

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology

Software Design Document

Version 1.3

Eco Bike Rental

Subject: ITSS-2021-1

Group 4

Nguyễn Viết Huy

Trần Đức Hiếu

Hoàng Mai Đức Long

Hanoi, <month, year>

<All notations inside the angle bracket are not part of this document, for its purpose is for extra instruction. When using this document, please erase all these notations and/or replace them with corresponding content as instructed>

<This document, written by Prof. NGUYEN Thi Thu Trang, is used as a case study for student with related courses. Any modifications and/or utilization without the consent of the author is strictly forbidden>

Table of Contents

Table of Contents	1
1 Introduction	5
1.1 Objective	5
1.2 Scope	5
1.3 Glossary	5
1.4 References	5
2 Overall Description	7
2.1 General Overview	7
2.2 Assumptions/Constraints/Risks	7
2.2.1 Assumptions	7
2.2.2 Constraints	7
2.2.3 Risks	8
3 System Architecture and Architecture Design	9
3.1 Architectural Patterns	9
3.2 Interaction Diagrams	9
3.3 Analysis Class Diagrams	9
3.4 Unified Analysis Class Diagram	9
3.5 Security Software Architecture	9
4 Detailed Design	10
4.1 User Interface Design	10
4.1.1 Screen Configuration Standardization	10
4.1.2 Screen Transition Diagrams	10
4.1.3 Screen Specifications	10
4.2 Data Modeling	10
4.2.1 Conceptual Data Modeling	10

4.2.2	Database Design	10
4.3	Non-Database Management System Files	11
4.4	Class Design	11
4.4.1	General Class Diagram	11
4.4.2	Class Diagrams	11
4.4.3	Class Design	11
5	Design Considerations	13
5.1	Goals and Guidelines	13
5.2	Architectural Strategies	13
5.3	Coupling and Cohesion	14
5.4	Design Principles	14
5.5	Design Patterns	14

List of Figures

No table of figures entries found.

List of Tables

No table of figures entries found.

1 Introduction

1.1 Objective

The purpose of this document is to present a detailed description of the implementation of the EcoBikeRental system. It will explain the architecture of the software, specify each of the components in the system, how they are implemented and interact with other modules. This document is intended for the developers of the system.

1.2 Scope

This software system, Ecobike Rental, will be a system for users to rent and return bikes automatically in the area of Ecopark township.

The software allows users to view information of all available docks and bikes; as well as to find a dock, rent a bike, return the bike and pay for rent. To pay for rent, users need to link the software to an Interbank account or E-wallet.

1.3 Glossary

No	Term	Definition
1	MVC	Stands for Model View Controller, is an architecture pattern for software development.
2	DBMS	Stands for Database Management System, is software that provides means to store, extract, update data.
3	GUI	Stands for Graphical User Interface, is the graphic interface that allows interaction between user and program.

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

EcoBikeRental is a desktop application that allows users to rent bikes in the area of Ecopark township. The user interacts with the app by clicking and entering information to the GUI. The interface then sends requests along with data to the program to process. Based on the process result, GUI will display the result to the user.

Below is the use case diagram for EcoBikeRental, as it should suggest which features, modules the project should include.



2.2 Assumptions/Constraints/Risks

2.2.1 Assumptions

Assuming, the user satisfied all requirements to run the app. First the user should have stable internet connections. Second, since this is a desktop/laptop application written in Java, the user should have a laptop/desktop that has Java JRE installed.

2.2.2 Constraints

- Low-end computers may affect user experience as it will produce lag and latency.
- Weak internet may result in data faults.

2.2.3 Risks

The user of this software will need to make a transaction to pay and make a deposit which may lead to the risk of leaking the user's private data.

→ Avoid storing user's transaction data or private data in the system. If user's information is needed, a data encoding procedure is required.

If the application or user's device is shut down during operation, the app may produce faulty results.

→ Backup crucial state of the operation, such as: store rental information and state to the database.

3 System Architecture and Architecture Design

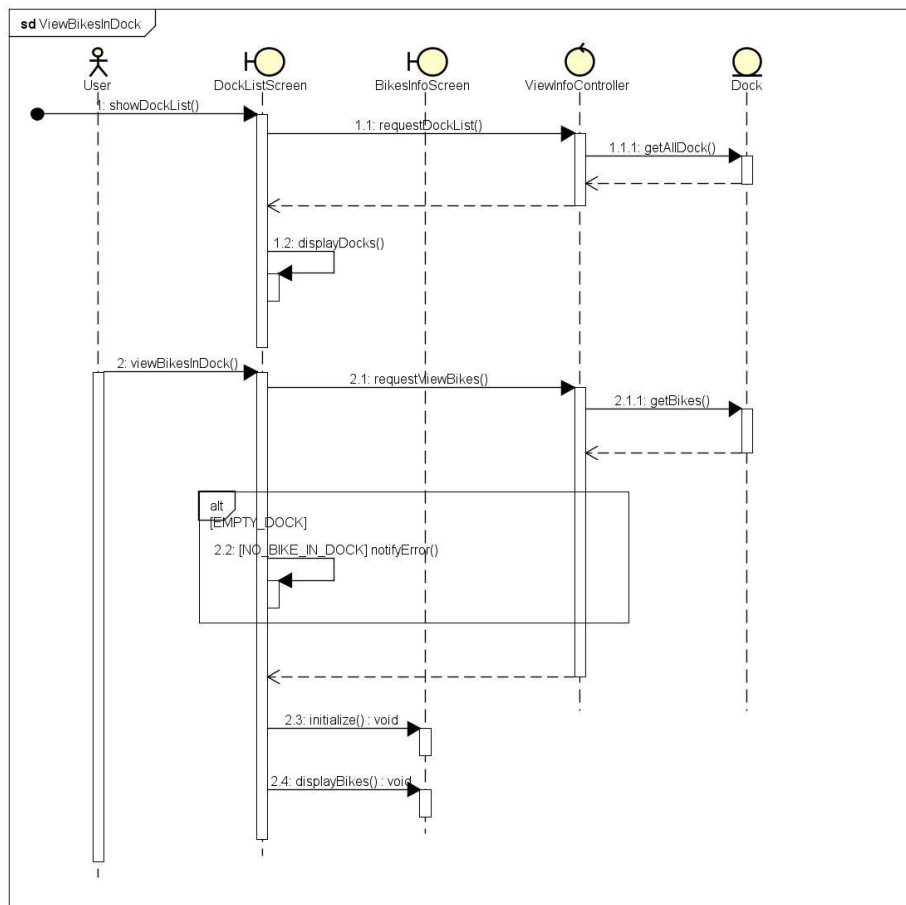
First we decide on an architecture which will become a framework for the design. Next, from the use case, we estimate the components and modules needed and specify their behaviours as well as the interactions between them. After assigning behaviours to class, we can design class's attributes and methods.

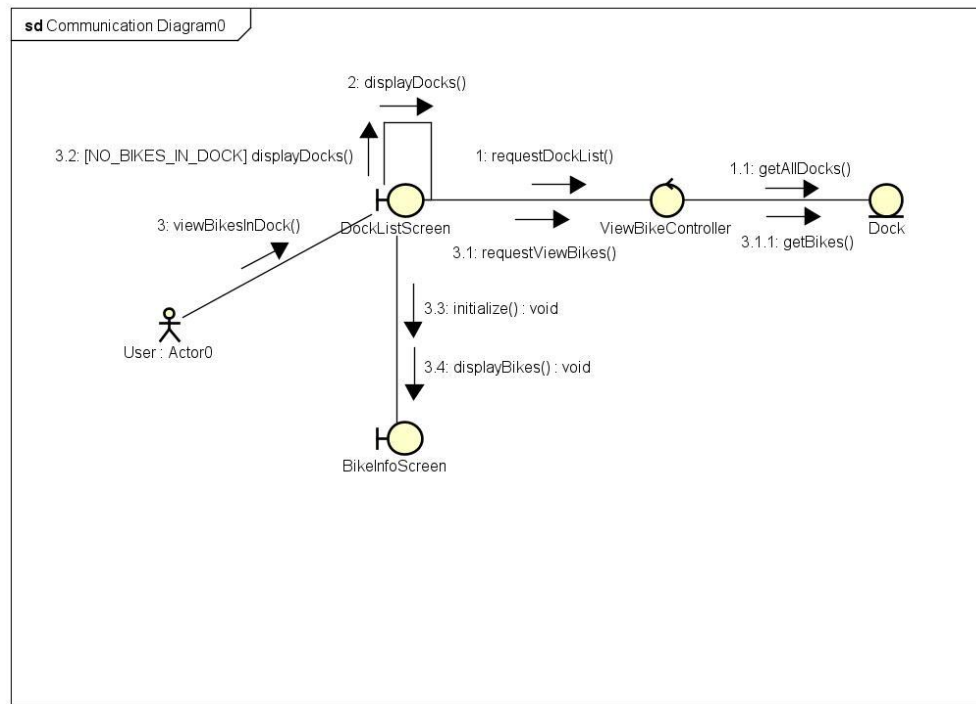
3.1 Architectural Patterns

For this project, the chosen architecture is MVC pattern. Entity class or Model shapes the data and represents real world objects in the system. Controller class processes the data according to the business flow. Boundary class or View is responsible for taking in user input and displaying processed data from the controller. The reason for MVC is that it gives a clear separation of concern and distributes responsibility to class logically.

3.2 Interaction Diagrams

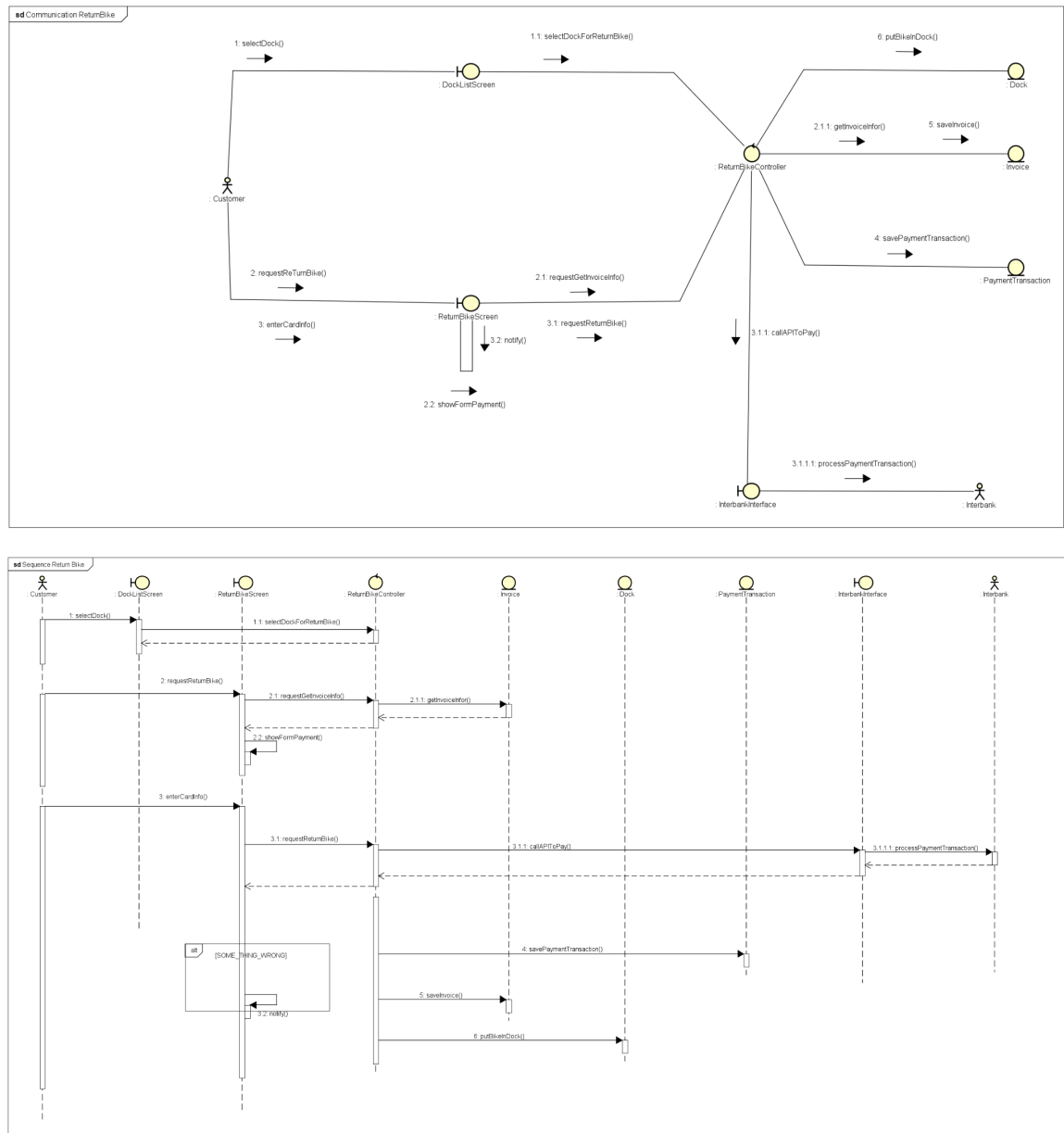
3.2.1. Use case View Bikes in Dock





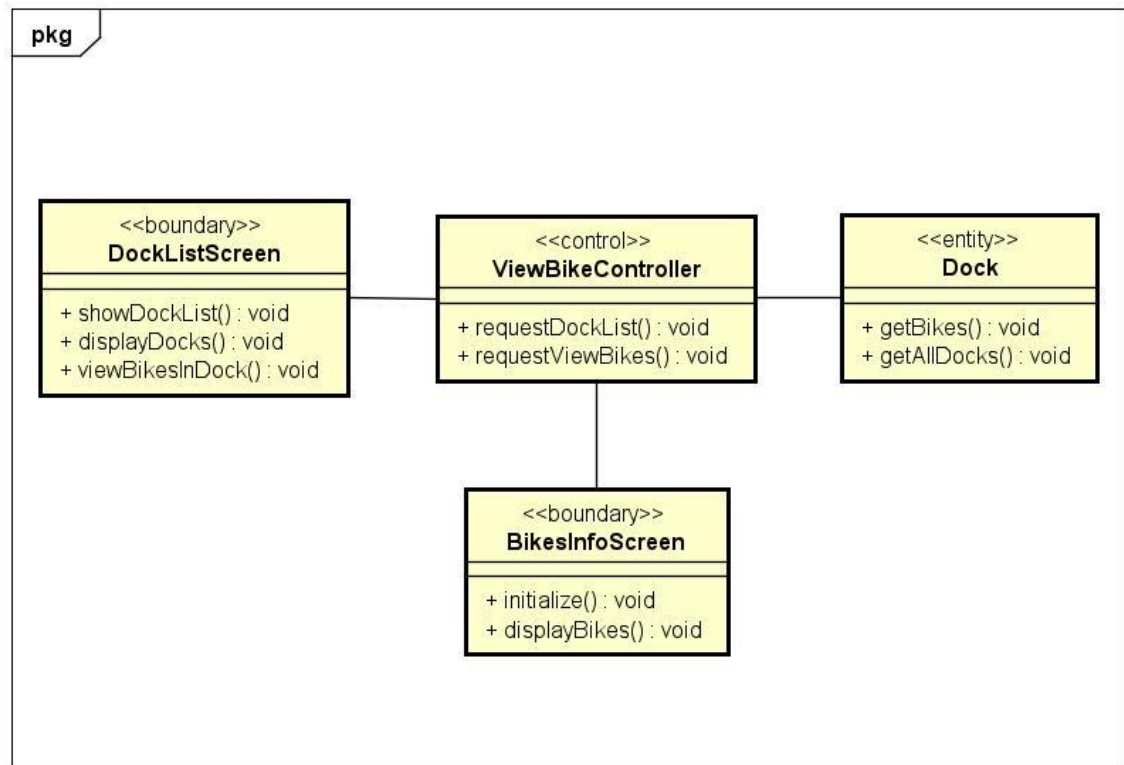
3.2.2. Use case Rent Bike

3.3.3. Use case Return Bike

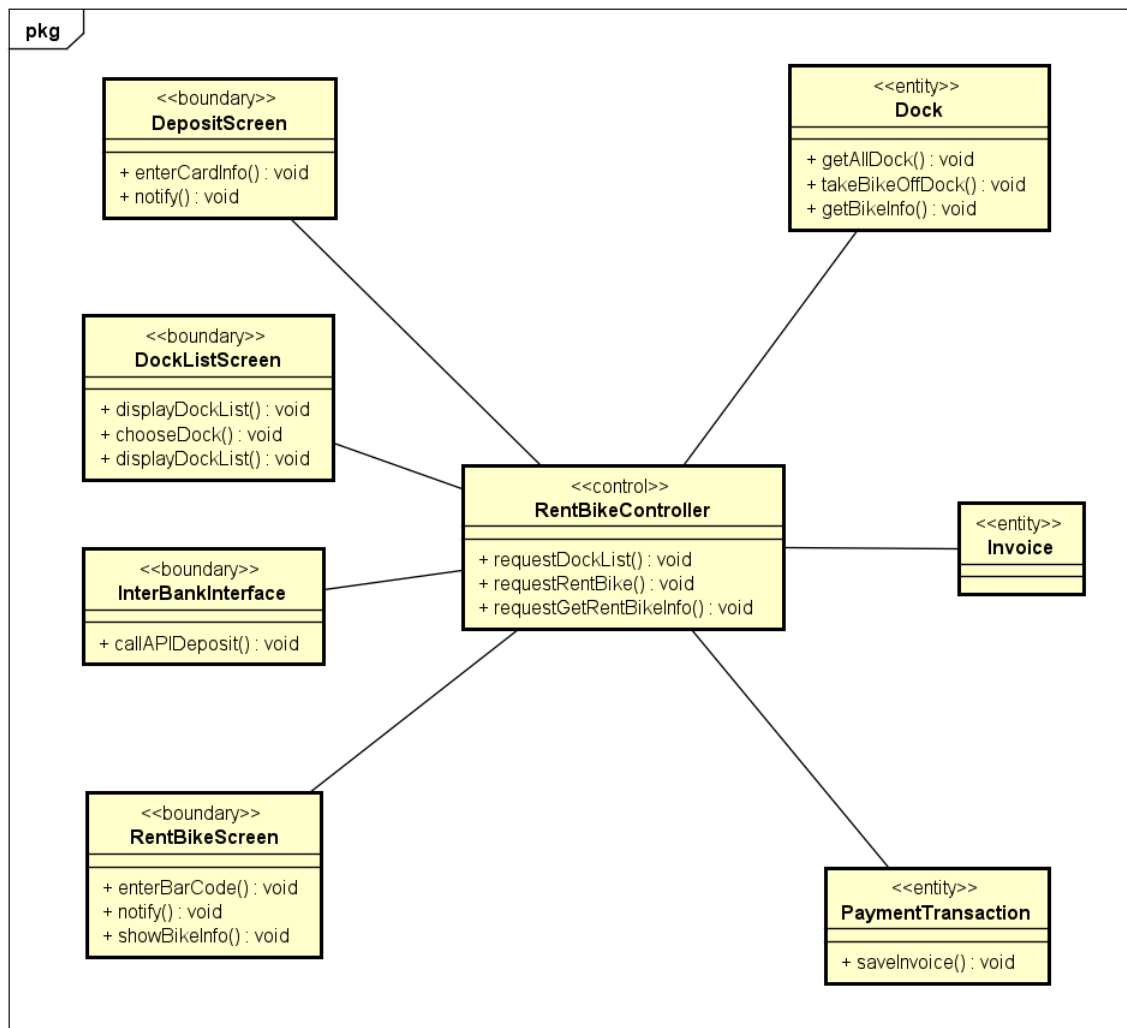


3.3 Analysis Class Diagrams

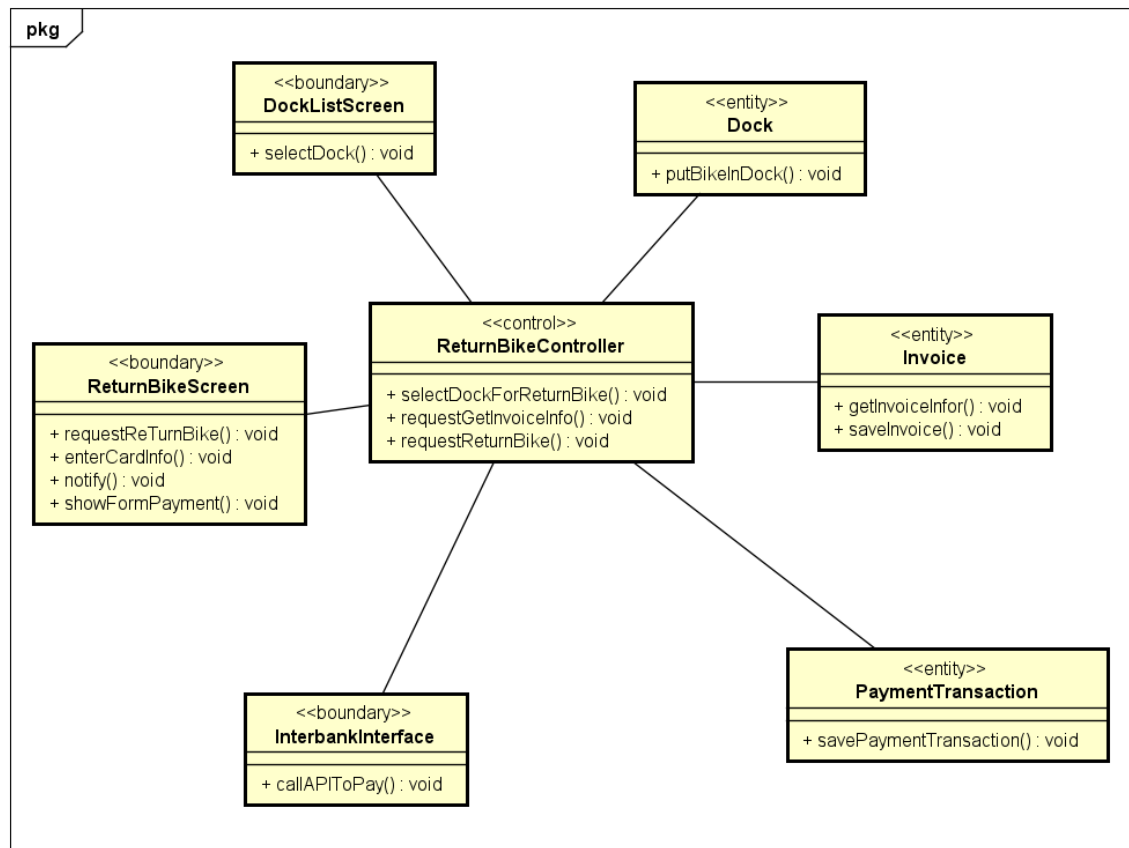
3.3.1. Use case View Bikes in Dock



3.3.2. Use case Rent Bike



3.3.3. Use case Return Bike



3.4 Unified Analysis Class Diagram

3.5 Security Software Architecture

In this project, as requirement has stated, the main focus of the software is 3 main functions: view bikes and docks, rent a bike and return a bike. Therefore user authentication is not implemented in the current version.

4 Detailed Design

4.1 User Interface Design

The following libraries/packages will be used for GUI design and execution:

- JavaFX 15
- SceneBuilder

4.1.1 Screen Configuration Standardization

a. Display

- Resolution: 900x600 pixels
- Number of color supported: 16.581.357

b. Screen

- Screen title location: At the upper-left corner of the screen.
- Location of standard button: At the bottom-center of the screen.
- Consistency of expressions:
 - + Comma is used to separate large numbers.
 - + Text will only consist of characters, digits, commas, dots, underscores, and hyphen symbols.
 - + Measurement and currency will be displayed in a consistent format.

c. Control

- Text: the text that user enter will be displayed in following settings
 - + Size: 20
 - + Color: Black
 - + Font: System (supported by JavaFX)
- Input validation process: All required input from the user will be validated, check empty or in the correct format.
- Sequence of screen:
 - + Home screen is compacted which contains dock list view and rent status view. The toggle between 2 views is done with tabs.
 - + Beside the home screen, other screens are separated and there will be no overlapping screen.

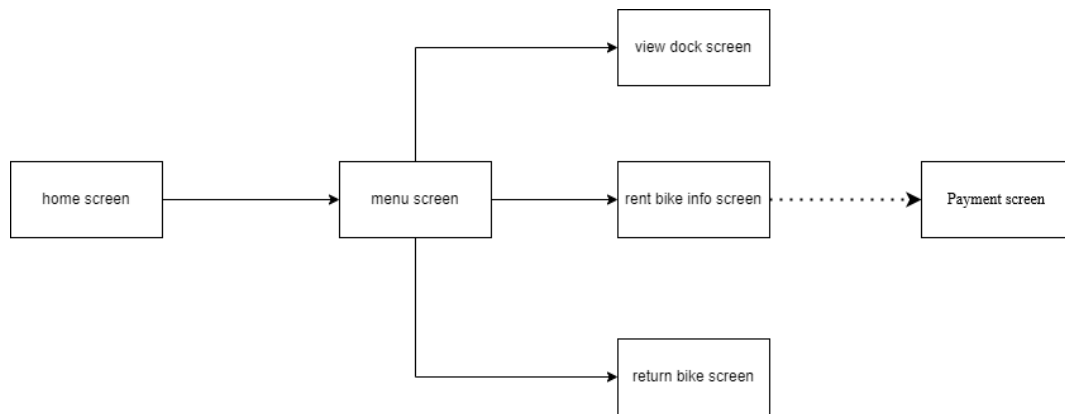
d. Direct input from keyboard

- There will be no shortcuts or keyboard key to control the screen.
- Each screen has a “Back” button to return to the previous screen.

e. Error


When an error occurs, error notification will be raised in the form of a pop up screen. Popup will contain details about the error and suggestions for the user to fix the error.

4.1.2 Screen Transition Diagrams



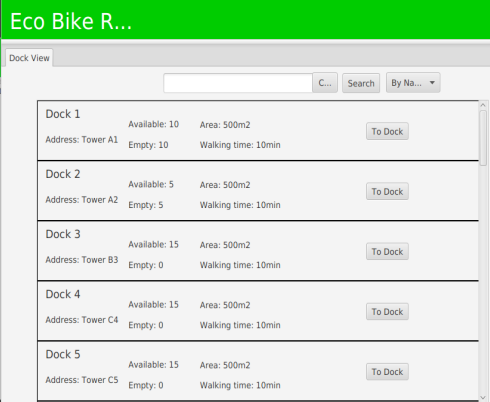
4.1.3 Screen Specifications

a. Splash screen

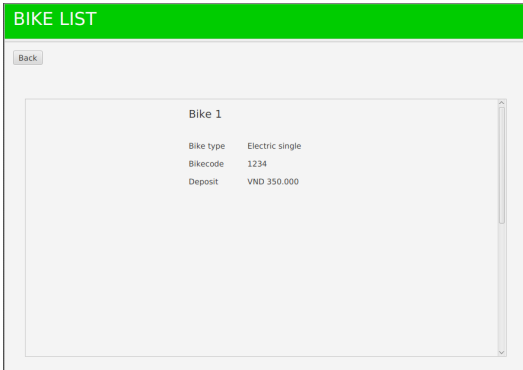
EcoBikeRental	Approved by	Reviewed by	In charge
Splash screen			Hoàng Mai Đức Long
	Control	Operation	Function
	Logo area	Display	Display software logo and name

b. Home screen

- Dock list view

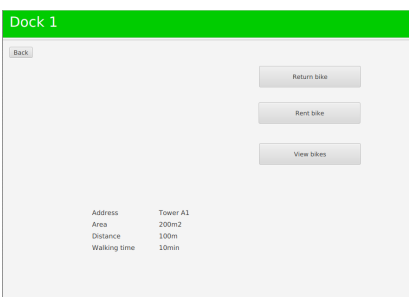
EcoBikeRental	Approved by	Reviewed by	In charge
Dock list			Hoàng Mai Đức Long
	Control	Operation	Function
	Dock list	Display	Display a list of dock
	Search field	Input	Take search keyword from user
	Clear button	Click	To clear search keyword and search result
	Search button	Click	To search for a dock that match the given keyword
	Search by button	Click	To choose search mode: by name or by address

- View rent bike info

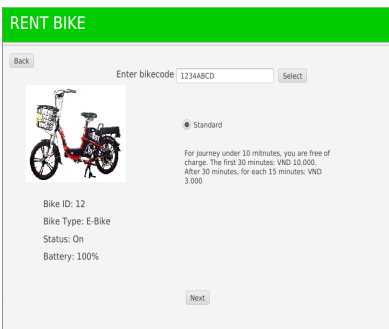
EcoBikeRental	Approved by	Reviewed by	In charge
			Hoàng Mai Đức Long
	Control	Operation	Function
	Bike info area	Display	Area to display bike detailed information in dock.

	Back button	Click	Go back to dock screen
--	-------------	-------	------------------------

c. Dock menu screen

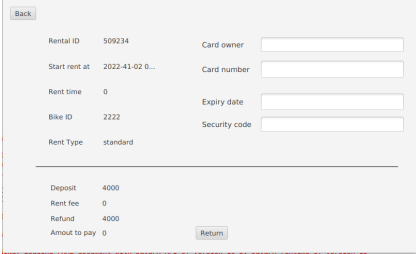
EcoBikeRental	Approved by	Reviewed by	In charge
Dock menu			Hoàng Mai Đức Long
	Control	Operation	Function
	Dock info area	Display	Area to display dock detailed information.
	View bike button	Click	To view bike in the dock.
	Return bike button	Click	To return the bike in your rental to that dock.
	Rent bike button	Click	To rent a bike from that dock.

d. Rent bike screen

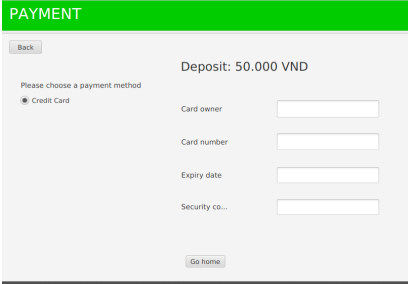
EcoBikeRental	Approved by	Reviewed by	In charge
Rent bike screen			Hoàng Mai Đức Long
	Control	Operation	Function
	Bike Code field	Input	For user to enter the bikecode of the bike to rent
	Rent type area	Click	For user to choose the type of rental
	Confirm button	Click	For user to submit bike code and rent type

	Back button	Click	To back to previous screen
--	-------------	-------	----------------------------

e. Return bike screen

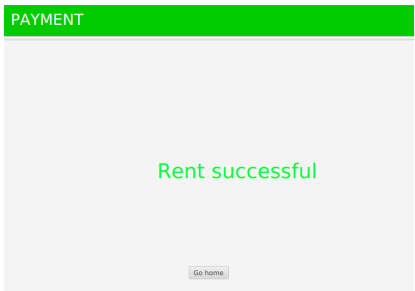
EcoBikeRental	Approved by	Reviewed by	In charge
Return bike screen			Hoàng Mai Đức Long
	Control	Operation	Function
	Rent bike info area	Display	For user to see rent bike info
	Owner field	Input	For user to input credit card owner name
	Card number field	Input	For user to input credit number
	Security code field	Input	For user to input card security code
	Expiry date field	Input	For user to input card's expiry date

f. Payment screen

EcoBikeRental	Approved by	Reviewed by	In charge
Payment screen			Hoàng Mai Đức Long
	Control	Operation	Function
	Payment method area	Click	Display result of the transaction
	Owner field	Input	For user to input credit card owner name
	Card number field	Input	For user to input credit number
	Security code	Input	For user to input card

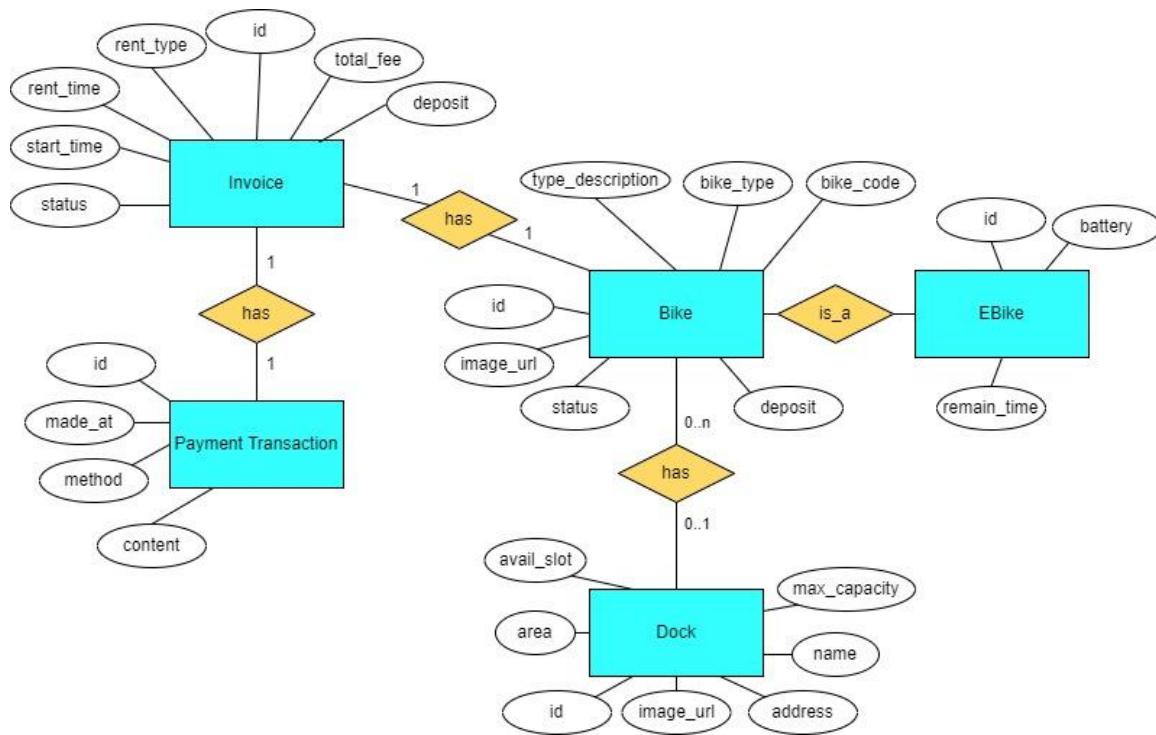
	field		security code
	Expiry date field	Input	For user to input card's expiry date

g. Result screen

EcoBikeRental	Approved by	Reviewed by	In charge
Result screen			Hoàng Mai Đức Long
	Control	Operation	Function
	Result area	Display	Display result of the transaction
	Message	Display	Message of the successful/failed transaction
	Go home button	Click	Go back to home screen

4.2 Data Modeling

4.2.1 Conceptual Data Modeling



- Entities:
 - + Bike: holds information about a bike, including:
 - id: id of the bike.
 - bike_type: type of the bikes: standard single, standard twin, electric single.
 - type_description: description about the frame (e.g. for standard single bike, 1 pair of paddles, 1 saddle).
 - bike_code: a unique of the bike, for the user to enter when renting a bike.
 - image_url: URL to the bike's image.
 - status: the status of the bike, being rented or not.
 - + Ebike: holds additional information about an electric bikes, including:
 - battery: the remaining battery percentage of the ebike.
 - remain_time: estimated remaining operating time of the ebike.
 - + Dock: holds information about a dock, including:
 - id: id of the dock.
 - name: name of the dock.

- address: the address of the dock.
- max_capacity: maximum number of slots of the dock.
- avail_slot: number of bikes available at the dock.
- area: the total area of the dock.
- image_url: URL to the image of the dock.
- + Invoice: holds information about a rental, including:
 - id: the id of the invoice.
 - rent_type: the type of rental, right now there is only 1 type of rental, which is 'standard rental'.
 - start_time: the datetime when the user starts renting a bike.
 - total_time: the total amount of time the user has rented the bike.
 - total_fee: the total amount of rental fee.
 - deposit: the amount of money that the user has made to deposit to rent a bike.
 - status: indicate if the rental is in-rental or have ended.
- + PaymentTransaction: holds information about a transaction made on the system, including:
 - id: the id of the transaction.
 - made_at: the datetime when the transaction was made.
 - method: the transaction method used to make the transaction.
 - content: a message about the transaction.
- Relationships:
 - + Ebike is a Bike: Ebike is a specific type of general bike, each Ebike corresponds to only 1 bike.
 - + Dock has Bike:
 - A dock can have many bikes or no bikes (all bikes being rented).
 - A bike can belong to 1 dock (when stationed) or not belong to any dock (when that bike is being rented).
 - + Invoice has Bike:
 - An invoice can only have 1 bike.
 - A bike can be rented in many invoices (not simultaneously) .
 - + PaymentTransaction has Invoice:
 - A payment transaction corresponds to only 1 invoice.

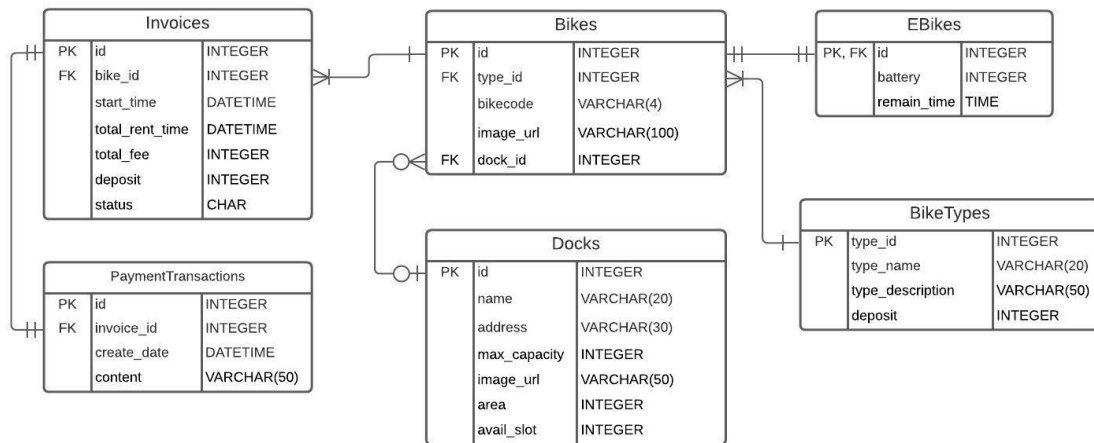
4.2.2 Database Design

4.2.2.1 Database Management System

The chosen DBMS: PostgreSQL

PostgreSQL is an open-sourced Database Management System. PostgreSQL is user-friendly, supports a lot of SQL concepts such as views, materialized views, role's privileges,... and comes with many useful extension/plugins (e.g. pgAgent for scheduling/cron job)

4.2.2.2 Database Diagram



- E Bike is a Bike: E Bike has id as primary key and at the same time references to bike's id.
- To avoid a lot of data duplication, BikeTypes table is made, to hold information about a specific bike type. A bike corresponds to 1 bike type.
- Bike has a nullable dock_id field which references the id of docks, to represent 'zero or 1' to 'zero or many' relationships.
- Invoice has a Bike: Invoice has bike_id field references to the id of bikes.
- Transaction has an Invoice: transaction has invoice_id field to references to the id of invoices.

4.2.2.3 Database Detail Design

Table 1. Docks

#	PK	FK	Column name	Data type	Default value	Mandatory	Description
1	x		id	integer		yes	auto increment
2			name	varchar(20)		yes	
3			address	varchar(30)		yes	
4			max_capacity	integer		yes	
5			image_url	varchar(50)		yes	
6			area	integer		yes	

7			avail_slot	integer		yes	
---	--	--	------------	---------	--	-----	--

Table 2. Biketypes

#	PK	FK	Column name	Data type	Default value	Mandatory	Description
1	x		type_id	integer		yes	
2			type_name	varchar(20)		yes	
3			type_description	varchar(50)		yes	
4			deposit	integer		yes	

Table 3. Bikes

#	PK	FK	Column name	Data type	Default value	Mandatory	Description
1	x		id	integer		yes	
2		x	type_id	integer		yes	
3			bike_code	varchar(4)		yes	
4			image_url	varchar(50)		yes	
5		x	dock_id	integer		no	

Table 4. Ebikes

#	PK	FK	Column name	Data type	Default value	Mandatory	Description
1	x	x	id	integer		yes	
2			battery	integer		yes	
3			remain_time	timestamp		yes	

Table 5. Invoices

#	PK	FK	Column name	Data type	Default value	Mandatory	Description
1	x		id	integer		yes	

2		x	bike_id	integer		yes	
3			start_time	timestamp		yes	
4			total_rent_time	integer		yes	in minutes
5			total_fee	integer		yes	
6			deposit	integer		yes	
7			status	char(1)		yes	

Table 6. PaymentTransaction

#	PK	FK	Column name	Data type	Default value	Mandatory	Description
1	x		id	integer		yes	
2		x	invoice_id	integer		yes	
3			create_date	timestamp		yes	
4			content	varchar(50)		yes	

4.3 Non-Database Management System Files

<Provide the detailed description of all non-DBMS files if any and include a narrative description of the usage of each file that identifies if the file is used for input, output, or both, and if the file is a temporary file. Also provide an indication of which modules read and write the file and include file structures (refer to the data dictionary). As appropriate, the file structure information should include the following:

- *Record structures, record keys or indexes, and data elements referenced within the records*
- *Record length (fixed or maximum variable length) and blocking factors*
- *Access method (e.g., index sequential, virtual sequential, random access, etc.)*
- *Estimate of the file size or volume of data within the file, including overhead resulting from file access methods*
- *Definition of the update frequency of the file (If the file is part of an online transaction-based system, provide the estimated number of transactions per unit of time, and the statistical mean, mode, and distribution of those transactions.)*

- *Backup and recovery specifications*>

4.4 Class Design

4.4.1 General Class Diagram

<General class diagram which shows the whole class diagram of the software. This diagram may have packages, subsystems and classes. Classes in this diagram may not have all attributes and operations>

4.4.2 Class Diagrams

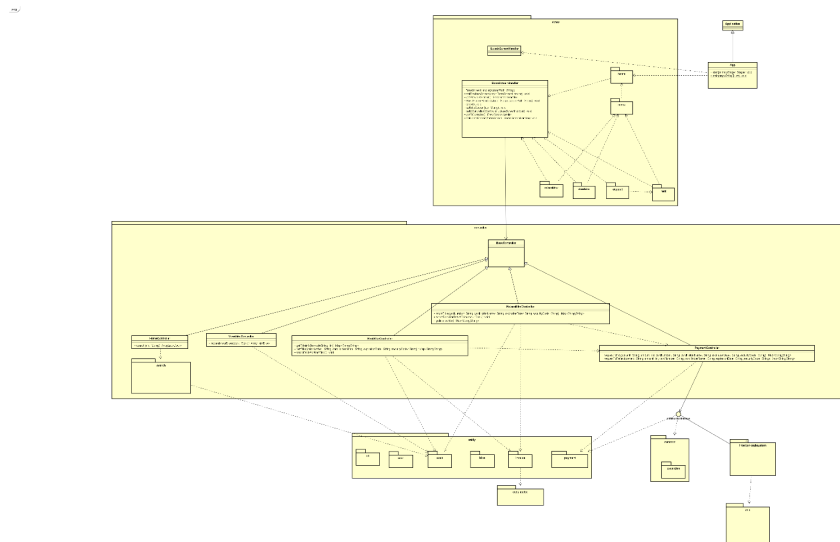
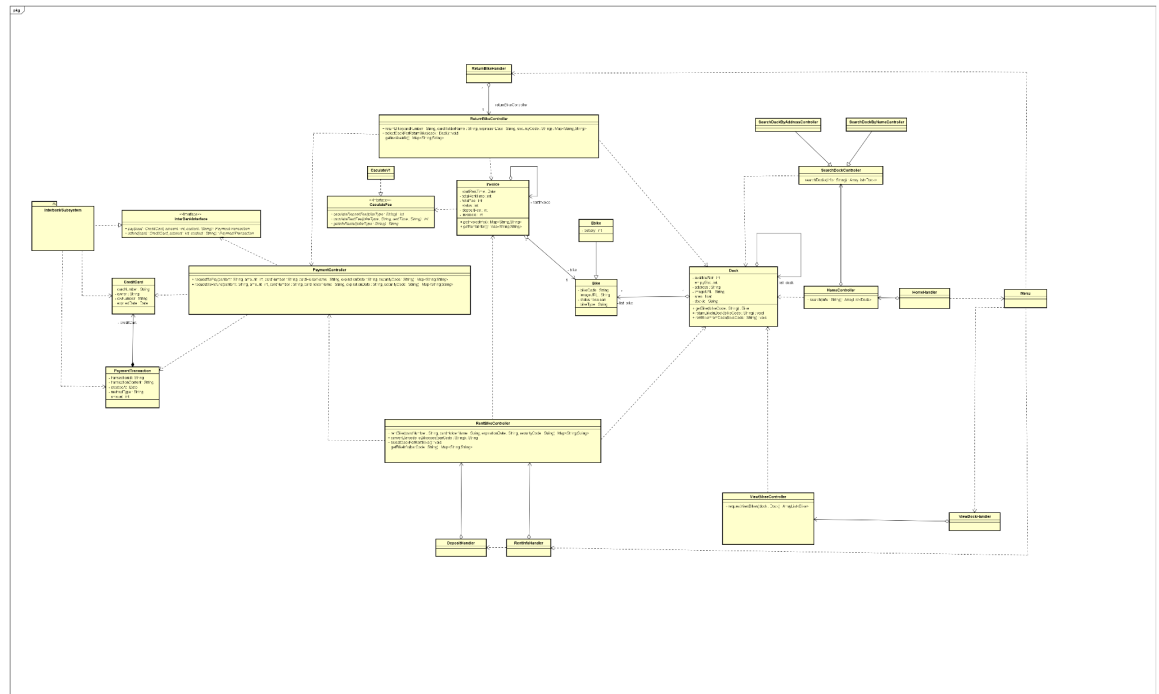
<Detail class diagram with full attributes and operations>

4.4.2.1 Class Diagram for Package A

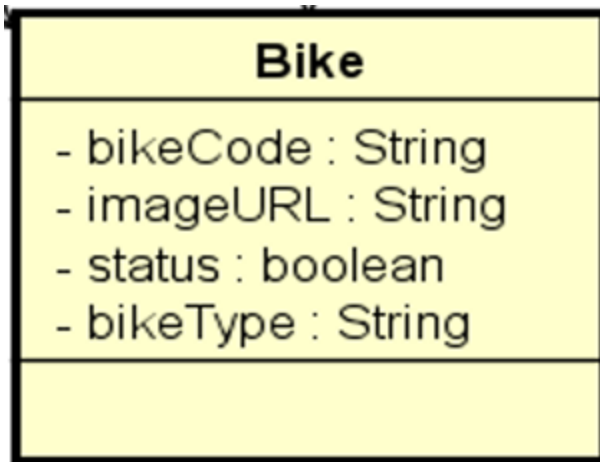
4.4.2.2 Class Diagram for Subsystem B

...

4.4.3 Class Design



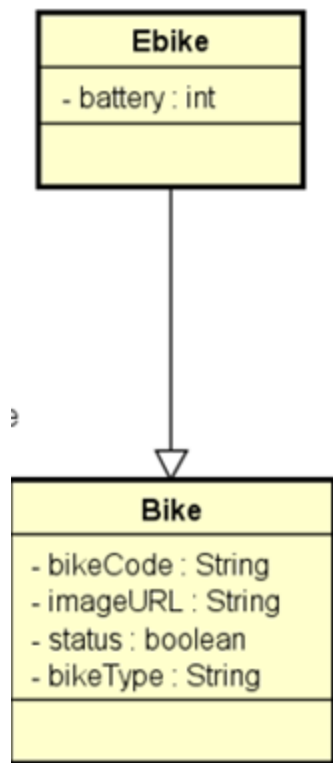
1. Class Bike



Attribute:

#	Name	Data type	Default value	Description
1	bikeType	String	""	Type of bike
2	bikeCode	String	""	Bike id
3	status	boolean	false	Is the car available?
4	imageUrl	String	""	where the pictures of the bike are stored

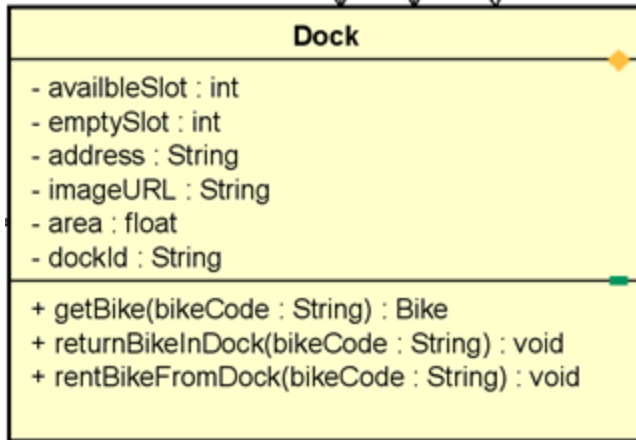
2. Class Ebike



Attribute:

#	Name	Data type	Default value	Description
1	battery	int	100	remaining battery of the electric bike

3. Class Dock



Attribute:

#	Name	Data type	Default value	Description
1	availableSlot	int	0	the number of bikes in the dock
2	emptySlot	int	0	number of the empty slots in the dock
3	address	String	""	Dock address
4	area	float	0	Dock area
5	dockId	String	""	Dock Id
6	imageURL	String	""	where the pictures of the are stored
7	dock_list	ArrayList<Dock>	null	list of all dock

#	Name	Return type	Description
1	rentBikeFromDock	None	Rent bike from the docck
2	returnBikeInDock	None	Return bike in the dock
3	getBike	Bike	Get insta bike from the dock

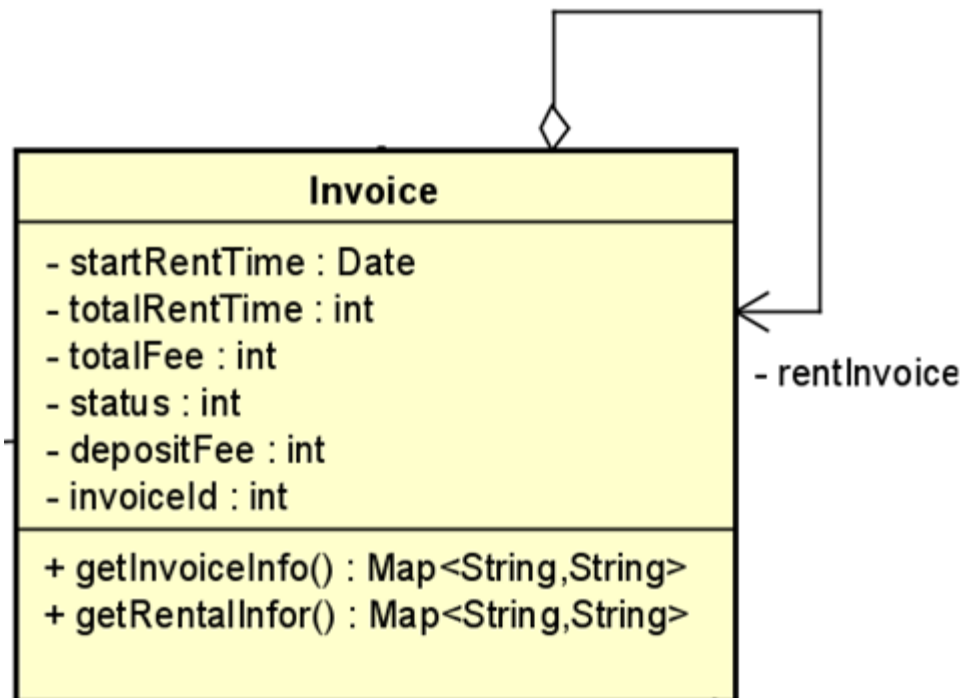
Parameter:

bikeCode : information to find the bike

Exception: None

Method: None

4. Class Invoice



Attribute:

#	Name	Data type	Default value	Description
1	startRentTime	Date	null	bike rental start time
2	totalRentTime	int	0	Bike rental total time
3	totalFee	int	0	Bike rental total fee
4	status	int	0	0: when the customer doesn't rent a bike 1: when the customer is renting a bike 2: when the customer returned bike

5	depositFee	int	0	When renting a bike, customers must advance money
6	invoiceId	String	""	Invoice id

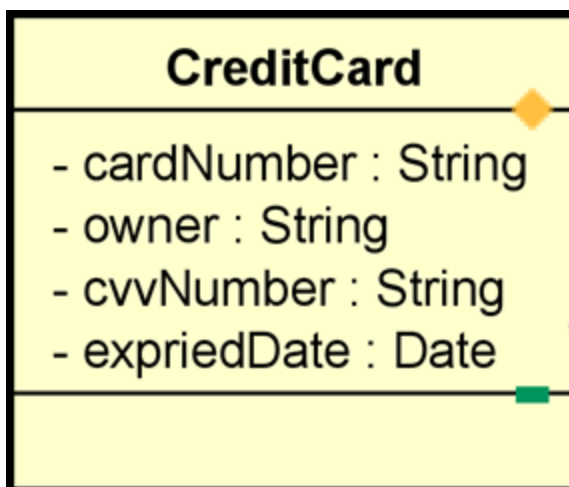
#	Name	Return type	Description
1	getInvoiceInfo	Map<String,String>	get invoice information
2	getRentalInfo	Map<String,String>	get related information about the calculation money

Parameter:None

Exception:None

Method:

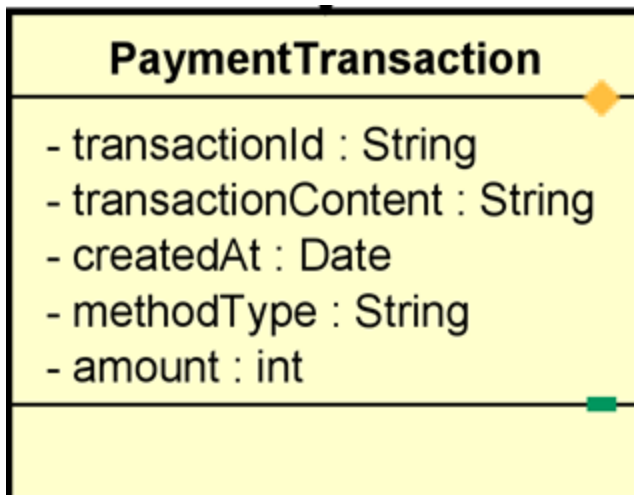
5. Class CreditCard



Attribute:

#	Name	Data type	Default value	Description
1	carNumber	String	""	Card number
2	owner	String	""	Cardholder
3	cvvNumber	String	""	Security code
4	expriedDate	Date	null	Expried date

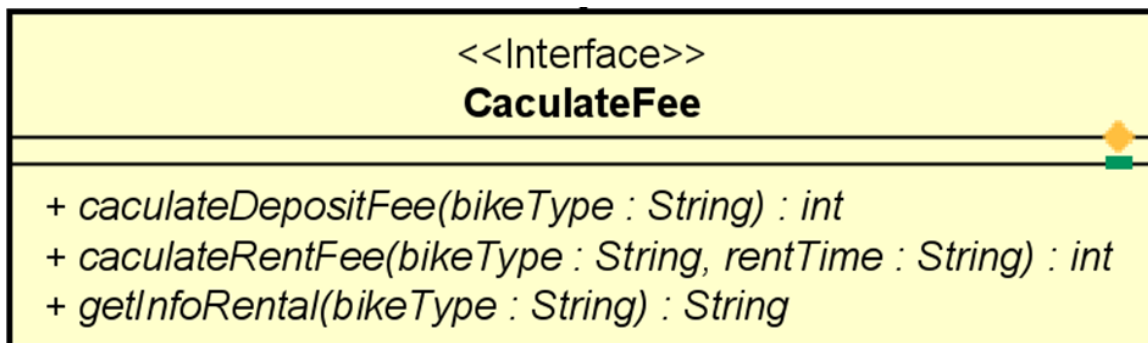
6. Class PaymentTransaction



Attribute:

#	Name	Data type	Default value	Description
1	transactionID	String	""	Transaction id
2	transactionContent	String	""	transaction notes
3	createAt	Date	null	Transaction execution time
4	methosType	String	""	"pay" or "refund"
5	amount	int	0	total transaction amount
6	card	CreditCard	null	Bank card

1. Interface CaculateFee



Attribute:

Operation:

#	Name	Return type	Description
1	calculateDepositFee	int	deposit fee
2	calculateRentFee	int	the amount paid when returning the car
3	getInfoRental	String	information related to bike rental

Parameter:

bikeType : type of bike

rentTime : bike rental time

Exception:None

Method:None

1. Class Rent Bike Controller

RentBikeController
+ rentBike(cardNumber : String, cardHolderName : String, expirationDate : String, securityCode : String) : Map<String,String> + convertBarcodeToBikecode(barCode : String) : String + selectDockForRentBike() : void + getBikeInfo(barCode : String) : Map<String,String>

Attribute:

#	Name	Data type	Default value	Description
1	invoice	Invoice	null	
2	curentDock	Dock	null	

Operation:

#	Name	Return type	Description
1	rentBike	Map<String,String>	Rent bike
2	selectDockForRentBike	Void	Choose a dock to rent a car
3	getBikeInfor	Map<String,String>	get information about rental bike

Parameter:

Exception:

Method:convertBarcodeToCardcode (Convert from barcode to cardcode)

7. Class ReturnBBikeController

ReturnBikeController	
<pre>+ returnBike(cardNumber : String, cardHolderName : String, expirationDate : String, securityCode : String) : Map<String,String> + selectDockForReturnBike(dock : Dock) : void + getInvoiceInfo() : Map<String,String></pre>	

Attribute:

#	Name	Data type	Default value	Description
1	invoice	Invoice	null	
2	curentDock	Dock	null	

Operation:

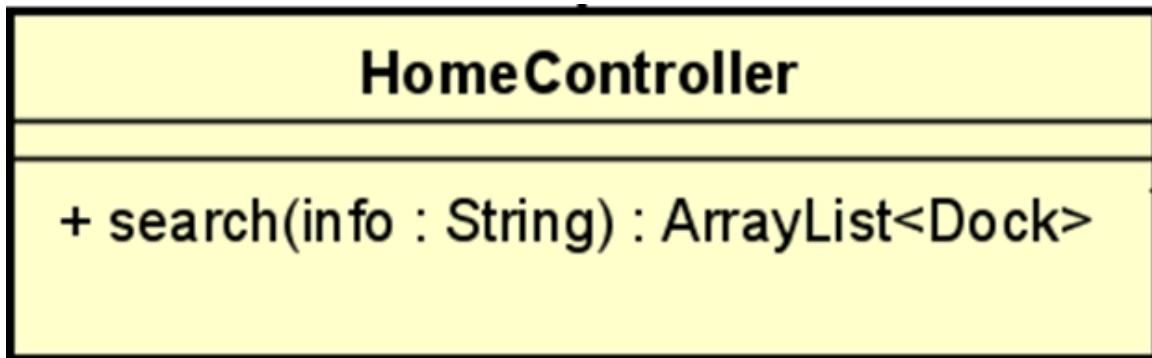
#	Name	Return type	Description
1	returnBike		
2	selectDockForReturnBike		
3	getInvoiceInfo		

Parameter:

Exception:

Method:

8. Class HomeController



Attribute:

Operation:

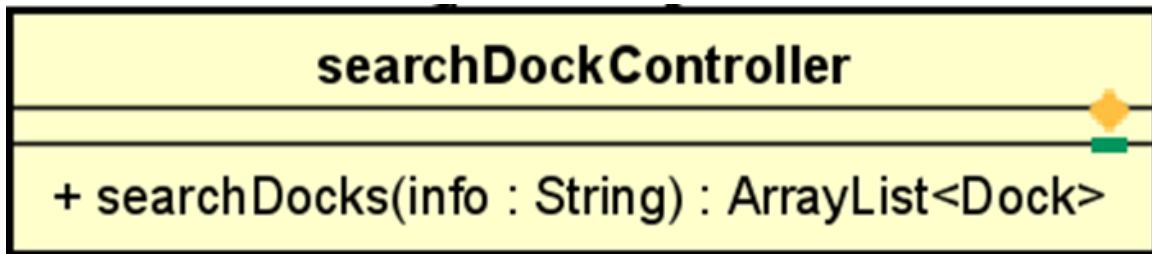
#	Name	Return type	Description
1	search	ArrayList<Dock >	search parking

Parameter: String info (information to search dock)

Exception:

Method:

1. Class SearchDockController



Attribute:

Operation:

#	Name	Return type	Description
1	searchDock	ArrayList<Dock>	Search docks

Parameter: String info (information to search parking)

Exception:

Method:

5 Design Considerations

5.1 Goals and Guidelines

Goals:

- To bring great experiences to the user.
- Easy to use and user-friendly.

Guidelines:

- Follow a unified coding convention, and OOP convention.
- Well documentation for better maintenance.

5.2 Architectural Strategies

Our design decisions focus on reusing components, unified system following

- + Programming Language: java
- + Database: PostgreSQL
- + Unified on error detection and recovery

We always aim to save memory and spaces, also speed up response time and nice looking. In the future, we plan to extend software: have a site for admin to add, delete bikes, statistics, business strategies. These targets make us concentrate totally on architectural design.

5.3 Coupling and Cohesion

The design is at Control Coupling level, as there are methods that behave differently depending on external outcome. For example,

5.4 Design Principles

The design is meant to follow SOLID principles. Still, there exists some part which violates the O principle. As 1 example, BikeFactory is currently being hard-coded to recognize to type of bikes

```
public BikeFactory() {  
    register("standard", new Bike());  
    register("electric", new Ebike());  
}
```

5.5 *Design Patterns*

- Factory:

A problem arises when creating a bike instance from the database query. If the bike is of type electric, the bike needs to set additional data which is battery and remain_time. To avoid if-else statements, a BikeFactory is implemented as singleton pattern.