HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology

Software Design Document

Version 1.3

EcoBike Application

Subject: ITSS Software Management

Group 6

No.	Student Name	Student ID
1	Nguyen Thi Minh Chau	20184238
2	Tran Le Hai Duong	20184248
3	Nguyen Thanh Long	20184287

Hanoi, 01/2022

Table of Contents

Table of Contents 3

List of Figures 6

List of Tables 7

1. Introduction 8
- 1.1. Objective 8
- 1.2. Scope 8
- 1.3. Glossary 8
- 1.4. References 9

2. Overall Description 10
- 2.1. General Overview 10
- 2.2. Assumptions/Constraints/Risks 11
  - 2.2.1. Assumptions 11
  - 2.2.2. Constraints 11
  - 2.2.3. Risks 11

3. System Architecture and Architecture Design 12
- 3.1. Architectural Patterns 12
- 3.2. Interaction Diagrams 12
  - 3.2.1. Communication Diagrams 12
  - 3.2.2. Sequence Diagrams 15
- 3.3. Analysis Class Diagrams 18
- 3.4. Unified Analysis Class Diagram 21
- 3.5. Security Software Architecture 21

4. Detailed Design 22
- 4.1. User Interface Design 22
  - 4.1.1. Screen Configuration Standardization 22
  - 4.1.2. Screen Transition Diagrams 22
  - 4.1.3. Screen Specifications 22
    - 4.1.3.1. Splash Screen 22
    - 4.1.3.2. Main Screen 23
    - 4.1.3.3. Dock Screen 23
    - 4.1.3.4. Bike Screen 24
    - 4.1.3.5. Payment Method Screen 25
    - 4.1.3.6. Deposit screen 25
    - 4.1.3.7. Payment screen 25
- 4.2. Data Modeling 27
  - 4.2.1. Conceptual Data Modeling 27
  - 4.2.2. Database Design 27

4.2.2.1. Database Management System 27
4.2.2.2. Database Diagram 28
4.2.2.3. Database Detail Design 28
4.3. Non-Database Management System Files 35
4.4. Class Design 35
4.4.1. General Class Diagram 35
4.4.2. Class Diagrams 36
4.4.2.1. Class Diagram for Package BikeInformation 36
4.4.2.2. Class Diagram for Subsystem RentBike 37
4.4.2.3. Class Diagram for Subsystem InterBank 37
4.4.3. Class Design 38
4.4.3.1. Class RentBikeController 38
4.4.3.2. Class ReturnBikeController 39
4.4.3.3. Class PaymentController 41
4.4.3.4. Class BikeInformationScreenHandler 43

5. Design Considerations 45
5.1. Goals and Guidelines 45
5.2. Architectural Strategies 45
5.3. Coupling and Cohesion 45
5.3.1. Coupling 45
5.3.1.1. Content coupling 45
5.3.1.2. Common coupling 46
5.3.1.3. Control coupling 46
5.3.1.4. Stamp coupling 46
5.3.1.5. Data coupling 46
5.3.2. Cohesion 47
5.3.2.1. Coincidental cohesion 47
5.3.2.2. Logical cohesion 47
5.3.2.3. Temporal cohesion 47
5.3.2.4. Procedure cohesion 47
5.3.2.5. Communicational cohesion 47
5.3.2.6. Sequential cohesion 48
5.3.2.7. Information cohesion 48
5.3.2.8. Functional cohesion 48
5.4. Design Principles 48
5.5. Design Patterns 48

List of Figures

Figure 1.1: General use case diagram 11

Figure 3.1: Communication Diagram for Rent Bike Use Case 13

Figure 3.2: Communication Diagram for Deposit Use Case 13

Figure 3.3: Communication Diagram for Update Payment Method Use Case 14

Figure 3.4: Communication Diagram for Return Bike Use Case 14

Figure 3.5: Communication Diagram for Return Deposit Use Case 15

Figure 3.6: Communication Diagram for Pay For Rental Use Case 15

Figure 3.7: Sequence Diagram for Rent Bike Use Case 16

Figure 3.8: Sequence Diagram for Deposit Use Case 16

Figure 3.9: Sequence Diagram for Update Payment Method Use Case 17

Figure 3.10: Sequence Diagram for Return Bike Use Case 17

Figure 3.11: Sequence Diagram for Return Deposit Use Case 18

Figure 3.12: Sequence Diagram for Pay For Rental Use Case 18

Figure 3.13: Class Diagram for View Bike Use Case 19

Figure 3.14: Class Diagram for View Bike Use Case 19

Figure 3.15: Class Diagram for Deposit Use Case 20

Figure 3.16: Class Diagram for Return Bike Use Case 20

Figure 3.17: Class Diagram for Return Deposit Use Case 21

Figure 3.18: Class Diagram for Pay Rental Use Case 21

Figure 3.19: Unified Class Diagram for EcoBike Application 21

Figure 4.1: Screen Transition Diagram for EcoBike Application 23

Figure 4.2. ER Diagram for EcoBike Application 28

Figure 4.3. Database Diagram for EcoBike Application 29

Figure 4.4. General Class Diagram for EcoBike Application 36

Figure 4.5. Class Diagram for Package BikeInformation 37

Figure 4.6. Class Diagram for Subsystem RentBike 38

Figure 4.7. Class Diagram for Subsystem InterBank *38*

Figure 4.8. RentBikeController Class Diagram *39*

Figure 4.9. ReturnBikeController Class Diagram *40*

Figure 4.10. PaymentControllerClass Diagram *42*

List of Tables

Table 1.1: Terms used in the document *10*

Table 4.1. Splash Screen Specification *24*

Table 4.2. Main Screen Specification *24*

Table 4.3. View Dock Screen Specification *25*

Table 4.4. View Bike Screen Specification *25*

Table 4.5. Payment Method Screen Specification *26*

Table 4.6. Deposit Screen Specification *26*

Table 4.7. Payment Screen Specification *27*

Table 4.8. Customer table design *29*

Table 4.9. Administrator table design *30*

Table 4.10. Dock table design *30*

Table 4.11. Bike table design *31*

Table 4.12. Bike In Dock table design *31*

Table 4.13. Bike Status table design *32*

Table 4.14. Invoice table design *32*

Table 4.15. Transaction table design *33*

Table 4.16. Rent Bike table design *33*

Table 4.17. Credit Card table design *34*

Table 4.18. RentBikeController attributes *39*

Table 4.19. RentBikeController operations *40*

Table 4.20. BikeTracker attributes *41*

Table 4.21. BikeTracker operations *41*

Table 4.22. PaymentController attributes *42*

Table 4.23. PaymentController operations *43*

Table 4.24. BikeInformationScreenHandler operations *45*

Table 4.25. BikeInformationScreenHandler operations *45*

1. Introduction

1.1. Objective

This Software Design Document provides the design of EcoBike Application. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli.

The expected audience is the user of the EcoBike Application, including Mrs. Trang and the ITSS Software development Course's students, the developer of the project, and the people who will maintain the EcoBike Application.

1.2. Scope

This document contains a complete description of the design of EcoBike Application.

The EcoBike Application is a mobile application that allows the resident of Hanoi to use the EcoBike service, including finding out information of nearby docks and renting bikes for personal usage and online payment for the renting process.

The objective of the EcoBike Application is to serve a maximum of 100.000 users concurrently, with a friendly and easy-to-use user interface with the aim of helping the user to find the most suitable place to rent or return the bike.

1.3. Glossary

Term	Definition
Administrator	The person who uses EcoBike application system for the purposes of monitoring list of bicycles in the system
Admin	as “administrator”
Bicycle	The transportation mean to be rent in this application system
Bike	as “bicycle”
Card number	The ID number of the credit card, printed on the credit card
Cardholder name	The name of the owner of the credit card, printed on the credit card
Credit card	A card connected to the interbank, used for performing transaction
Customer	The person who uses EcoBike application system for the purposes of renting bike

Database	Collection of all information monitored by this system
Deposit	An amount of money customer has to pay at first in order to rent a bike
Dock	A place where bicycles are put
Interbank	The organization in charges of performing payment and return deposit transactions in the system
Payment	An amount of money customer has to pay to rent a bike, including deposit and rental fee
Rent a bike	The action of using a bike in a period of time, with paying deposit and rental fee
Rental fee	An amount of money customer has to pay, outside of the deposit, which depends on the rental time
Rental time	The time period when the bike is being rented
Return a bike	The action of stopping using a bike after having rented
Software Requirement Specification	A document that completely describes all of the functions of a proposed system and the constraints under which it must operate. For example, this document.
Station	as “dock”
Transaction	The action of paying for bike deposit, bike rental or returning deposit
User	Customer or Administrator

Table 1.1: Terms used in the document

1.4. References

[1] Centers for Medicare & Medicaid Services, "System Design Document Template," [Online]. Available: https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SystemDesignDocument.docx.

2. Overall Description

2.1. General Overview

EcoBike Application is a desktop application through which users can view docks and rent or return bikes. We design a clean and clear interface for users. Users can interact with the apps by clicking on the interface, and the request is processed by the controller.

Additionally, we have our own database to store information and data that is related to our system, as well as a subsystem to proceed payment transactions.

The below figure is the general use-case diagram for our design:

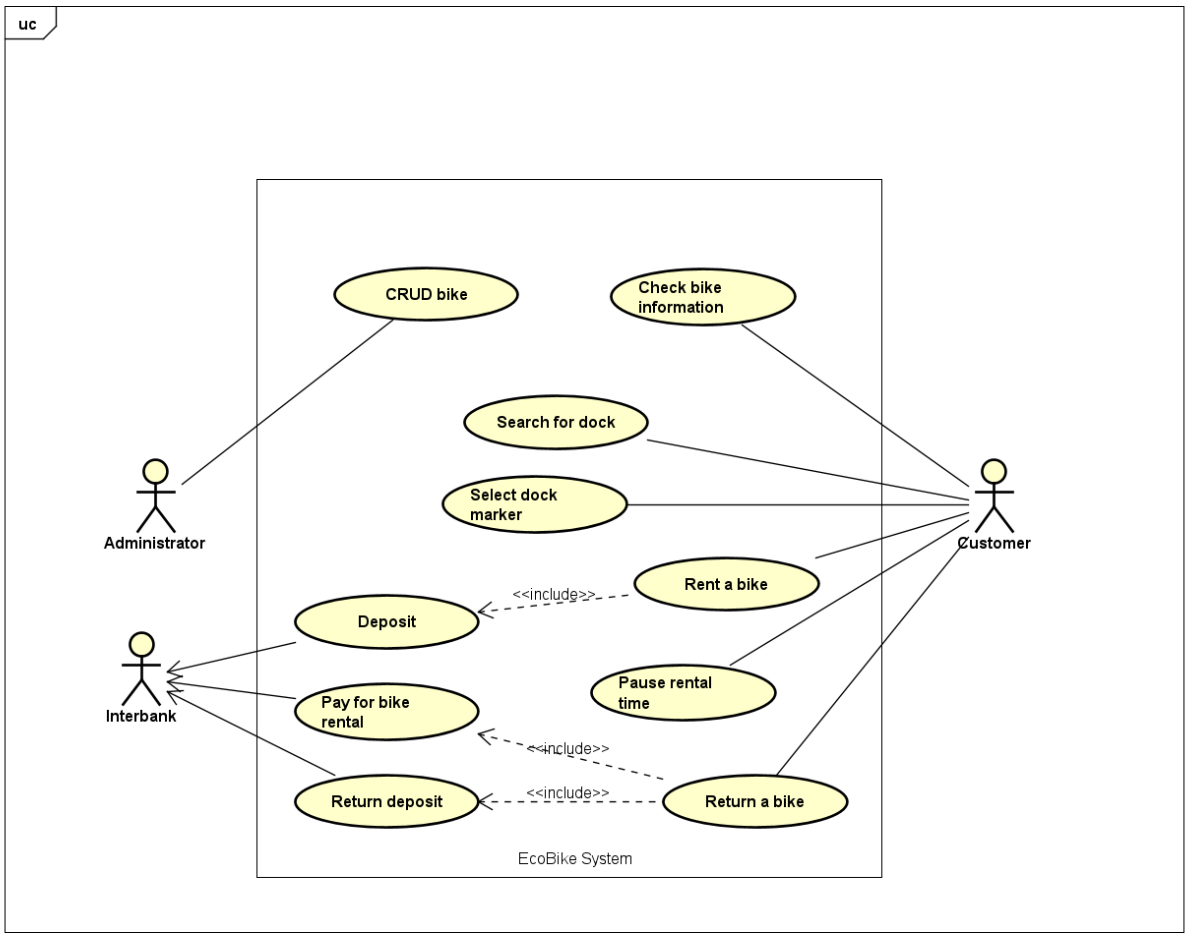


Figure 1.1: General use case diagram

2.2. Assumptions/Constraints/Risks

2.2.1. Assumptions

In order to use the application, users must have an internet connection as well as a personal computer to run the app. We would also require the latest version of JRE in order to ensure the application' stability.

2.2.2. Constraints

· *Hardware or software environment*

· *End-user environment*

· *Availability or volatility of resources*

· *Standards compliance*

· *Interoperability requirements*

· *Interface/protocol requirements*

· *Licensing requirements*

· *Data repository and distribution requirements*

· *Security requirements (or other such regulations)*

· *Memory or other capacity limitations*

· *Performance requirements*

· *Network communications*

· *Verification and validation requirements (testing)*

· *Other means of addressing quality goals*

· *Other requirements described in the Requirements Document*

2.2.3. Risks

3. System Architecture and Architecture Design

3.1. Architectural Patterns

<Specify and briefly describe the chosen architectural patterns and the reasons why they were chosen>

3.2. Interaction Diagrams

3.2.1. Communication Diagrams

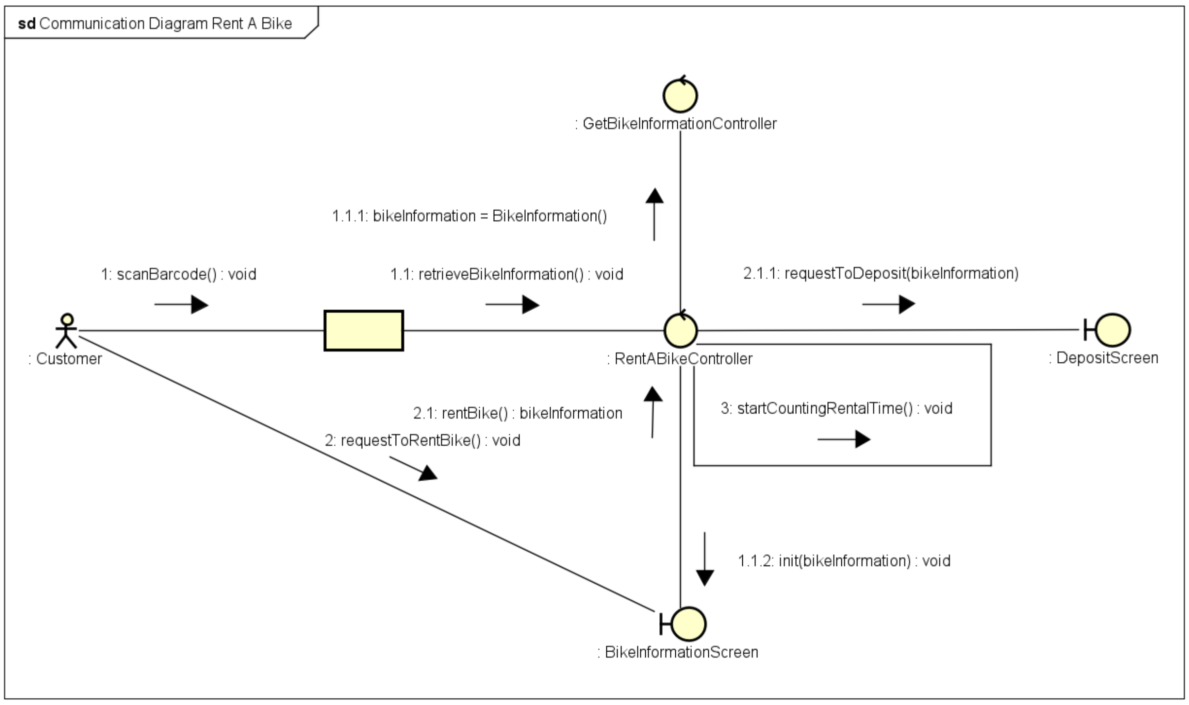

sd Communication Diagram Rent A Bike
: GetBikeInformationController
1.1.1: bikeInformation = BikeInformation()
1: scanBarcode() : void
1.1: retrieveBikeInformation() : void
2.1.1: requestToDeposit(bikeInformation)
: Customer
: RentABikeController
: DepositScreen
2.1: rentBike() : bikeInformation
2: requestToRentBike() : void
3: startCountingRentalTime() : void
1.1.2: init(bikeInformation) : void
: BikeInformationScreen

Figure 3.1: Communication Diagram for Rent Bike Use Case

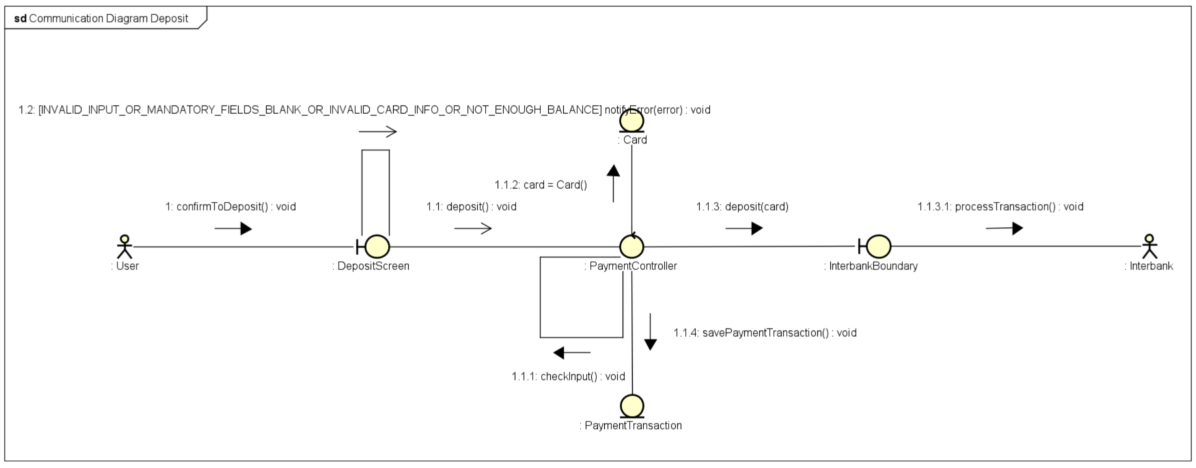

sd Communication Diagram Deposit
1.2: [INVALID_INPUT_OR_MANDATORY_FIELDS_BLANK_OR_INVALID_CARD_INFO_OR_NOT_ENOUGH_BALANCE] notifyError(error) : void
: Card
1.1.2: card = Card()
1: confirmToDeposit() : void
1.1: deposit() : void
1.1.3: deposit(card)
1.1.3.1: processTransaction() : void
: User
: DepositScreen
: PaymentController
: InterbankBoundary
: Interbank
1.1.4: savePaymentTransaction() : void
1.1.1: checkInput() : void
: PaymentTransaction

Figure 3.2: Communication Diagram for Deposit Use Case

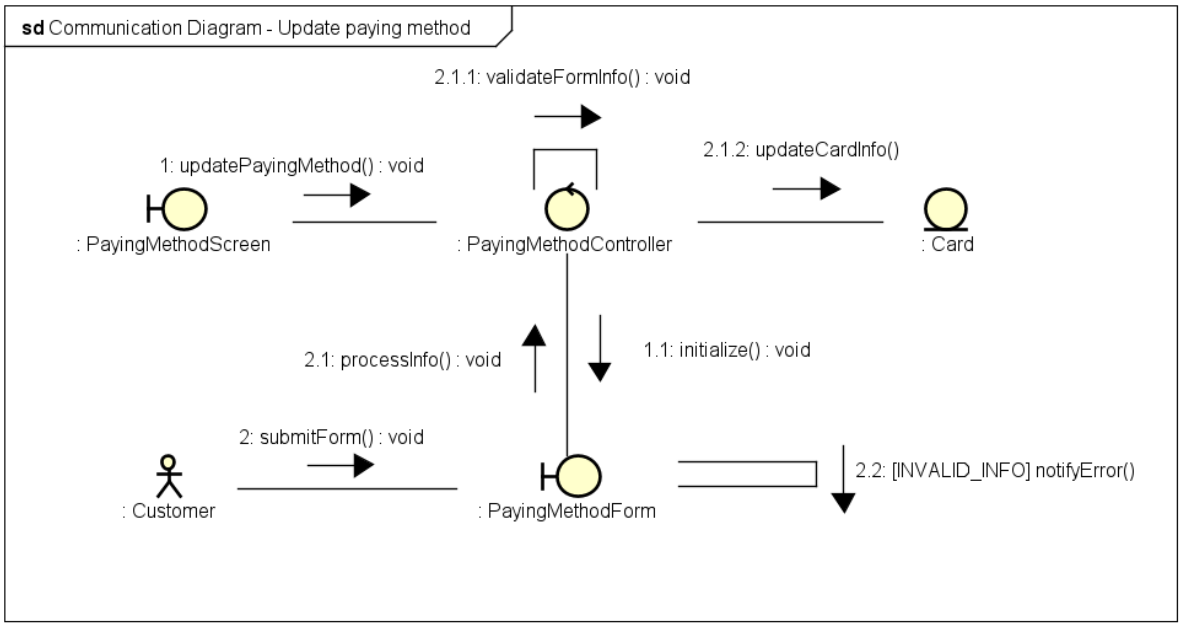

sd Communication Diagram - Update paying method
2.1.1: validateFormInfo() : void
1: updatePayingMethod() : void
2.1.2: updateCardInfo()
: PayingMethodScreen
: PayingMethodController
: Card
2.1: processInfo() : void
1.1: initialize() : void
2: submitForm() : void
: Customer
: PayingMethodForm
2.2: [INVALID_INFO] notifyError()

Figure 3.3: Communication Diagram for Update Payment Method Use Case

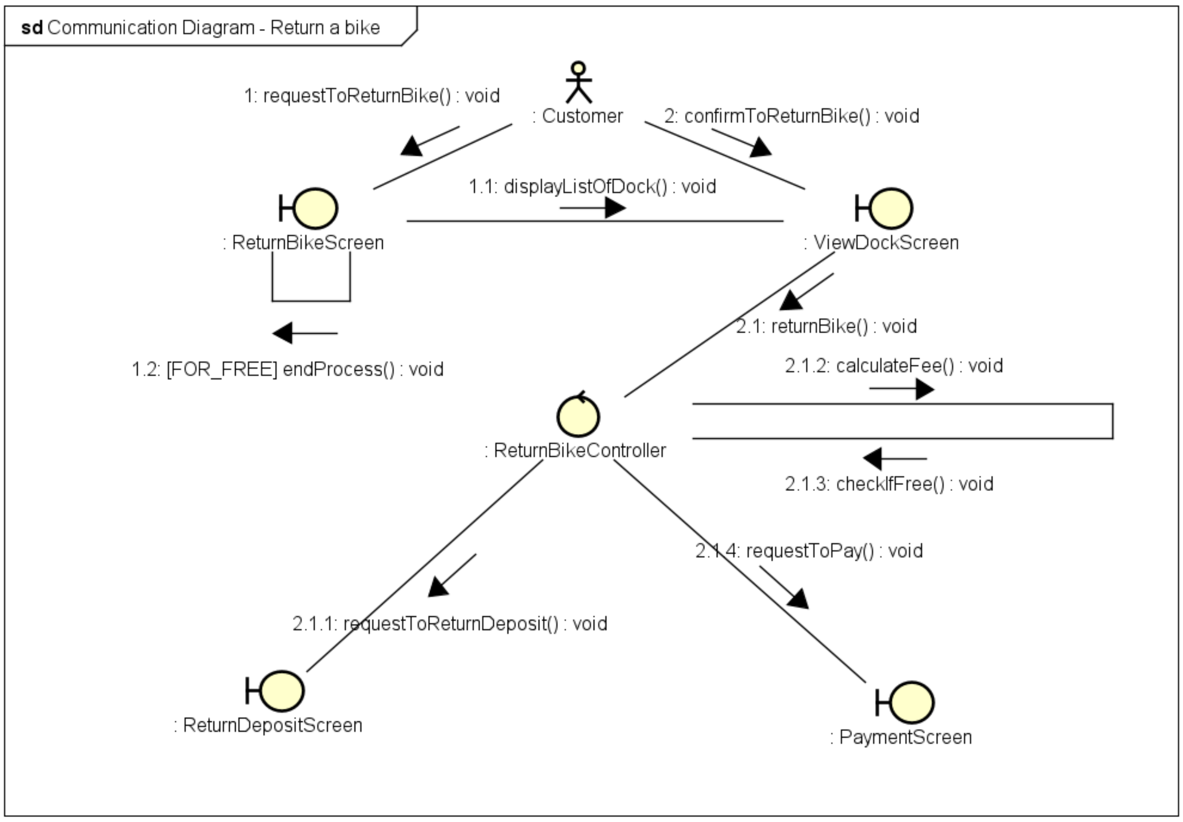

sd Communication Diagram - Return a bike
1: requestToReturnBike() : void
: Customer
2: confirmToReturnBike() : void
1.1: displayListOfDock() : void
: ReturnBikeScreen
: ViewDockScreen
2.1: returnBike() : void
1.2: [FOR_FREE] endProcess() : void
2.1.2: calculateFee() : void
: ReturnBikeController
2.1.3: checkIfFree() : void
2.1.4: requestToPay() : void
2.1.1: requestToReturnDeposit() : void
: ReturnDepositScreen
: PaymentScreen

Figure 3.4: Communication Diagram for Return Bike Use Case

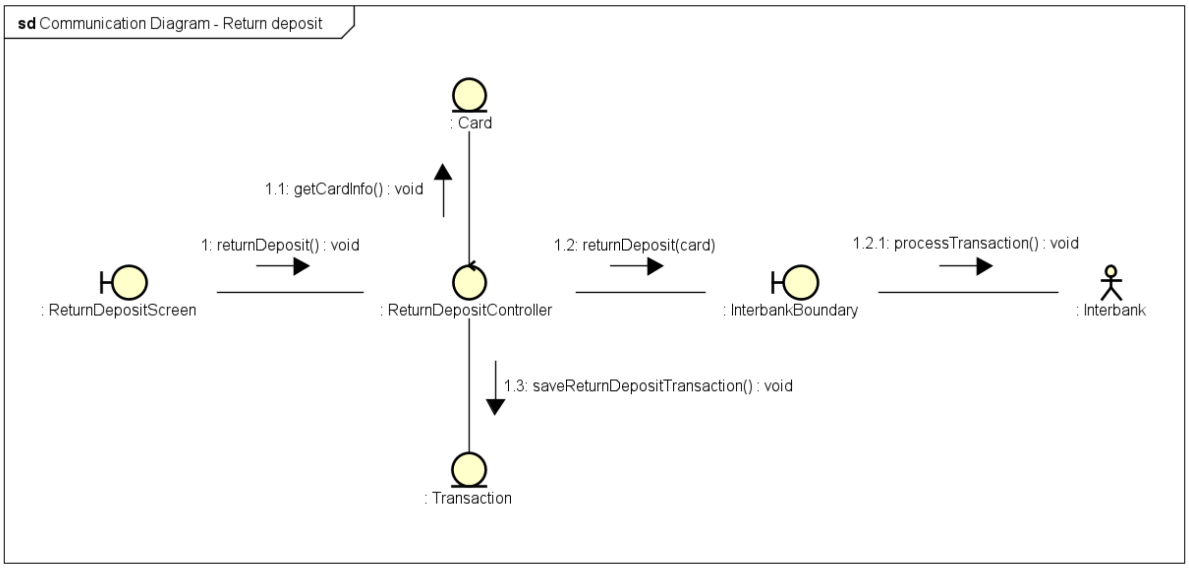

sd Communication Diagram - Return deposit
: Card
1.1: getCardInfo() : void
1: returnDeposit() : void
1.2: returnDeposit(card)
1.2.1: processTransaction() : void
: ReturnDepositScreen
: ReturnDepositController
: InterbankBoundary
: Interbank
1.3: saveReturnDepositTransaction() : void
: Transaction

Figure 3.5: Communication Diagram for Return Deposit Use Case

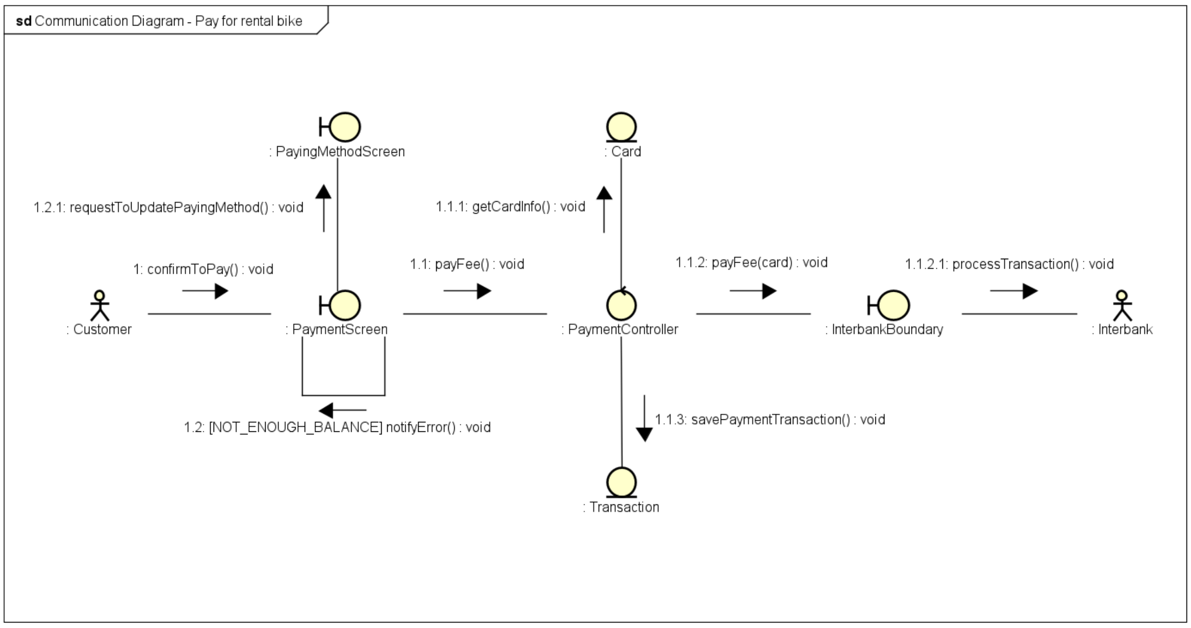

sd Communication Diagram - Pay for rental bike
: PayingMethodScreen
: Card
1.2.1: requestToUpdatePayingMethod() : void
1.1.1: getCardInfo() : void
1: confirmToPay() : void
1.1: payFee() : void
1.1.2: payFee(card) : void
1.1.2.1: processTransaction() : void
: Customer
: PaymentScreen
: PaymentController
: InterbankBoundary
: Interbank
1.2: [NOT_ENOUGH_BALANCE] notifyError() : void
1.1.3: savePaymentTransaction() : void
: Transaction

Figure 3.6: Communication Diagram for Pay For Rental Use Case

3.2.2. Sequence Diagrams

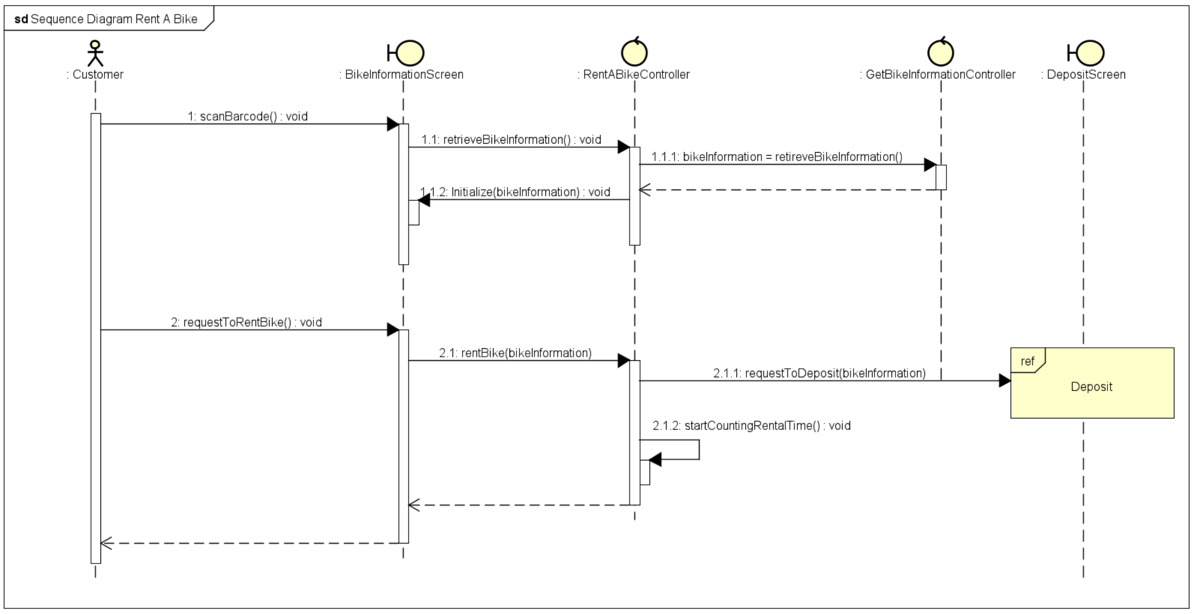

sd Sequence Diagram Rent A Bike
: Customer
: BikeInformationScreen
: RentABikeController
: GetBikeInformationController
: DepositScreen
1: scanBarcode() : void
1.1: retrieveBikeInformation() : void
1.1.1: bikeInformation = retireveBikeInformation()
1.1.2: Initialize(bikeInformation) : void
2: requestToRentBike() : void
2.1: rentBike(bikeInformation)
2.1.1: requestToDeposit(bikeInformation)
ref
Deposit
2.1.2: startCountingRentalTime() : void

Figure 3.7: Sequence Diagram for Rent Bike Use Case

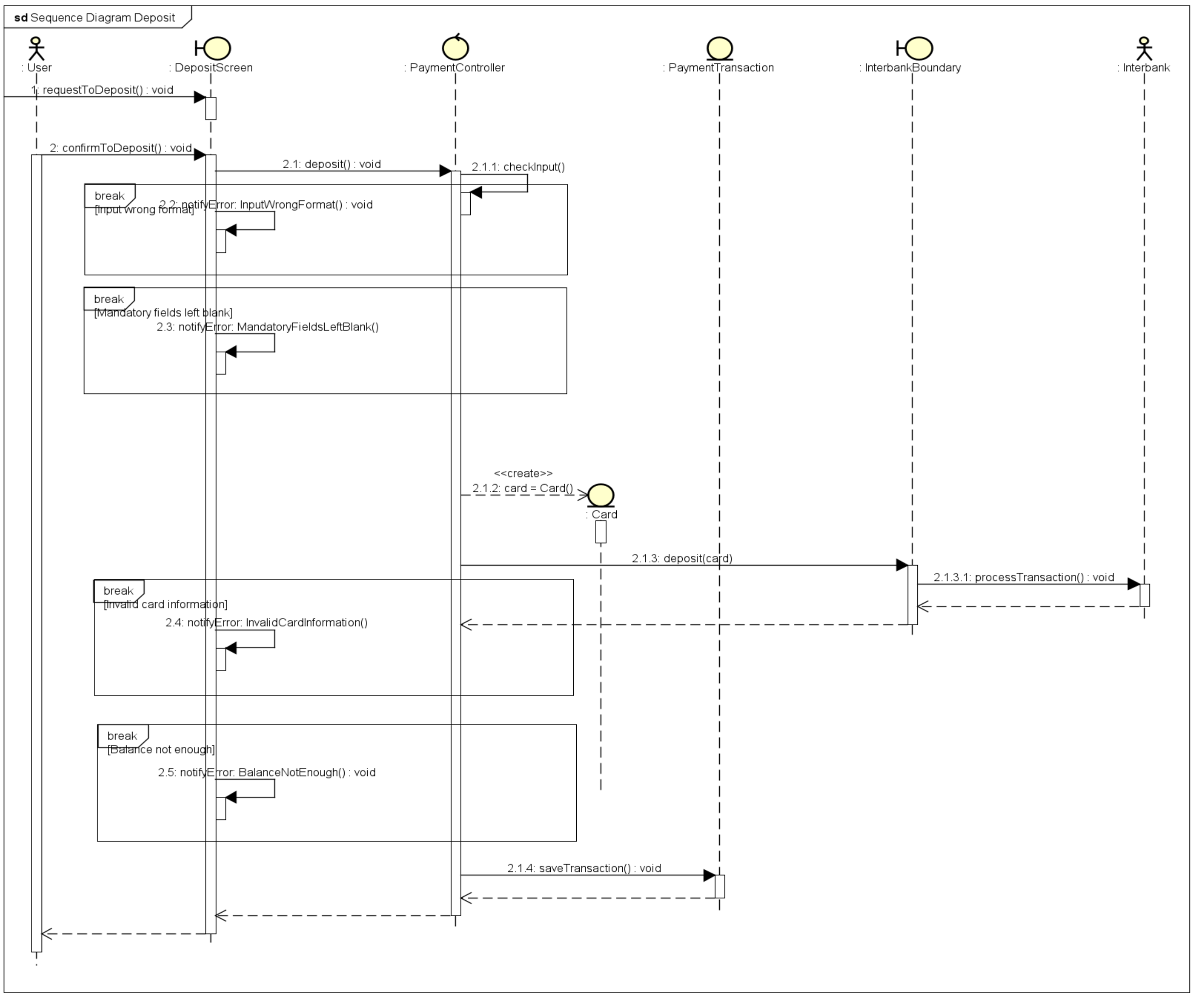

sd Sequence Diagram Deposit
: User
: DepositScreen
: PaymentController
: PaymentTransaction
: InterbankBoundary
: Interbank
1: requestToDeposit() : void
2: confirmToDeposit() : void
2.1: deposit() : void
2.1.1: checkInput()
break
[Input wrong format]
2.2: notifyError: InputWrongFormat() : void
break
[Mandatory fields left blank]
2.3: notifyError: MandatoryFieldsLeftBlank()
<<create>>
2.1.2: card = Card()
: Card
2.1.3: deposit(card)
2.1.3.1: processTransaction() : void
break
[Invalid card information]
2.4: notifyError: InvalidCardInformation()
break
[Balance not enough]
2.5: notifyError: BalanceNotEnough() : void
2.1.4: saveTransaction() : void

Figure 3.8: Sequence Diagram for Deposit Use Case

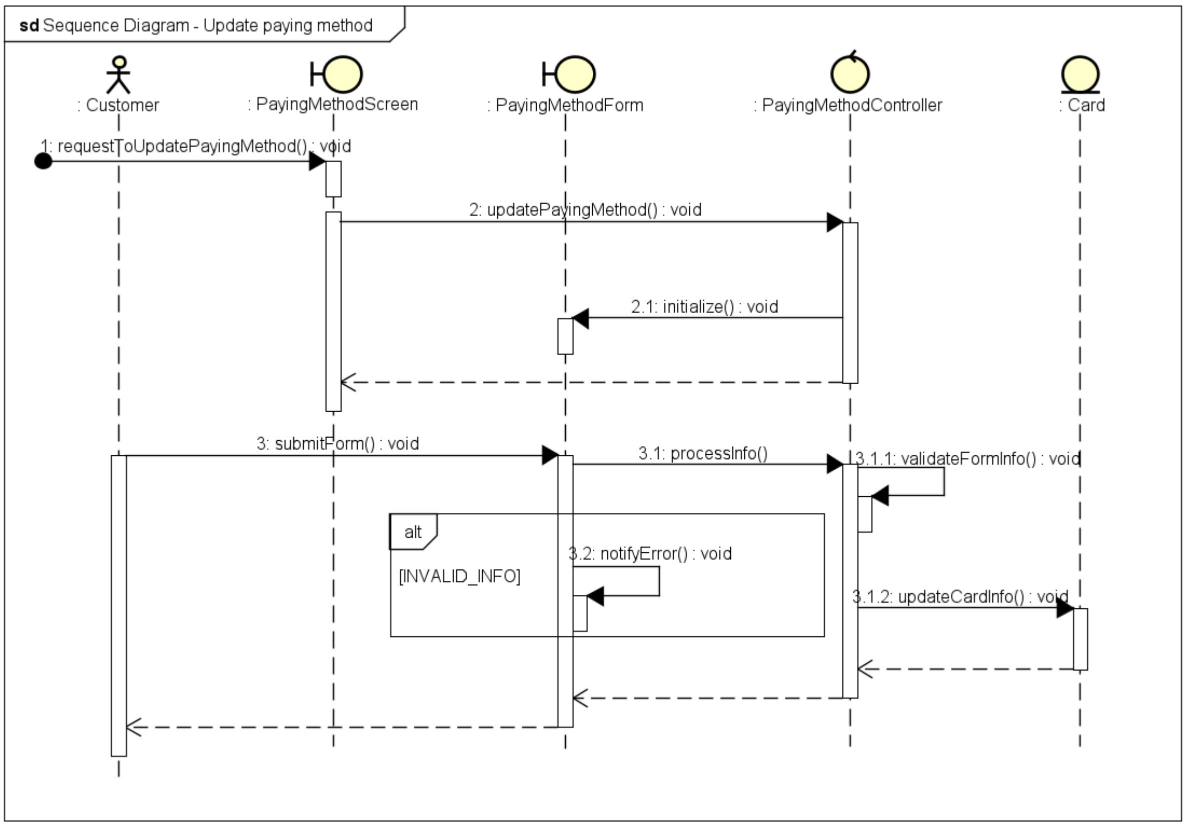

sd Sequence Diagram - Update paying method
: Customer
: PayingMethodScreen
: PayingMethodForm
: PayingMethodController
: Card
1: requestToUpdatePayingMethod() : void
2: updatePayingMethod() : void
2.1: initialize() : void
3: submitForm() : void
3.1: processInfo()
3.1.1: validateFormInfo() : void
alt
[INVALID_INFO]
3.2: notifyError() : void
3.1.2: updateCardInfo() : void

Figure 3.9: Sequence Diagram for Update Payment Method Use Case

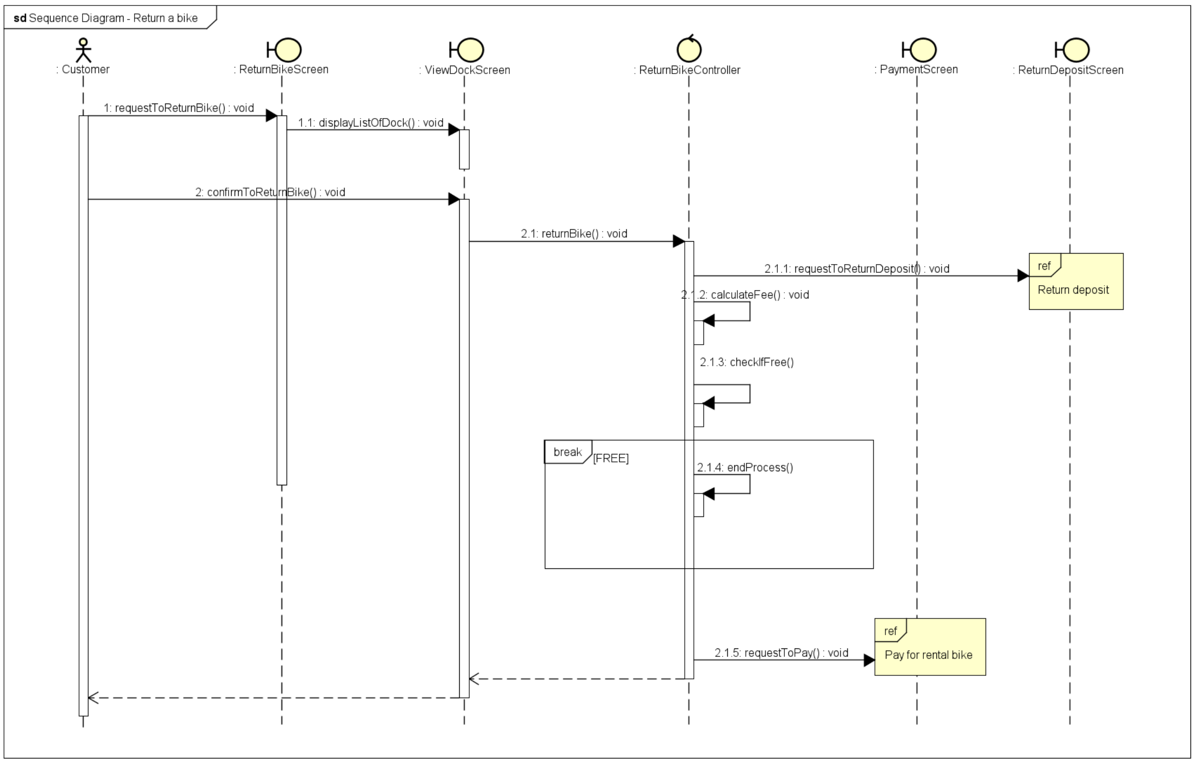
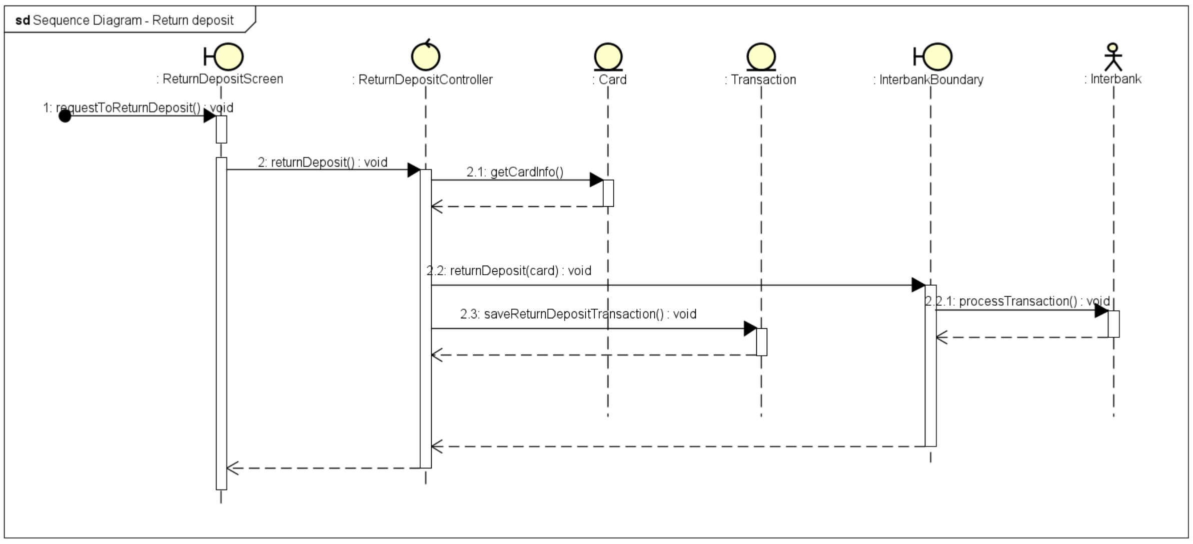

sd Sequence Diagram - Return a bike
: Customer
: ReturnBikeScreen
: ViewDockScreen
: ReturnBikeController
: PaymentScreen
: ReturnDepositScreen
1: requestToReturnBike() : void
1.1: displayListOfDock() : void
2: confirmToReturnBike() : void
2.1: returnBike() : void
2.1.1: requestToReturnDeposit() : void
ref
Return deposit
2.1.2: calculateFee() : void
2.1.3: checkIfFree()
break
[FREE]
2.1.4: endProcess()
2.1.5: requestToPay() : void
ref
Pay for rental bike

Figure 3.10: Sequence Diagram for Return Bike Use Case



sd Sequence Diagram - Return deposit
: ReturnDepositScreen
: ReturnDepositController
: Card
: Transaction
: InterbankBoundary
: Interbank
1: requestToReturnDeposit() : void
2: returnDeposit() : void
2.1: getCardInfo()
2.2: returnDeposit(card) : void
2.2.1: processTransaction() : void
2.3: saveReturnDepositTransaction() : void

Figure 3.11: Sequence Diagram for Return Deposit Use Case

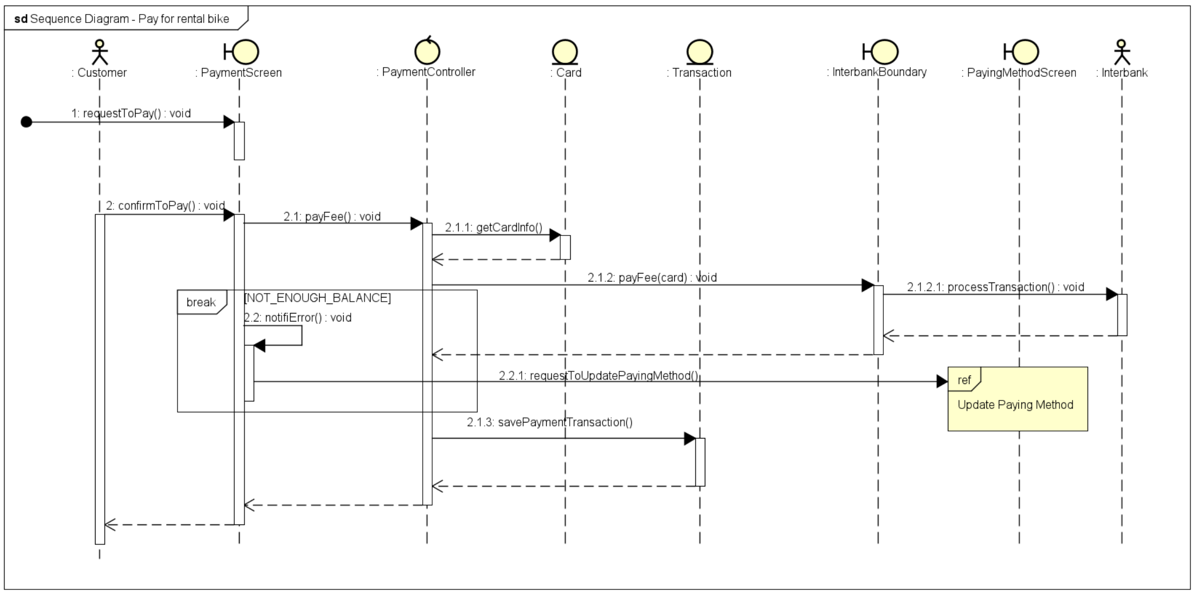


Figure 3.12: Sequence Diagram for Pay For Rental Use Case

3.3. Analysis Class Diagrams

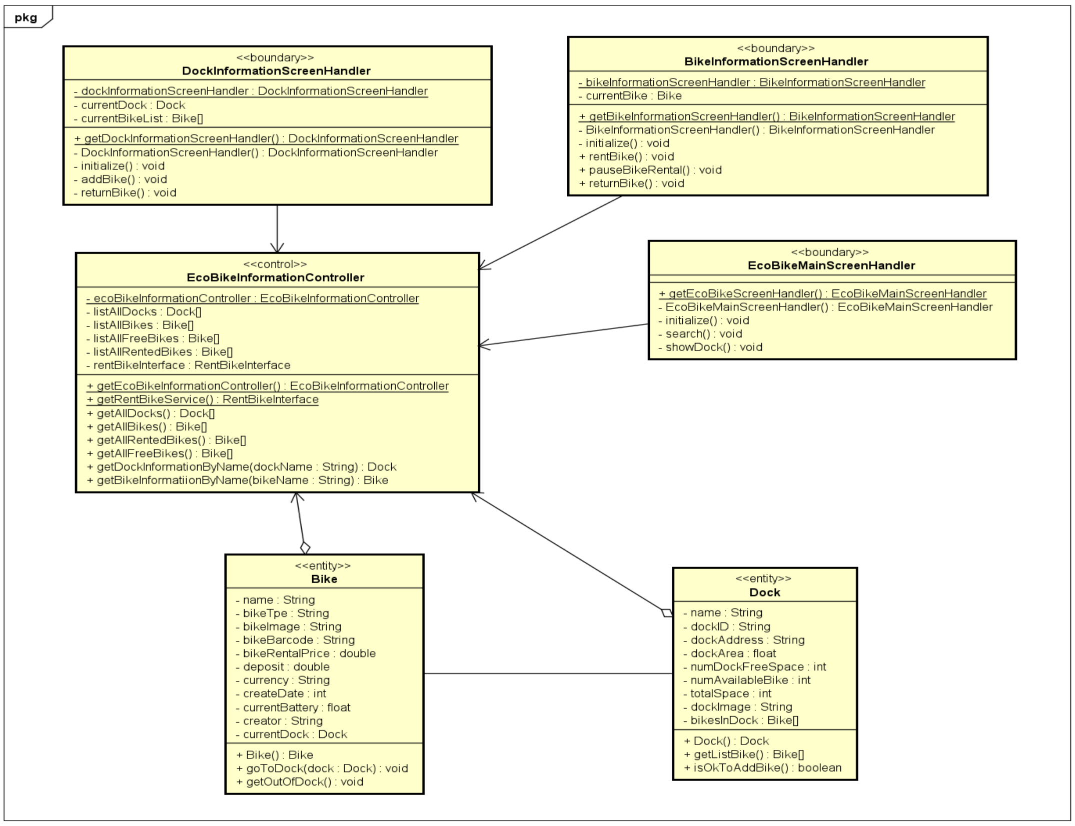

pkg
<<boundary>>
DockInformationScreenHandler
- dockInformationScreenHandler : DockInformationScreenHandler
- currentDock : Dock
- currentBikeList : Bike[]
+ getDockInformationScreenHandler() : DockInformationScreenHandler
- DockInformationScreenHandler() : DockInformationScreenHandler
- initialize() : void
- addBike() : void
- returnBike() : void
<<boundary>>
BikeInformationScreenHandler
- bikeInformationScreenHandler : BikeInformationScreenHandler
- currentBike : Bike
+ getBikeInformationScreenHandler() : BikeInformationScreenHandler
- BikeInformationScreenHandler() : BikeInformationScreenHandler
- initialize() : void
+ rentBike() : void
+ pauseBikeRental() : void
+ returnBike() : void
<<control>>
EcoBikeInformationController
- ecoBikeInformationController : EcoBikeInformationController
- listAllDocks : Dock[]
- listAllBikes : Bike[]
- listAllFreeBikes : Bike[]
- listAllRentedBikes : Bike[]
- rentBikeInterface : RentBikeInterface
+ getEcoBikeInformationController() : EcoBikeInformationController
+ getRentBikeService() : RentBikeInterface
+ getAllDocks() : Dock[]
+ getAllBikes() : Bike[]
+ getAllRentedBikes() : Bike[]
+ getAllFreeBikes() : Bike[]
+ getDockInformationByName(dockName : String) : Dock
+ getBikeInformatiionByName(bikeName : String) : Bike
<<boundary>>
EcoBikeMainScreenHandler
+ getEcoBikeScreenHandler() : EcoBikeMainScreenHandler
- EcoBikeMainScreenHandler() : EcoBikeMainScreenHandler
- initialize() : void
- search() : void
- showDock() : void
<<entity>>
Bike
- name : String
- bikeTpe : String
- bikeImage : String
- bikeBarcode : String
- bikeRentalPrice : double
- deposit : double
- currency : String
- createDate : int
- currentBattery : float
- creator : String
- currentDock : Dock
+ Bike() : Bike
+ goToDock(dock : Dock) : void
+ getOutOfDock() : void
<<entity>>
Dock
- name : String
- dockID : String
- dockAddress : String
- dockArea : float
- numDockFreeSpace : int
- numAvailableBike : int
- totalSpace : int
- dockImage : String
- bikesInDock : Bike[]
+ Dock() : Dock
+ getListBike() : Bike[]
+ isOkToAddBike() : boolean

Figure 3.13: Class Diagram for View Bike Use Case

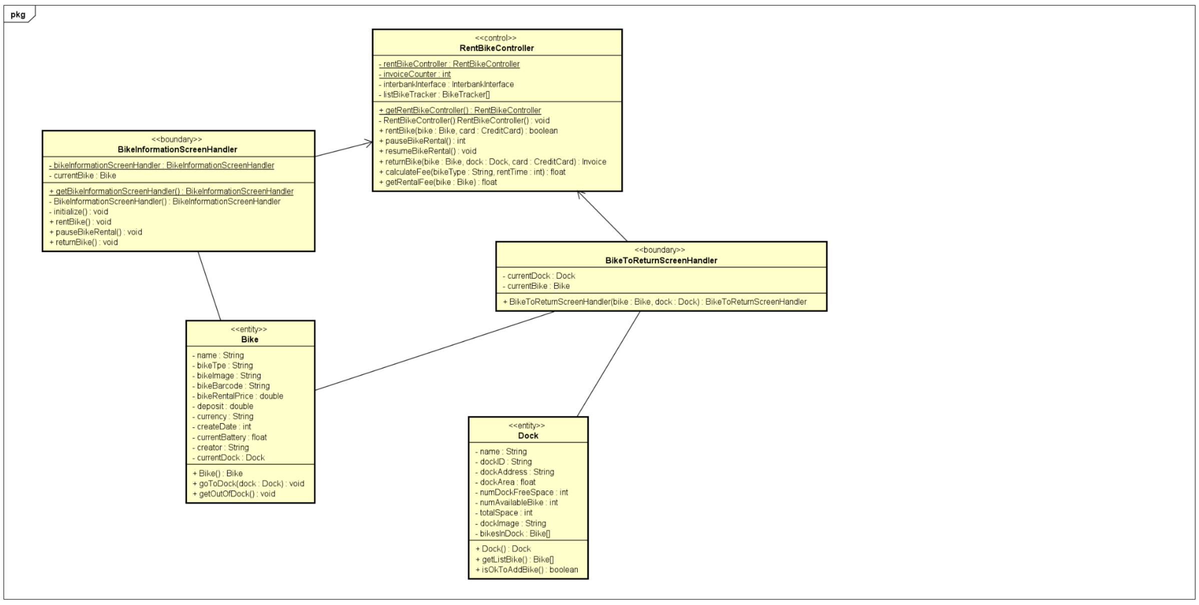

pkg
<<control>>
RentBikeController
- rentBikeController : RentBikeController
- invoiceCounter : int
- interbankInterface : InterbankInterface
- listBikeTracker : BikeTracker[]
+ getRentBikeController() : RentBikeController
- RentBikeController():RentBikeController() : void
+ rentBike(bike : Bike, card : CreditCard) : boolean
+ pauseBikeRental() : int
+ resumeBikeRental() : void
+ returnBike(bike : Bike, dock : Dock, card : CreditCard) : Invoice
+ calculateFee(bikeType : String, rentTime : int) : float
+ getRentalFee(bike : Bike) : float
<<boundary>>
BikeInformationScreenHandler
- bikeInformationScreenHandler : BikeInformationScreenHandler
- currentBike : Bike
+ getBikeInformationScreenHandler() : BikeInformationScreenHandler
- BikeInformationScreenHandler() : BikeInformationScreenHandler
- initialize() : void
+ rentBike() : void
+ pauseBikeRental() : void
+ returnBike() : void
<<boundary>>
BikeToReturnScreenHandler
- currentDock : Dock
- currentBike : Bike
+ BikeToReturnScreenHandler(bike : Bike, dock : Dock) : BikeToReturnScreenHandler
<<entity>>
Bike
- name : String
- bikeTpe : String
- bikeImage : String
- bikeBarcode : String
- bikeRentalPrice : double
- deposit : double
- currency : String
- createDate : int
- currentBattery : float
- creator : String
- currentDock : Dock
+ Bike() : Bike
+ goToDock(dock : Dock) : void
+ getOutOfDock() : void
<<entity>>
Dock
- name : String
- dockID : String
- dockAddress : String
- dockArea : float
- numDockFreeSpace : int
- numAvailableBike : int
- totalSpace : int
- dockImage : String
- bikesInDock : Bike[]
+ Dock() : Dock
+ getListBike() : Bike[]
+ isOkToAddBike() : boolean

Figure 3.14: Class Diagram for Rent Bike Use Case

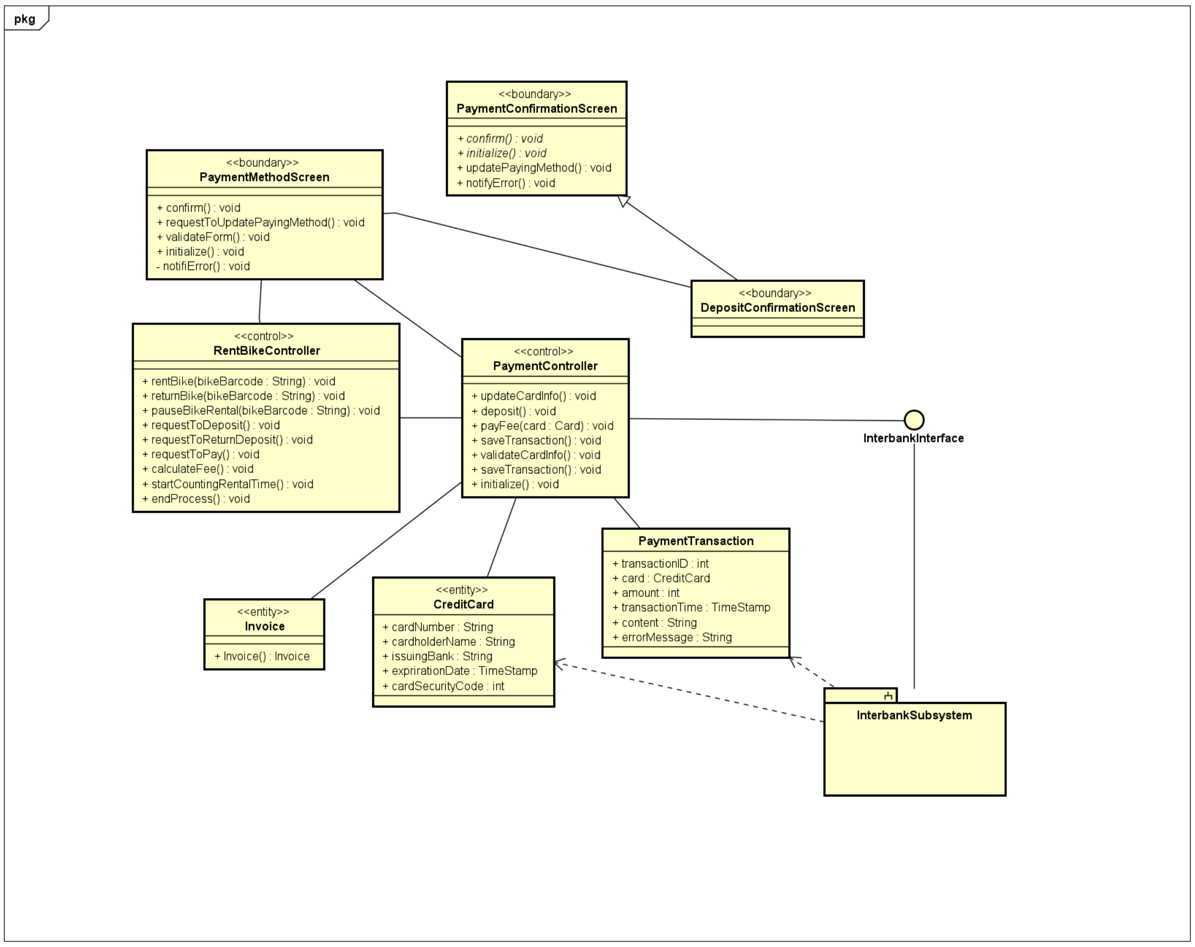


Figure 3.15: Class Diagram for Deposit Use Case

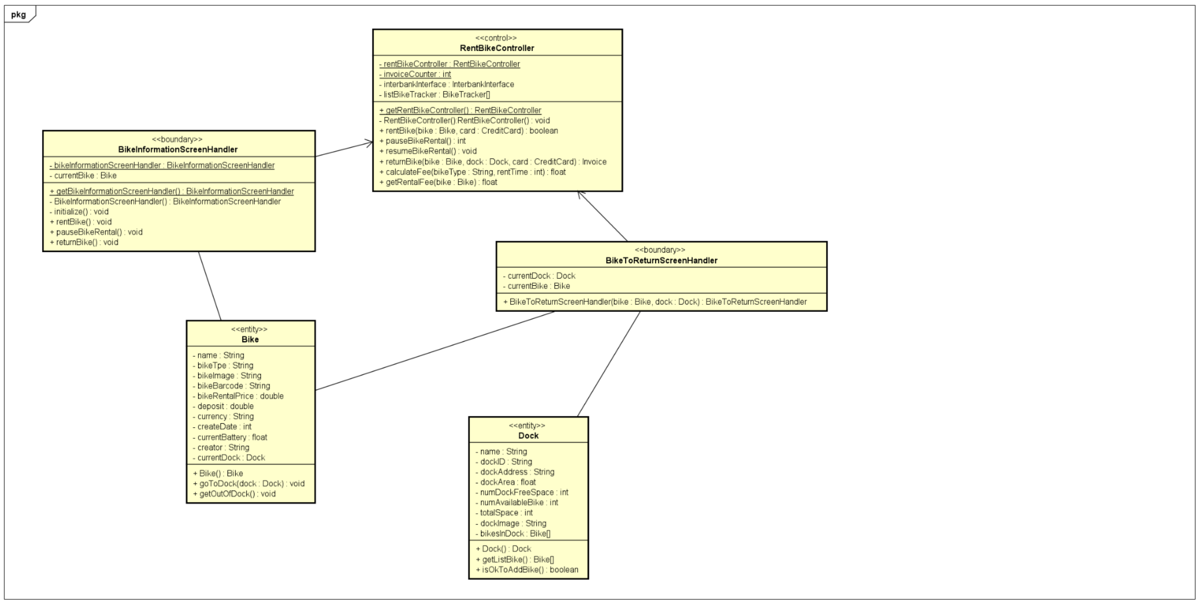

pkg
<<control>>
RentBikeController
- rentBikeController : RentBikeController
- invoiceCounter : int
- interbankInterface : InterbankInterface
- listBikeTracker : BikeTracker[]
+ getRentBikeController() : RentBikeController
- RentBikeController():RentBikeController() : void
+ rentBike(bike : Bike, card : CreditCard) : boolean
+ pauseBikeRental() : int
+ resumeBikeRental() : void
+ returnBike(bike : Bike, dock : Dock, card : CreditCard) : Invoice
+ calculateFee(bikeType : String, rentTime : int) : float
+ getRentalFee(bike : Bike) : float
<<boundary>>
BikeInformationScreenHandler
- bikeInformationScreenHandler : BikeInformationScreenHandler
- currentBike : Bike
+ getBikeInformationScreenHandler() : BikeInformationScreenHandler
- BikeInformationScreenHandler() : BikeInformationScreenHandler
- initialize() : void
+ rentBike() : void
+ pauseBikeRental() : void
+ returnBike() : void
<<boundary>>
BikeToReturnScreenHandler
- currentDock : Dock
- currentBike : Bike
+ BikeToReturnScreenHandler(bike : Bike, dock : Dock) : BikeToReturnScreenHandler
<<entity>>
Bike
- name : String
- bikeTpe : String
- bikeImage : String
- bikeBarcode : String
- bikeRentalPrice : double
- deposit : double
- currency : String
- createDate : int
- currentBattery : float
- creator : String
- currentDock : Dock
+ Bike() : Bike
+ goToDock(dock : Dock) : void
+ getOutOfDock() : void
<<entity>>
Dock
- name : String
- dockID : String
- dockAddress : String
- dockArea : float
- numDockFreeSpace : int
- numAvailableBike : int
- totalSpace : int
- dockImage : String
- bikesInDock : Bike[]
+ Dock() : Dock
+ getListBike() : Bike[]
+ isOkToAddBike() : boolean

Figure 3.16: Class Diagram for Return Bike Use Case

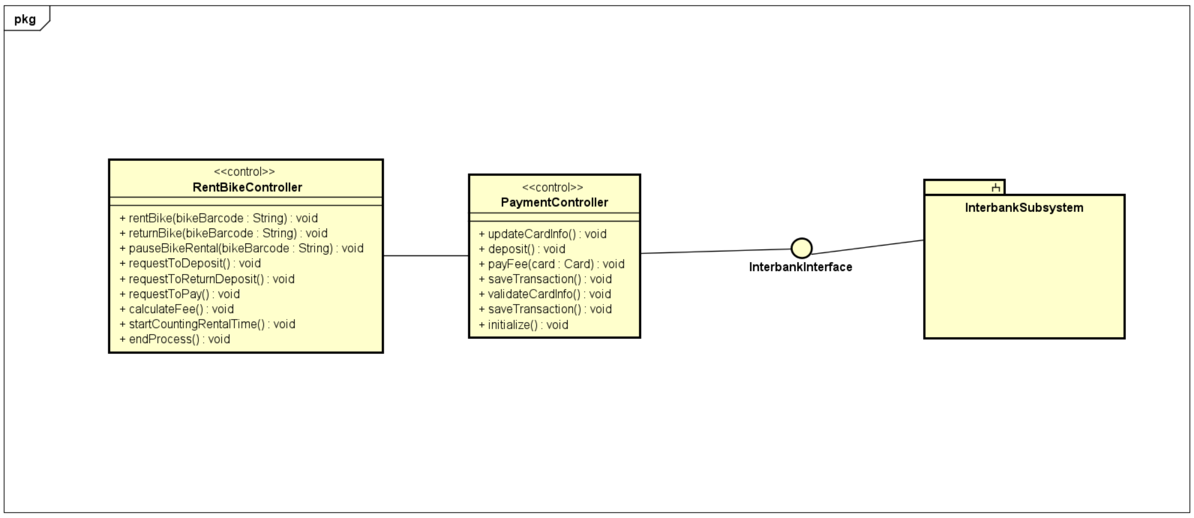

pkg
<<control>>
RentBikeController
+ rentBike(bikeBarcode : String) : void
+ returnBike(bikeBarcode : String) : void
+ pauseBikeRental(bikeBarcode : String) : void
+ requestToDeposit() : void
+ requestToReturnDeposit() : void
+ requestToPay() : void
+ calculateFee() : void
+ startCountingRentalTime() : void
+ endProcess() : void
<<control>>
PaymentController
+ updateCardInfo() : void
+ deposit() : void
+ payFee(card : Card) : void
+ saveTransaction() : void
+ validateCardInfo() : void
+ saveTransaction() : void
+ initialize() : void
InterbankInterface
InterbankSubsystem

Figure 3.17: Class Diagram for Return Deposit Use Case

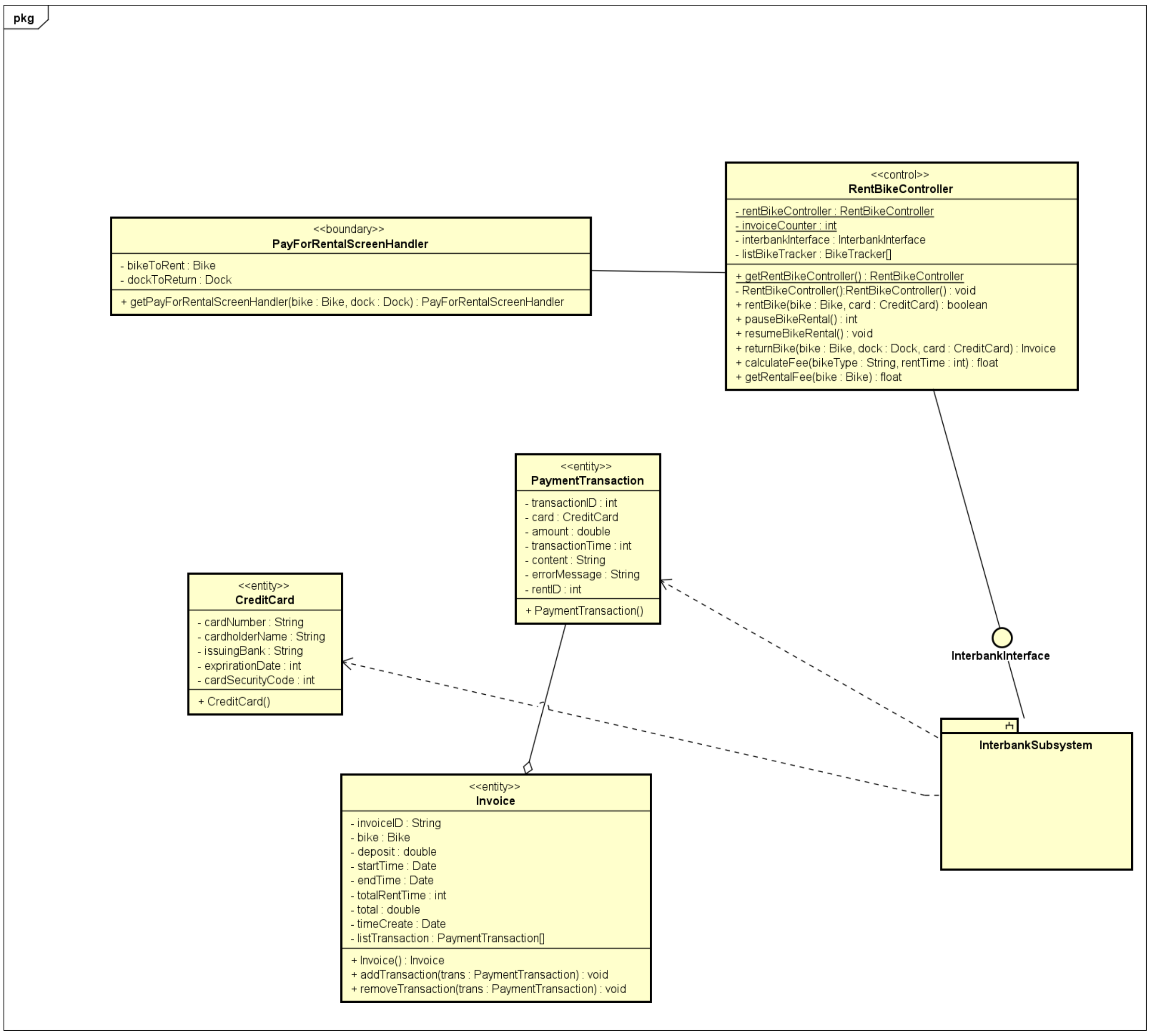

pkg
<<boundary>>
PayForRentalScreenHandler
- bikeToRent : Bike
- dockToReturn : Dock
+ getPayForRentalScreenHandler(bike : Bike, dock : Dock) : PayForRentalScreenHandler
<<control>>
RentBikeController
- rentBikeController : RentBikeController
- invoiceCounter : int
- interbankInterface : InterbankInterface
- listBikeTracker : BikeTracker[]
+ getRentBikeController() : RentBikeController
- RentBikeController().RentBikeController() : void
+ rentBike(bike : Bike, card : CreditCard) : boolean
+ pauseBikeRental() : int
+ resumeBikeRental() : void
+ returnBike(bike : Bike, dock : Dock, card : CreditCard) : Invoice
+ calculateFee(bikeType : String, rentTime : int) : float
+ getRentalFee(bike : Bike) : float
<<entity>>
PaymentTransaction
- transactionID : int
- card : CreditCard
- amount : double
- transactionTime : int
- content : String
- errorMessage : String
- rentID : int
+ PaymentTransaction()
<<entity>>
CreditCard
- cardNumber : String
- cardholderName : String
- issuingBank : String
- exprirationDate : int
- cardSecurityCode : int
+ CreditCard()
InterbankInterface
InterbankSubsystem
<<entity>>
Invoice
- invoiceID : String
- bike : Bike
- deposit : double
- startTime : Date
- endTime : Date
- totalRentTime : int
- total : double
- timeCreate : Date
- listTransaction : PaymentTransaction[]
+ Invoice() : Invoice
+ addTransaction(trans : PaymentTransaction) : void
+ removeTransaction(trans : PaymentTransaction) : void

Figure 3.18: Class Diagram for Pay Rental Use Case

3.4. Unified Analysis Class Diagram

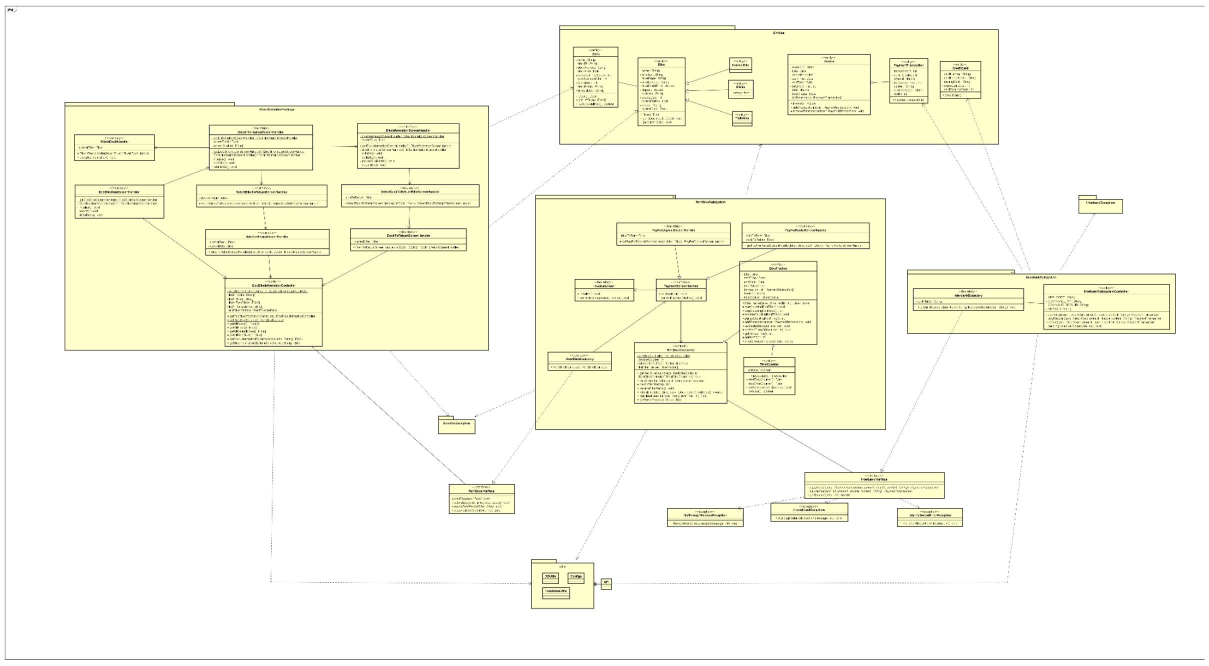


Figure 3.19: Unified Class Diagram for EcoBike Application

3.5. Security Software Architecture

In this project, we will not consider features such as user authentication (e.g., sign up, sign in, sign out), we only focus on features related to rent and return bikes.

4. Detailed Design

4.1. User Interface Design

4.1.1. Screen Configuration Standardization

Display
Screen resolution: 1366x768px
Number of colors supported: 16,177,216 colors
Screen
Size: 1200 x 600px
Main background color: #e6ebbc (R: 230, G: 235, B: 188)
Location of buttons: Bottom center of the frame
Logo: 100x100 px
Header logo: 100x100 px, located top left of the screen
Header/Screen title: Segoe UI, Bold, 24px, black
Numbers: comma for thousand separation, dot for decimal separation
Text: Segoe UI, size at most 24px
Frame border (if necessary): bounded rectangle, dashed line with width of 3px, color #afc139 (R: 175, G:193, B:57)

4.1.2. Screen Transition Diagrams

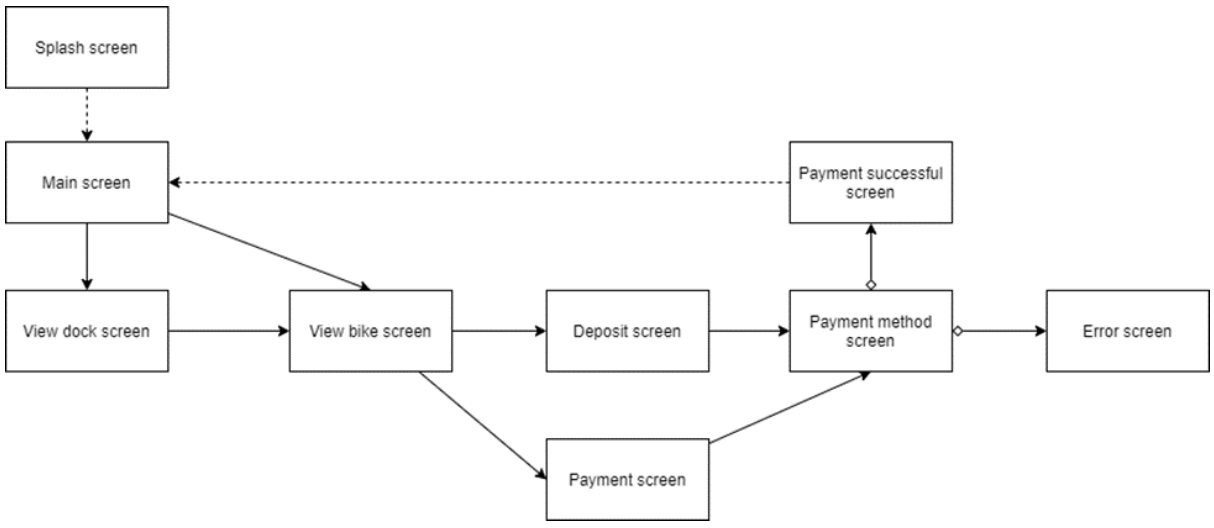


Figure 4.1: Screen Transition Diagram for EcoBike Application

4.1.3. Screen Specifications

4.1.3.1. Splash Screen

EcoBike Software	***Date of creation***	***Approved by***	***Reviewed by***	***Person in charge***

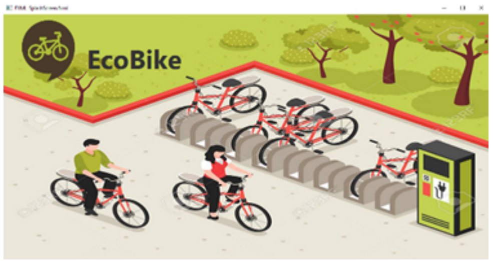
Screen specification	***Splash screen***	*28/10/2021*			*Chau*
		***Control***	***Operation***	***Function***	
		Main area	*None*	*Introduce the application*	

Table 4.1. Splash Screen Specification

4.1.3.2. Main Screen

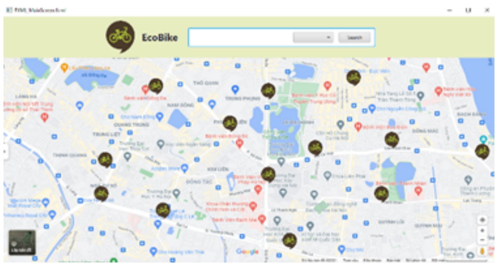
EcoBike Software		***Date of creation***	***Approved by***	***Reviewed by***	***Person in charge***
Screen specification	***Main screen***	*28/10/2021*			*Chau*
		***Control***	***Operation***	***Function***	
		Header logo	*Click*	*Return immediately to main screen*	
		Search bar	*Type, select & click*	*Type in information and select search type to search for docks or bikes*	
		Main area	*Initial*	*Display map at current location of users and nearby docks in term. The pins of docks can be clicked to see docks details*	

Table 4.2. Main Screen Specification

4.1.3.3. Dock Screen

EcoBike Software		***Date of creation***	***Approved by***	***Reviewed by***	***Person in charge***
Screen specification	***View Dock screen***	*28/10/2021*			*Chau*

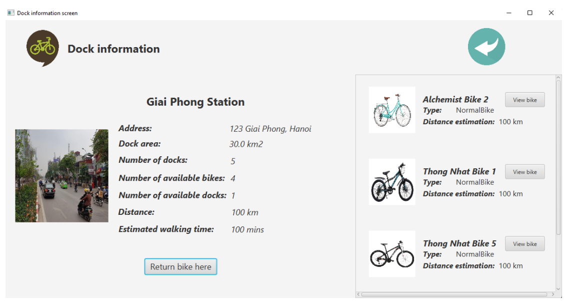
	Control	***Operation***	***Function***
 Dock information screen Dock information Giai Phong Station *Address:* *123 Giai Phong, Hanoi* *Dock area:* *30.0 km2* *Number of docks:* *5* *Number of available bikes:* *4* *Number of available docks:* *1* *Distance:* *100 km* *Estimated walking time:* *100 mins* Return bike here *Alchemist Bike 2* View bike *Type:* NormalBike *Distance estimation:* 100 km *Thong Nhat Bike 1* View bike *Type:* NormalBike *Distance estimation:* 100 km *Thong Nhat Bike 5* View bike *Type:* NormalBike *Distance estimation:* 100 km	*Logo*	*Click*	*Return to the main screen immediately*
	Header	*Initial*	*Display title of screen*
	Dock information	*Initial*	*Display dock information*
	Return bike	*Click*	*Allow user to start return bike process at the dock*
	Bike list	*Click*	*Display brief details about bikes available in the current dock. Allow choosing each bike to see detailed information*

Table 4.3. View Dock Screen Specification

4.1.3.4. Bike Screen

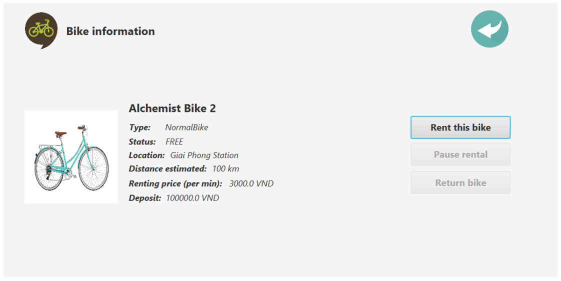
EcoBike Software		***Date of creation***	***Approved by***	***Reviewed by***	***Person in charge***
Screen specification	***View bike screen***	*28/10/2021*			*Chau*
 Bike information Alchemist Bike 2 *Type:* *NormalBike* *Status:* *FREE* *Location:* *Giai Phong Station* *Distance estimated:* *100 km* *Renting price (per min):* *3000.0 VND* *Deposit:* *100000.0 VND* Rent this bike Pause rental Return bike		***Control***	***Operation***	***Function***	
		Logo	*Click*	*Return to the main screen immediately*	
		Header	*Initial*	*Display title of screen*	
		Bike information	*Initial*	*Display bike information*	
		Option pane	*Click*	*Allow customer to perform renting, pause or return bike*	

Table 4.4. View Bike Screen Specification

4.1.3.5. Payment Method Screen

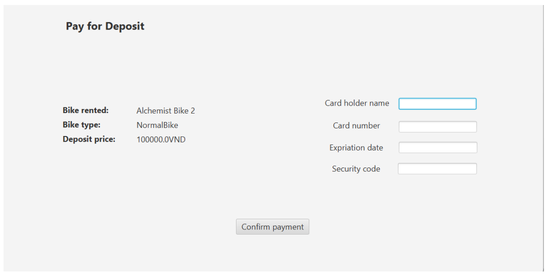
EcoBike Software		***Date of creation***	***Approved by***	***Reviewed by***	***Person in charge***
Screen specification	***Paying method screen***	*29/10/2021*			*Duong*
 **Pay for Deposit** **Bike rented:** Alchemist Bike 2 **Bike type:** NormalBike **Deposit price:** 100000.0VND Card holder name Card number Expriation date Security code Confirm payment		***Control***	***Operation***	***Function***	
		Logo	*Click*	*Return to the main screen immediately*	
		Header	*Initial*	*Display title of screen*	
		Payment	*Initial*	*Display information of paying method*	
		Button	*Click*	*Allow customer confirm to the paying method*	

Table 4.5. Payment Method Screen Specification

4.1.3.6. Deposit screen

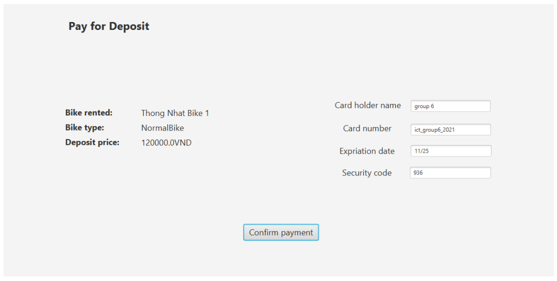
EcoBike Software		***Date of creation***	***Approved by***	***Reviewed by***	***Person in charge***
Screen specification	***Payment screen***	*29/10/2021*			*Long*
 **Pay for Deposit** **Bike rented:** Thong Nhat Bike 1 **Bike type:** NormalBike **Deposit price:** 120000.0VND Card holder name: group 6 Card number: ict_group6_2021 Expriation date: 11/25 Security code: 936 Confirm payment		***Control***	***Operation***	***Function***	
		Logo	*Click*	*Return to the main screen immediately*	
		Header	*Initial*	*Display title of screen*	
		Information of payment	*Initial*	*Display information of payment*	
		Button	*Click*	*Allow customer confirm to deposit the bike*	

Table 4.6. Deposit Screen Specification

4.1.3.7. Payment screen

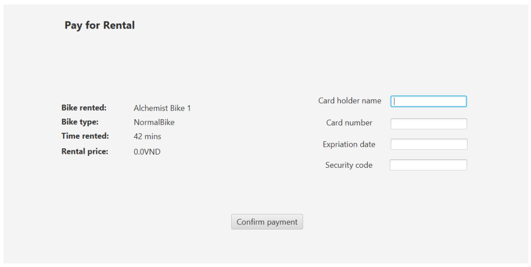
EcoBike Software		***Date of creation***	***Approved by***	***Reviewed by***	***Person in charge***
Screen specification	***Payment screen***	*29/10/2021*			*Duong*
 **Pay for Rental** **Bike rented:** Alchemist Bike 1 **Bike type:** NormalBike **Time rented:** 42 mins **Rental price:** 0.0VND Card holder name Card number Expriation date Security code Confirm payment		***Control***	***Operation***	***Function***	
		Logo	*Click*	*Return to the main screen immediately*	
		Header	*Initial*	*Display title of screen*	
		Information of payment	*Initial*	*Display information of payment*	
		Buttons	*Click*	*Allow customer confirm to pay or update card info*	

Table 4.7. Payment Screen Specification

4.2. Data Modeling

4.2.1. Conceptual Data Modeling

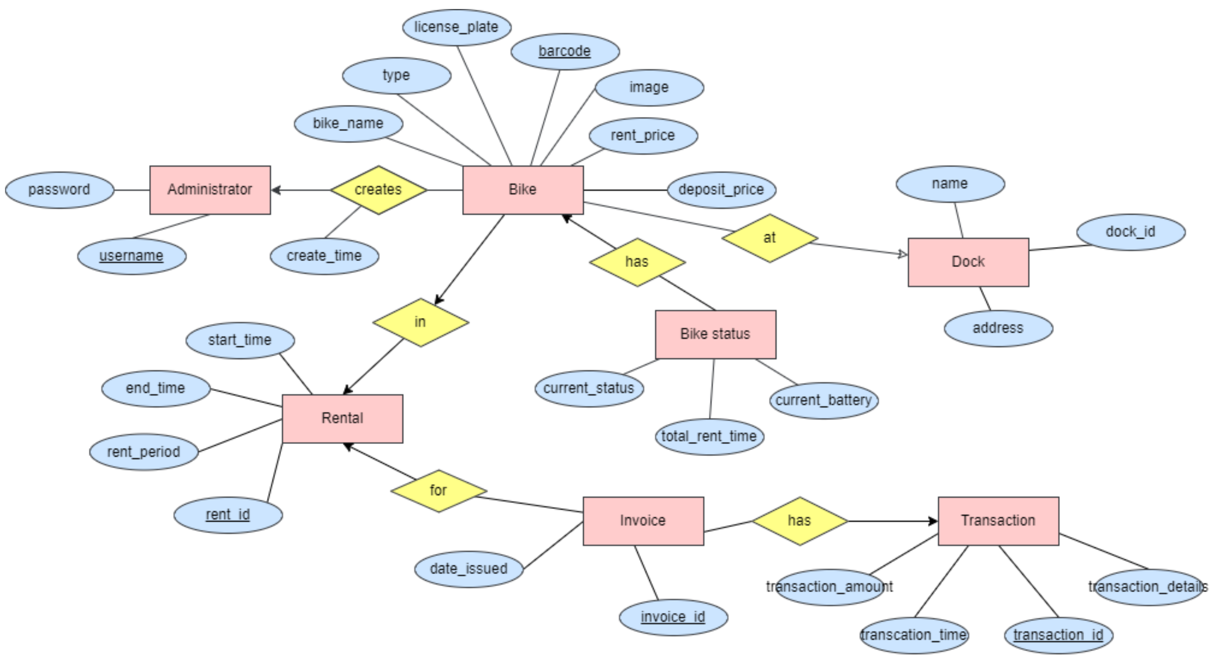


Figure 4.2. ER Diagram for EcoBike Application

4.2.2. Database Design

4.2.2.1. Database Management System

Database Management System: SQLite

4.2.2.2. Database Diagram

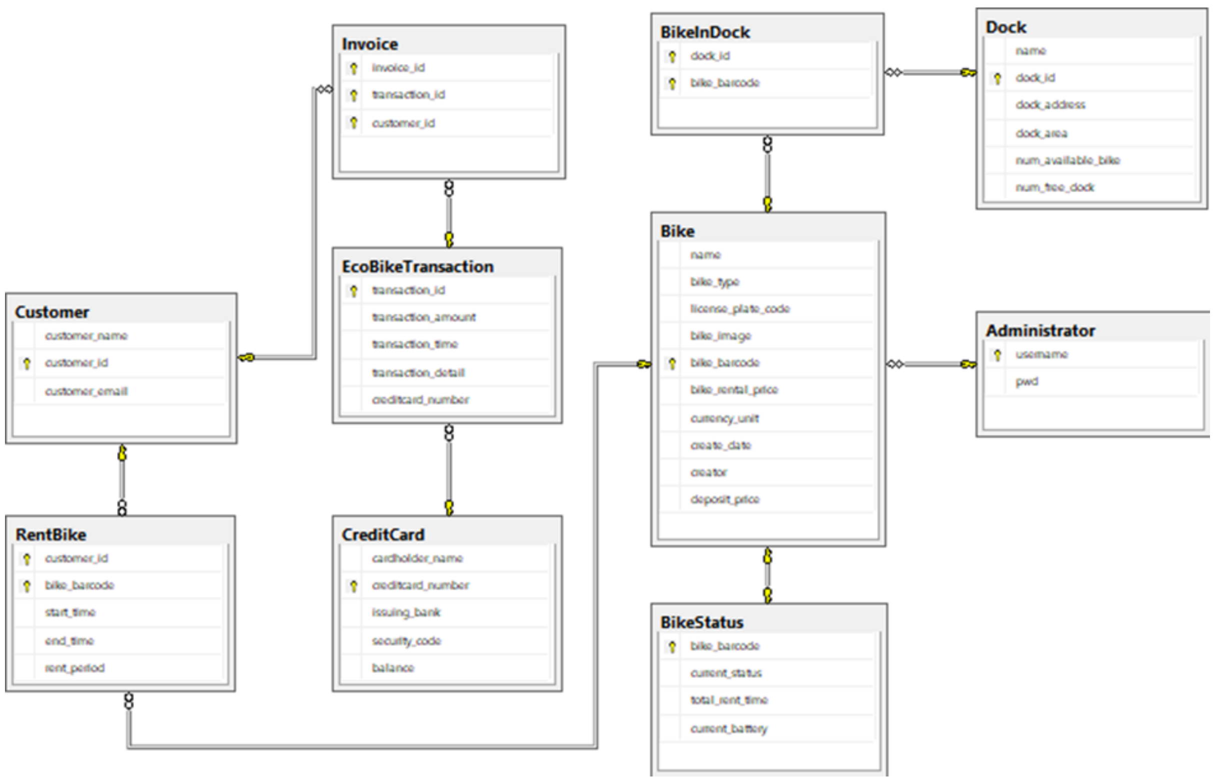


Figure 4.3. Database Diagram for EcoBike Application

4.2.2.3. Database Detail Design

No.	***PK***	***FK***	***Name***	***Data type***	***Mandatory***	***Description***
1	*x*	*x*	*customer_id*	*int*	*X*	*ID of customer*
2			*customer_name*	*varchar(256)*	*x*	*Name of customer renting bike*
3			*customer_email*	*varchar(256)*	*x*	*Email of customer renting bike for sending invoice*

Table 4.8. Customer table design

No.	***PK***	***FK***	***Name***	***Data type***	***Mandatory***	***Description***
1	*x*	*x*	*username*	*varchar(256)*	*x*	*Username of the administrator*
2			*pwd*	*varchar(256)*	*x*	*Password of the administrator used to login*

Table 4.9. Administrator table design

No.	***PK***	***FK***	***Name***	***Data type***	***Mandatory***	***Description***
1	*x*	*x*	*dock_id*	*int*	*x*	*ID of dock*
2			*dock_name*	*varchar(256)*	*x*	*Name of the dock*
3			*dock_address*	*varchar(256)*	*x*	*Address of the dock*
4			*dock_area*	*float*		*Area of the dock*
5			*num_available_bike*	*int*	*x*	*Number of current available bike in dock*
6			*num_free_dock*	*int*	*x*	*Number of current available bike slot in dock for returning bike*

Table 4.10. Dock table design

No.	*PK*	*FK*	*Name*	*Data type*	*Mandatory*	*Description*
1			*bike_name*	*varchar(256)*	*x*	*Name of the bike*
2			*bike_type*	*varchar(16)*	*x*	*Type of bike*
3			*license_plate_code*	*varchar(32)*	*x*	*Code of the license plate of the bike*
4			*bike_image*	*varchar(256)*		*Path to image of the bike*
5	*x*	*x*	*bike_barcode*	*int*	*x*	*Barcode of the bike*
6			*bike_rental_price*	*float*	*x*	*Price to rent the bike*
7			*deposit_price*	*float*	*x*	*Deposit cost to rent the bike*
8			*currency_unit*	*varchar(3)*	*x*	*Currency unit used to calculate rental fee and deposit fee*
9			*create_date*	*date*	*x*	*Day imported bike data*
10			*creator*	*varchar(256)*	*x*	*The administrator who create data for the bike*

Table 4.11. Bike table design

No.	*PK*	*FK*	*Name*	*Data type*	*Mandatory*	*Description*
1			*dock_id*	*int*	*x*	*Id of the dock*
2			*bike_barcode*	*int*	*x*	*Barcode of the bike in dock*

Table 4.12. Bike In Dock table design

No.	**PK**	**FK**	**Name**	**Data type**	**Mandatory**	**Description**
1			*bike_barcode*	*int*	*x*	*Barcode of the bike*
2			*current_status*	*varchar(4)*	*x*	*‘free’/’rent’*
3			*total_rent_time*	*int*	*x*	*Total time that the bike is rented (in minute)*
4			*current battery*	*float*	*x*	*Current battery status of the bike*

Table 4.13. Bike Status table design

No.	**PK**	**FK**	**Name**	**Data type**	**Mandatory**	**Description**
1	*x*		*invoice_id*	*int*	*X*	*ID of the invoice*
2		*x*	*transaction_id*	*int*	*x*	*ID of the transaction*
3		*x*	*customer_id*	*int*	*x*	*ID of the customer*

Table 4.14. Invoice table design

No.	**PK**	**FK**	**Name**	**Data type**	**Mandatory**	**Description**
1	*x*		*transaction_id*	*int*	*x*	*ID of transaction*
2			*transaction_amount*	*int*	*x*	*The amount of money for the transaction*
3			*transaction_time*	*DATETIME*	*x*	*Time the transaction is made*

4			*transaction_detail*	*varchar(256)*		*The content of the transaction*
5		*x*	*creditcard_number*	*int*	*x*	*The number of the credit card*

Table 4.15. Transaction table design

No.	***PK***	***FK***	***Name***	***Data type***	***Mandatory***	***Description***
1	*x*	*x*	*customer_id*	*int*	*X*	*ID of customer*
2	*x*		*bike_barcode*	*int*	*x*	*Barcode of the bike being rented*
3			*start_time*	*time*	*x*	*Time start renting*
4			*end_time*	*time*		*Time end renting (null if the bike is currently being rented)*
5			*rent_period*	*int*		*Total time renting the bike, in terms of minutes (null if the bike is currently being rented)*

Table 4.16. Rent Bike table design

No.	***PK***	***FK***	***Name***	***Data type***	***Mandatory***	***Description***
1	*x*	*x*	*card_number*	*int*	*X*	*Number of the credit card*
2			*cardholder_name*	*varchar(256)*	*x*	*Name of the cardholder*

3			*issuing_bank*	*varchar(256)*	*x*	*Bank in charge of the card*
4			*security_code*	*varchar(16)*	*x*	*Security code on the credit card for transaction*
5			*balance*	*float*	*x*	*Current balance of the credit card*

Table 4.17. Credit Card table design

Database script:

```
create table Administrator(username varchar(256) not null primary key,
                                        pwd varchar(256) not null);

create table Bike(name varchar(256) not null,
                             bike_type varchar(16) not null,
                             license_plate_code varchar(32) not null,
                             bike_image varchar(256),
                             bike_barcode int not null primary key
identity(1,1),
                             bike_rental_price float not null,
                             currency_unit varchar(3) not null,
                             create_date date,
                             creator varchar(256),
                             constraint FK_Bike_Creator foreign key
(creator) references Administrator(username));

create table Dock(name varchar(256),
                       dock_id int not null primary key identity(1,1),
                       dock_address varchar(256),
                       dock_area float,
                       num_available_bike int,
                       num_free_dock int);

create table BikeInDock(dock_id int not null,
                                   bike_barcode int not null,
                                   constraint PK_Bike_In_Dock primary
key (dock_id, bike_barcode),
                                   constraint FK_BikeInDock_Dock foreign
key (dock_id) references Dock(dock_id),
                                   constraint FK_BikeInDock_Bike foreign
key (bike_barcode) references Bike(bike_barcode));
```

```
create table Customer(customer_name varchar(256) not null,
                                  customer_id int not null
identity(1,1) primary key,
                                  customer_email varchar(128) not
null);

create table CreditCard(cardholder_name varchar(256) not null,
                                  creditcard_number varchar(25) not
null primary key,
                                  issuing_bank varchar(128) not null,
                                  security_code varchar(8) not null,
                                  balance float,
                                  constraint Check_CardBalance check
(balance >=0));

create table RentBike(customer_id int not null,
                            bike_barcode int not null,
                            start_time time not null,
                            end_time time,
                            rent_period int,
                            constraint PK_Rent_Bike primary key
(customer_id, bike_barcode),
                            constraint FK_RentBike_Bike foreign key
(bike_barcode) references Bike(bike_barcode),
                            constraint FK_RenBike_Customer foreign key
(customer_id) references Customer(customer_id),
                            constraint Check_RenBike_Time check
(end_time > start_time));


create table BikeStatus(bike_barcode int not null primary key,
                                  current_status varchar(4),
                                  total_rent_time int,
                                  current_battery float,
                                  constraint FK_BikeStatus_Barcode
foreign key (bike_barcode) references Bike(bike_barcode),
                                  constraint
Check_BikeStatus_Total_Rent_Time check (total_rent_time >=0),
                                  constraint Check_BikeStatus_Battery
check (current_battery >=0),
                                  constraint Check_BikeStatus_Status
check (current_status = 'free' or current_status = 'rent'));

create table EcoBikeTransaction(transaction_id varchar(32) not null
primary key,
                                  transaction_amount float not null,
                                  transaction_time datetime not null,
                                  transaction_detail varchar(256),
                                  creditcard_number varchar(25) not
null,
                                   constraint FK_Transaction_CreditCard
foreign key (creditcard_number) references
CreditCard(creditcard_number),
                                   constraint Check_Amount check
(transaction_amount > 0));

create table Invoice(invoice_id varchar(256) not null,
                             transaction_id varchar(32) not null,
                             customer_id int not null,
                             constraint PK_InvoiceTransactionCustomer
primary key (invoice_id, transaction_id, customer_id),
                             constraint FK_Invoice_Transaction foreign
key (transaction_id) references EcoBikeTransaction(transaction_id),
                             constraint FK_Invoice_Customer foreign key
(customer_id) references Customer(customer_id));
```

4.3. Non-Database Management System Files

4.4. Class Design

4.4.1. General Class Diagram

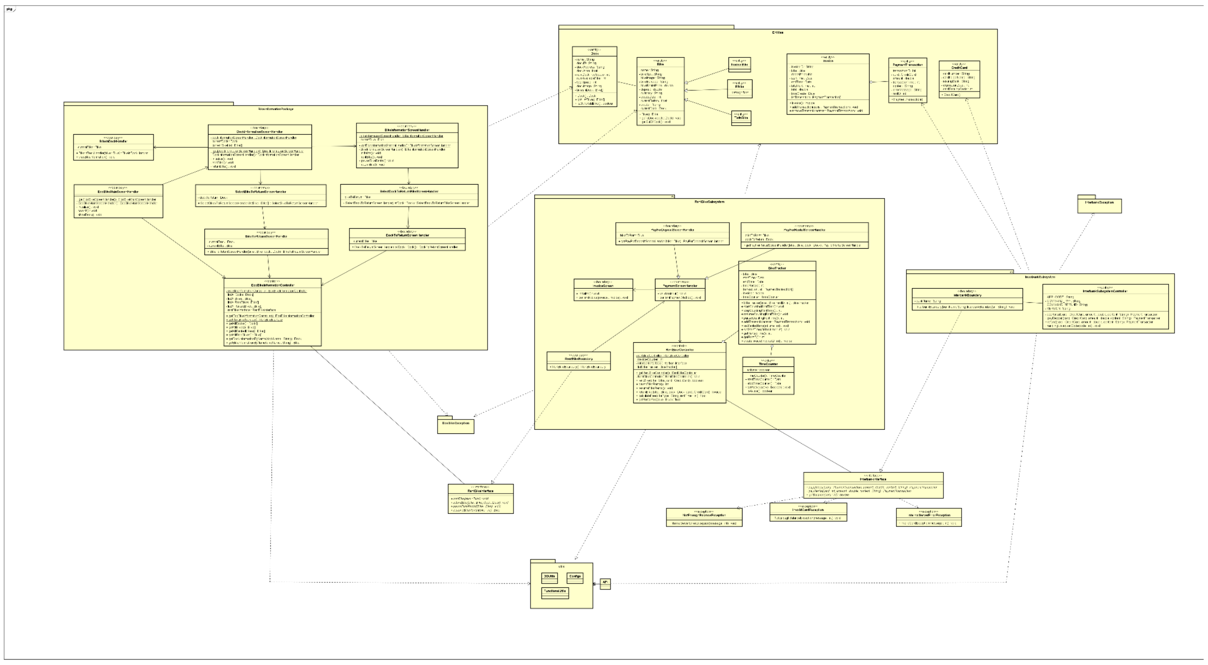


Figure 4.4. General Class Diagram for EcoBike Application

4.4.2. Class Diagrams

4.4.2.1. Class Diagram for Package BikeInformation

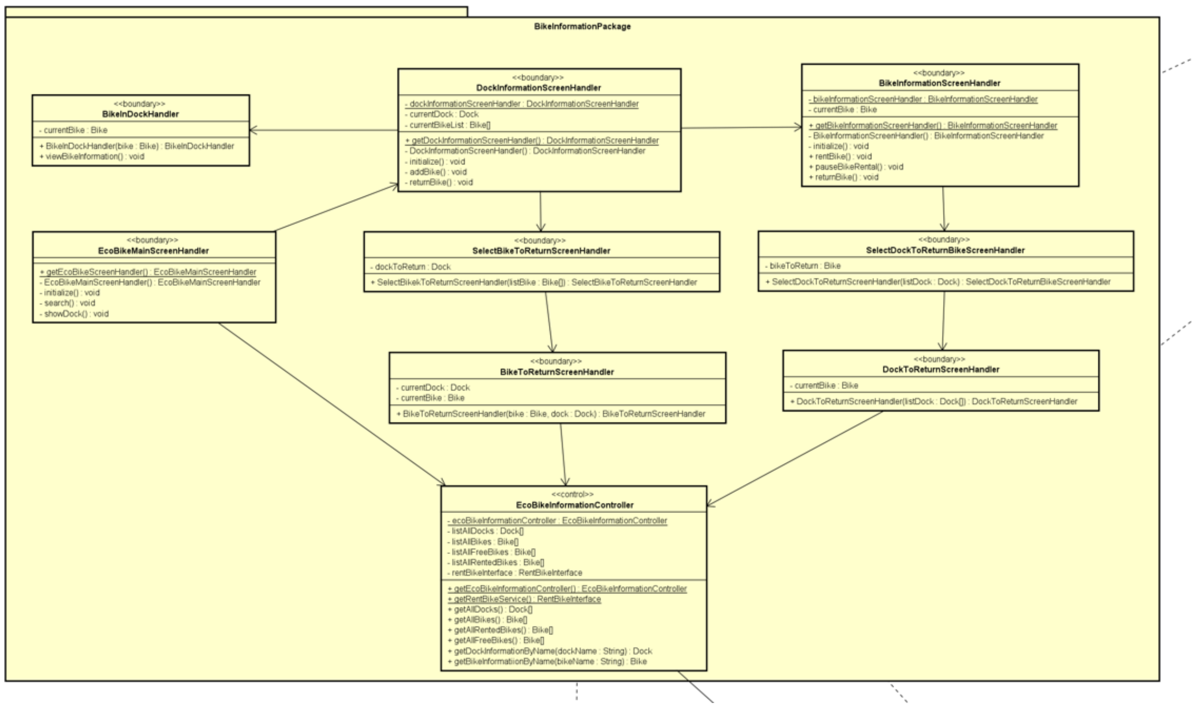

BikeInformationPackage
<<boundary>>
BikeInDockHandler
- currentBike : Bike
+ BikeInDockHandler(bike : Bike) : BikeInDockHandler
+ viewBikeInformation() : void
<<boundary>>
DockInformationScreenHandler
- dockInformationScreenHandler : DockInformationScreenHandler
- currentDock : Dock
- currentBikeList : Bike[]
+ getDockInformationScreenHandler() : DockInformationScreenHandler
- DockInformationScreenHandler() : DockInformationScreenHandler
- initialize() : void
- addBike() : void
- returnBike() : void
<<boundary>>
BikeInformationScreenHandler
- bikeInformationScreenHandler : BikeInformationScreenHandler
- currentBike : Bike
+ getBikeInformationScreenHandler() : BikeInformationScreenHandler
- BikeInformationScreenHandler() : BikeInformationScreenHandler
- initialize() : void
+ rentBike() : void
+ pauseBikeRental() : void
+ returnBike() : void
<<boundary>>
EcoBikeMainScreenHandler
+ getEcoBikeScreenHandler() : EcoBikeMainScreenHandler
- EcoBikeMainScreenHandler() : EcoBikeMainScreenHandler
- initialize() : void
- search() : void
- showDock() : void
<<boundary>>
SelectBikeToReturnScreenHandler
- dockToReturn : Dock
+ SelectBikekToReturnScreenHandler(listBike : Bike[]) : SelectBikeToReturnScreenHandler
<<boundary>>
SelectDockToReturnBikeScreenHandler
- bikeToReturn : Bike
+ SelectDockToReturnScreenHandler(listDock : Dock) : SelectDockToReturnBikeScreenHandler
<<boundary>>
BikeToReturnScreenHandler
- currentDock : Dock
- currentBike : Bike
+ BikeToReturnScreenHandler(bike : Bike, dock : Dock) : BikeToReturnScreenHandler
<<boundary>>
DockToReturnScreenHandler
- currentBike : Bike
+ DockToReturnScreenHandler(listDock : Dock[]) : DockToReturnScreenHandler
<<control>>
EcoBikeInformationController
- ecoBikeInformationController : EcoBikeInformationController
- listAllDocks : Dock[]
- listAllBikes : Bike[]
- listAllFreeBikes : Bike[]
- listAllRentedBikes : Bike[]
- rentBikeInterface : RentBikeInterface
+ getEcoBikeInformationController() : EcoBikeInformationController
+ getRentBikeService() : RentBikeInterface
+ getAllDocks() : Dock[]
+ getAllBikes() : Bike[]
+ getAllRentedBikes() : Bike[]
+ getAllFreeBikes() : Bike[]
+ getDockInformationByName(dockName : String) : Dock
+ getBikeInformatiionByName(bikeName : String) : Bike

Figure 4.5. Class Diagram for Package BikeInformation

4.4.2.2. Class Diagram for Subsystem RentBike

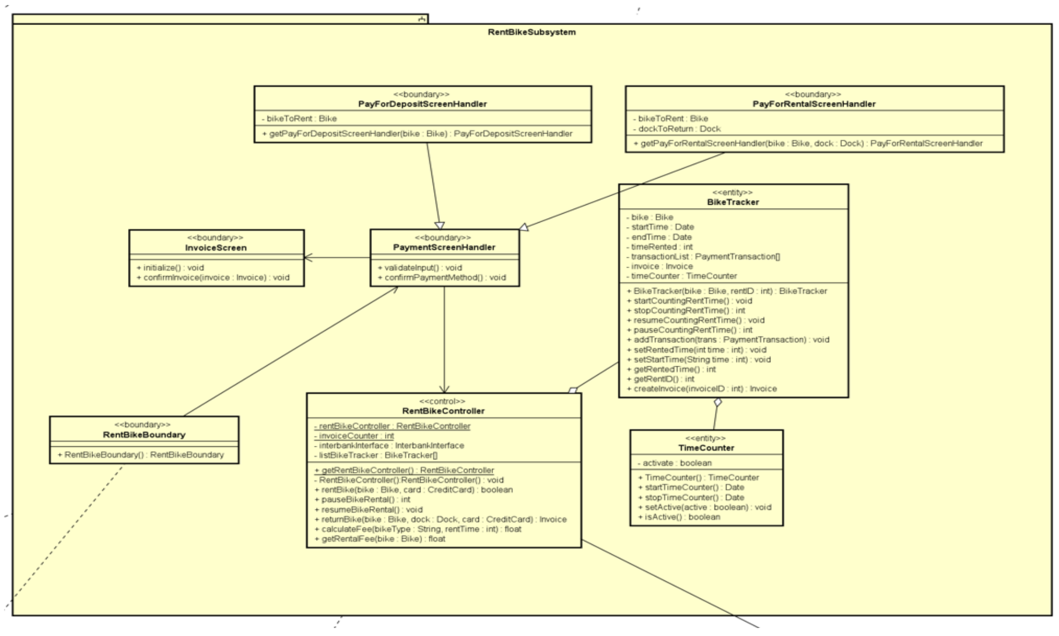

RentBikeSubsystem
<<boundary>>
PayForDepositScreenHandler
- bikeToRent : Bike
+ getPayForDepositScreenHandler(bike : Bike) : PayForDepositScreenHandler
<<boundary>>
PayForRentalScreenHandler
- bikeToRent : Bike
- dockToReturn : Dock
+ getPayForRentalScreenHandler(bike : Bike, dock : Dock) : PayForRentalScreenHandler
<<boundary>>
InvoiceScreen
+ initialize() : void
+ confirmInvoice(invoice : Invoice) : void
<<boundary>>
PaymentScreenHandler
+ validateInput() : void
+ confirmPaymentMethod() : void
<<entity>>
BikeTracker
- bike : Bike
- startTime : Date
- endTime : Date
- timeRented : int
- transactionList : PaymentTransaction[]
- invoice : Invoice
- timeCounter : TimeCounter
+ BikeTracker(bike : Bike, rentID : int) : BikeTracker
+ startCountingRentTime() : void
+ stopCountingRentTime() : int
+ resumeCountingRentTime() : void
+ pauseCountingRentTime() : int
+ addTransaction(trans : PaymentTransaction) : void
+ setRentedTime(int time : int) : void
+ setStartTime(String time : int) : void
+ getRentedTime() : int
+ getRentID() : int
+ createInvoice(invoiceID : int) : Invoice
<<boundary>>
RentBikeBoundary
+ RentBikeBoundary() : RentBikeBoundary
<<control>>
RentBikeController
- rentBikeController : RentBikeController
- invoiceCounter : int
- interbankInterface : InterbankInterface
- listBikeTracker : BikeTracker[]
+ getRentBikeController() : RentBikeController
- RentBikeController():RentBikeController() : void
+ rentBike(bike : Bike, card : CreditCard) : boolean
+ pauseBikeRental() : int
+ resumeBikeRental() : void
+ returnBike(bike : Bike, dock : Dock, card : CreditCard) : Invoice
+ calculateFee(bikeType : String, rentTime : int) : float
+ getRentalFee(bike : Bike) : float
<<entity>>
TimeCounter
- activate : boolean
+ TimeCounter() : TimeCounter
+ startTimeCounter() : Date
+ stopTimeCounter() : Date
+ setActive(active : boolean) : void
+ isActive() : boolean

Figure 4.6. Class Diagram for Subsystem RentBike

4.4.2.3. Class Diagram for Subsystem InterBank

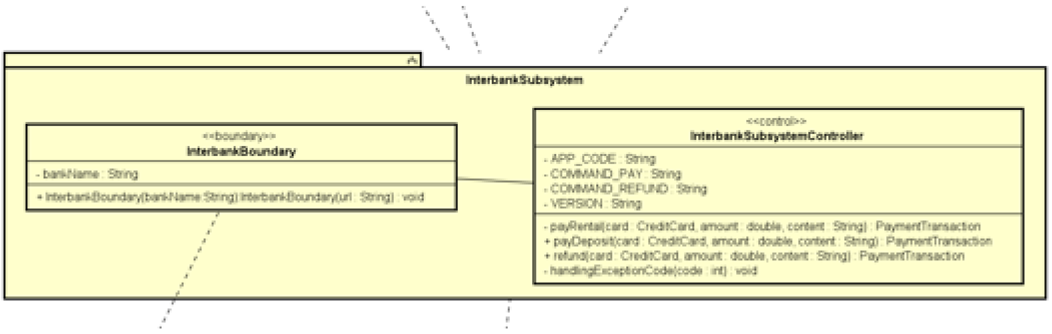

InterbankSubsystem
<<boundary>>
InterbankBoundary
- bankName : String
+ InterbankBoundary(bankName:String):InterbankBoundary(url : String) : void
<<control>>
InterbankSubsystemController
- APP_CODE : String
- COMMAND_PAY : String
- COMMAND_REFUND : String
- VERSION : String
- payRental(card : CreditCard, amount : double, content : String) : PaymentTransaction
+ payDeposit(card : CreditCard, amount : double, content : String) : PaymentTransaction
+ refund(card : CreditCard, amount : double, content : String) : PaymentTransaction
- handlingExceptionCode(code : int) : void

Figure 4.7. Class Diagram for Subsystem InterBank

4.4.3. Class Design

4.4.3.1. Class RentBikeController

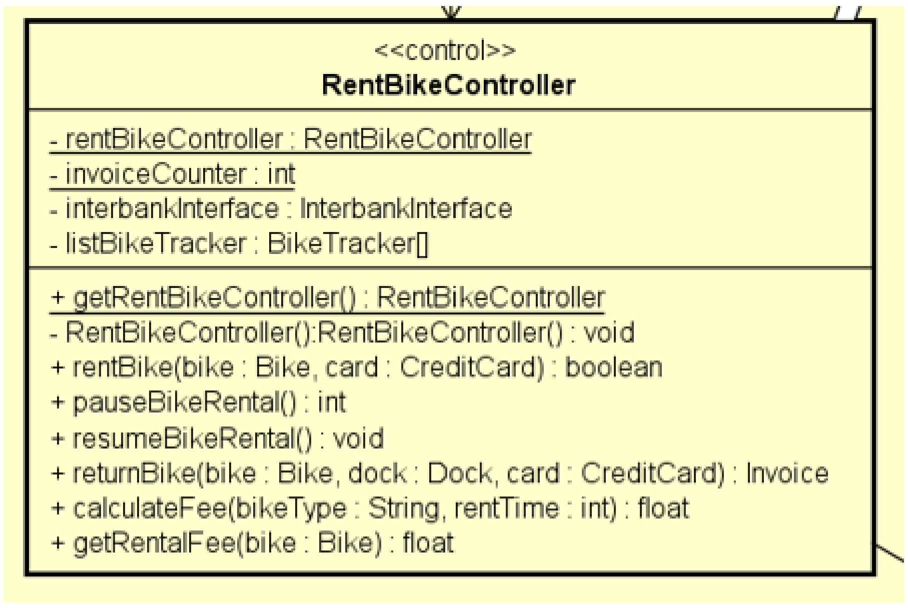

<<control>>
RentBikeController
- rentBikeController : RentBikeController
- invoiceCounter : int
- interbankInterface : InterbankInterface
- listBikeTracker : BikeTracker[]
+ getRentBikeController() : RentBikeController
- RentBikeController():RentBikeController() : void
+ rentBike(bike : Bike, card : CreditCard) : boolean
+ pauseBikeRental() : int
+ resumeBikeRental() : void
+ returnBike(bike : Bike, dock : Dock, card : CreditCard) : Invoice
+ calculateFee(bikeType : String, rentTime : int) : float
+ getRentalFee(bike : Bike) : float

Figure 4.8. RentBikeController Class Diagram

#	*Name*	*Data type*	*Default value*	*Description*
1	interbankInterface	InterbankInterface	InterbankInterface	Interbank to proceed transaction
2	rentBikeController	RentBikeController	null	static instance of the RentBikeController

Table 4.18. RentBikeController attributes

#	*Name*	*Return type*	*Description (purpose)*
1	getRentBikeServiceController	RentBikeController	return static instance rentBikeController of class RentBikeController
2	rentBike	void	start renting process
3	pauseBikeRental	void	pause counting rental time
4	startCountingRentBike	void	start counting renal time
5	resumeBikeRental	void	resume counting rental time
6	calculateFee	float	calculate the renting fee
7	returnBike	Invoice	start returning bike process

Table 4.19. RentBikeController operations

Parameter:

- bikeBarcode: bar code of the bike to rent
- card: the credit card to perform transaction (deposit)
- bikeToRent: the bike entities represent the bike to be rented

Exception:

- IOException
- RentBikeException If the bike is not currently available, the barcode is not valid
- EcoBikeUndefinedException If there is an unexpected error occurs during the renting process

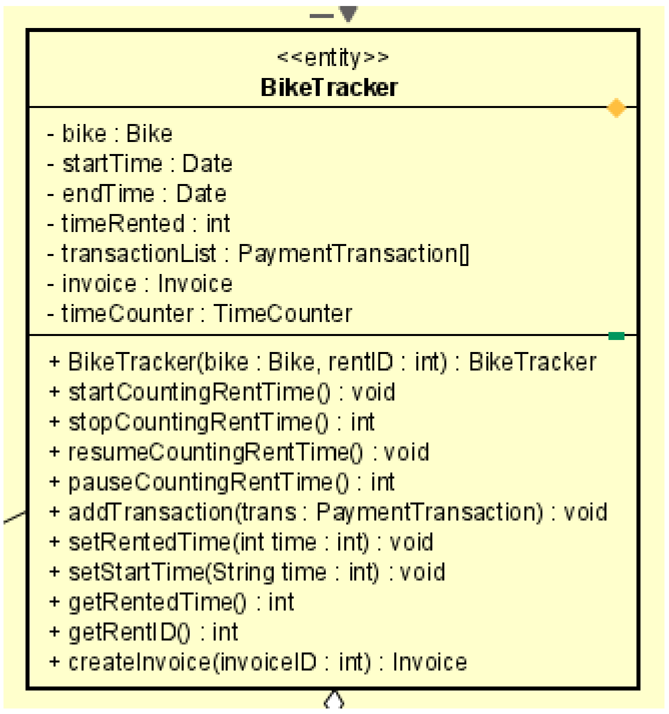
Method:

None

State

None

4.4.3.2. Class BikeTracker



<<entity>>
BikeTracker
- bike : Bike
- startTime : Date
- endTime : Date
- timeRented : int
- transactionList : PaymentTransaction[]
- invoice : Invoice
- timeCounter : TimeCounter
+ BikeTracker(bike : Bike, rentID : int) : BikeTracker
+ startCountingRentTime() : void
+ stopCountingRentTime() : int
+ resumeCountingRentTime() : void
+ pauseCountingRentTime() : int
+ addTransaction(trans : PaymentTransaction) : void
+ setRentedTime(int time : int) : void
+ setStartTime(String time : int) : void
+ getRentedTime() : int
+ getRentID() : int
+ createInvoice(invoiceID : int) : Invoice

Figure 4.9. BikeTracker Class Diagram

#	*Name*	*Data type*	*Default value*	*Description*
1	bike	Bike	null	the bike currently tracking
2	startTime	Date	null	start renting time
3	endTime	Date	null	end renting time
4	transactionList	PaymentTransaction[]	null	list of transaction on current bike
5	invoice	Invoice	null	invoice of
6	timeCounter	TimeCounter	TimeCounter	time counter for the renting process

Table 4.20. BikeTracker attributes

#	*Name*	*Return type*	*Description (purpose)*

1	startingCountingRentTime	void	start counting the renting time of current bike
2	stopCountingRentTime	void	stop counting the renting time of current bike
3	stopCountingRentTime	void	resume counting the renting time of current bike
4	stopCountingRentTime	void	pause counting the renting time of current bike
5	createInvoice	Invoice	create invoice

Table 4.21. BikeTracker operations

Parameter:

- bike: currently tracked bike
- rentID: ID of the rental
- paymentTransaction: transaction to add to the history
- invoiceID: ID of the invoice

Exception:

- RentBikeException If the bike is not currently available, the barcode is not valid
- EcoBikeUndefinedException If there is an unexpected error occurs during the renting process

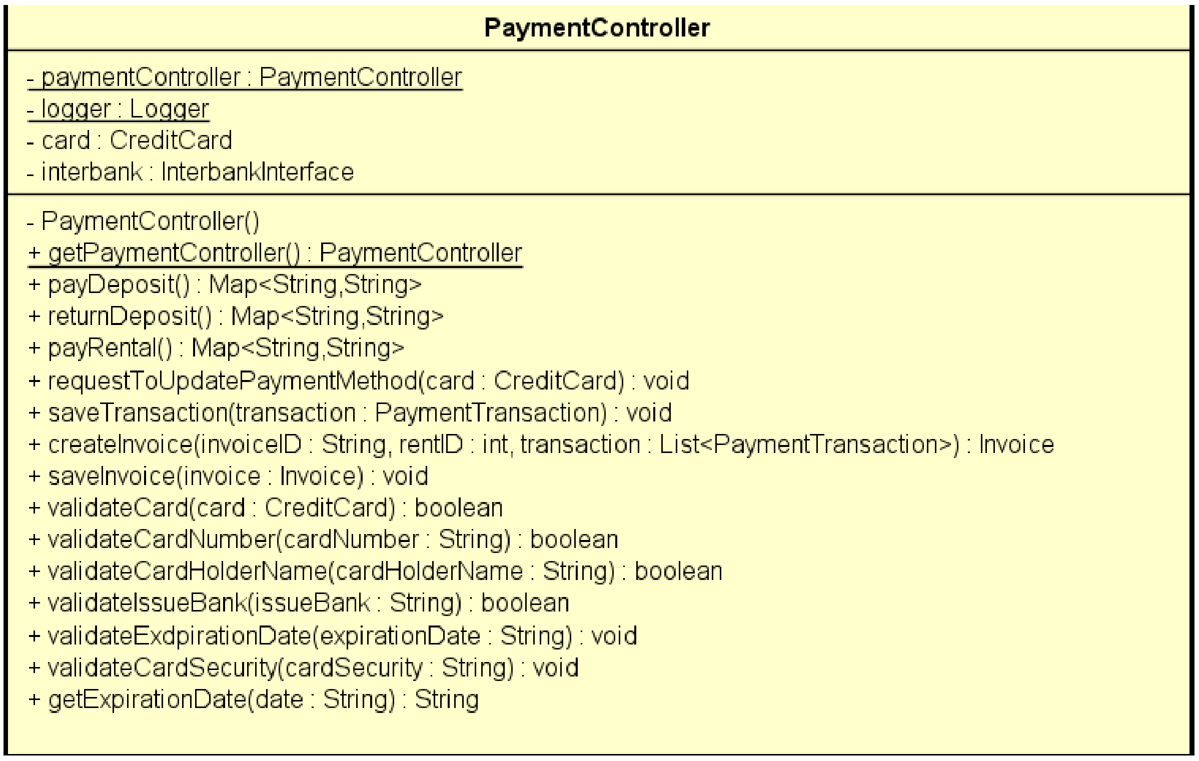
Method

None

State

None

4.4.3.3. Class PaymentController



```
PaymentController
- paymentController : PaymentController
- logger : Logger
- card : CreditCard
- interbank : InterbankInterface
- PaymentController()
+ getPaymentController() : PaymentController
+ payDeposit() : Map<String,String>
+ returnDeposit() : Map<String,String>
+ payRental() : Map<String,String>
+ requestToUpdatePaymentMethod(card : CreditCard) : void
+ saveTransaction(transaction : PaymentTransaction) : void
+ createInvoice(invoiceID : String, rentID : int, transaction : List<PaymentTransaction>) : Invoice
+ saveInvoice(invoice : Invoice) : void
+ validateCard(card : CreditCard) : boolean
+ validateCardNumber(cardNumber : String) : boolean
+ validateCardHolderName(cardHolderName : String) : boolean
+ validateIssueBank(issueBank : String) : boolean
+ validateExdpirationDate(expirationDate : String) : void
+ validateCardSecurity(cardSecurity : String) : void
+ getExpirationDate(date : String) : String
```

Figure 4.10. PaymentControllerClass Diagram

#	*Name*	*Data type*	*Default value*	*Description*
1	paymentController	PaymentController	null	static instance of PaymentController
2	logger	Logger	Logger	logger to log interaction
3	card	CreditCard	null	credit card to perform transaction
4	interbank	Interbank	Interbank	interbank to perform transaction

Table 4.22. PaymentController attributes

#	*Name*	*Return type*	*Description (purpose)*
1	payDeposit	Map<String, String>	start pay deposit process

2	returnDeposit	Map<String, String>	start return deposit process
3	payRental	Map<String, String>	start pay rental process
4	requestToUpdate PaymentMethod	void	start request to update payment method process
5	saveTransaction	void	save current transaction
6	createInvoice	Invoice	create invoice for current transaction
7	saveInvoice	void	save current invoice

Table 4.23. PaymentController operations

Parameter:

- transaction: the entity represent a transaction
- invoice: the entity represent a invoice
- card: the entity represent a card

Exception:

- EcoBikeUndefinedException If there is an unexpected error occurs during the renting process
- SQLException if there is an unexpected error with the database

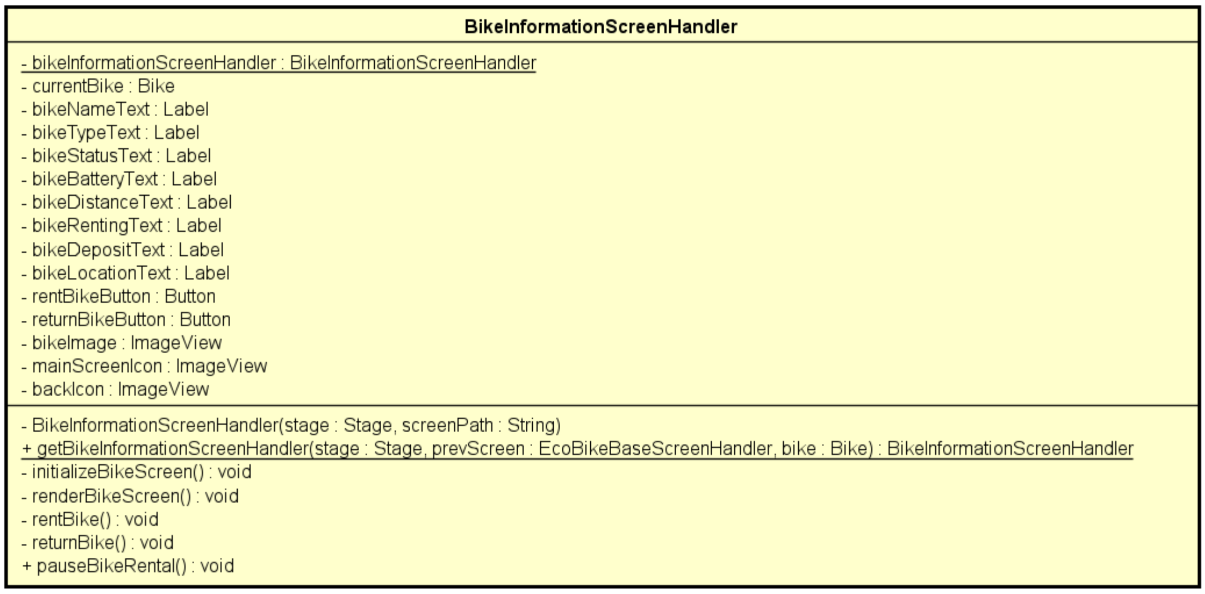
Method

None

State

None

4.4.3.4. Class BikeInformationScreenHandler



```
BikeInformationScreenHandler
- bikeInformationScreenHandler : BikeInformationScreenHandler
- currentBike : Bike
- bikeNameText : Label
- bikeTypeText : Label
- bikeStatusText : Label
- bikeBatteryText : Label
- bikeDistanceText : Label
- bikeRentingText : Label
- bikeDepositText : Label
- bikeLocationText : Label
- rentBikeButton : Button
- returnBikeButton : Button
- bikeImage : ImageView
- mainScreenIcon : ImageView
- backIcon : ImageView
- BikeInformationScreenHandler(stage : Stage, screenPath : String)
+ getBikeInformationScreenHandler(stage : Stage, prevScreen : EcoBikeBaseScreenHandler, bike : Bike) : BikeInformationScreenHandler
- initializeBikeScreen() : void
- renderBikeScreen() : void
- rentBike() : void
- returnBike() : void
+ pauseBikeRental() : void
```

Figure 4.11. BikeInformationScreenHandler Class Diagram

#	*Name*	*Data type*	*Default value*	*Description*
1	currentBike	Bike	null	the bike the screen is showing
2	bikeNameText	Label	null	label of the name of the bike
3	bikeTypeText	Label	null	label of the type of the bike
4	bikeStatusText	Label	null	label of the status of the bike
5	bikeBatteryText	Label	null	label of the battery of the bike
6	bikeDistanceText	Label	null	label of the estimation distance that the bike can travel
7	bikeRentingText	Label	null	label of the renting price per hour of the bike
8	bikeDepositText	Label	null	label of the amount of money needed to deposit to rent the bike

9	bikeLocation Text	Label	null	label of the location of the bike
10	returnBikeBu tton	Button	null	button to return this bike
11	rentBikeButt on	Button	null	button to rent this bike

Table 4.24. BikeInformationScreenHandler *operations*

#	*Name*	*Return type*	*Description (purpose)*
1	renderBikeScreen	void	render the screen with information of the bike
2	rentBikeS	void	start rent bike process
3	returnBike	void	start return bike process

Table 4.25. BikeInformationScreenHandler *operations*

Parameter:

None

Exception:

- IOException if there is unexpected error with the IO

Method

None

State

None

5. Design Considerations

5.1. Goals and Guidelines

Goals:

- Provide a user-friendly application
- Provide an eye-catching interface and convenient experience for users
- The response time for the system is 1 second at normal and 2 seconds during a peak load

Guidelines:

- Obligate the coding convention in Java, and OOP principles.
- Avoid hard-coding
- Write comments for codes
- Structure the doc for maintenance

5.2. Architectural Strategies

Our intention is to reuse components

- Programming Language: Java
- Database: MySQL
- UML: Astah
- GUI: Scene Builder

We're always looking toward minimizing the memory and space usage; reduce the complexity to speed up the response time, and improve the performance. We're also concerned about the maintenance. For the future, we're looking forward to updating the system, integrating new features such as admin to manage the crud, the statistics, the profit.

5.3. Coupling and Cohesion

5.3.1. Coupling

5.3.1.1. Content coupling

Related modules	Description	Improvement
No related module	Our modules are self-contained and don't rely on other modules to operate	No improvement

5.3.1.2. Common coupling

Related modules	Description	Improvement
No related module	We only use static with Singleton pattern to share the controller instance between boundaries to control the flow of the programs. Some constants exist in the system, but only with careful usage shared between the related modules	No improvement

5.3.1.3. Control coupling

Related modules	Description	Improvement
No related module	Our methods are designed to carry out only one specific task, so no control coupling existed	No improvement

5.3.1.4. Stamp coupling

Related modules	Description	Improvement
RentBikeServiceController	In module RentBikeServiceController, the Bike entities was used as an argument for the calculateFee method, which only need bikeType and totalRentTime as arguments	Fix the method to accept only needed arguments instead of the accepting Bike entities as the argument

5.3.1.5. Data coupling

Related modules	Description	Improvement
Controllers and Boundaries modules	Boundaries need data to render GUI, which is acceptable	No improvement

5.3.2. Cohesion

5.3.2.1. Coincidental cohesion

Related modules	Description	Improvement
No module	The only visible coincidental cohesion in our project might be the class Configs, which contains some constant share between some controllers and entities	No improvement

5.3.2.2. Logical cohesion

Related modules	Description	Improvement
No module		No improvement

5.3.2.3. Temporal cohesion

Related modules	Description	Improvement
Controller View,	In our project, we put all controllers into a Controller package, screen handlers into a View package, which might be considered temporal cohesion	No improvement

5.3.2.4. Procedure cohesion

Related modules	Description	Improvement
RentBikeServiceCon-troller	Consist of validating methods	No improvement

5.3.2.5. Communicational cohesion

Related modules	Description	Improvement
No module		No improvement

5.3.2.6. Sequential cohesion

Related modules	Description	Improvement
No module		No improvement

5.3.2.7. Information cohesion

Related modules	Description	Improvement
DBUtils JSONUtils	All methods are to perform database queries or manipulate json string	No improvement

5.3.2.8. Functional cohesion

Related modules	Description	Improvement
Most of the modules		No improvement

In our software design, we detect that there are still some components that have Control Coupling and Communicational Cohesion problems.

We are trying our best to resolve these problems, decrease Coupling level and increase Cohesion level. However, due to lack of time, we might not be able to fix this before the announced deadline.

5.4. Design Principles

We design simple classes follow SOLID principles that means:

- A class should have only one job, one responsibility.
- Software entities are open for extension but close for modification.
- We also use interfaces, abstract classes. So, subclasses should be substitutable for their base classes.
- Use specific interfaces if necessary instead of using general purpose interfaces which do not use.
- We put all classes with the same properties into one package to manage easily. Therefore, we can reuse source code, adapt to any changing requirements.

5.5. Design Patterns

Facade pattern:

- We use InterbankInterface for communication between software and interbank subsystem. It decrease the overall complexity of our application and provides an easier interface for communication

Singleton pattern:

- We use singleton pattern for screen handler so that we do not need to create new instance of screen handler each time we change the screen

Factory pattern:

- The factory pattern is used to create different type of bike by stamp

Observer pattern:

- The observer pattern is used to observe the change in the dock so each time a bike is rented or returned, the GUI can be updated accordingly without having to query the database. We only need to store the status back to the database.