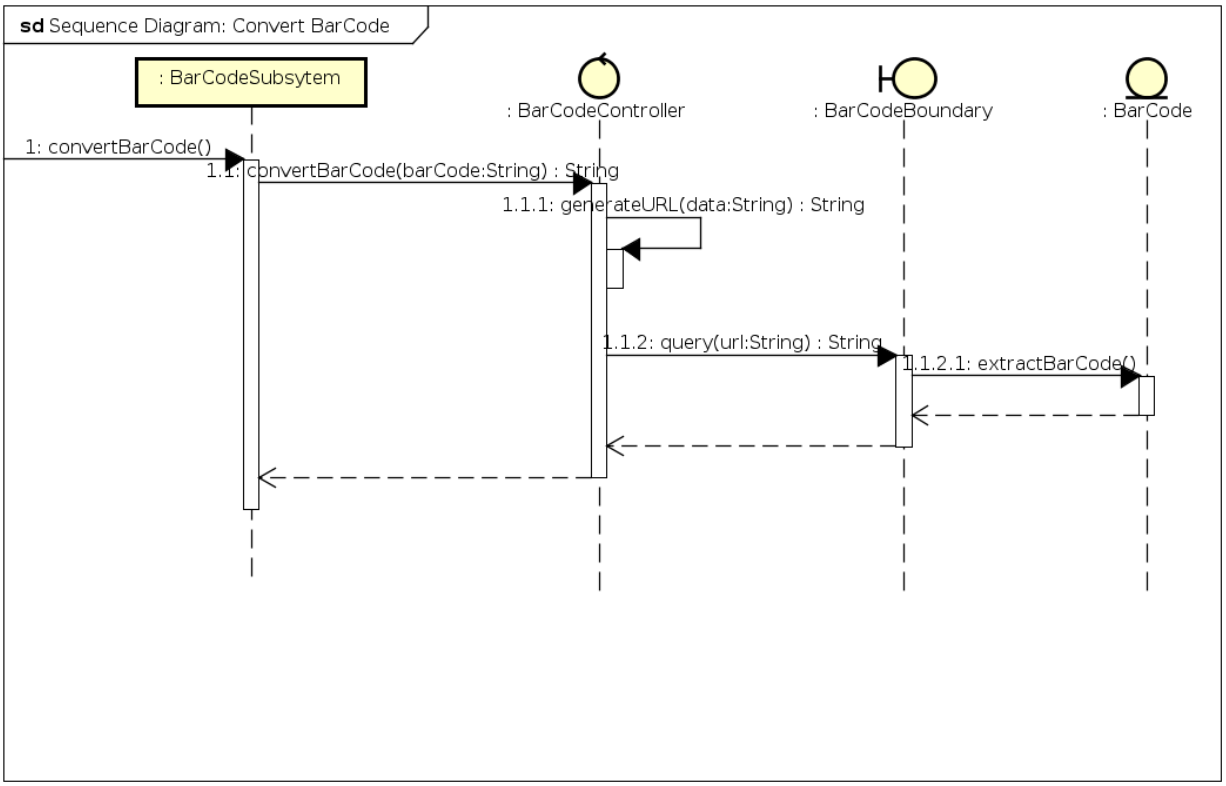
- Check balance:



b. Barcode Subsystem

The barcode subsystem has one function, which is converting the barcode to the corresponding bike code stored in the database.





2. User Interface Design

a. Screen Specification

- General information
    - Main color: #bec913
    - All Screen have return button in the top right corner
- Detail
    1. Splash Screen

2. Main Screen

- Function: Home screen of application
- Contain:
    - Logo
    - Name: EcoBike
    - Button “Xem xe đang thuê”: Show Rented Bike Info Screen when you click and you is renting 1 bike or show List Rented Bike Screen when you is renting more than 1 bike
    - List of available station:
        - show all available station by row show Station Screen when you click one available station

3. Station Screen

- Precondition: You click one station in Main Screen
- Function: Show information of station
- Contain:
    - Logo
    - Name: Bãi xe
    - Button “Thuê xe”: Show Rent Bike Screen when you click
    - Button “Quay lai”: Close and return previous screen

- Information of station:
    - Logo of station:
    - Name of station: Bãi xe Bách Khoa
    - Address of station: số 1 Đại Cồ Việt
    - Number of slots:
    - Number of available bike:
- List of available bike:
    - show all bike by row but unavailable bike information is lighter in color
    - show Detail Bike Screen when you double click one bike
    - change BIke image when you choose one bike
- Bike image:

4. Bike Detail Screen

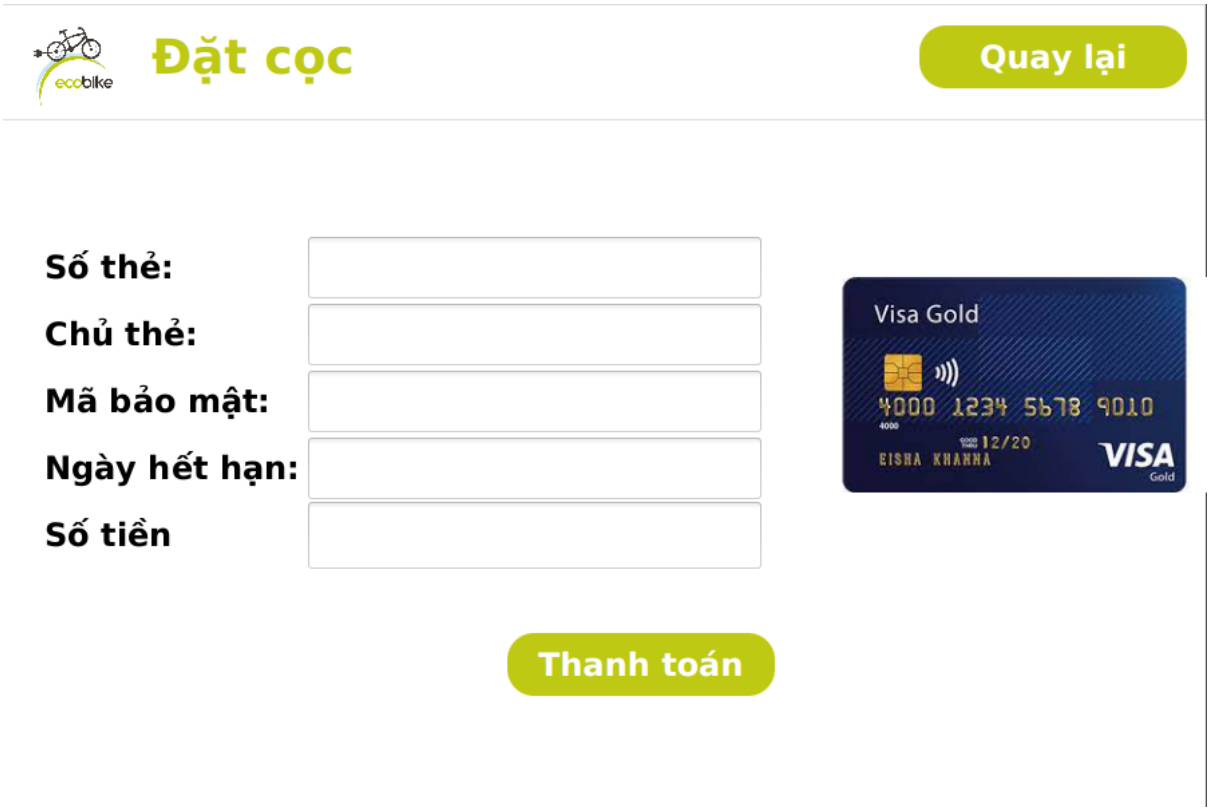
- Precondition:
    - You double click one bike in Station Screen
- Function: Show information of bike
- Contain:
    - Logo
    - Name: Thông tin chi tiết xe

- Button “Quay lại”: Close and show previous screen
- Information of bike:

5. Rent BIke Screen

- Precondition:
  - You click “Thuê xe” in Station Screen
- Function: Enter barcode of rented bike
- Contain:
  - Logo
  - Name: Thuê xe
  - Button “Thuê xe”: Show Deposit when you click
  - Button “Quay lại”: Close and show previous screen
  - Barcode:

6. Deposit Screen

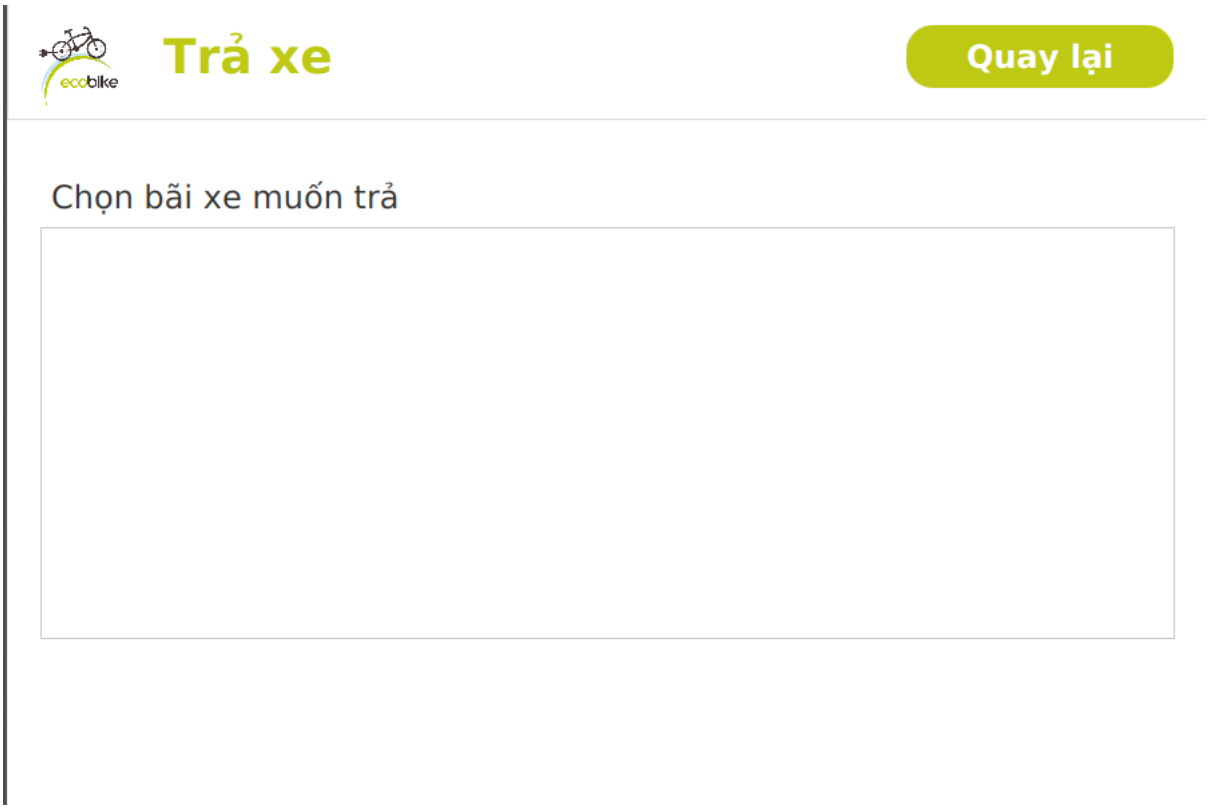


- Precondition:
  - You click “Thuê xe” in Rent Bike Screen
- Function: Deposit for rented bike
- Contain:
  - Logo
  - Name: Đăt cọc
  - Card Info:
    - Card image
    - Card number
    - Card owner
    - Code
    - Amount of money
  - Button “Thanh toán”: Process deposit
  - Button “Quay lại”: Close and show previous screen

7. Rented Bike Info Screen

- Precondition:
  - You click “Xem xe đang thuê” in Main Screen
- Function: Show information of rented bike
- Contain:
  - Logo
  - Name: Xe đang thuê
  - Rented Bike Info:
  - Button “Quay lại”
  - Button “Tạm dừng”
  - Button “Trả xe”: show Return BIke Screen when you click

8. Return Bike Screen

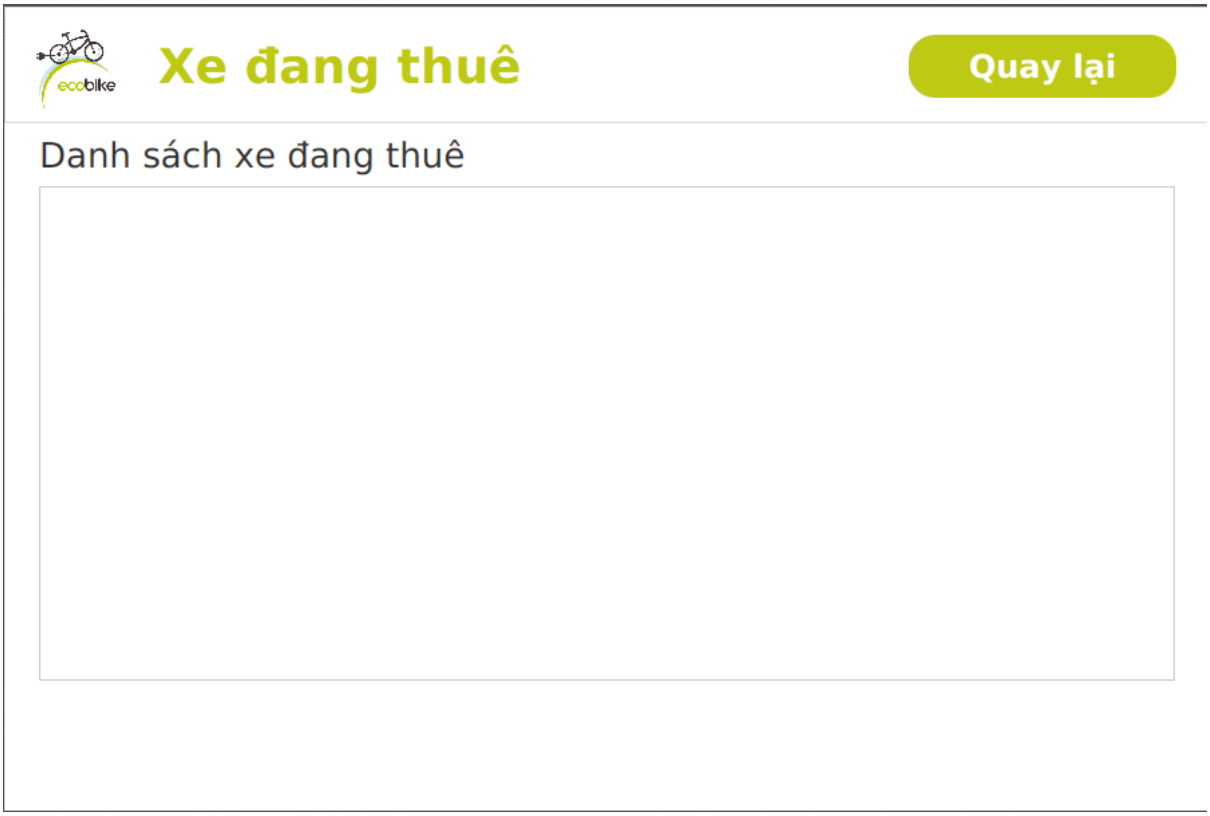


- Precondition:
  - You click “Trả xe” in Rented Bike Info Screen
- Function: Choose station you want to return
- Contain:
  - Logo
  - Name: Trả xe
  - List of available station:
    - show all station by row but unavailable station information is lighter in color
    - show Rented BIke Transaction Screen when you click one available station

9. Rented BIke Transaction Screen

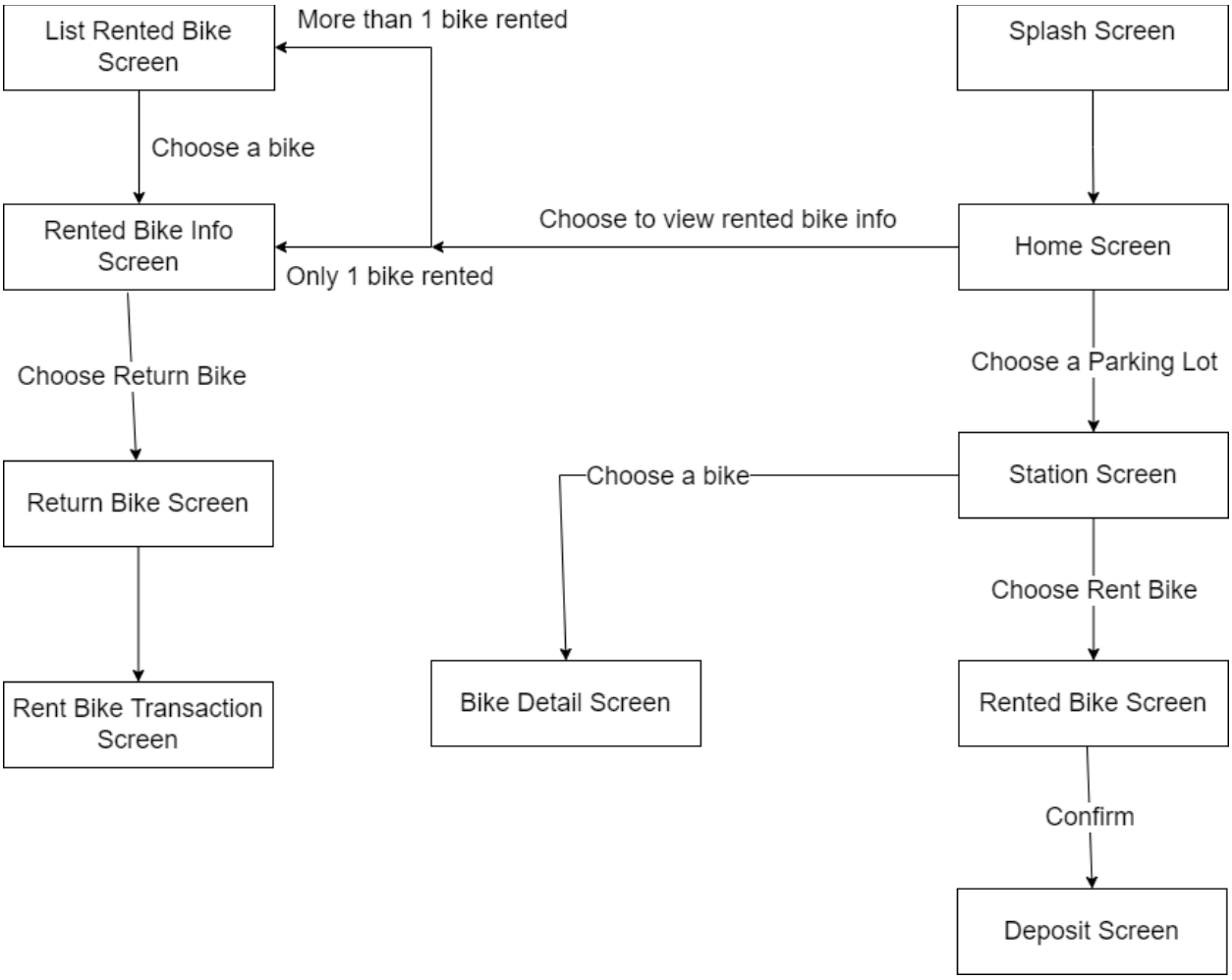
- Precondition:
  - You click one available station in Return BIke Screen
- Function: Pay Order
- Contain:
  - Logo
  - Name: Trả xe
  - Transaction Info
  - Button “Xac nhan”

10. List Rented Bike Screen



- Precondition:
  - You click “Xem xe dang thue” in Main Screen and you are renting more than 1 bike
- Function: Show list rented bike
- Contain:
  - Logo
  - Button “Quay lai”
  - List of rented bike
    - Show rented bike information when you double click in one bike

b. Screen Transition



G. Design Considerations

I. Goals and guildlines

1. Goals

- Create an application which can help users with finding and renting/returning bikes conveniently.
- Develop a good design for the system.

2. Guildlines

- Do not rent a bike for too long because the account balance may not be enough.

II. Architectural Strategies

1. Using JavaFX to create UI and Java language to support multi-platform.
2. Using mySQL to manage the SQL database

III. Coupling and Cohension

This system has high cohension. The modules in the system are separated by the function it manages. No module has more than one responsibility. For example, ReturnBikeController only has one main responsibility, which is to handle the return bike request from the view class. Besides, RentBikeController also has only one main function is handling the deposit request to rent bike from users.

Howerver, the system has not yet achived loose coupling. It can only achieve Stamp Coupling. In some methods, a whole object is passed in but not all attributes of it are used in the function.

Example: + In the function saveReturnTransaction(String park_id, int bike_id, Event event, InterbankTransaction interbankTransaction) of class Return Bike Controller, an Event object was passed in the function but we do not use the attribute 'time' of it.

IV. Design principles

1. Single Responsibility Principle

The responsibility of the system is delegated to the packages with separate functions. For example, we have package ‘controllers’ to handle backend operations; package ‘view’ to handle the screen events, etc. In each package, each class is responsible for a single function.

2. Open/Closed Principle

The system is open for extension since we use inheritance to separate child classes. For example, if we need to add one more type of bike, we just need to add a class corresponding to that bike type and make sure it extends the base Bike class. Another example is when we need to add one more cost calculator strategy, we

only need to implement the method calculate() in another class and make it extend the base Calculator class.
Besides, the Ecobike system communicates with the subsystems through the interfaces, so that the subsystems can be extended easily provided that the interface remains the same.

3. Liskov Substitution Principle

This principle states that objects of a superclass shall be replaceable with objects of its subclasses without breaking the application. That requires the objects of the subclasses to behave in the same way as the objects of the superclass. In this system, for example, there are four cost calculation strategies corresponding to 4 types of bike. All of them extend the same interface and can replace the interface without causing any errors.

4. Interface Segregation Principle

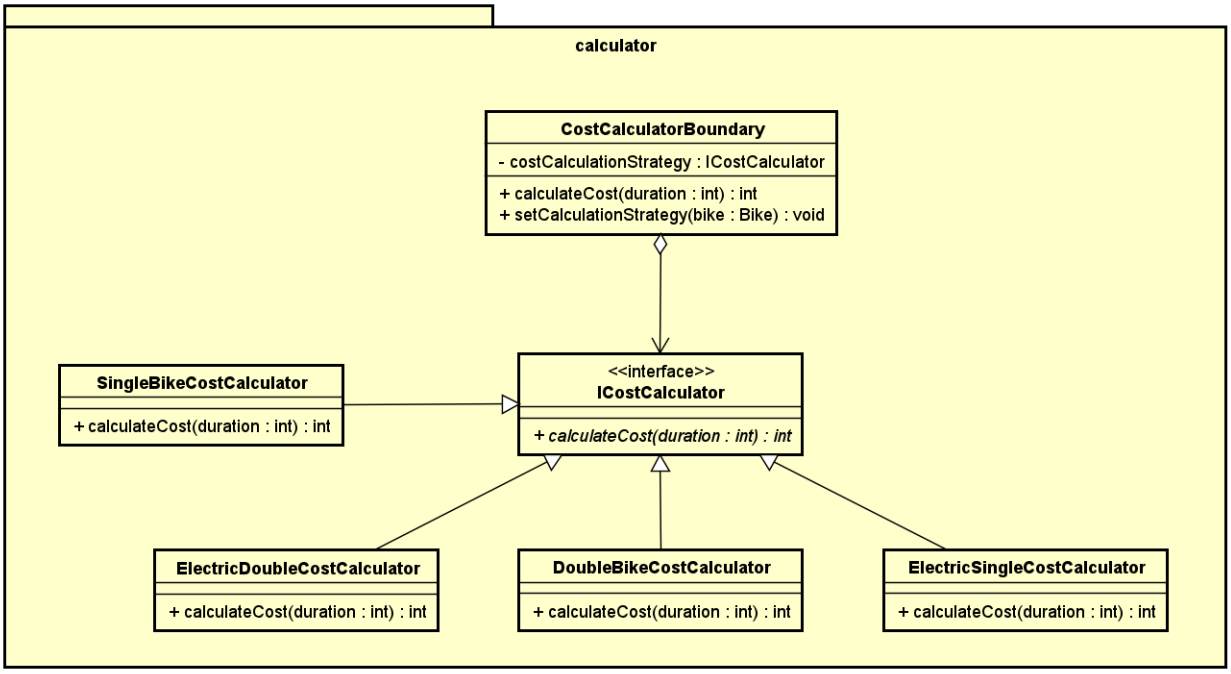
The ISP states that a class should not be forced to implement interfaces it does not use. Instead, it is better to split those interfaces into smaller and more specific ones so that the classes can only implement the methods that are relevant to them. This can lead to a better separation of concerns and more maintainable code.
However in this system, the Interbank Interface has 2 methods processTransaction() and reset() which can be separated into 2 interfaces with regards to ISP.

5. Dependency Inversion Principle

It states that high-level modules should not depend on low-level modules, but both should depend on abstractions. This means that the implementation details of low-level classes should be decoupled from the higher-level classes that use them. This can be achieved by depending on abstractions, rather than concrete implementations.
However, in this system, the RentBikeController depends directly on Card class. Card is a concrete implementation class and should not be used in this case. Instead, an abstract class should be used to follow DIP. In the future, there may be new payment methods so abstraction is highly necessary.

V. Design patterns

1. Strategy Pattern



The Strategy Design Pattern is a behavioral design pattern that defines a family of algorithms, encapsulates each one of them, and makes them interchangeable. This allows the algorithm to vary independently from the clients that use it.

The pattern involves creating objects which represent various strategies and a context object whose behavior varies as per its strategy object. The strategy object changes the executing algorithm of the context object.

It provides a way to create and switch between multiple algorithms dynamically, based on the requirements, without affecting the client code that uses them. This pattern helps to avoid hard-coding a specific algorithm in the client code, making it more flexible and maintainable.

The Strategy Design Pattern is often used in situations where there is a need to switch between multiple algorithms at runtime, based on user inputs or system

conditions. Some common use cases include, sorting algorithms, compression algorithms, and payment methods.

In the package ‘calculator’, there are 4 stategies to calculate cost. The Boundary class can switch between those with respect to the bike type. The client code calls the method of the boundary class object without any awareness of the underlying calculating code.

H. Conclusion

In this document, we have eleborated on every step of designing the system. Due to the lack of time, the application is not yet of the highest quality. In the future, our team can develop the system to satisfy the SOLID principles as well as taking advantages of more design paterrns to make the system better. Besides, we will also add multiple functions into the app, such as login/logout, pause/resume renting, etc.