HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology

Software Design Document

EcoBike Application

Subject: ITSS Software Management

Group 01

No.	Student Name	Student ID
1	Nguyen Quoc Anh	20194726
2	Pham Thanh Bien	20194731
3	Dinh Huu Dai	20194735

Hanoi, 02/2023

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Contribution

No	Team member	Student ID	Assignment	Contribution
1	Dinh Huu Dai	20194735	Team leader, refining documentation, backend implementation	35%
2	Pham Thanh Bien	20194731	Preparing documentation, implementing backend	35%
3	Nguyen Quoc Anh	20194726	Preparing documentation, implementing frontend	30%

1.2. 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 stimulation.

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.

1.3. Scope

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

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

The objective of the EcoBike Application is to serve a maximum of 10.000 users concurrently, with a friendly user interface with the aim of helping the user to find the most suitable service to rent or return the bike.

1.4. Glossary

Term	Definition
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 and returning 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 bikes are placed
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 returned 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

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:
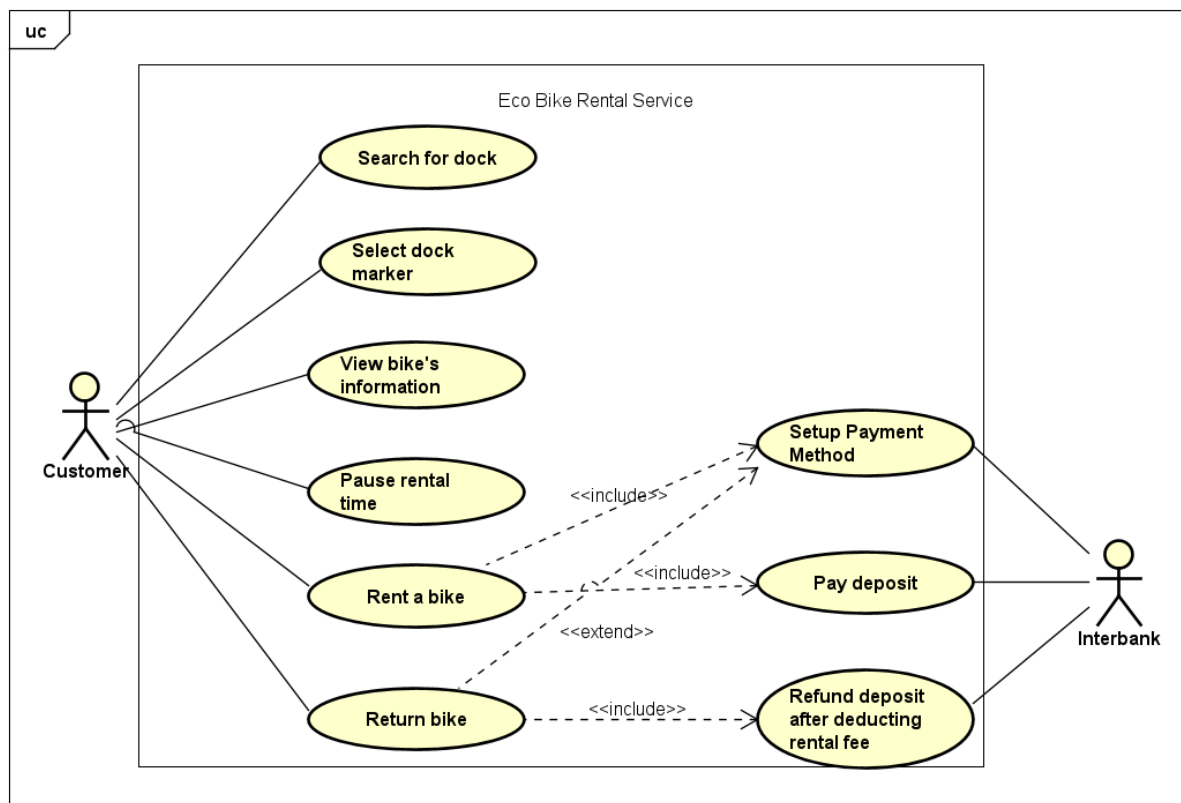
Figure 1.1: General use case diagram

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's 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*

3. System Architecture and Architecture Design

3.1. Interaction Diagrams

3.1.1. Communication Diagrams
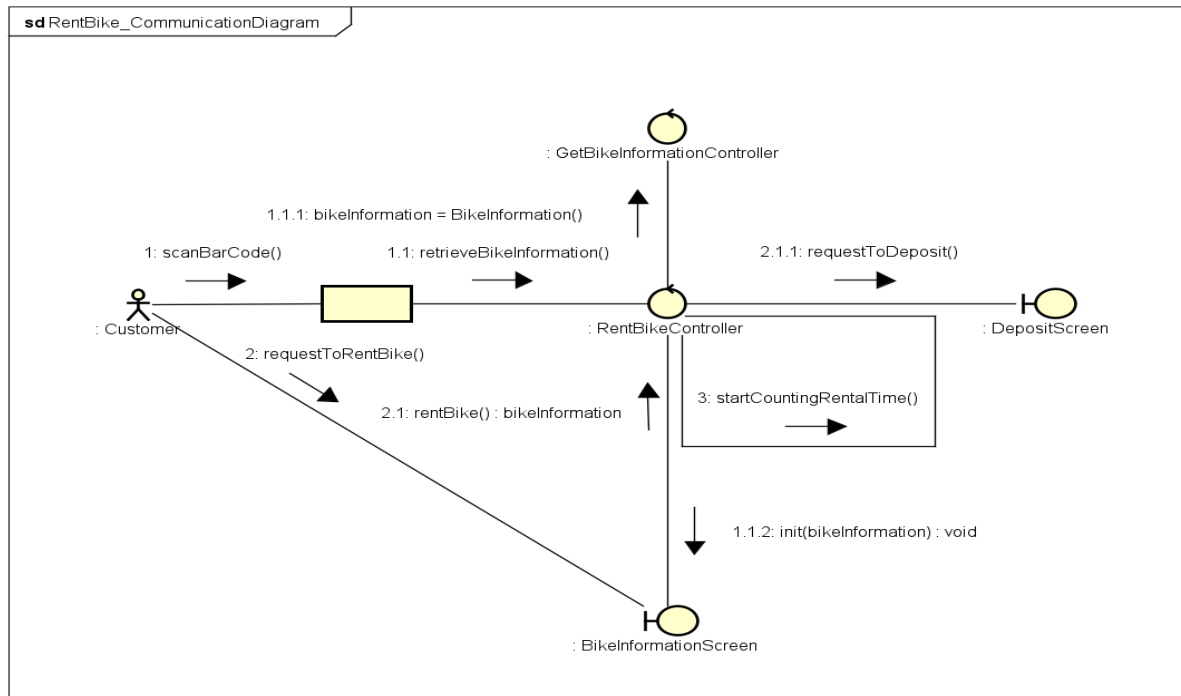
Figure 3.1: Communication Diagram for Rent Bike Use Case
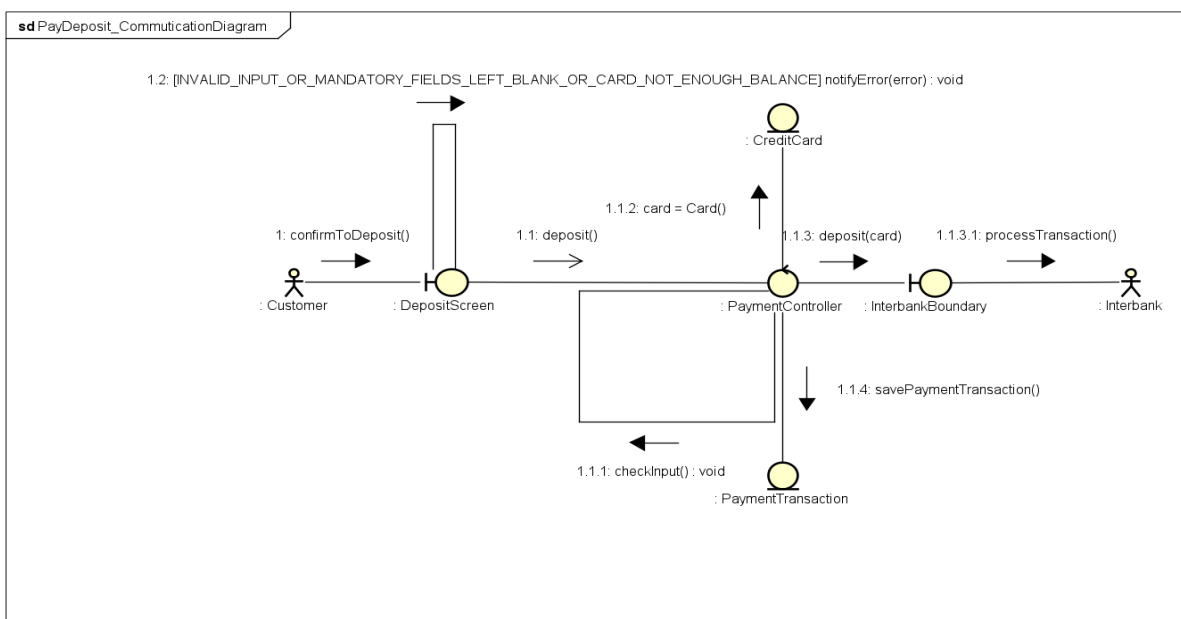
Figure 3.2: Communication Diagram for Deposit Use Case
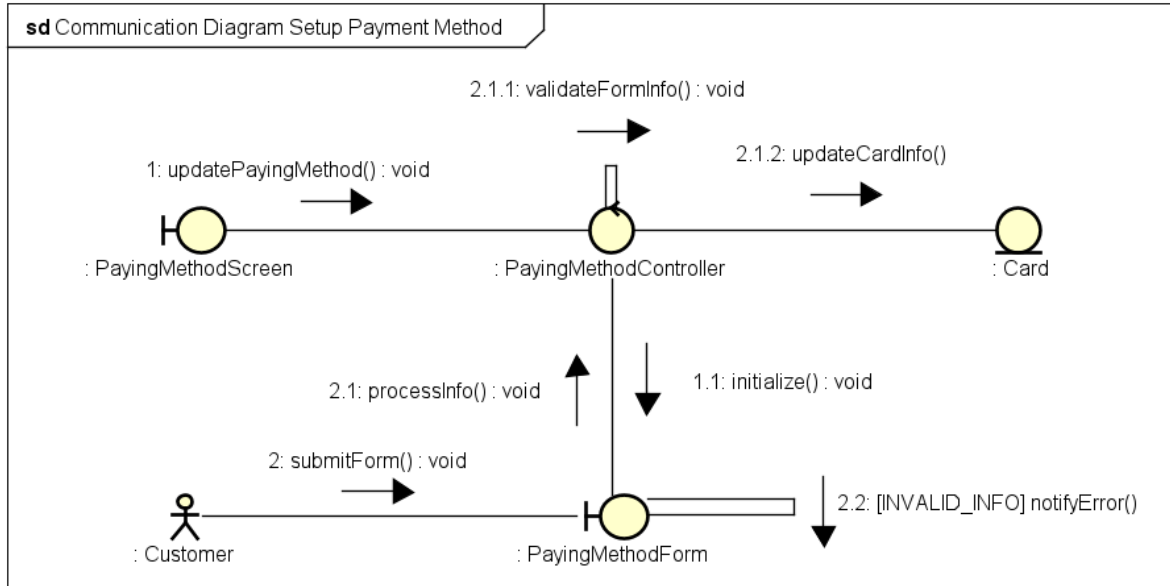
Figure 3.3: Communication Diagram for Setup Payment Method Use Case
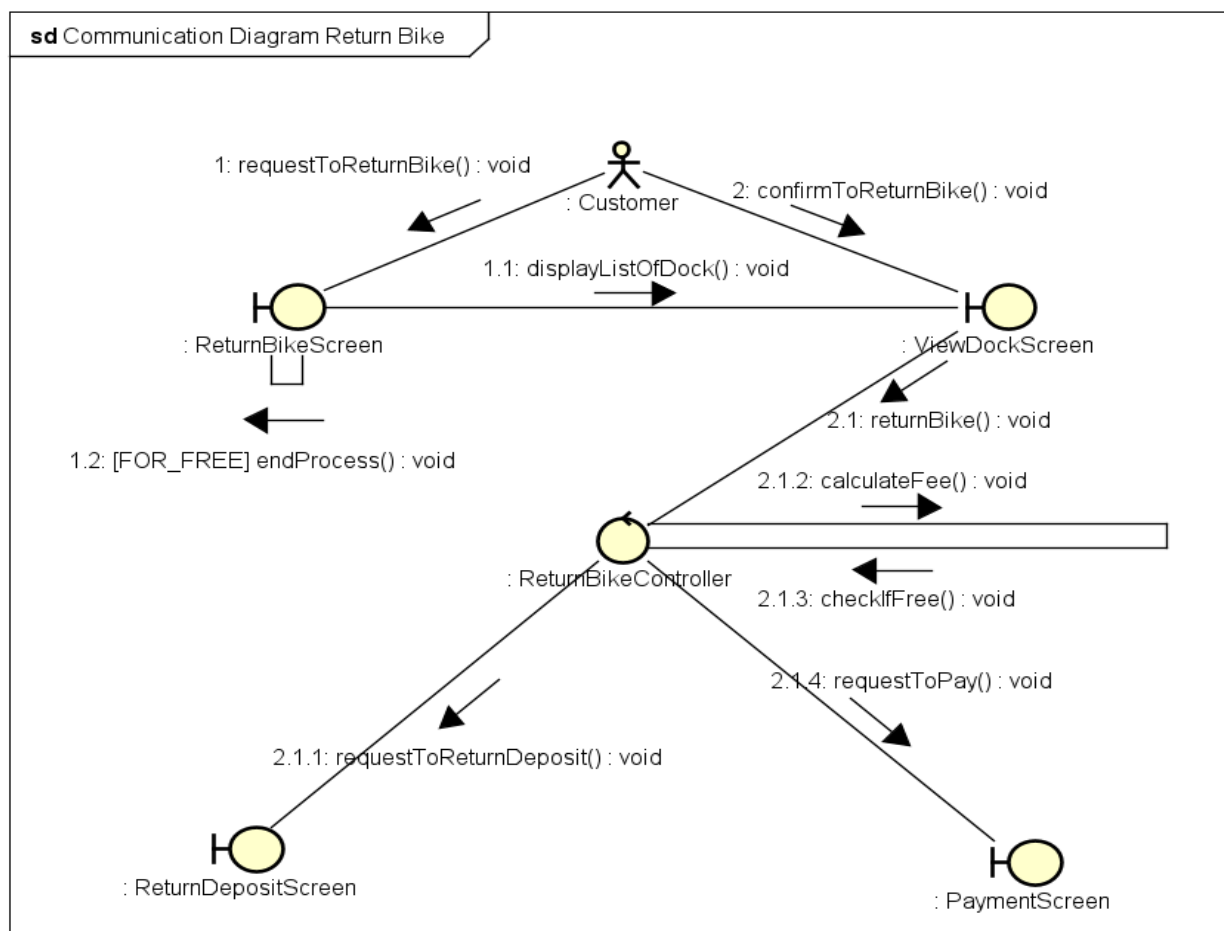
Figure 3.4: Communication Diagram for Return Bike Use Case
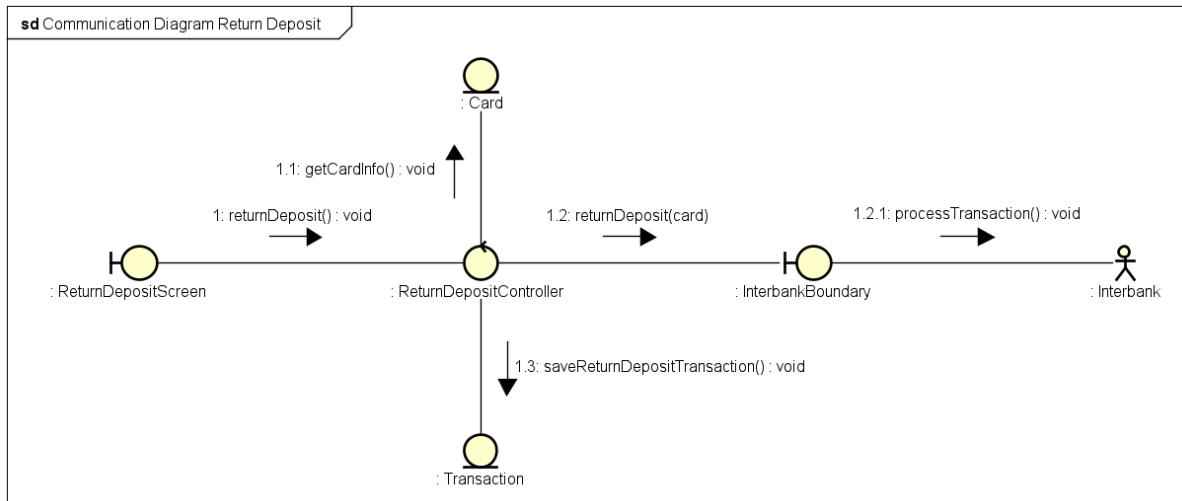
Figure 3.5: Communication Diagram for Return Deposit Use Case
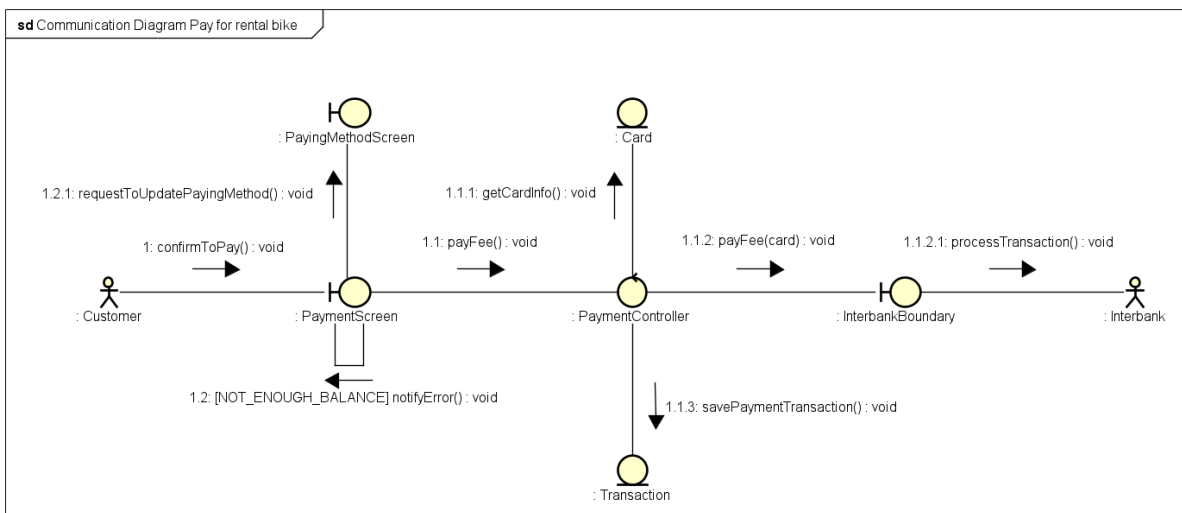
Figure 3.6: Communication Diagram for Pay For Rental Use Case

3.1.2. Sequence Diagrams
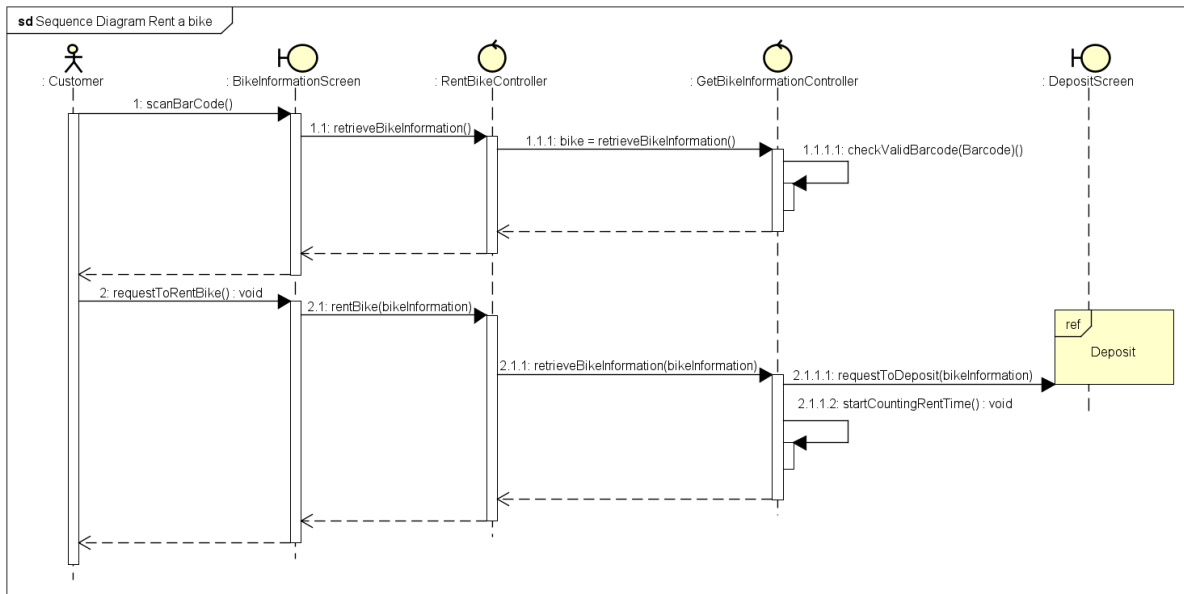
Figure 3.7: Sequence Diagram for Rent A Bike Use Case
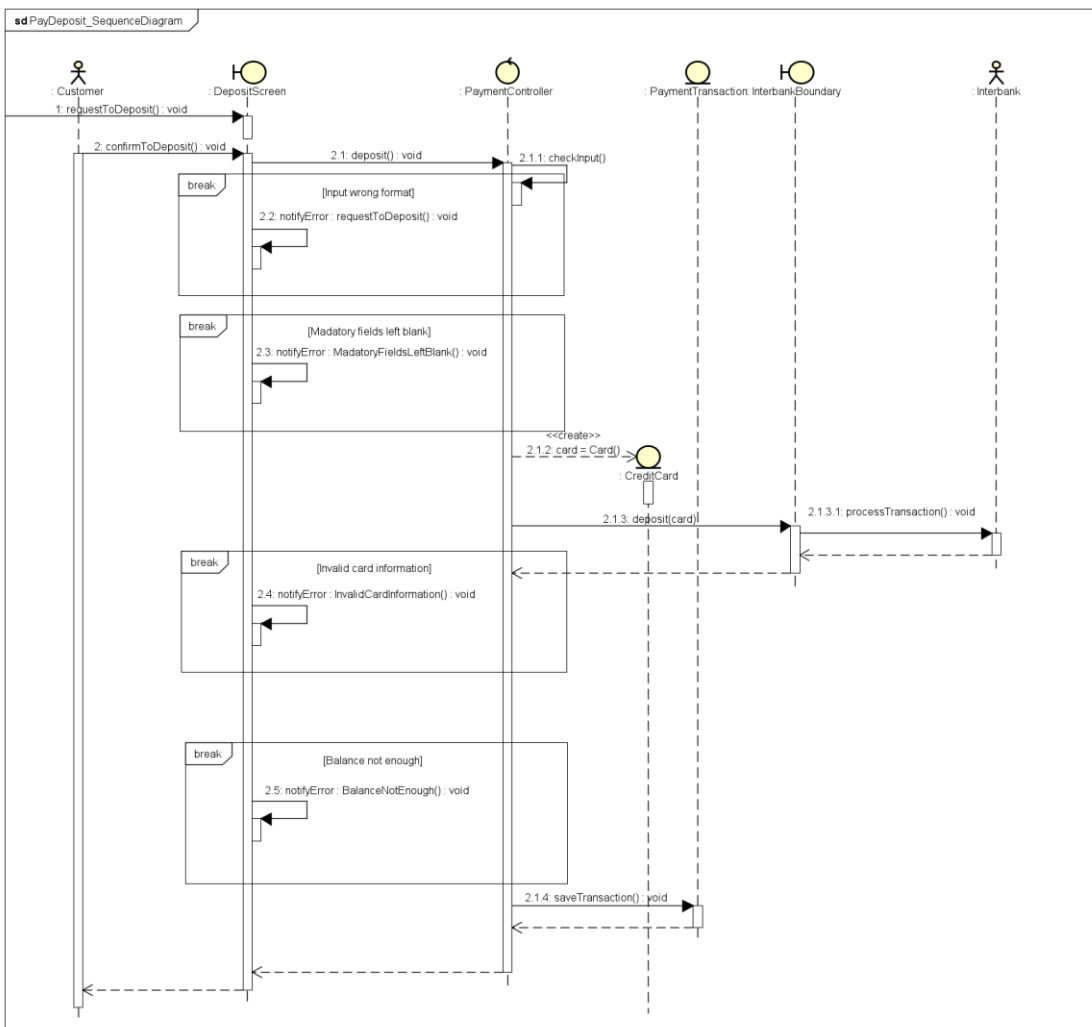
Figure 3.8: Sequence Diagram for Deposit Use Case
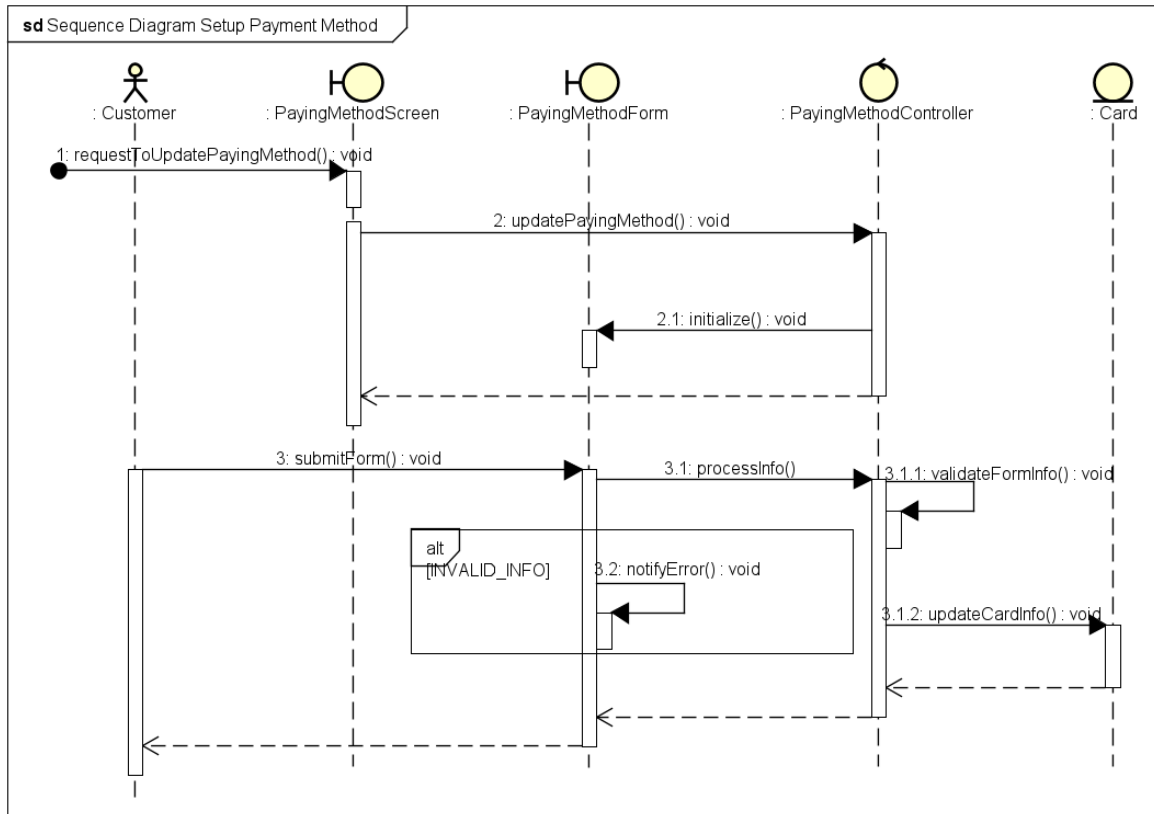
Figure 3.9: Sequence Diagram for Setup Payment Method Use Case
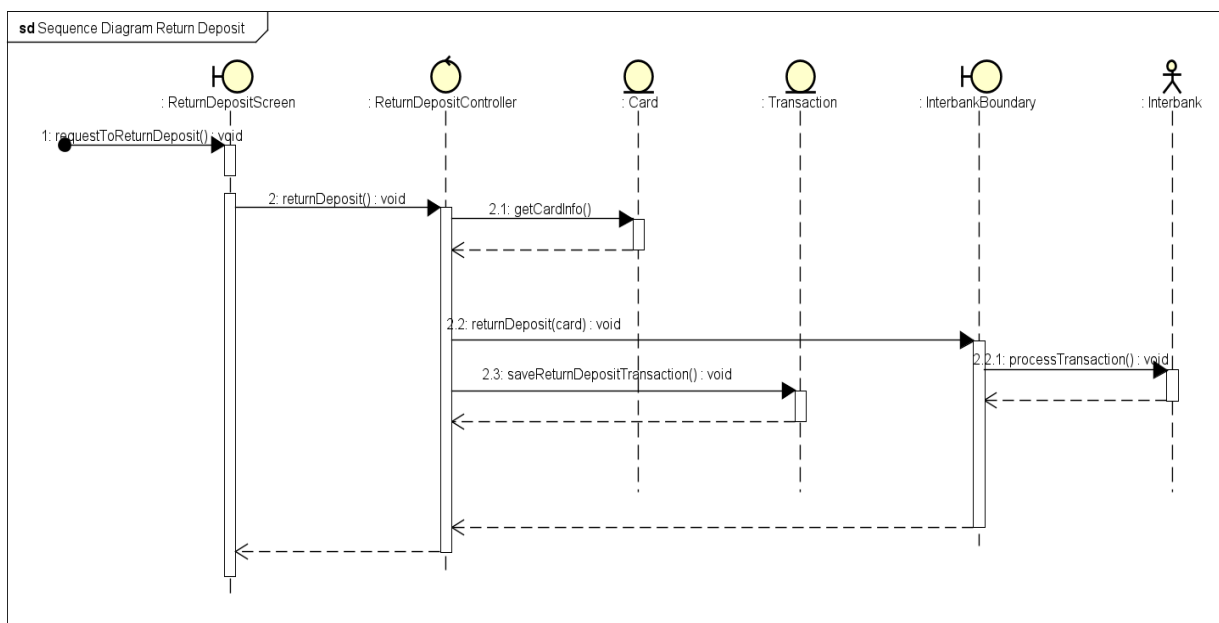
Figure 3.10: Sequence Diagram for Return Bike Use Case
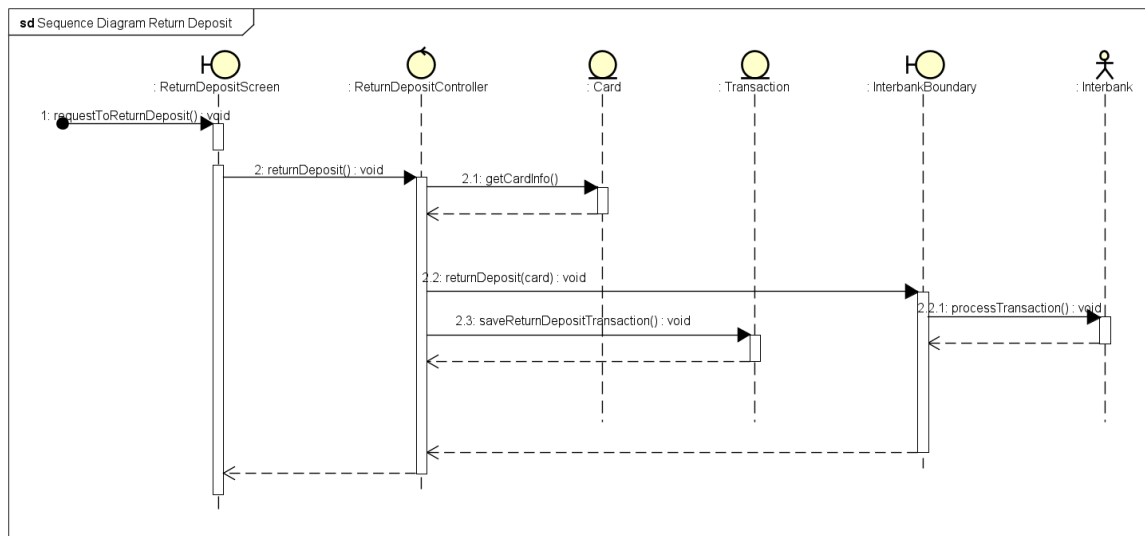
Figure 3.11: Sequence Diagram for Return Deposit Use Case
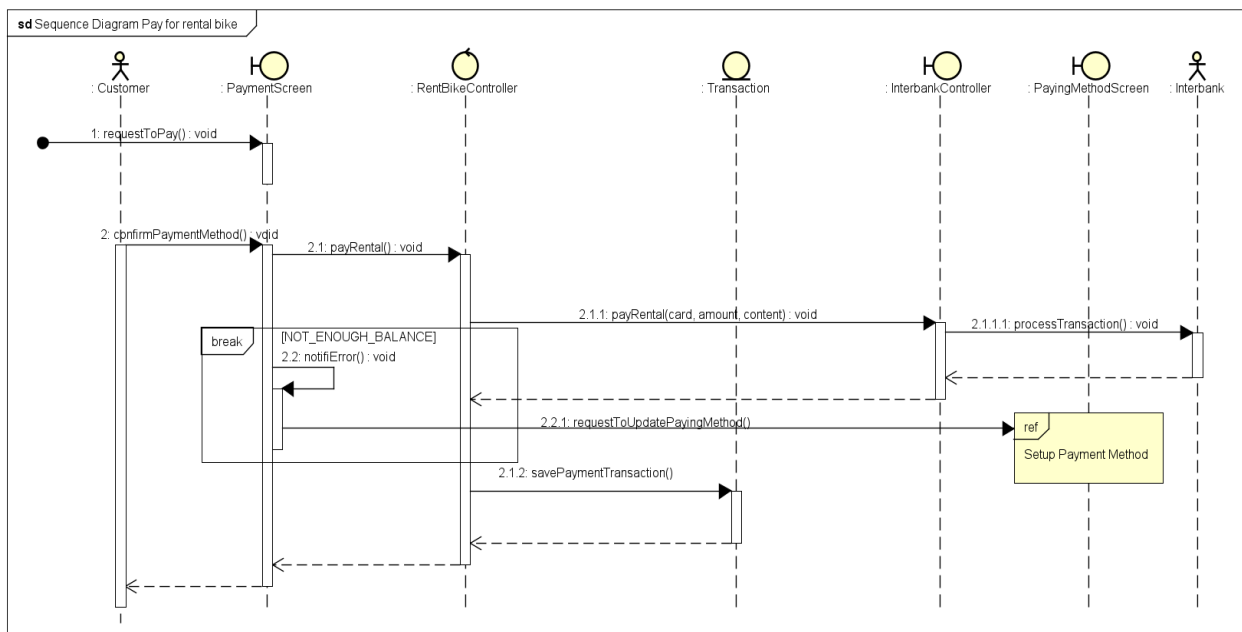
Figure 3.12: Sequence Diagram for Pay For Rental Use Case

3.2. Analysis Class Diagrams
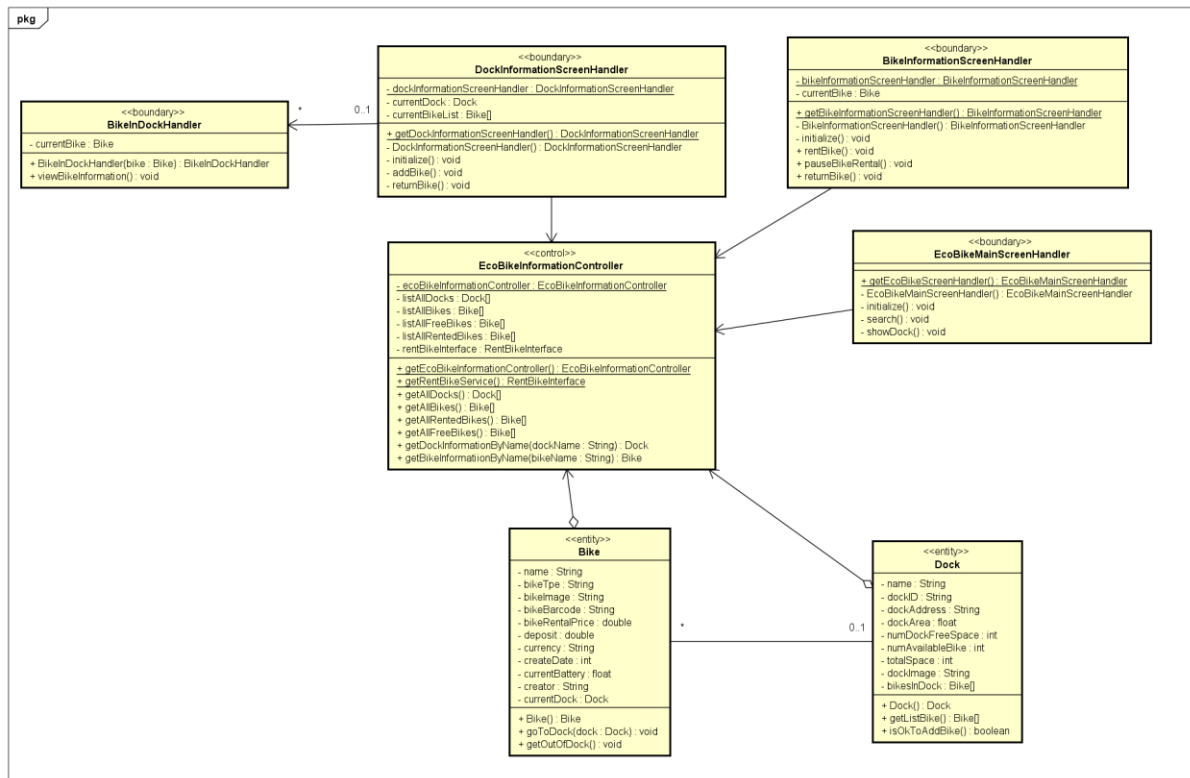
Figure 3.13: Class Diagram for View Bike Use Case
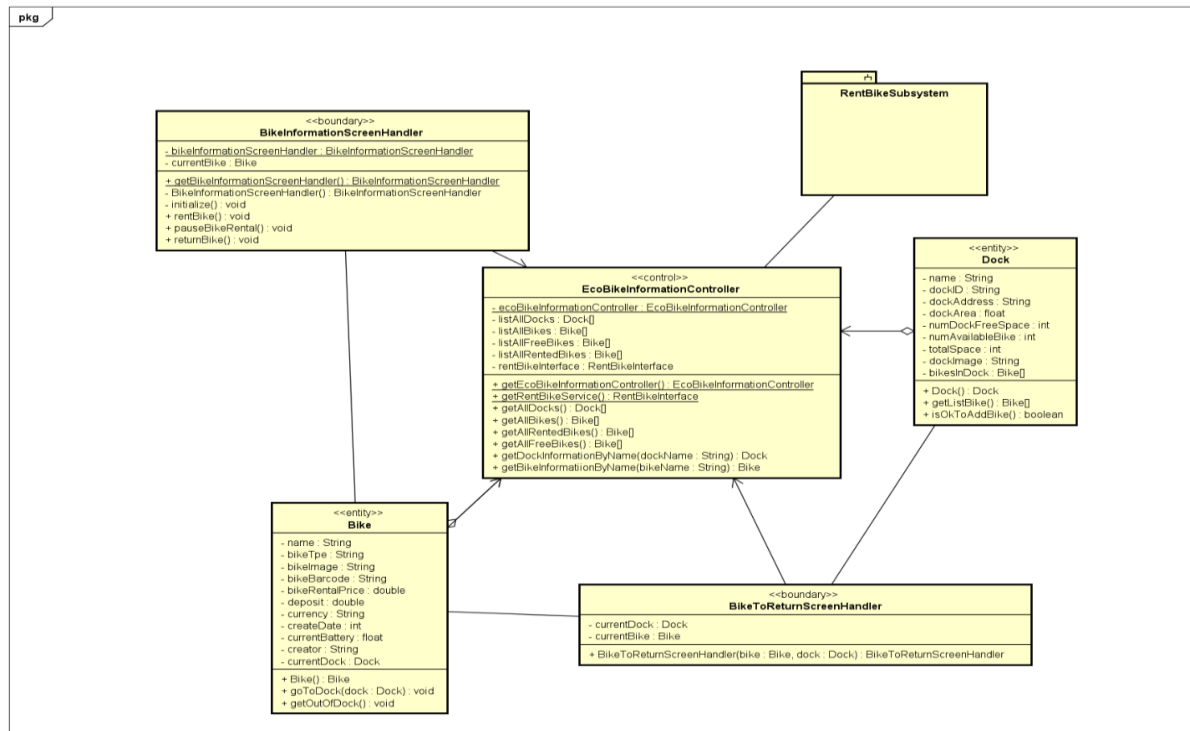
Figure 3.14: Class Diagram for Rent Bike Use Case
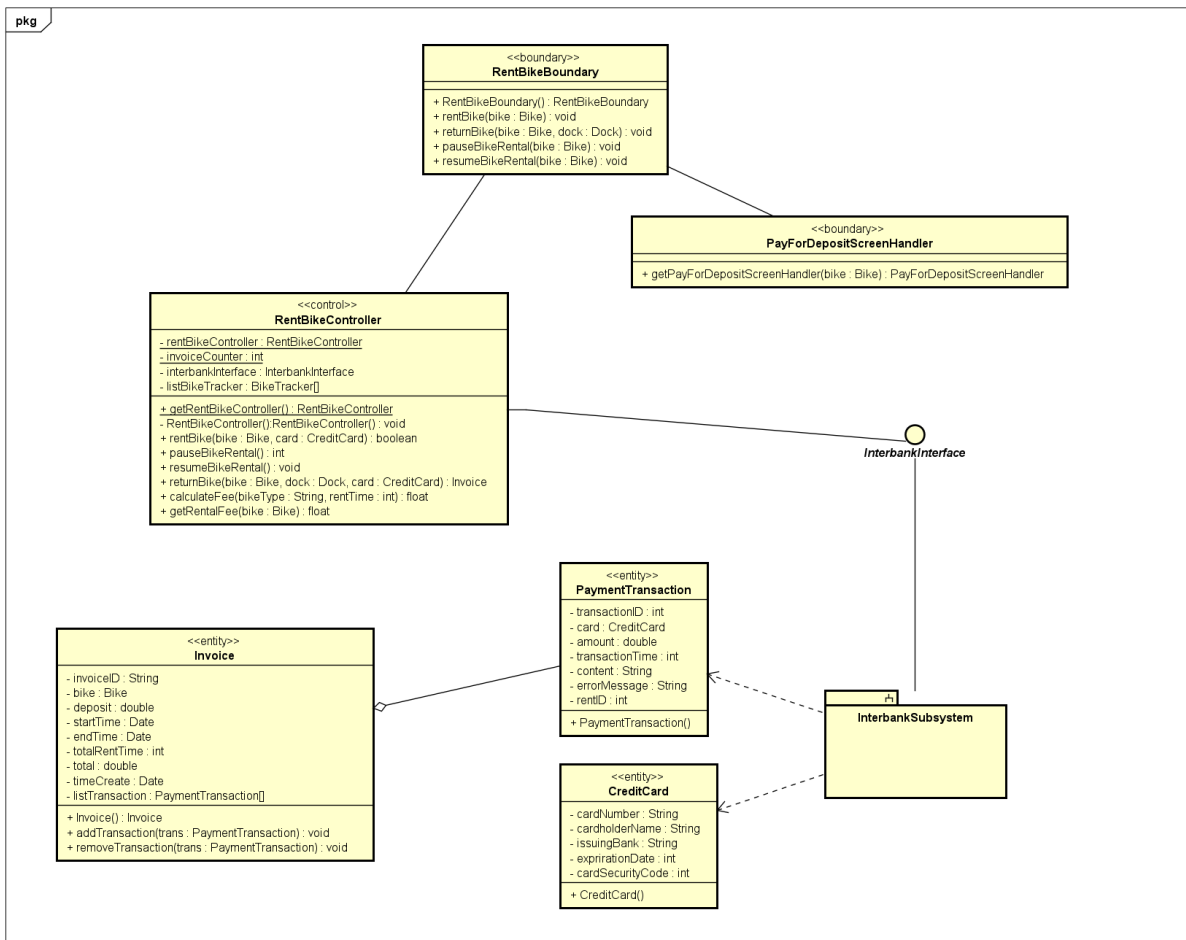
Figure 3.15: Class Diagram for Deposit Use Case
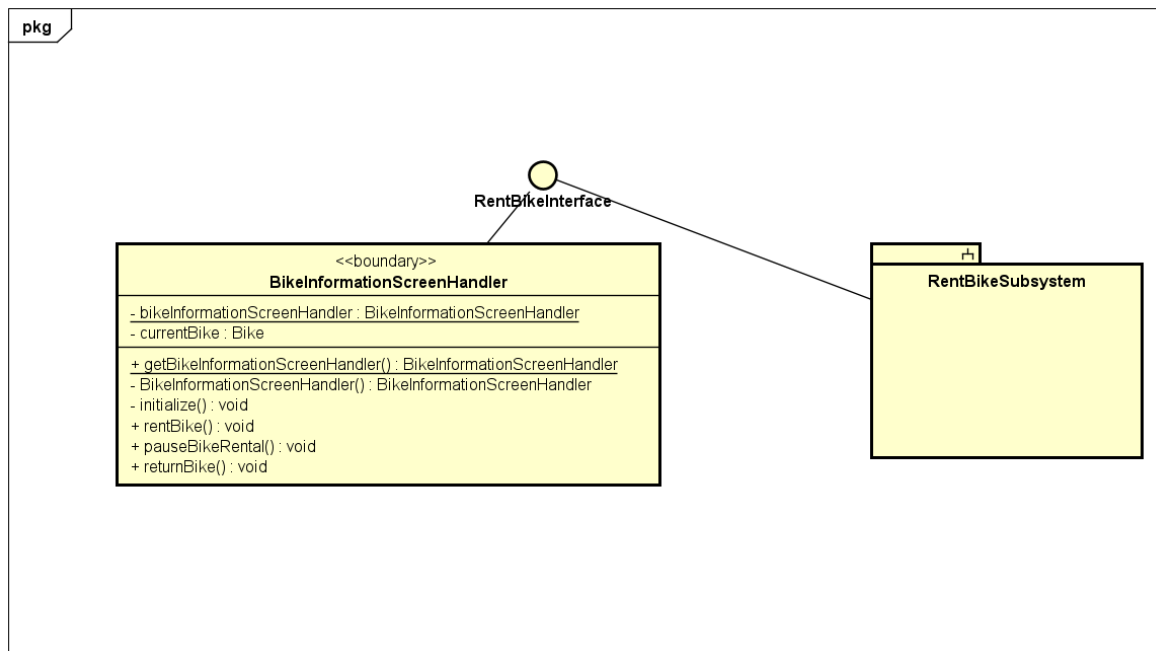
Figure 3.16: Class Diagram for Return Bike Use Case

19
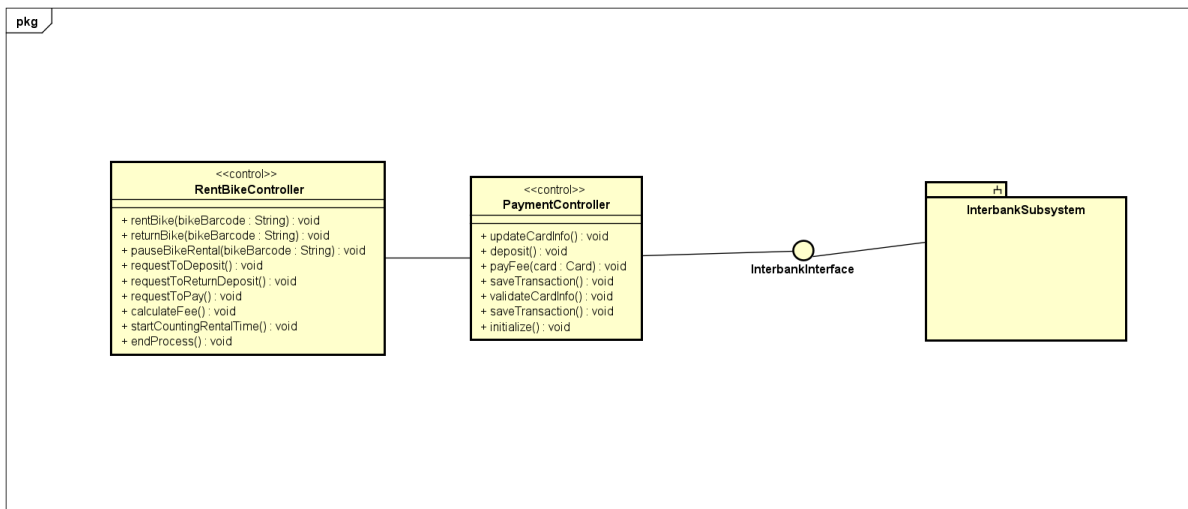
Figure 3.17: Class Diagram for Return Deposit Use Case
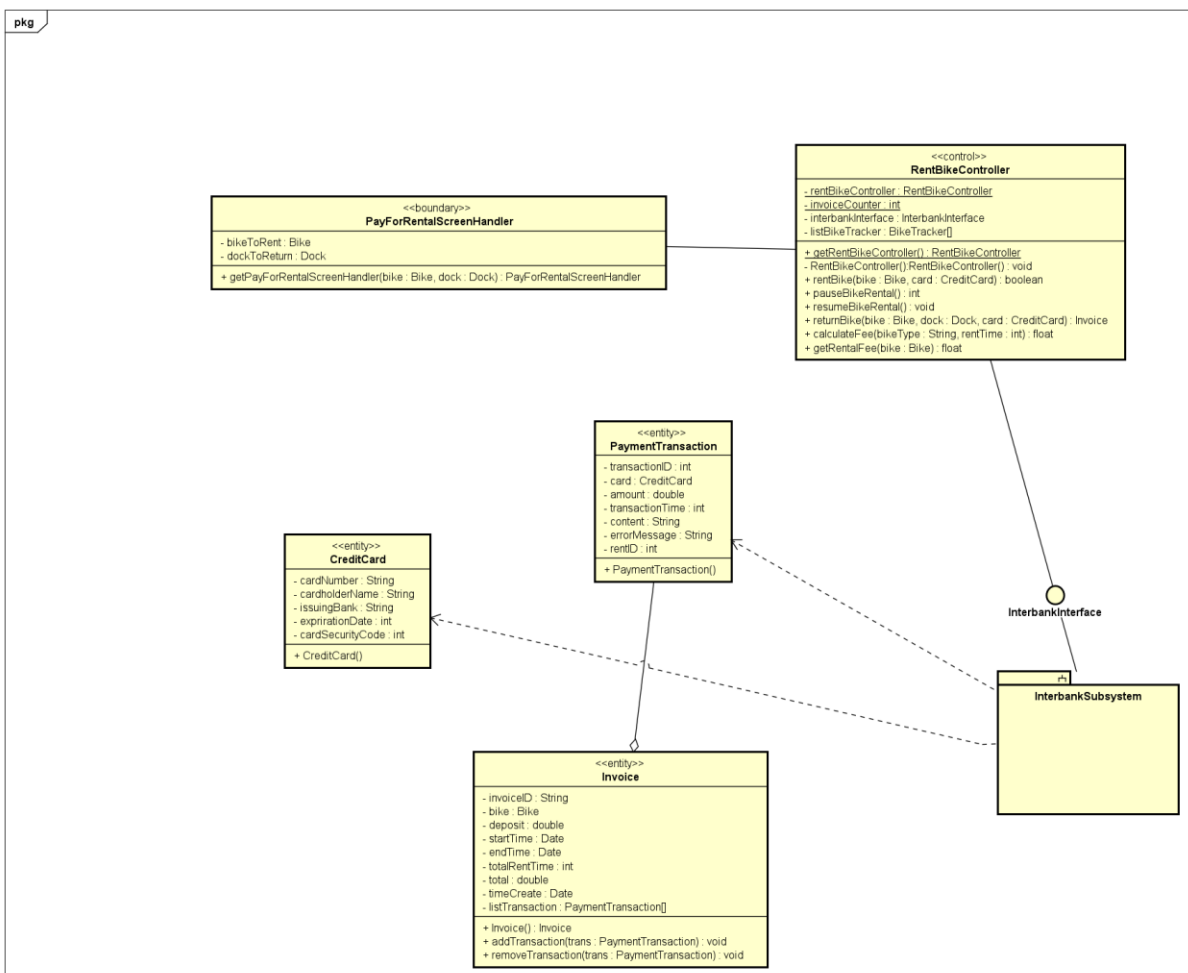
Figure 3.18: Class Diagram for Pay Rental Use Case

20

3.3. Unified Analysis Class Diagram
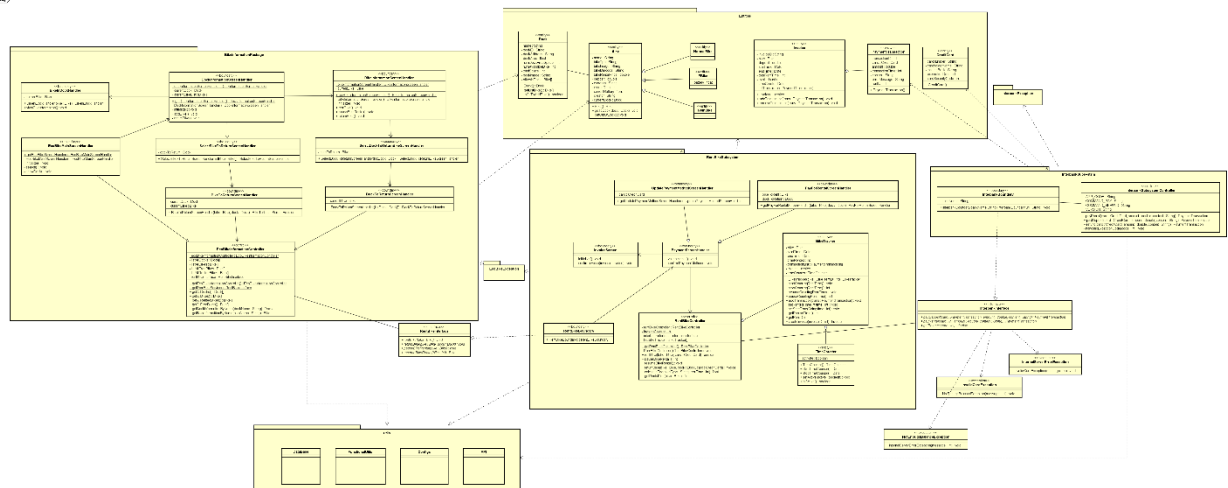
Figure 3.19: Unified Class Diagram for EcoBike Application

3.4. 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: #e5ffcc (R: 229, G: 255, B: 204)
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, Green
Numbers: comma for thousand separation, dot for decimal separation
Text: Segoe UI, size at most 24px

4.1.2. Screen Transition Diagrams
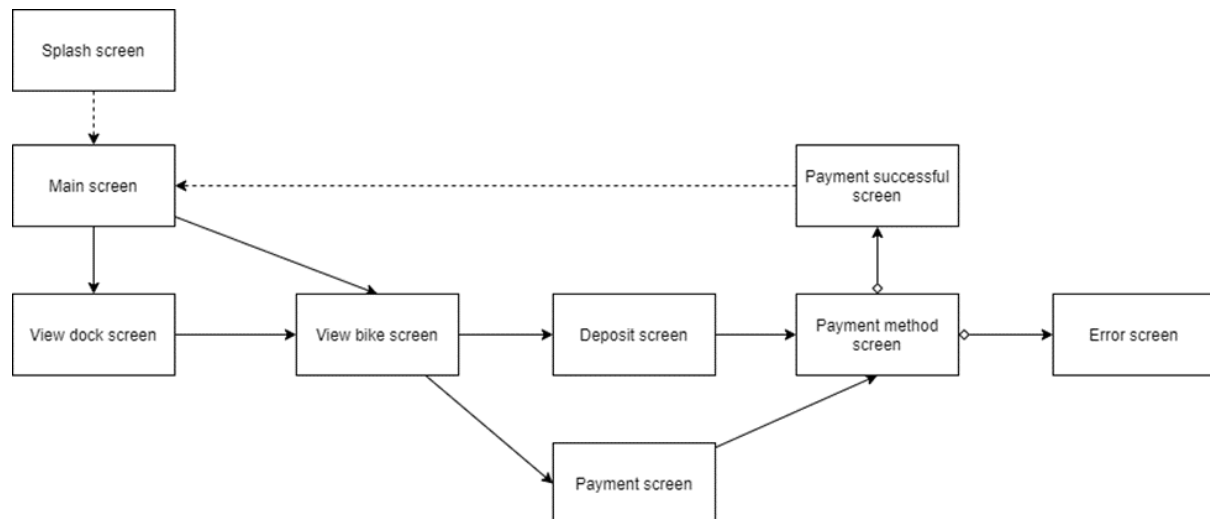
Figure 4.1: Screen Transition Diagram for EcoBike Application

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**	01/01/2023			Bien
		Control	**Operation**	**Function**	

		Main area	None	Introduce the application
ECO BIKE				

Table 4.1. Splash Screen Specification

4.1.3.2. Main Screen
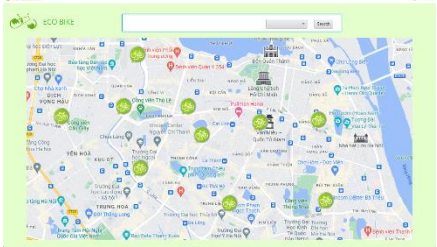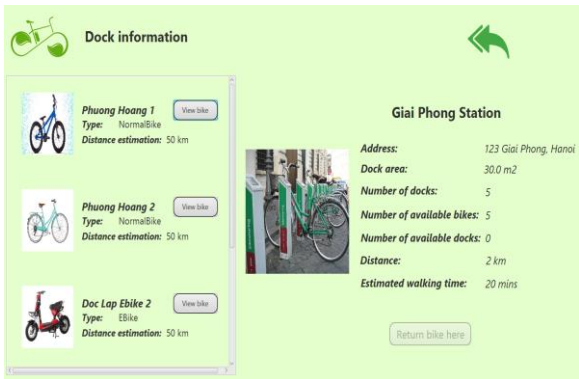
EcoBike Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	**Main screen**	01/01/2023			Bien
		Control	**Operation**	**Function**	
		Header logo	Click	Return immediately to main screen	
		Search bar	Type, select & click	Type in information and select search type 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**	01/01/2023			Bien
		Control	**Operation**	**Function**	
		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 current dock. Allow	

			choosing each bike to see detailed information

Table 4.3. View Dock Screen Specification

4.1.3.4. Bike Screen
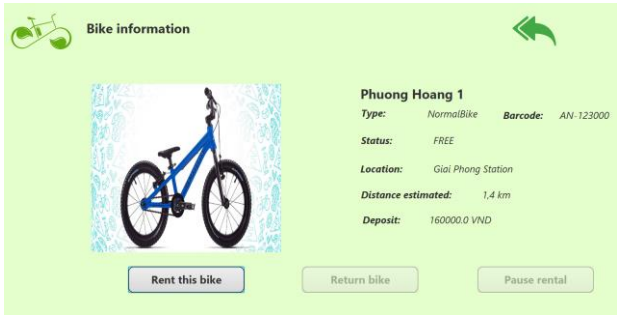
EcoBike Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	**View bike screen**	01/01/2023			Bien
		Control	**Operation**	**Function**	
		Logo	Click	Return to the main screen immediately	
		Header	Initial	Display title of screen	
		Bike information	Initial	Display 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
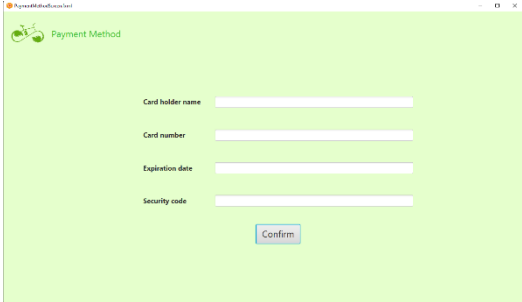
EcoBike Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	**Paying method screen**	01/01/2023			Bien
		Control	**Operation**	**Function**	
		Logo	Click	Return to the main screen immediately	
		Header	Initial	Display title of screen	
		Payment	Initial	Display information 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
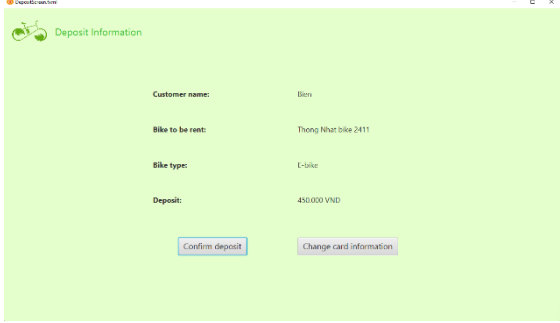
Screen specification	Payment screen	01/01/2023			Bien
			Control	Operation	Function
			Logo	Click	Return to the main screen immediately
			Header	Initial	Display title of screen
			Information of payment	Initial	Display information of payment
			Button	Click	Allow customer confirm to deposit the bike

Table 4.6. Deposit Screen Specification

4.1.3.7. Payment screen
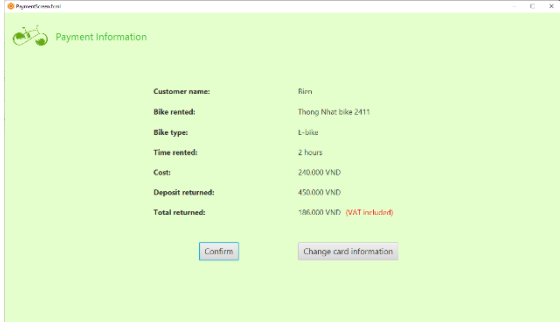
EcoBike Software			Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Payment screen		01/01/2023			Bien
			Control	Operation	Function	
			Logo	Click	Return to the main screen immediately	
			Header	Initial	Display title of screen	
			Information of payment	Initial	Display 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
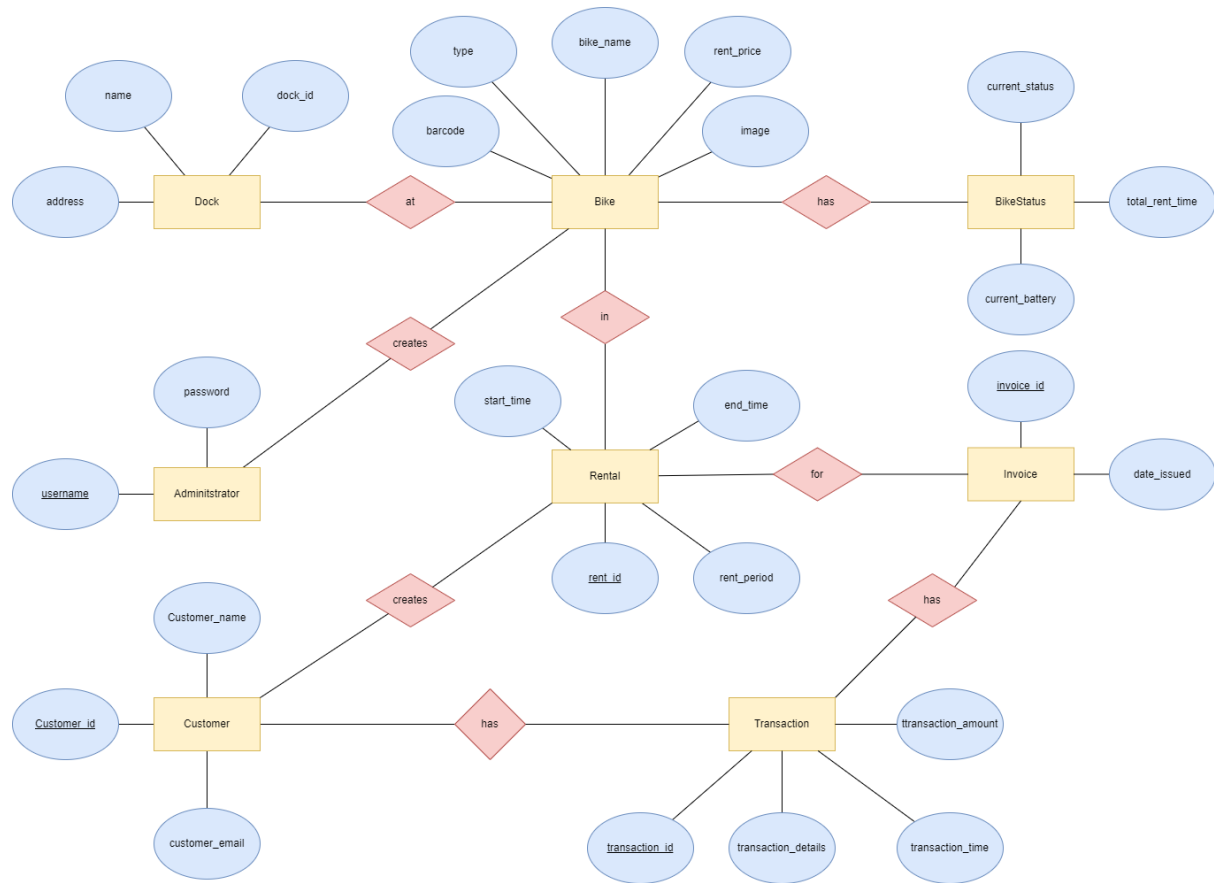
Figure 4.2. ER Diagram for EcoBike Application

4.2.2. Database Design

4.2.2.1. Database Management System

Database Management System: SQLite

4.2.2.2. Database Diagram
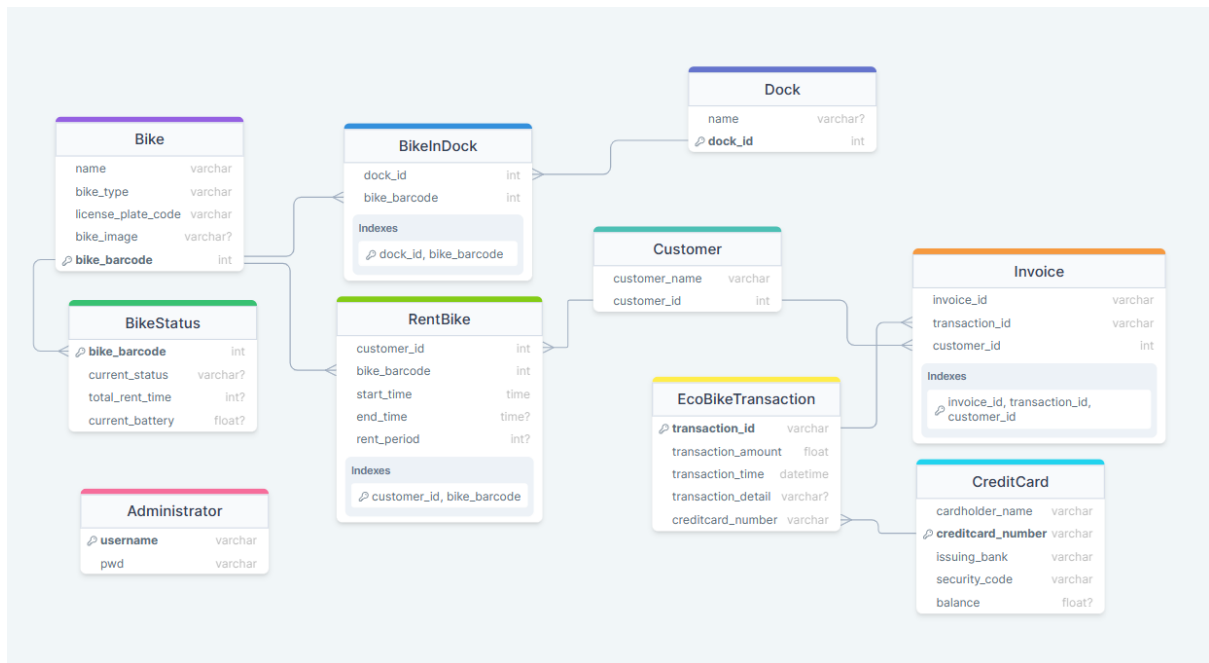
Figure 4.3. Database Diagram for EcoBike Application

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

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

No.	PK	FK	Name	Data type	Mandatory	Description
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

No.	PK	FK	Name	Data type	Mandatory	Description
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
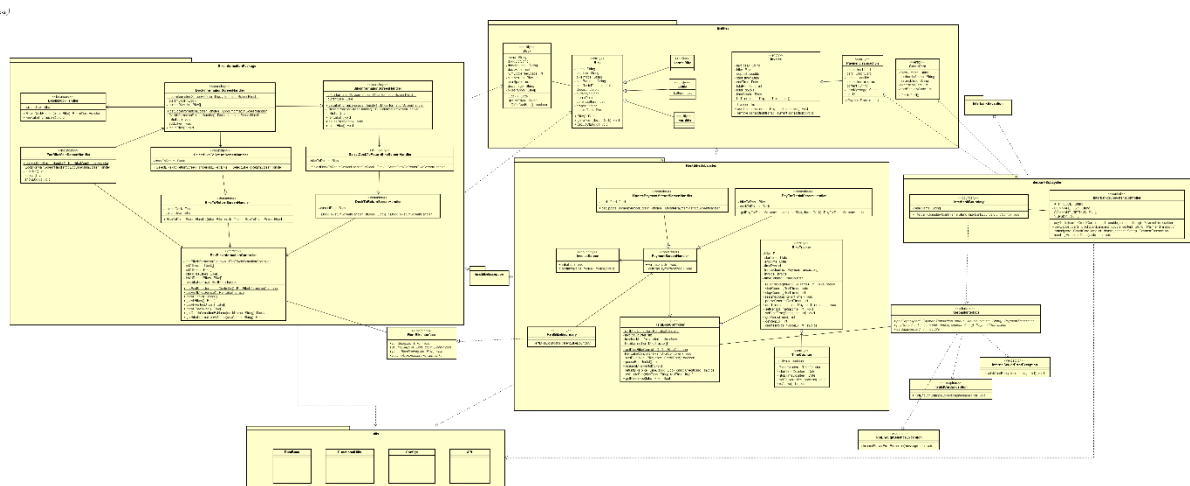
Figure 4.4. General Class Diagram for EcoBike Application

4.4.2. Class Diagrams

4.4.2.1. Class Diagram for Package BikeInformation
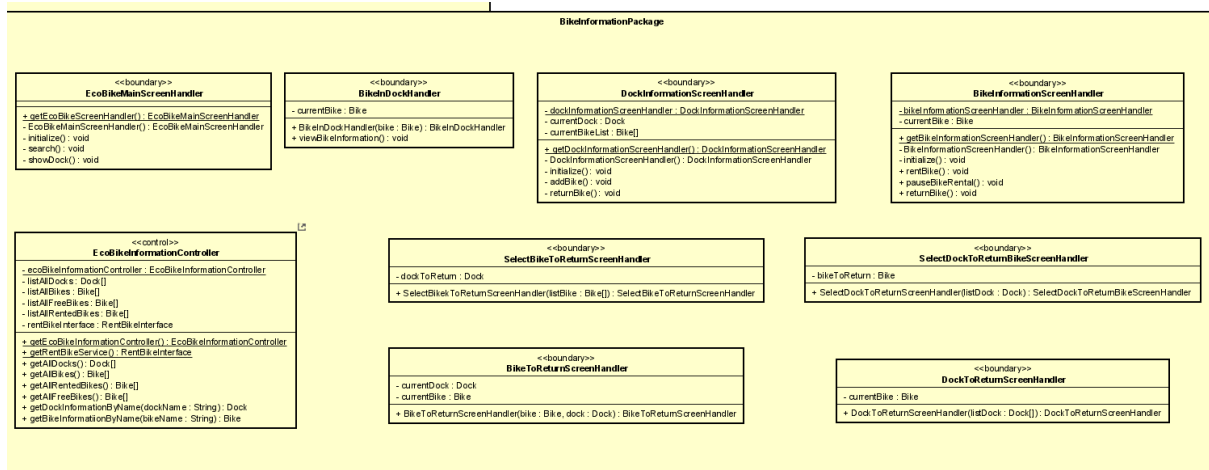
Figure 4.5. Class Diagram for Package BikeInformation

4.4.2.2. Class Diagram for Subsystem RentBike
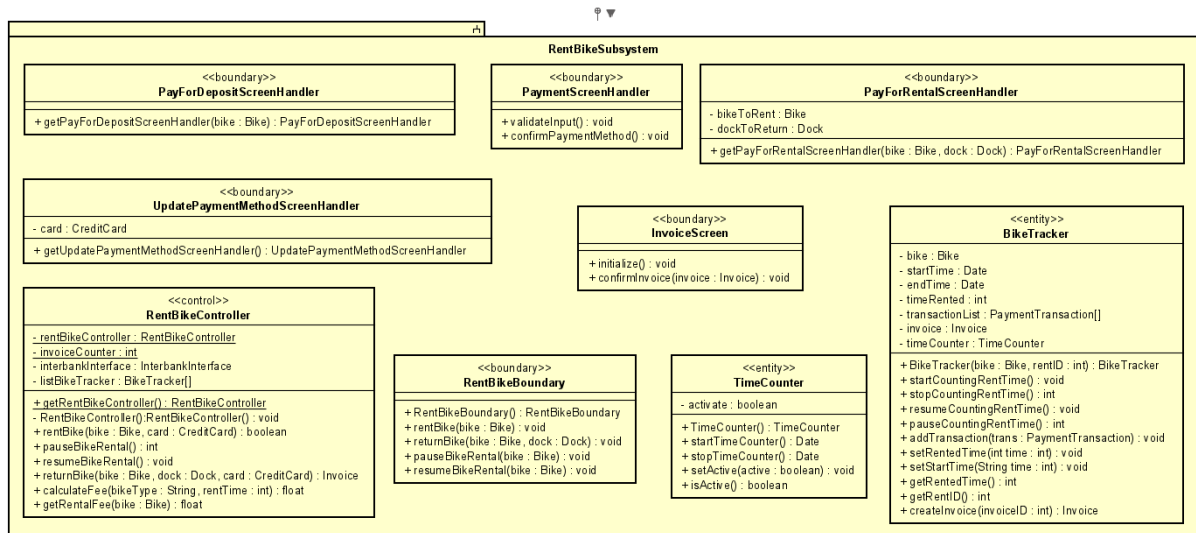
Figure 4.6. Class Diagram for Subsystem RentBike

4.4.2.3. Class Diagram for Subsystem InterBank
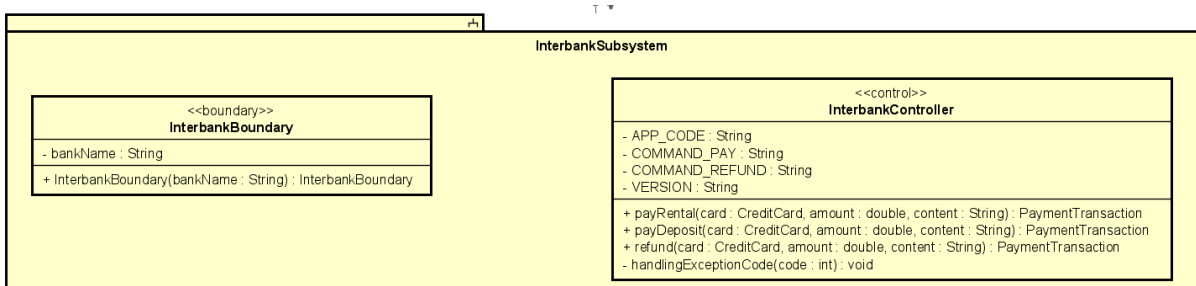
Figure 4.7. Class Diagram for Subsystem InterBank

4.4.3. Class Design

4.4.3.1. Class RentBikeController
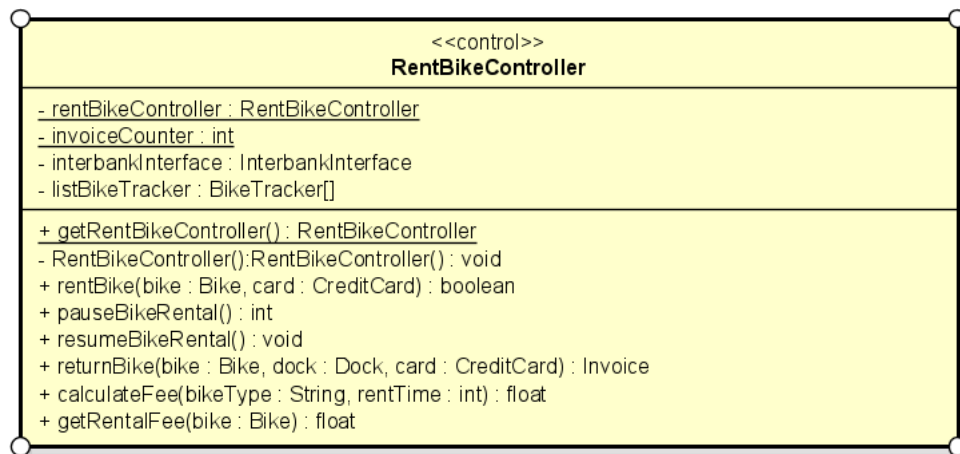
Figure 4.8. RentBikeController Class Diagram

#	Name	Data type	Default value	Description
1	interbankInterface	InterbankInterface	InterbankInterface	Interbank to proceed transaction
2	rentBikeController	RentBikeController	null	static instance of the RentBikeController
3	invoiceCounter	int	0	Calculate invoice follow time
4	listBikeTracker	BikeTracker[]	null	List of bike tracker

Table 4.18. RentBikeController attributes

#	Name	Return type	Description (purpose)
1	getRentBikeController	RentBikeController	return static instance rentBikeController of class RentBikeController
2	rentBike	void	start renting proceed

3	pauseBikeRental	void	pause counting rental time
4	startCountingRentBike	void	start counting renal time
5	calculateFee	float	Calculate rantal fee
6	getRentalFee	float	Get rental fee result

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 InterbankController
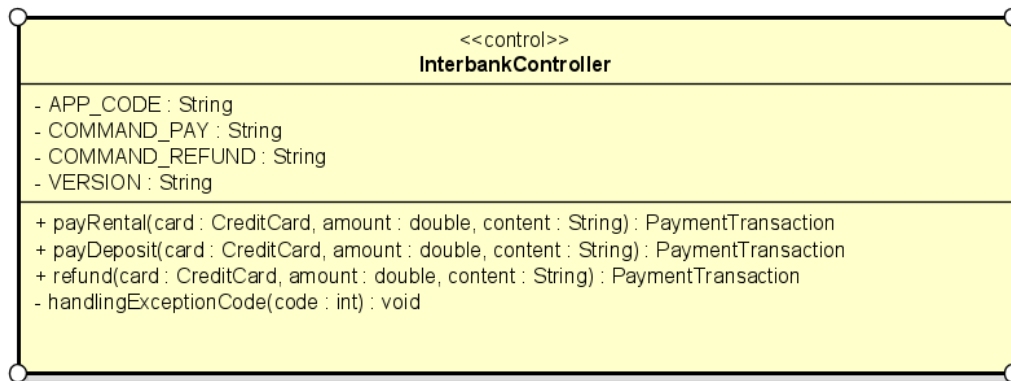
*Figure 4.9. InterbankController Class Diagram*

#	Name	Data type	Default value	Description
1	APP_CODE	String	raQdFQH5jg==	Code of app
2	COMMAND_ PAY	String	PAY	Command to pay
3	COMMAND_ REFUND	String	refund	Command to refund
4	VERSION	String	1.0.1	Version to pay

Table 4.20. ReturnBikeController attributes

#	Name	Return type	Description (purpose)
1	payRental	PaymentTransaction	Pay rental fee
2	payDeposit	PaymentTransaction	pay deposit
3	refund	PaymentTransaction	Return deposit and pay money

| 4 | handlingExce ptionCode | void | Handle exception code |

Table 4.21. ReturnBikeController operations

Parameter:

- card: your card for interbank
- amount: the fee you must pay
- content: content of comment in paying process

Exception:

- handlingExceptionCode If the bankcode is not currently available, the CvvCode is not valid

Method

None

State

None

4.4.3.3. Class PaymentController
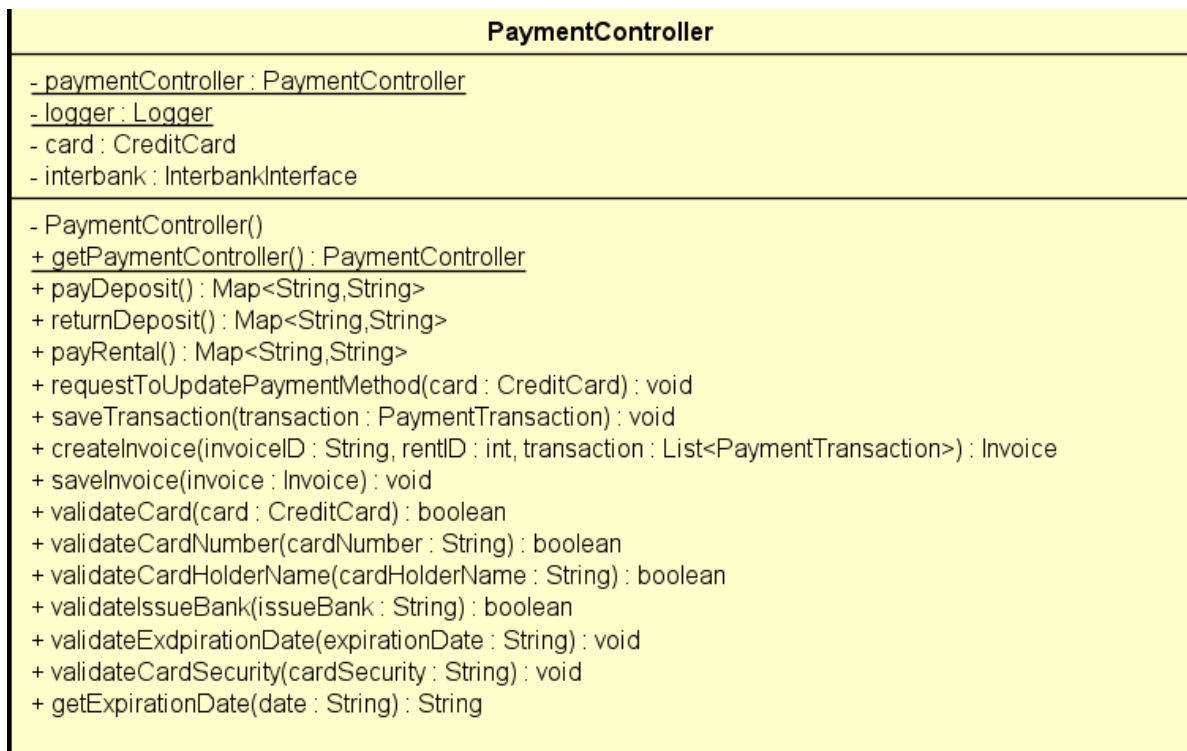
Figure 4.10. PaymentControllerClass Diagram

#	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	requestToUpdatePaymentMethod	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	bikeDistance Text	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	returnBikeButton	Button	null	button to return this bike
11	rentBikeButton	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: SQLite
- 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

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 a new instance of screen handler each time we change the screen

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.