HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology

# Software Design Document

Version 1.0

# EcoBikeRental

## Subject: IT Software Development

**Group 10**

Vu Trung Dung

Nguyen Xuan Hoang

Nguyen Trung Nghia

Nguyen Ngoc Quy

*Hanoi, 11/2020*

## Table of Contents

## List of Figures

No table of figures entries found.

## List of Tables

No table of figures entries found.

# 1 Introduction

## *1.1 Objective*

The objective of the document is to describe the requirements for EcoBikeRental Software. The goal is to have the EcoBikeRental Software requirements specification which is usable for the EcoBikeRental Software Design.

The document describes the potential users, domains and user-studies for EcoBikeRental Software. The document contains also EcoBikeRental Software conceptual model (as UML class diagrams), functional requirements (as UML use-case model and usage scenarios), and non-functional requirements in the level of details required for the first sprints. Thus, the requirements specification covers full-functionality in the low details, and the usage scenarios for the first sprints have been described in detail.

## *1.2 Scope*

This software system will be a Eco Park Bike Rental System for everyone including novice users to use without any training. This system will be designed to allow for approximately 100 average concurrent users with no perceivable performance difference and can be operated upto 200 hours continuously. The system is also very responsive with typical response time around 1 second and only requires 2 hours of downtime for maintenance.

## *1.3 Glossary*

| Term | Definition |
|---|---|
| User | Main actor of the system |
| Map | The entire area of Eco Park, with detailed location of all docking stations |
| Docking station | The area to store all bikes available to the user |
| E-bike | Standard bike with an integrated electric motor for assisted propulsion |
| Twin bike | Standard bike with 2 saddles, 2 pedal and no electric motor |

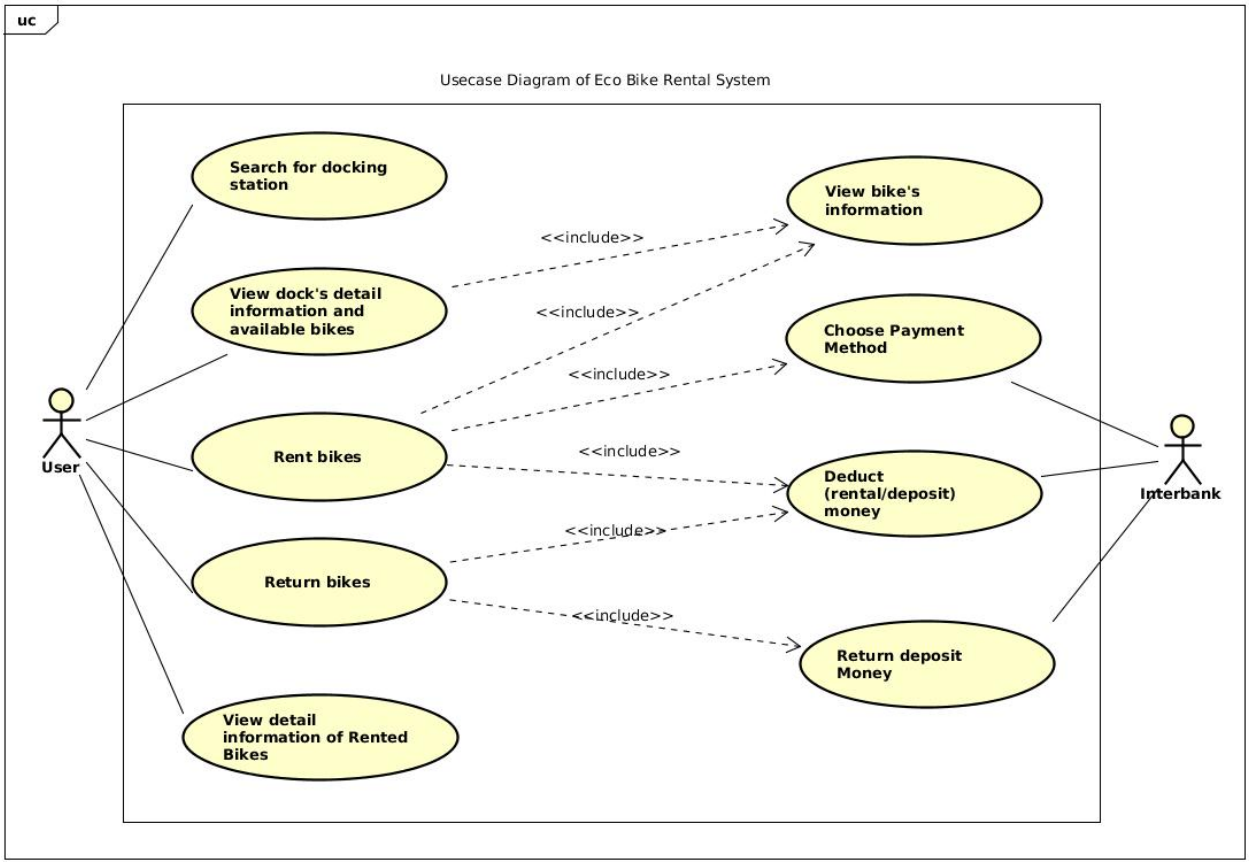
## *1.4 References*


- IEEE. IEEE Std 1016-2009 IEEE Standard for Information Technology—Systems Design—Software Design Descriptions. IEEE Computer Society, 2009

# 2 Overall Description

## *2.1 General Overview*

EcoBikeRental Software allows for interaction between 2 main actors: the Customer and the Interbank, across a variety of use cases




## *2.2 Assumptions/Constraints/Risks*

### 2.2.1 Assumptions

The software assumes each client device to be equipped with a GPS-capable mobile device, connected to the internet for the duration of rental service, legibility with at least one supported interbank for the payment process.

### 2.2.2 Constraints

- For the time being, each user must have their own client installed and configured with their own payment card
- The software must be online at all times to ensure all bike and dock station status
- Users must agree to the terms and conditions about location privacy concerns

### 2.2.3 Risks

The software currently has no protection against attacks via direct contact with the client software due to no implemented features surrounding account based authentication

# 3 System Architecture and Architecture Design

## *3.1 Architectural Patterns*

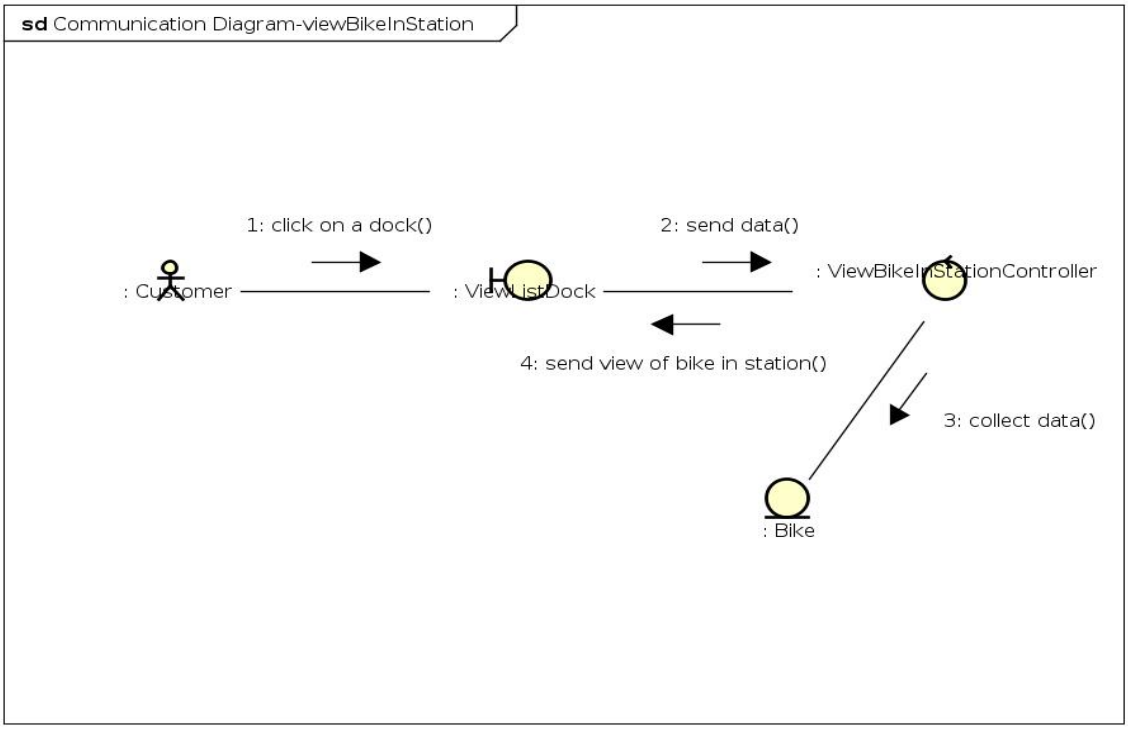
This architectural pattern is created following the MVC-model in order to build a system for renting bikes in our Ecopark residential. Each part of the architectural pattern normally contains a controller to process all the business requirements inside.

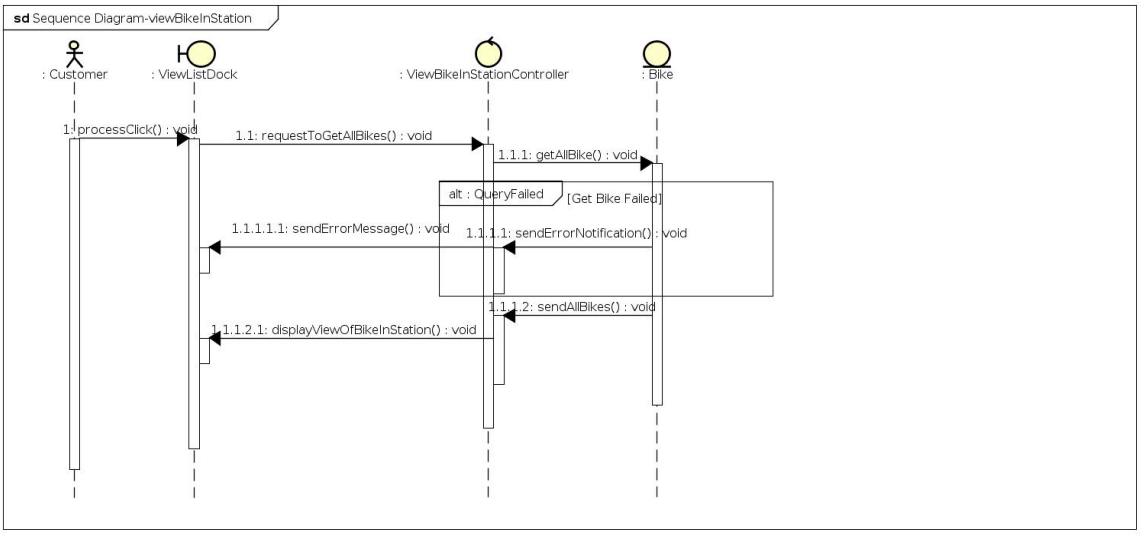
We choose this kind of architectural patterns because it's simple, light, and can be scalable for this project and also MVC is a very well-known design pattern that developers normally use for this kind of project

## *3.2 Interaction Diagrams*

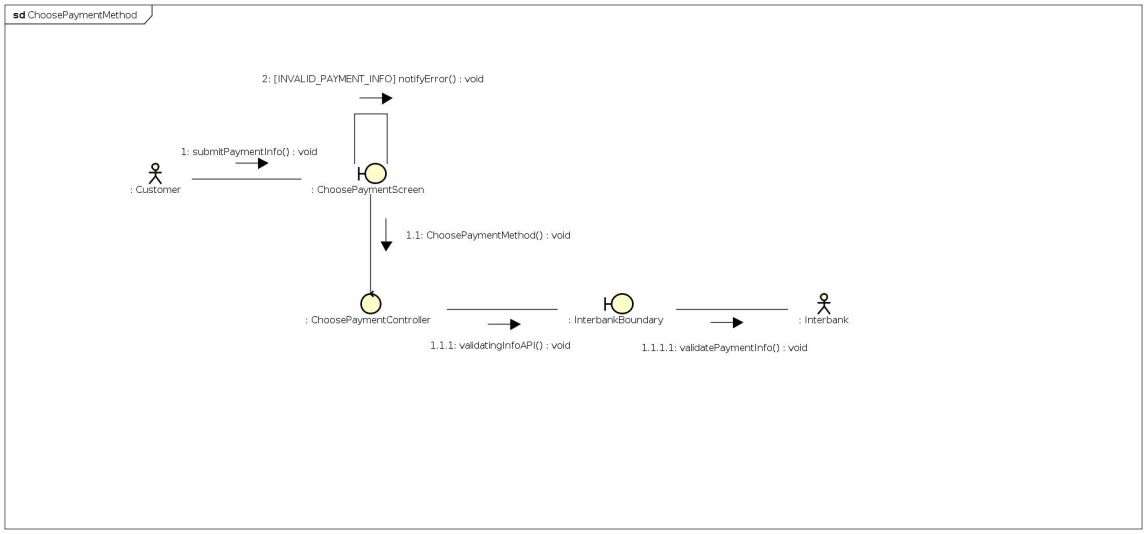
Sequence diagrams and communication diagram

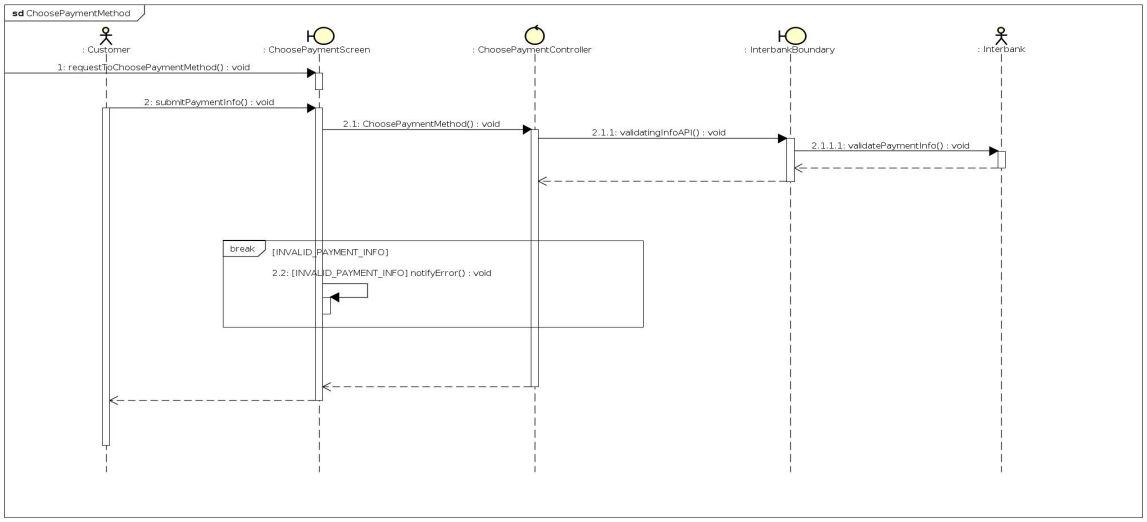
*viewBikeInStation sequence diagram + communication diagram*



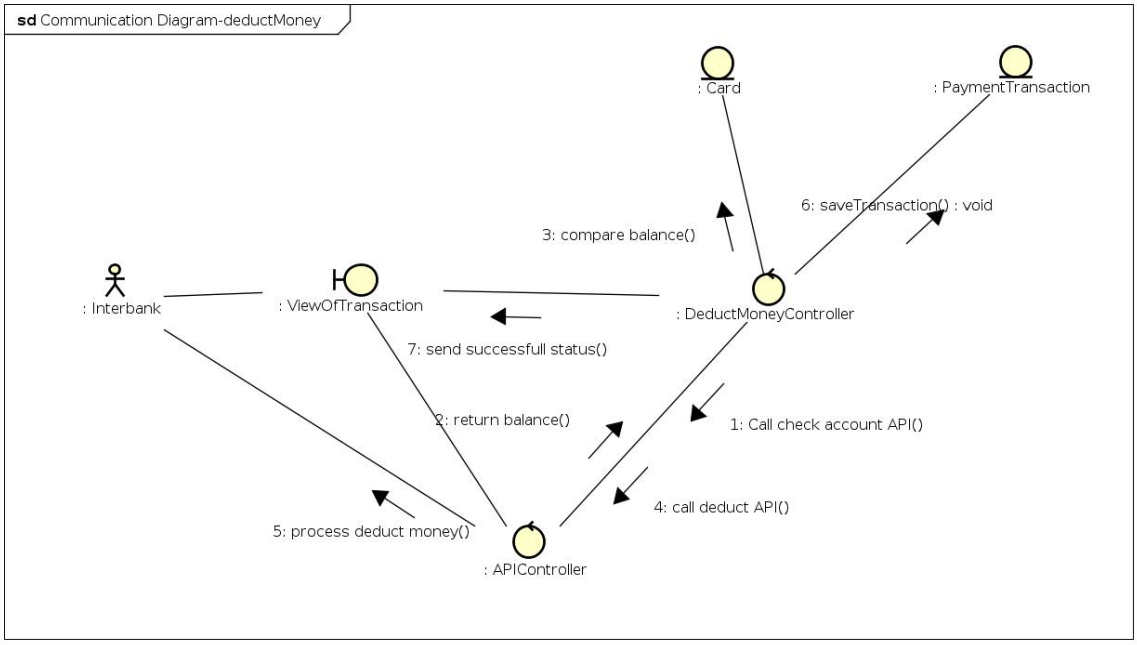



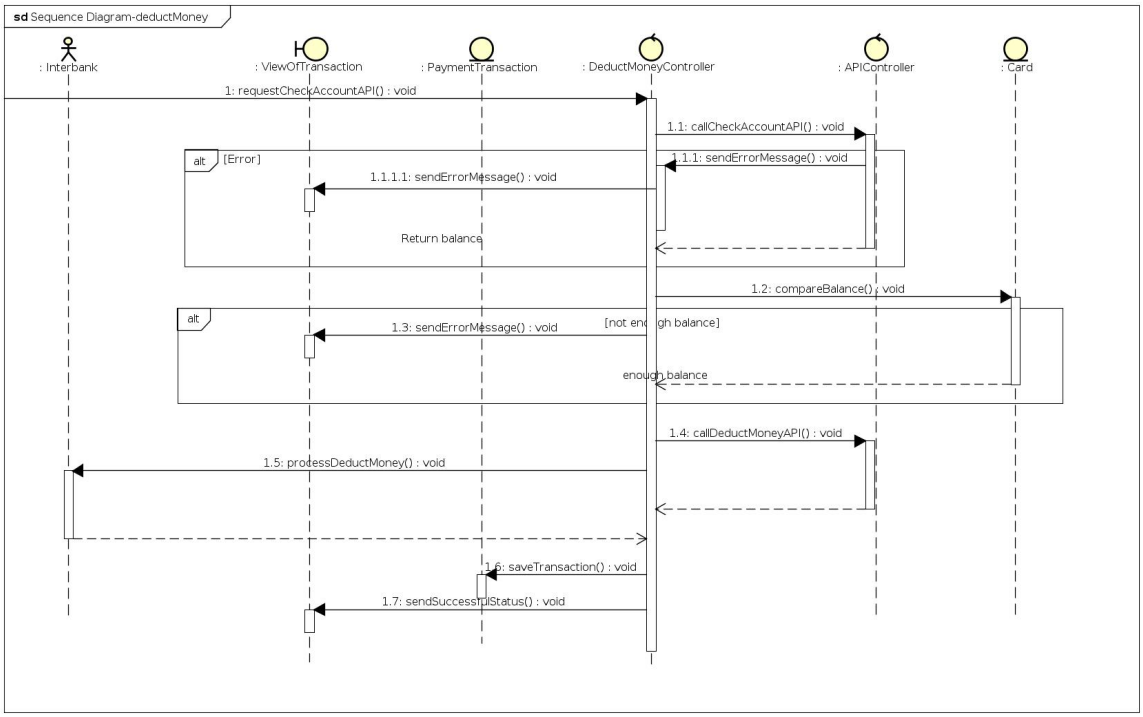
*choosePaymentMethod sequence diagram + communication diagram*



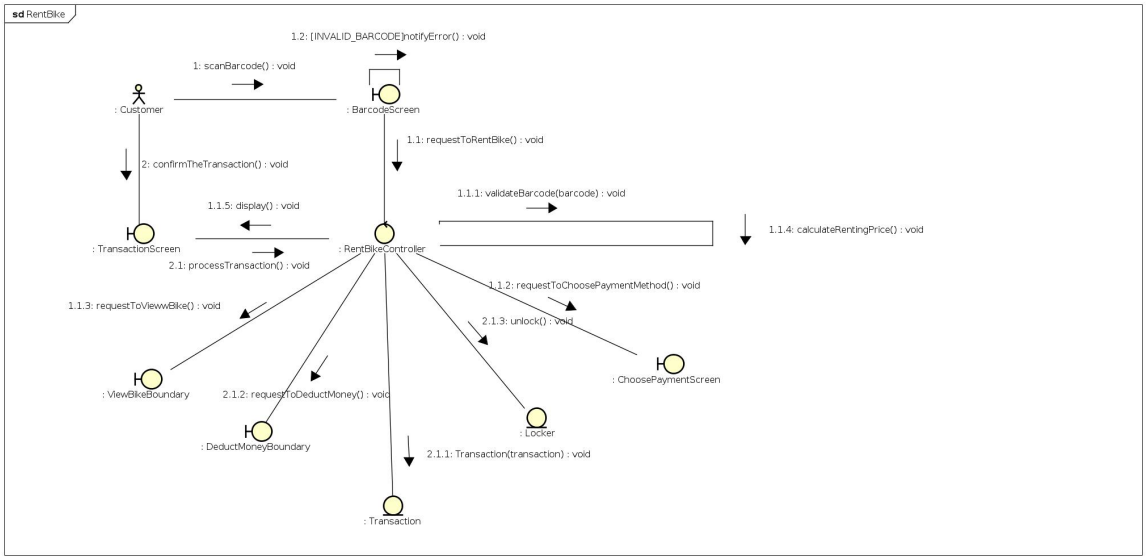



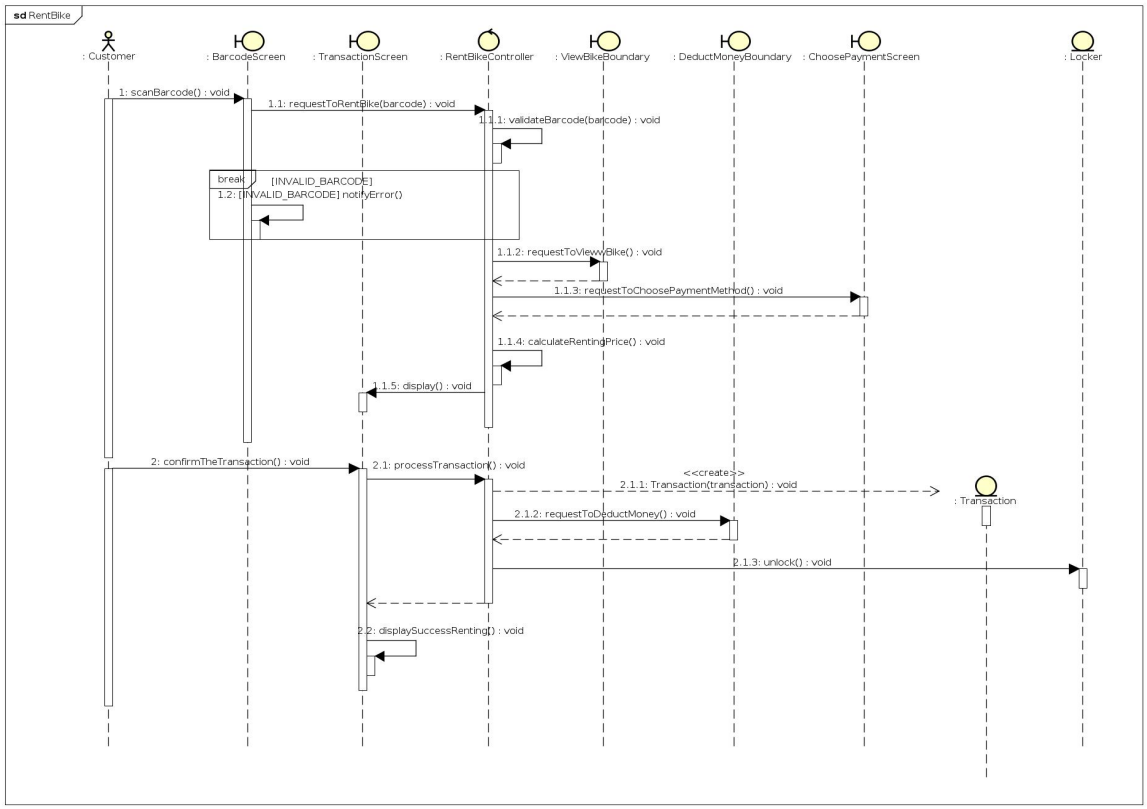
*deductMoney sequence diagram + communication diagram*



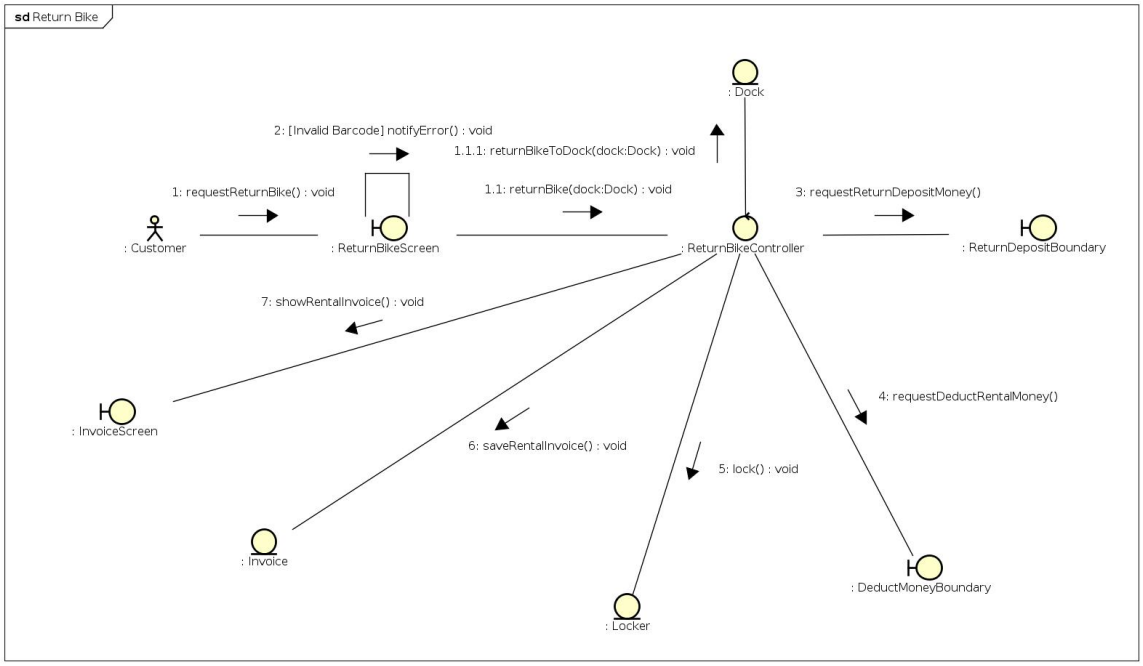



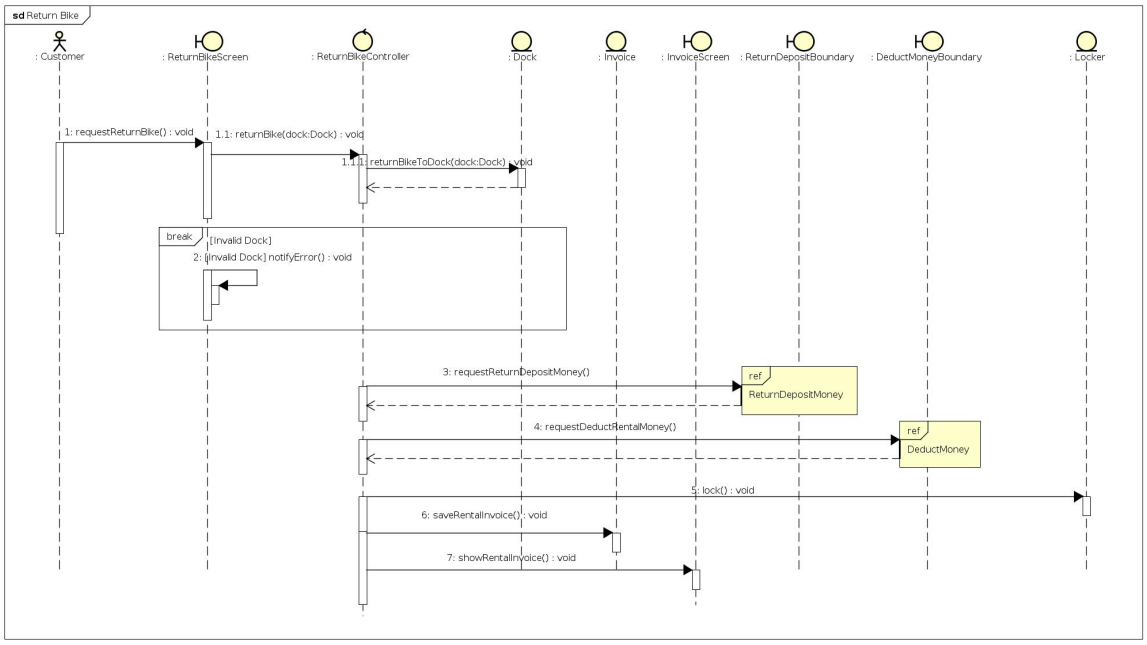

*rentBike sequence diagram + communication diagram*



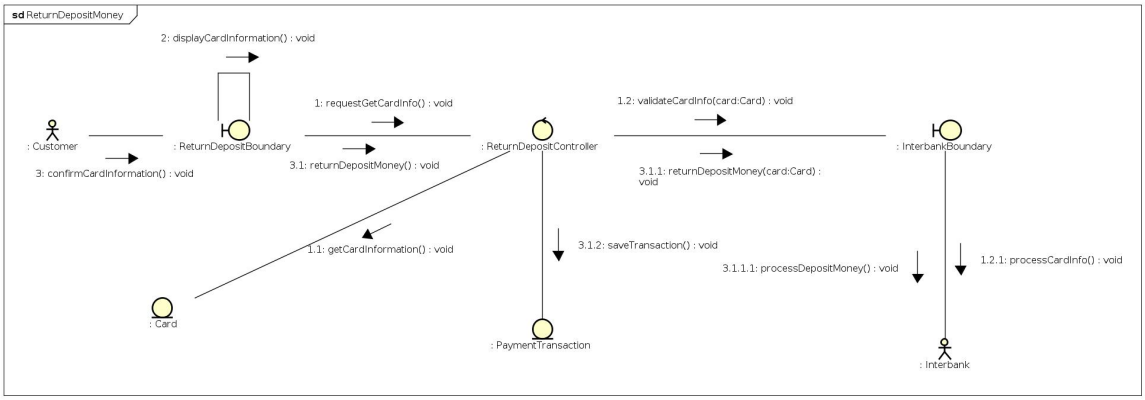



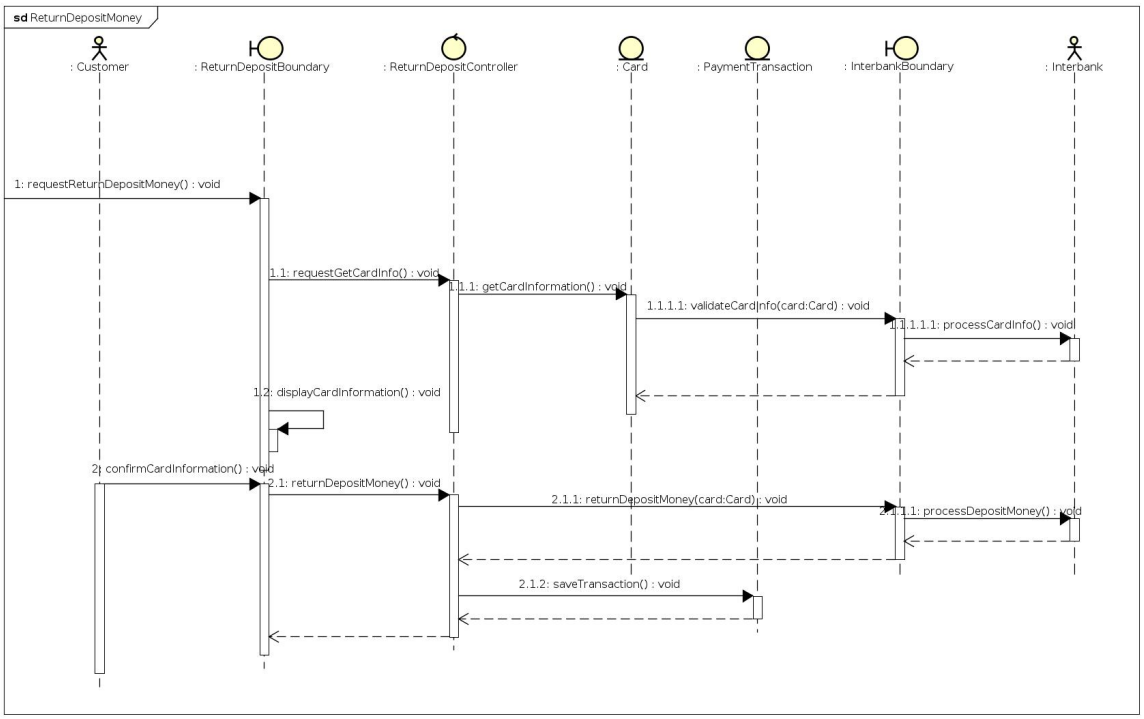

*returnBike sequence diagram + communication diagram*



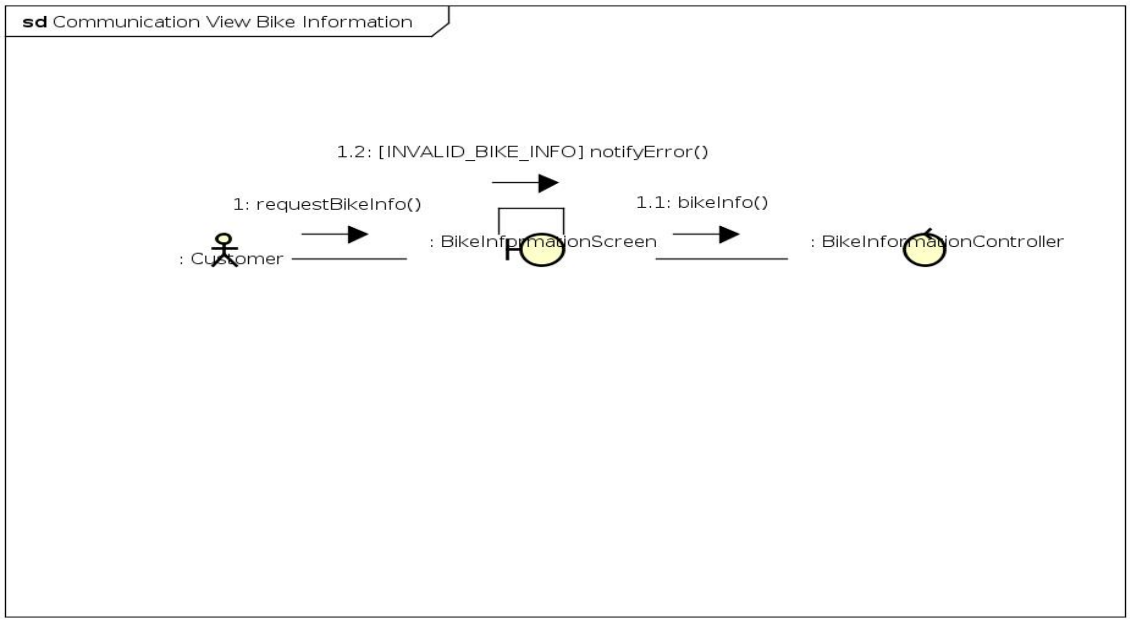



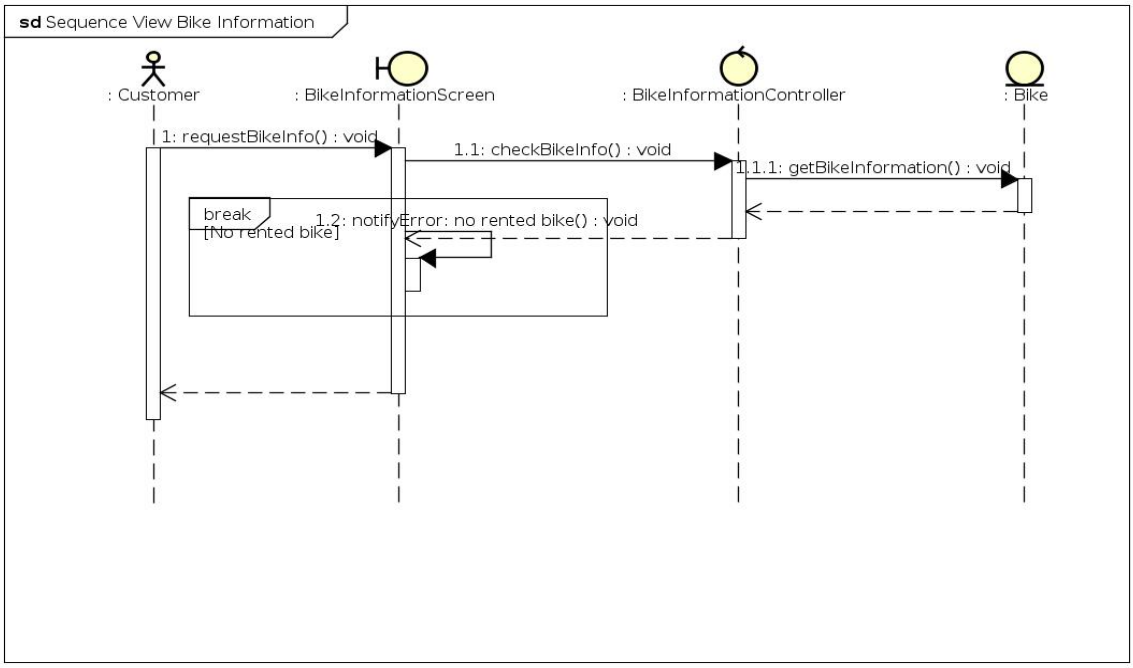

*returnDepositMoney sequence diagram + communication diagram*




*viewBikeInformation sequence diagram + communication diagram*


sd Communication View Bike Information
1.2: [INVALID_BIKE_INFO] notifyError()
1: requestBikeInfo()
1.1: bikeInfo()
: Customer
: BikeInformationScreen
: BikeInformationController


sd Sequence View Bike Information
: Customer
: BikeInformationScreen
: BikeInformationController
: Bike
1: requestBikeInfo() : void
1.1: checkBikeInfo() : void
1.1.1: getBikeInformation() : void
break
[No rented bike]
1.2: notifyError: no rented bike() : void

## 3.3 Analysis Class Diagrams

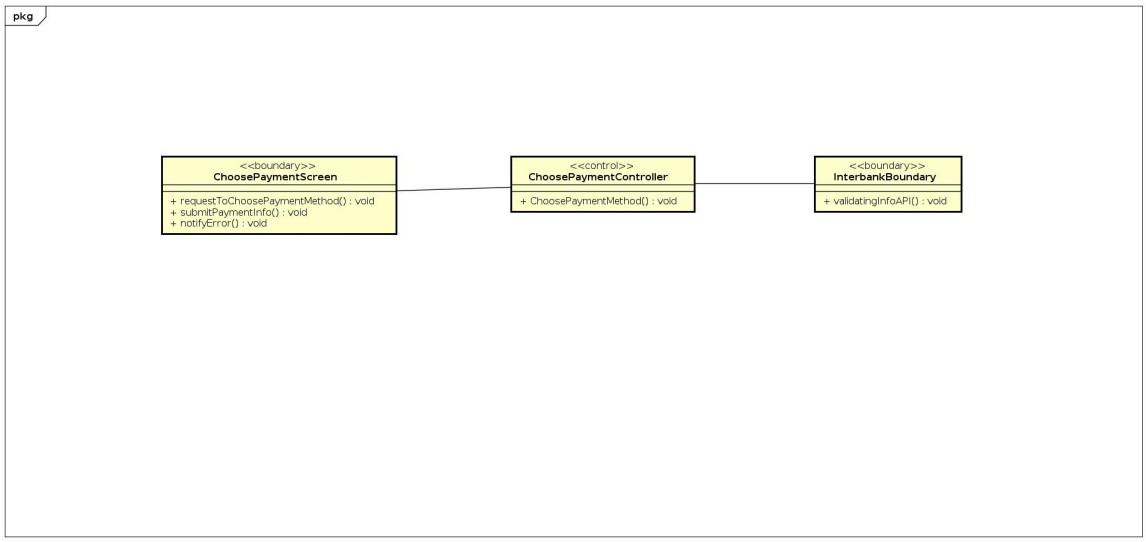



*figure 1: choosePaymentMethod class diagram*

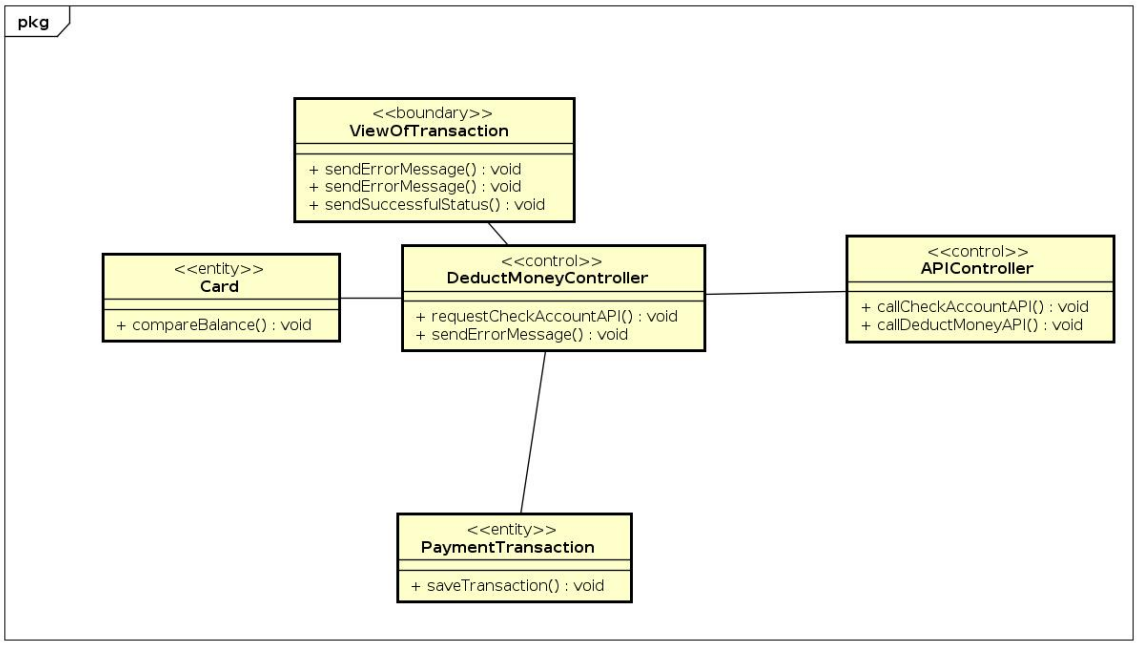



*figure 2: deductMoney class diagram*

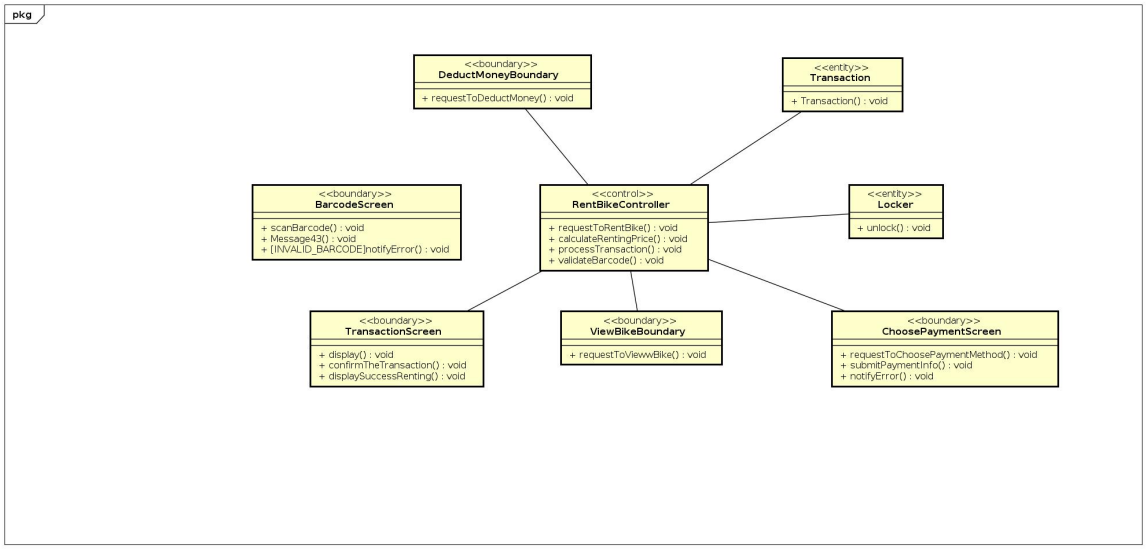



*figure 3: rentBike class diagram*

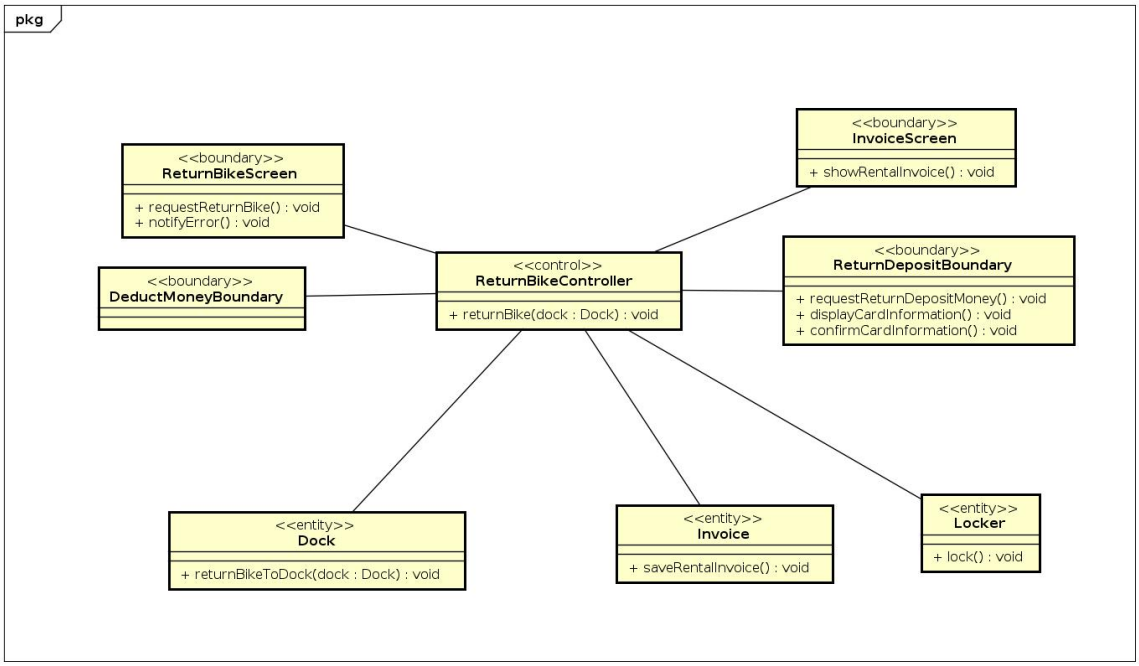



*figure 4: returnBike class diagram*

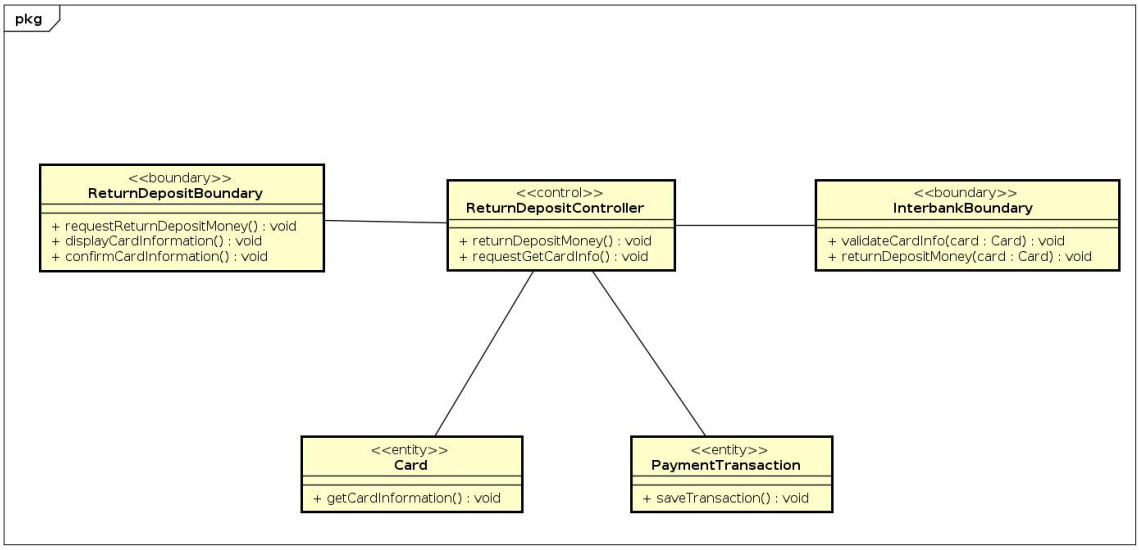



*figure 5: returnDepositMoney class diagram*

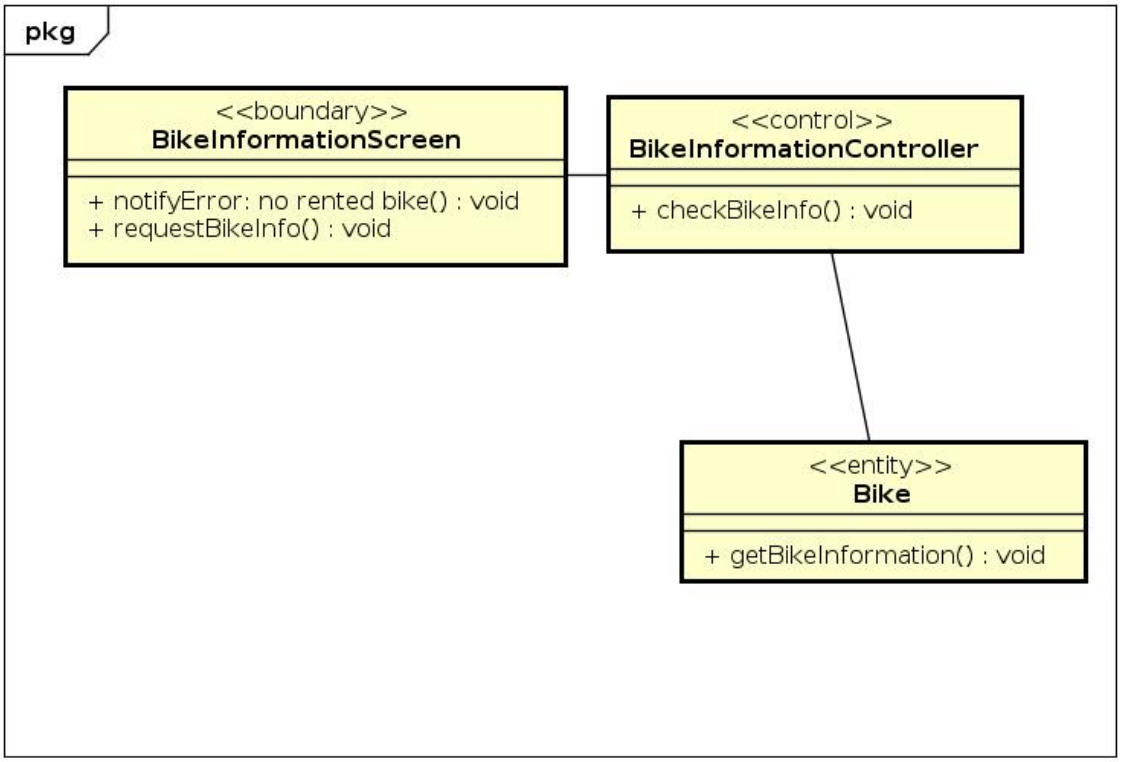



*figure 6: viewBikeInformation class diagram*

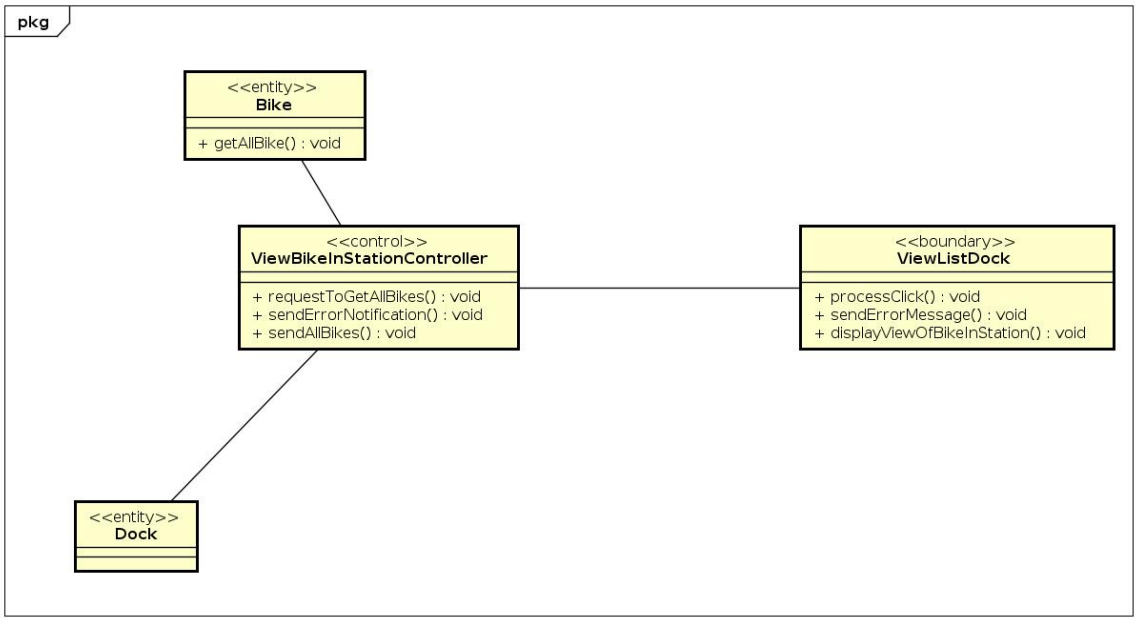
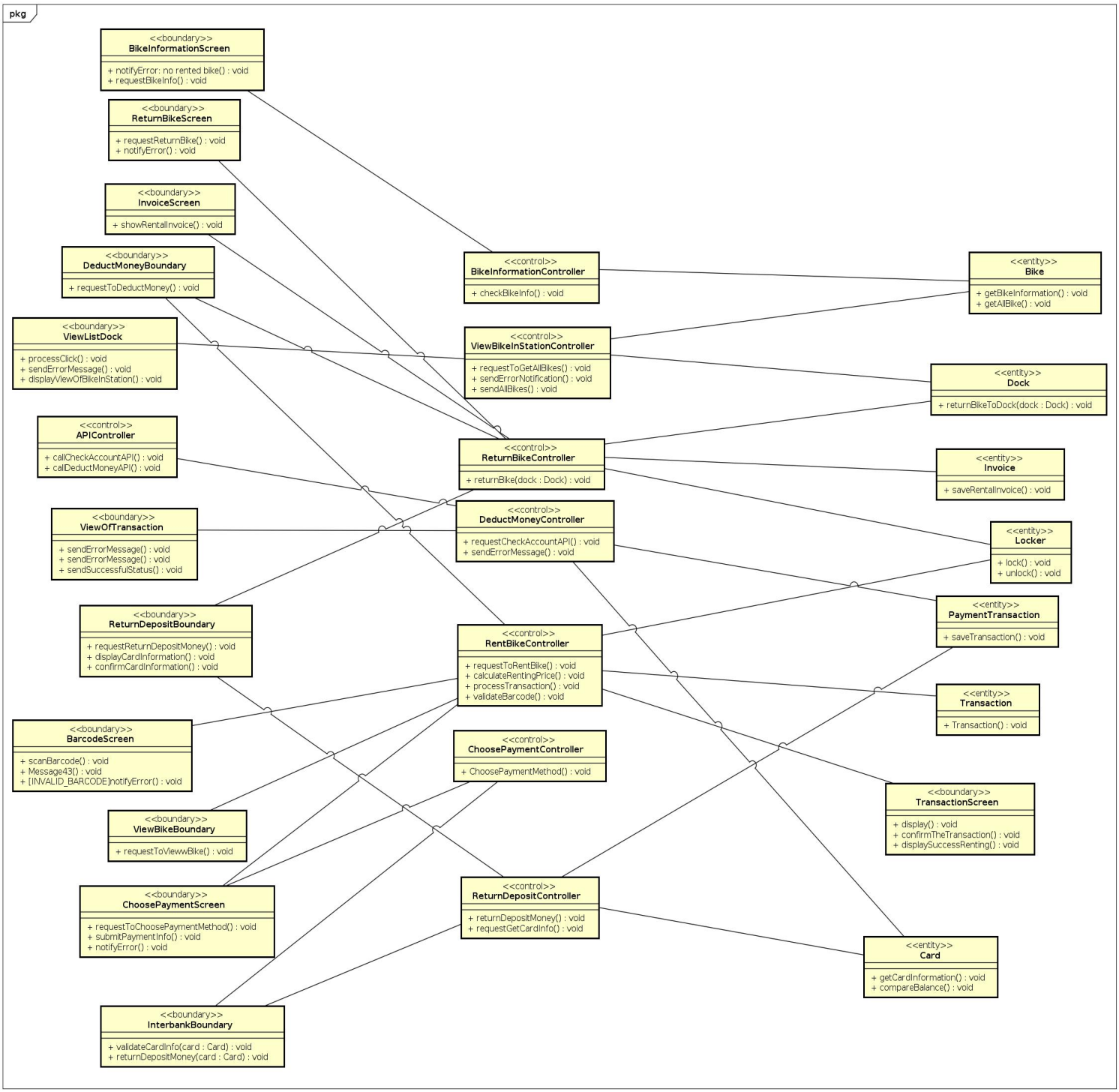



*figure 7: viewBikeInStaion class diagram*

## 3.4 Unified Analysis Class Diagram




## 3.5 Security Software Architecture

*<Describe the software components and configuration supporting the security and privacy of the system. Specify the architecture for (1) authentication to validate user identity before allowing access to the system;(2) authorization of users to perform*

*functional activity once logged into the system, (3) encryption protocol to support the business risks and the nature of information, and (4) logging and auditing design, if required.>*

# 4 Detailed Design

## *4.1 User Interface Design*

*<Suppose that you design a Graphical User Interface (GUI)>*

### 4.1.1 Screen Configuration Standardization

#### *4.1.1.1 Screen Configuration Standardizations Display*

Number of colors supported: 16,777,216 colors

Resolution: 1792 x 828 - Phone Resolution

#### *4.1.1.2 Screen*

Location of standard buttons: At the bottom (vertically) and in the middle (horizontally) of the frame

Location of the messages: Starting from the top vertically and in the middle horizontally of the frame down to the bottom.

Display of the screen title: The title is located at the top of the frame in the middle.

Consistency in expression of alphanumeric numbers: comma for separator of thousand while strings only consist of characters, digits, commas, dots, spaces, underscores, and hyphen symbol.

#### *4.1.1.3 Control*

Size of the text: medium size (mostly 24px). Font: Material UI. Color: #000000

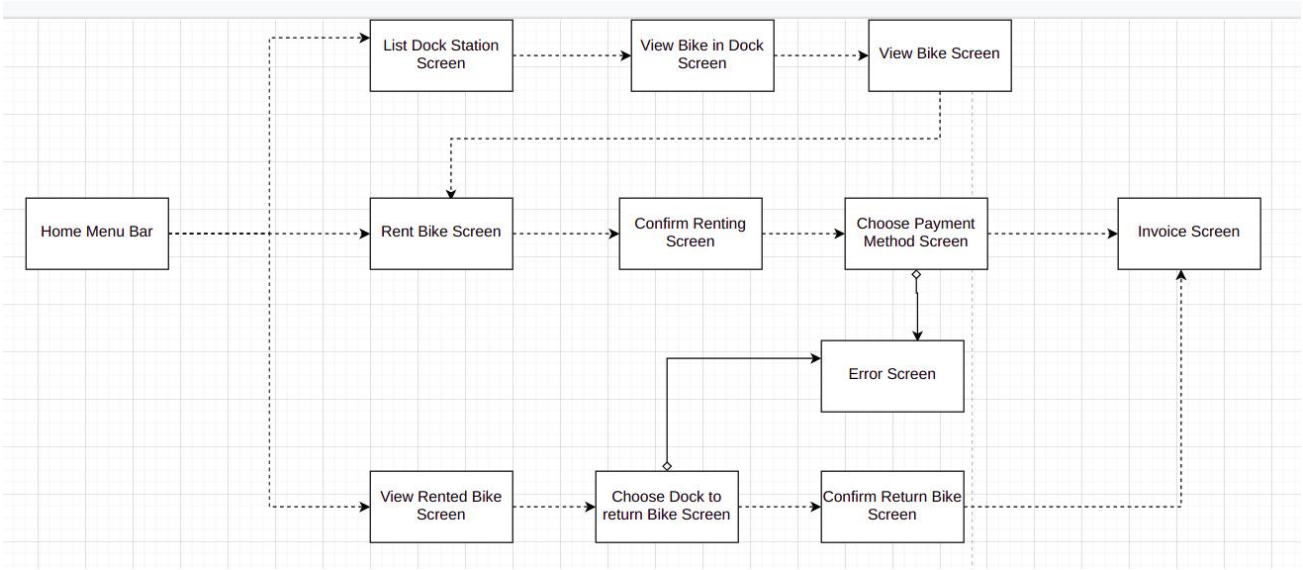
Input check process: Should check if it is empty or not. Next, check if the input is in the correct format or not

Sequence of moving the focus: There will be no stack frames. Each screen will be separated. However, the manual is considered a popup message, as the main screen cannot be operated while the manual screen is shown. After the opening screen, the app will start with splash screen, and then the first screen (home screen) will appear.

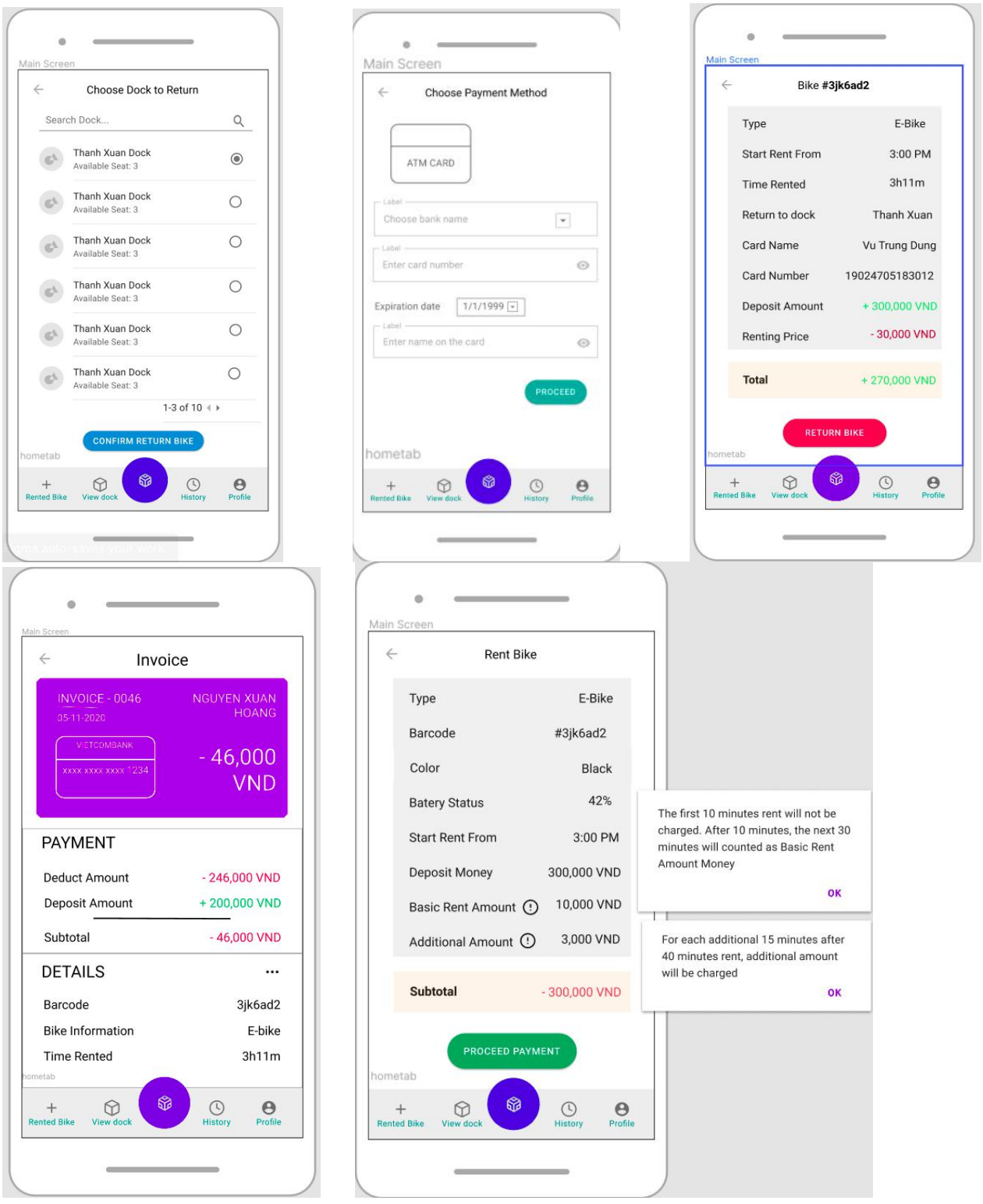
#### *4.1.1.4 Sequences of the system screens:*

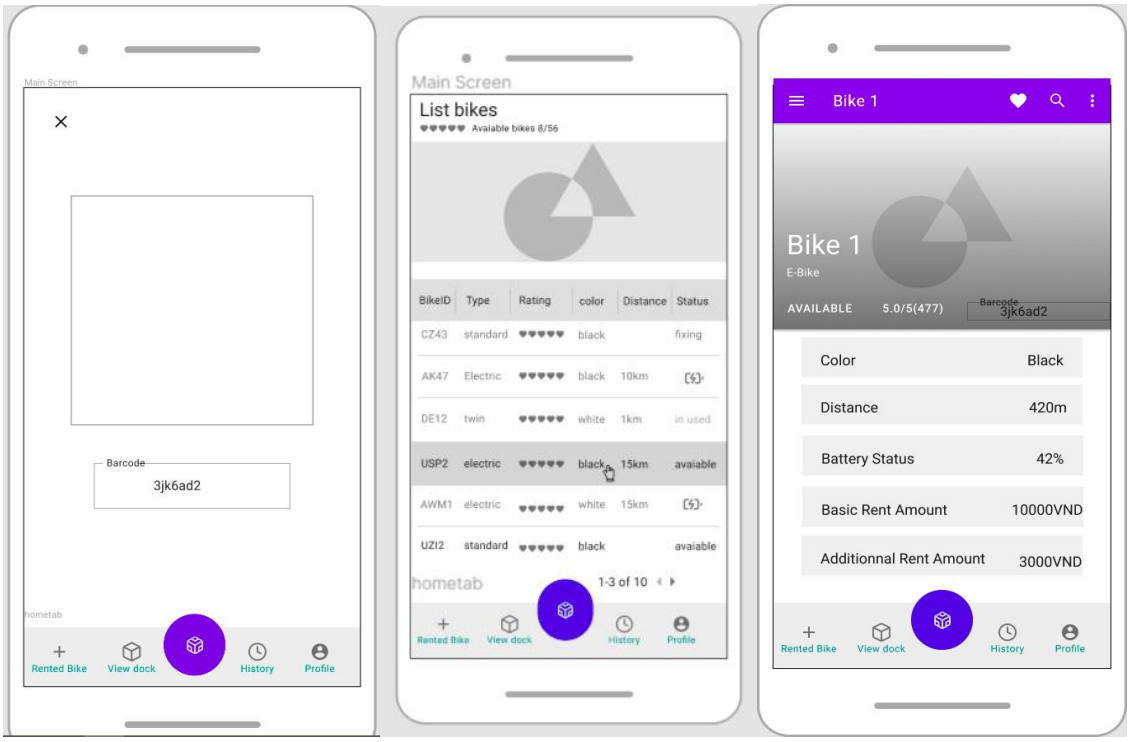
1. List Dock Screen
2. View Bike in Dock Screen
3. View Bike Screen
4. Enter Barcode to Rent Bike Screen
5. Confirm Renting Bike Screen
6. Choose Payment Method Screen
7. View Rented Bike Screen
8. Choose Dock to Return Bike Screen
9. Confirm Return Bike Screen
10. Invoice Screen

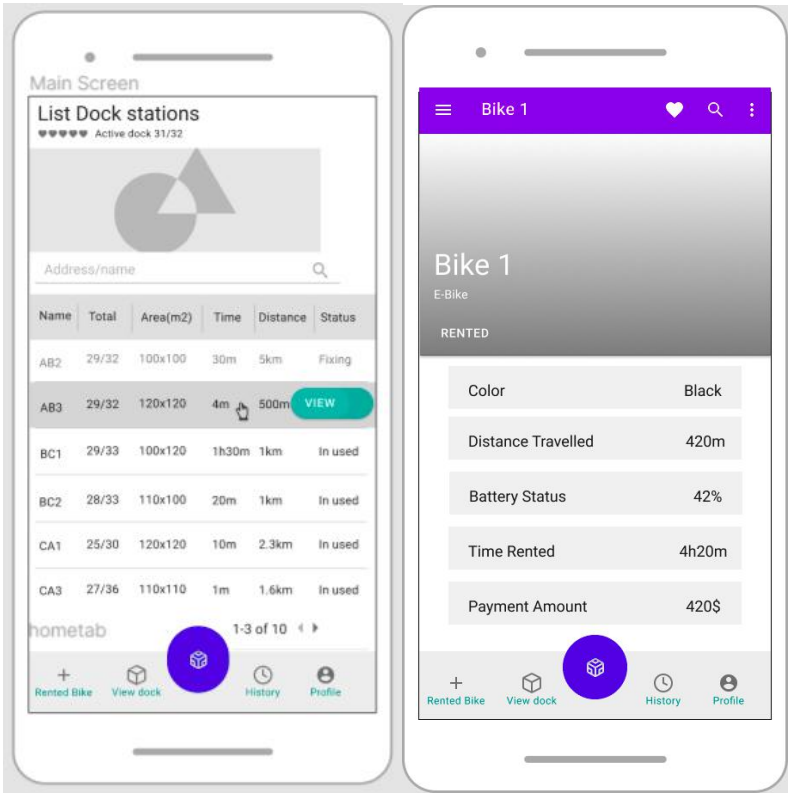
### 4.1.2 Screen Transition Diagrams



### 4.1.3 Screen Specifications



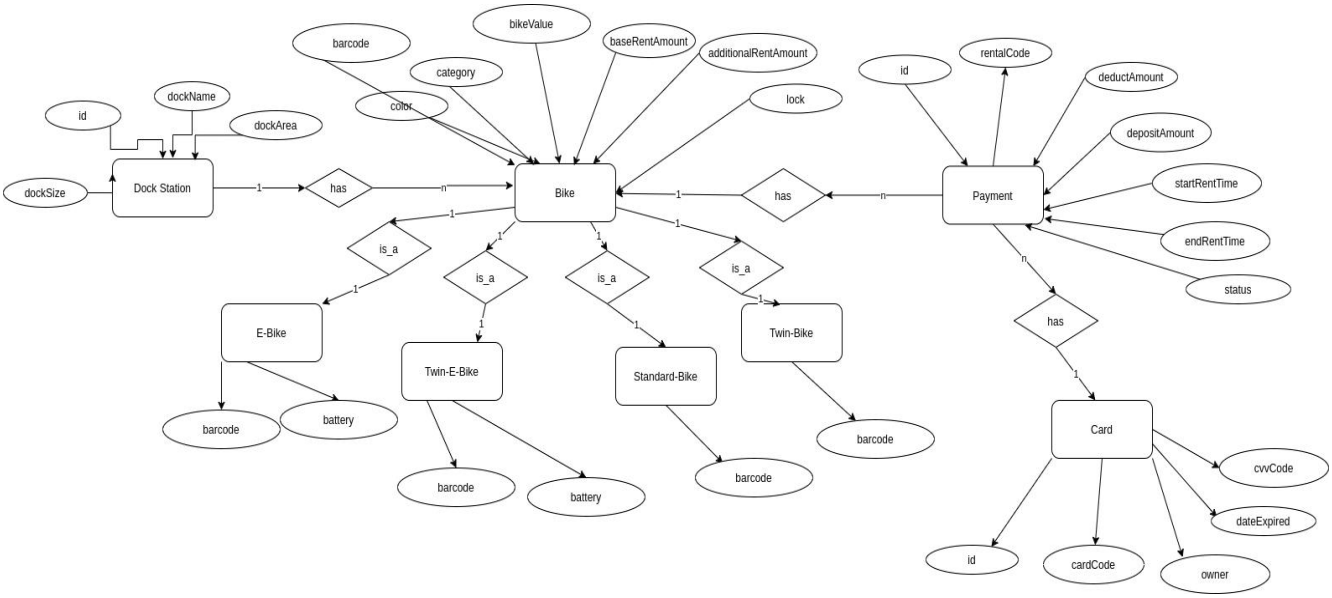

Main Screen
Barcode
3jk6ad2
Rented Bike
View dock
History
Profile
Main Screen
List bikes
Avaiable bikes 8/56
BikeID Type Rating color Distance Status
CZ43 standard black fixing
AK47 Electric black 10km
DE12 twin white 1km in used
USP2 electric black 15km avaiable
AWM1 electric white 15km
UZI2 standard black avaiable
hometab
1-3 of 10
Bike 1
Bike 1
E-Bike
AVAILABLE
5.0/5(477)
Barcode
3jk6ad2
Color Black
Distance 420m
Battery Status 42%
Basic Rent Amount 10000VND
Additionnal Rent Amount 3000VND


Main Screen
List Dock stations
Active dock 31/32
Address/name
Name Total Area(m2) Time Distance Status
AB2 29/32 100x100 30m 5km Fixing
AB3 29/32 120x120 4m 500m VIEW
BC1 29/33 100x120 1h30m 1km In used
BC2 28/33 110x100 20m 1km In used
CA1 25/30 120x120 10m 2.3km In used
CA3 27/36 110x110 1m 1.6km In used
hometab
1-3 of 10
Rented Bike
View dock
History
Profile
Bike 1
Bike 1
E-Bike
RENTED
Color Black
Distance Travelled 420m
Battery Status 42%
Time Rented 4h20m
Payment Amount 420$

## 4.2 Data Modeling

### 4.2.1 Conceptual Data Modeling

*<E-R Diagram image and description of entities and relationships>*

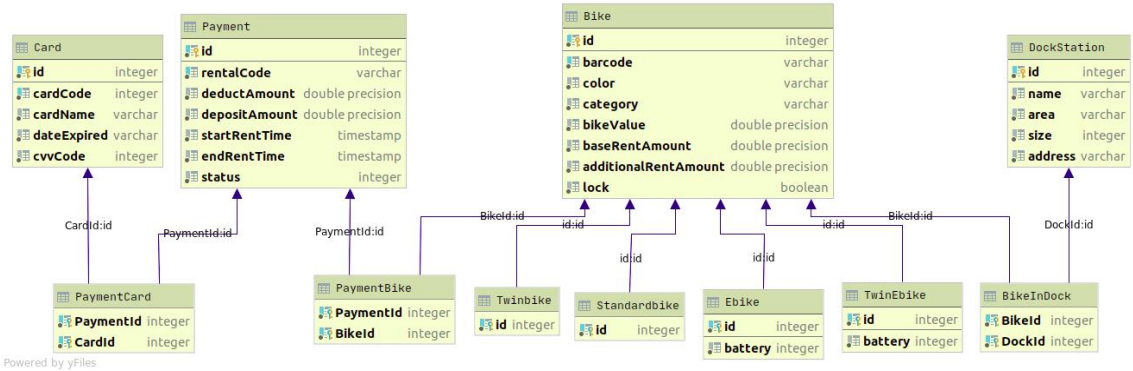



### 4.2.2 Database Design

#### *4.2.2.1 Database Management Systems*

- Database Management System: PostgreSQL
- PostgreSQL follows SQL standards but does not conflict with traditional features or could lead to harmful architectural decisions.
- PostgreSQL is open source, powerful DBMS and there are a wide variety of communities. Therefore, it will be much more easier to find a solution when having concern or error.

#### *4.2.2.2 Logical Data Model*



#### *4.2.2.3 Physical Data Model*

**Payment**

| # | PK | FK | Column Name | Data Type | Mandatory | Description |
|---|---|---|---|---|---|---|
| **1** | x | | id | serial | yes | Payment id |
| **2** | | | rentalCode | Integer | yes | Rental code |
| **3** | | | deductAmount | float | Yes | Deduct amount |
| **4** | | | DepositAmount | Float | yes | Deposit amount |
| **5** | | | startRentTime | TIMESTAMP | yes | Starting rent time |
| **6** | | | endRentTime | TIMESTAMP | yes | Ending rent time |
| **7** | | | Status | integer | yes | Status of transaction |

- **Payment Bike**

| # | PK | FK | Column Name | Data Type | Mandatory | Description |
|---|---|---|---|---|---|---|
| **1** | | x | PaymentId | Serial | Yes | Payment id |
| **2** | | x | BikeId | Serial | Yes | Bike id |

- DockStation

| # | PK | FK | Column Name | Data Type | Mandator | Description |
|---|---|---|---|---|---|---|

| | | | | | *y* | |
|---|---|---|---|---|---|---|
| 1. | x | | Id | Integer | Yes | ID, auto increment |
| 2. | | | Name | VARCHAR | Yes | Name of dock |
| 3. | | | area | VARCHAR | Yes | Area of the dock |
| 4. | | | size | Integer | Yes | Max size of dock |
| 5. | | | Address | VARCHAR | Yes | Address of dock |

- BikeInDock

| **#** | *PK* | *FK* | *Column Name* | *Data Type* | *Mandatory* | *Description* |
|---|---|---|---|---|---|---|
| 1. | x | x | BikeId | Integer | Yes | Bike Id |
| 2. | x | x | DockId | Integer | Yes | Dock Id |

- Card

| **#** | *PK* | *FK* | *Column Name* | *Data Type* | *Mandatory* | *Description* |
|---|---|---|---|---|---|---|
| 1. | x | | id | Integer | Yes | ID |
| 2. | | | cardCode | VARCHAR | Yes | Card’s Code |

| | | | | | | |
|---|---|---|---|---|---|---|
| 3. | | | cardName | VARCHAR | Yes | Name of the card's owner |
| 4. | | | cvvCode | INT | Yes | CVV Code of the Card |
| 5. | | | dateExpired | VARCHAR | Yes | Card's Expiration Date |

- PaymentCard

| # | *PK* | *FK* | *Column Name* | *Data Type* | *Mandatory* | *Description* |
|---|---|---|---|---|---|---|
| 1. | X | X | PaymentID | Integer | Yes | Payment ID |
| 2. | X | X | CardID | Integer | Yes | Card ID |

- Bike

| # | *PK* | *FK* | *Column Name* | *Data Type* | *Mandatory* | *Description* |
|---|---|---|---|---|---|---|
| **1** | x | | Id | Integer | Yes | ID, auto increment |
| **2** | | | barcode | VARCHAR | Yes | Bike's barcode |
| **3** | | | color | VARCHAR | Yes | Bike's color |
| **4** | | | category | VARCHAR | Yes | Bike's category |
| **5** | | | bikeValue | float | Yes | Bike's value |

| | | | | | | |
|---|---|---|---|---|---|---|
| **6** | | | baseRentAmount | float | Yes | Bike’s base rent amount |
| **7** | | | additionalRentAmount | float | Yes | Bike’s additional rent amount |
| **8** | | | lock | BOOLEAN | Yes | Bike’s lock status |

- E-Bike

| **#** | *PK* | *FK* | *Column Name* | *Data Type* | *Mandatory* | *Description* |
|---|---|---|---|---|---|---|
| 1. | x | x | id | Integer | Yes | Bike Id |
| 2. | | | battery | Integer | Yes | Bike’s battery status |

- Twin E-bike

| **#** | *PK* | *FK* | *Column Name* | *Data Type* | *Mandatory* | *Description* |
|---|---|---|---|---|---|---|
| 1. | x | x | id | Integer | Yes | Bike Id |
| 2. | | | battery | Integer | Yes | Bike’s battery status |

- Standard Bike

| **#** | *PK* | *FK* | *Column Name* | *Data Type* | *Mandatory* | *Description* |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| 1. | x | x | id | Integer | Yes | Bike Id |

- Twin Bike

| # | PK | FK | Column Name | Data Type | Mandatory | Description |
|---|---|---|---|---|---|---|
| 1. | x | x | id | Integer | Yes | Bike Id |

*- Database script*

```
create table "ecoBikeSystem"."DockStation"
(
  id serial not null,
  name VARCHAR not null,
  area VARCHAR not null,
  size int not null,
  address VARCHAR not null
);

create unique index dockstation_id_uindex
  on "ecoBikeSystem"."DockStation" (id);

alter table "ecoBikeSystem"."DockStation"
  add constraint dockstation_pk
    primary key (id);

create table "ecoBikeSystem"."Bike"
(
  id serial not null,
  barcode VARCHAR not null,
  color VARCHAR not null,
  category VARCHAR not null,
```

```
  "bikeValue" float not null,
  "baseRentAmount" float not null,
  "additionalRentAmount" float not null,
  lock BOOLEAN default FALSE not null
);

create unique index bike_barcode_uindex
  on "ecoBikeSystem"."Bike" (barcode);

create unique index bike_id_uindex
  on "ecoBikeSystem"."Bike" (id);

alter table "ecoBikeSystem"."Bike"
  add constraint bike_pk
    primary key (id);

create table "ecoBikeSystem"."BikeInDock"
(
  "BikeId" int not null
    constraint bikeindock_bike_id_fk
      references "ecoBikeSystem"."Bike"
        on update cascade on delete cascade,
  "DockId" int not null
    constraint bikeindock_dockstation_id_fk
      references "ecoBikeSystem"."DockStation"
        on update cascade on delete cascade,
  constraint bikeindock_pk
    primary key ("BikeId", "DockId")
);

create table "ecoBikeSystem"."Payment"
(
  id serial not null,
```

```
  "rentalCode" VARCHAR not null,
  "deductAmount" float not null,
  "depositAmount" float not null,
  "startRentTime" TIMESTAMP not null,
  "endRentTime" TIMESTAMP not null,
  status int not null
);

create unique index payment_id_uindex
  on "ecoBikeSystem"."Payment" (id);

create unique index payment_rentalcode_uindex
  on "ecoBikeSystem"."Payment" ("rentalCode");

alter table "ecoBikeSystem"."Payment"
  add constraint payment_pk
    primary key (id);


create table "ecoBikeSystem"."PaymentBike"
(
  "PaymentId" int not null
    constraint paymentbike_payment_id_fk
      references "ecoBikeSystem"."Payment"
        on update cascade on delete cascade,
  "BikeId" int not null
    constraint paymentbike_bike_id_fk
      references "ecoBikeSystem"."Bike"
        on update cascade on delete cascade,
  constraint paymentbike_pk
    primary key ("PaymentId", "BikeId")
);
```

```
create table "ecoBikeSystem"."Card"
(
  id serial not null,
  "cardCode" int not null,
  "cardName" VARCHAR not null,
  "dateExpired" VARCHAR not null,
  "cvvCode" int not null
);

create unique index card_cardcode_uindex
  on "ecoBikeSystem"."Card" ("cardCode");

create unique index card_id_uindex
  on "ecoBikeSystem"."Card" (id);

alter table "ecoBikeSystem"."Card"
  add constraint card_pk
    primary key (id);

create table "ecoBikeSystem"."PaymentCard"
(
  "PaymentId" int not null
    constraint paymentcard_payment_id_fk
      references "ecoBikeSystem"."Payment"
        on update cascade on delete cascade,
  "CardId" int not null
    constraint paymentcard_card_id_fk
      references "ecoBikeSystem"."Card"
        on update cascade on delete cascade,
  constraint paymentcard_pk
    primary key ("PaymentId", "CardId")
);
```

```
create table "ecoBikeSystem"."Ebike"
(
  id int not null
    constraint ebike_pk
      primary key
    constraint ebike_bike_id_fk
      references "ecoBikeSystem"."Bike"
        on update cascade on delete cascade,
  battery int not null
);

create table "ecoBikeSystem"."TwinEbike"
(
  id int not null
    constraint twinebike_pk
      primary key
    constraint twinebike_bike_id_fk
      references "ecoBikeSystem"."Bike"
        on update cascade on delete cascade,
  battery int not null
);

create table "ecoBikeSystem"."Standardbike"
(
  id int not null
    constraint standardbike_pk
      primary key
    constraint standardbike_bike_id_fk
      references "ecoBikeSystem"."Bike"
        on update cascade on delete cascade
);

create table "ecoBikeSystem"."Twinbike"
```

```
(
  id int not null
    constraint twinbike_pk
      primary key
    constraint twinbike_bike_id_fk
      references "ecoBikeSystem"."Bike"
        on update cascade on delete cascade
);
```

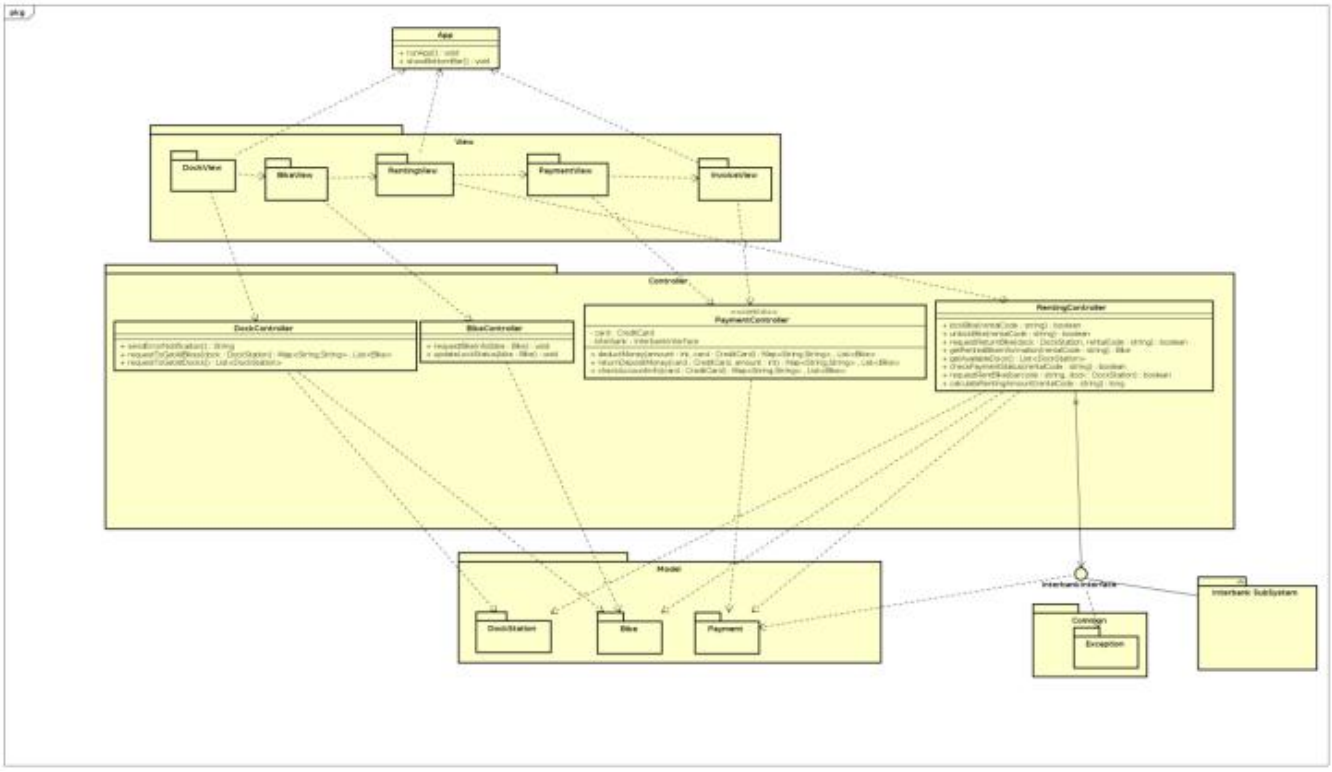
### *4.3 Non-Database Management System Files*

*<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>*

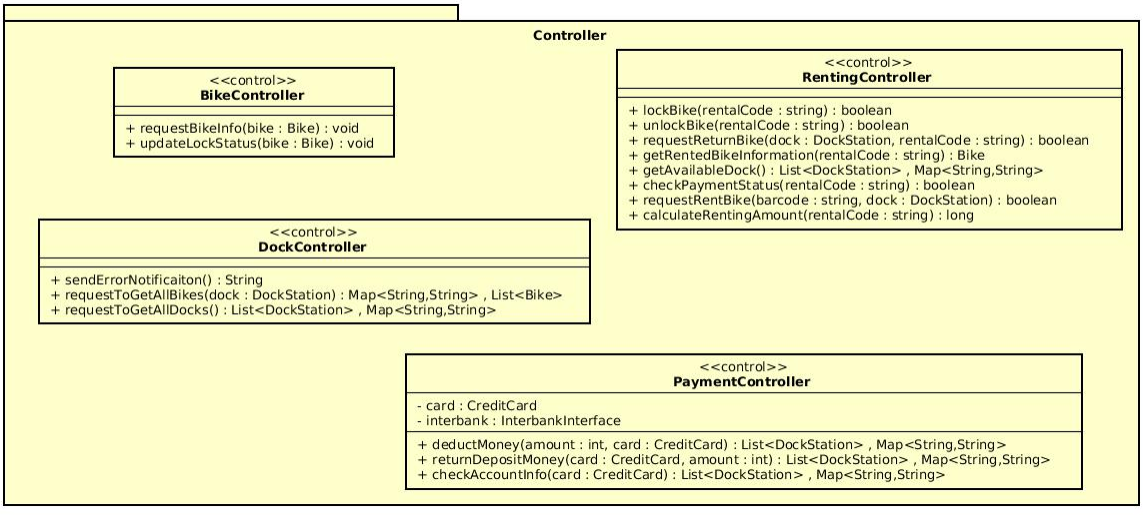
## 4.4 Class Design

### 4.4.1 General Class Diagram

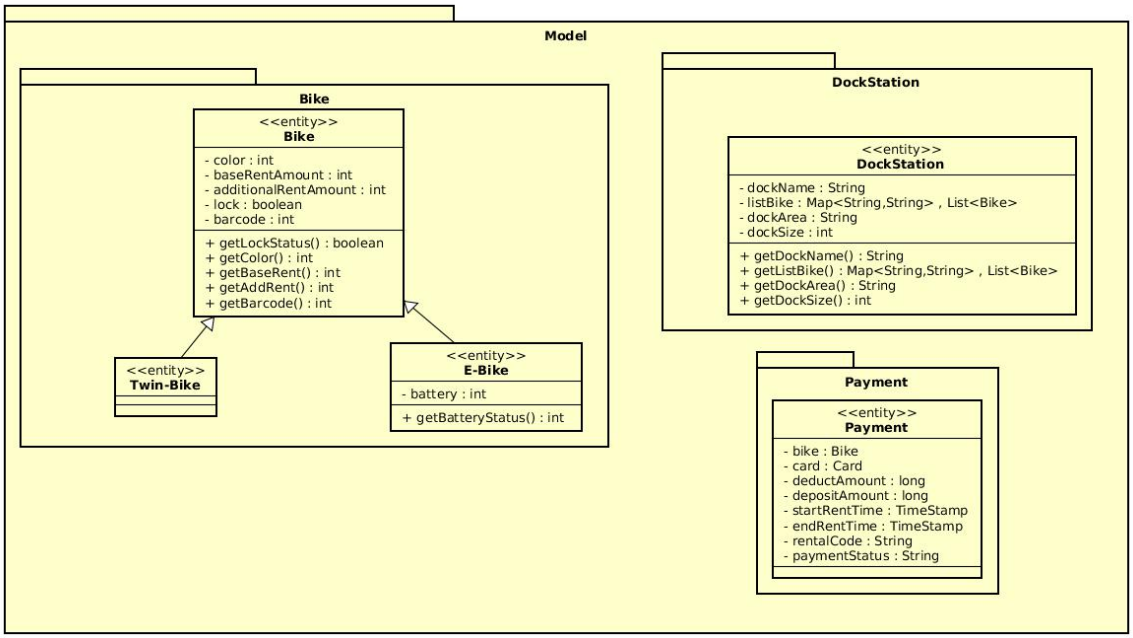


### 4.4.2 Class Diagrams

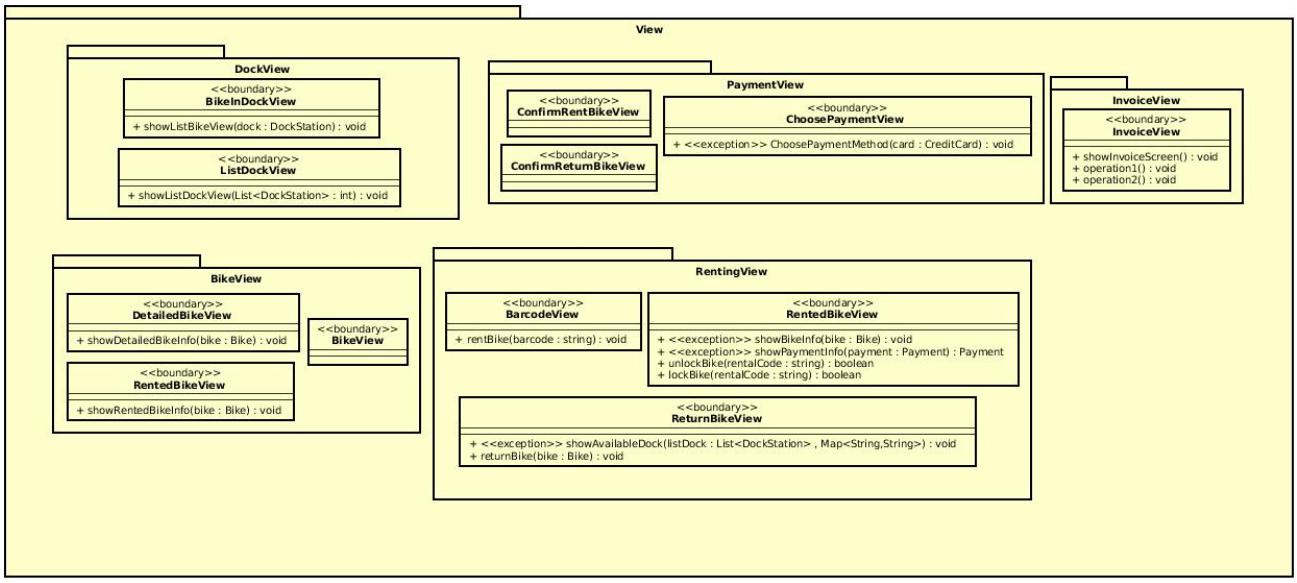
#### 4.4.2.1 Class Diagram for Package Controller



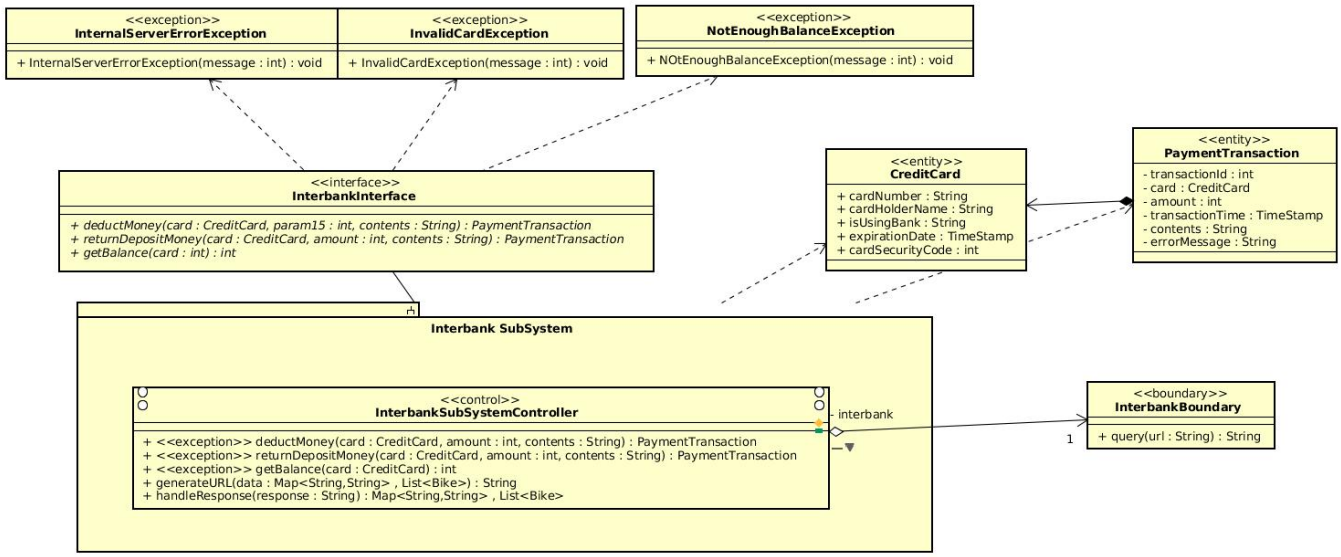

#### 4.4.2.2 Class Diagram for Package Controller




### 4.4.2.3 Class Diagram for Package View




### 4.4.2.4 Class Diagram for SubSystem Interbank




### 4.4.3 Class Design

#### *4.4.3.1 Class "ListDockView"*


**Attribute**

None

**Operation**

| # | *Name* | *Return type* | *Description (purpose)* |
|---|---|---|---|
| 1 | showListDockView | void | Display the list dock station view |
| 2 | | | |

*Parameter*:

- DockStation: List<DockStation> - a list of all dock stations

*Exception*:

**Method**

**State**

#### *4.4.3.2 Class "BikeInDockView"*


**Attribute**

None

**Operation**

| # | *Name* | *Return type* | *Description (purpose)* |
|---|---|---|---|
| 1 | ShowListBikeView | void | Show the list Bike in dock view |

*Parameter*:

- Dock: dockStation – information of the dock

*Exception*:

**Method**

**State**

#### *4.4.3.3 Class "DockController"*


**Attribute**

None

**Operation**

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

| | | | |
|---|---|---|---|
| 1 | requestToGetAllBikes | List<Bike> | Get all bikes from a dock station |
| 2 | requestToGetAllDocks | List<DockStation> | Update the cart information |
| 3 | sendErrorNotification | String | Send error notification |

*Parameter*:

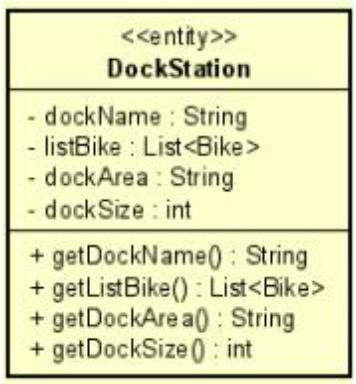
- Dock: dockStation – information of the dock

*Exception*:

**Method**

**State**

#### *4.4.3.4 Class "DockStation"*




**Attribute**

| # | *Name* | *Data type* | *Default value* | *Description* |
|---|---|---|---|---|
| 1 | dockName | String | null | name of the dock |

| | | | | |
|---|---|---|---|---|
| 2 | listBike | List<Bike> | null | list bike in the dock |
| 3 | dockArea | String | null | example: 120x100 (m2) |
| 4 | dockSize | Int | null | total number of bike can be fit in dock |

**Operation**

| # | *Name* | *Return type* | *Description (purpose)* |
|---|---|---|---|
| 1 | getDockName | String | Get name of the dock station |
| 2 | getListBike | List<Bike> | Get list bike in the dock station |
| 3 | getDockArea | String | Get dock area |

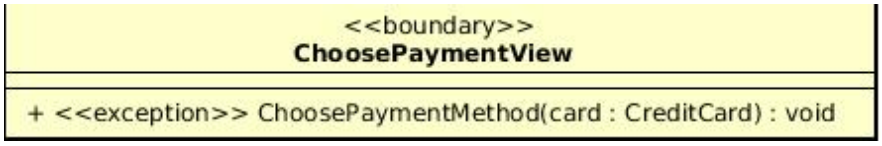
*Parameter*:

*Exception*:

**Method**

**State**

#### *4.4.3.5 Class "PaymentView"*




**Attribute**

None

**Operation**

| # | *Name* | *Return type* | *Description (purpose)* |
|---|---|---|---|
| 1 | ChoosePaymentMethod | Void | Choose Payment method for payment/refund |

*Parameter*:

- card – credit card used for payment/refund

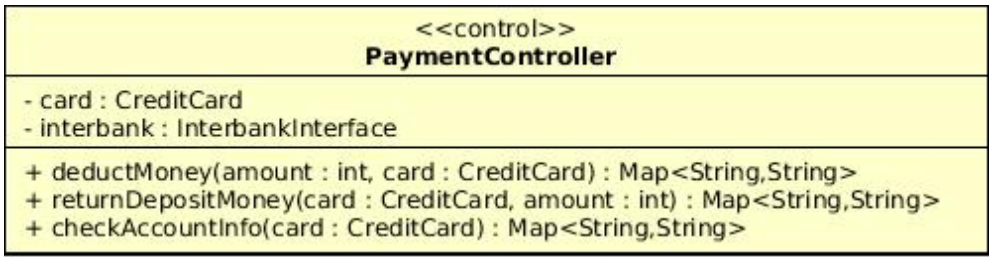
*Exception*:

- PaymentException - If responded with a pre-defined error code
- UnrecognizedException – If responded with unknown error code or something goes wrong

**Method**

**State**

#### *4.4.3.6 Class "PaymentController"*




**Attribute**

| # | *Name* | *Data type* | *Default value* | *Description* |
|---|---|---|---|---|
| 1 | Card | CreditCard | null | Represent the card used for payment |
| 2 | Interbank | InterbankInterface | null | Represent the Interbank system |

**Operation**

| # | *Name* | *Return type* | *Description (purpose)* |
|---|---|---|---|
| 1 | DeductMoney | Map<String,String> | Deduct money for renting and return the result with a message |
| 2 | ReturnDepositMoney | Map<String,String> | Return deposit money when user return bike |
| 3 | CheckAccountInfo | Map<String,String> | Check credit card information and return result with a message |

*Parameter*:

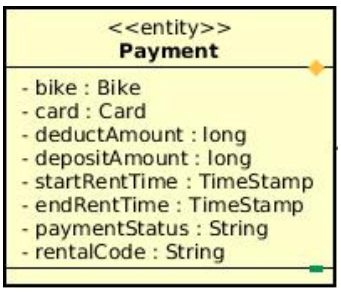
- card – credit card used for payment/refund
- amount – the amount to pay

*Exception*:

**Method**

**State**

#### *4.4.3.7 Class "Payment"*




**Attribute**

| # | *Name* | *Data type* | *Default value* | *Description* |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| 1 | Bike | Bike | NULL | Represent the bike which user rented |
| 2 | card | Card | NULL | Represent the card used for payment |
| 3 | DeductAmount | Long | NULL | Amount of deduct money |
| 4 | DepositeAmount | Long | NULL | Amount of deposit money |
| 5 | StartRentTime | TimeStamp | NULL | Time when user start renting |
| 6 | EndRentTime | TimeStamp | NULL | Time when user end renting |
| 7 | paymentStatus | String | NULL | Status of the payment |
| 8 | RentalCode | String | NULL | Unique code of every rental |

**Operation**

None

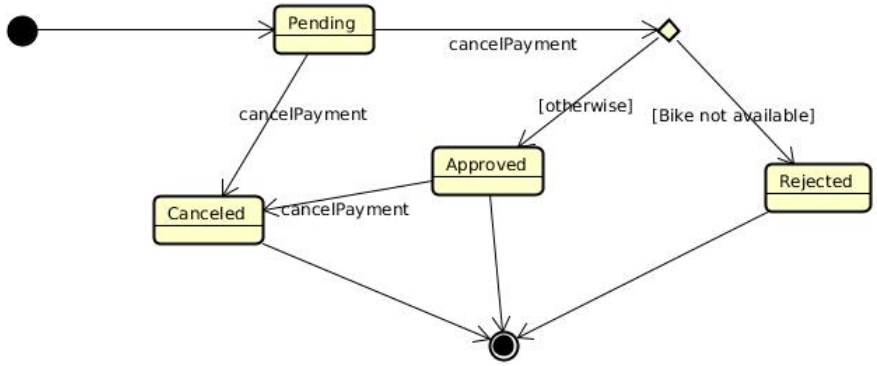
*Parameter*:

- card – credit card used for payment/refund
- amount – the amount to pay

*Exception*:

**Method**

**State**




#### *4.4.3.8 Class "DetailedBikeView"*


**Attribute**

None

**Operation**

| # | *Name* | *Return type* | *Description (purpose)* |
|---|---|---|---|
| 1 | showDetailedBikeInfo | Void | Display the detailed bike information view |

*Parameter*:

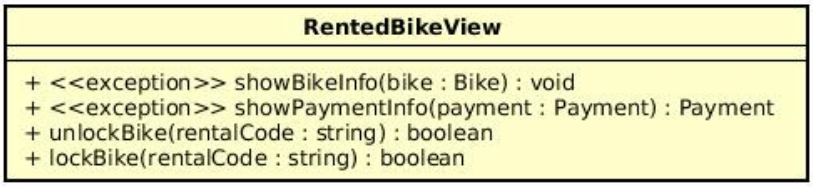
- bike – the bike selected

*Exception*:

**Method**

**State**

#### *4.4.3.9 Class "RentedBikeView"*




**Attribute**

None

**Operation**

| # | *Name* | *Return type* | *Description (purpose)* |
|---|---|---|---|
| 1 | showBikeInfo | void | Show Information of rented bike |
| 2 | showPaymentInfo | Void | Show information about rented bike fee |
| 3 | unlockBike | Void | Lock the rented bike from user |
| 4 | lockBike | Void | Unlock the rented bike from user |

*Parameter*:

- listDock: List<DockStation> - list of DockStation model
- bike:Bike – the bike selected

*Exception*:

- rentedBikeException - exception raise if the rented bike is not eligible
- paymentException - exception raise if there is no payment with the bike

**Method**

**State**

#### *4.4.3.10 Class "BikeController"*

**Attribute**

None

**Operation**

| # | *Name* | *Return type* | *Description (purpose)* |
|---|---|---|---|
| 1 | requestBikeInfo | Void | Request all related information for selected bike from database |
| 2 | updateLockStatus | Void | Change rented bike's lock status |

*Parameter*:

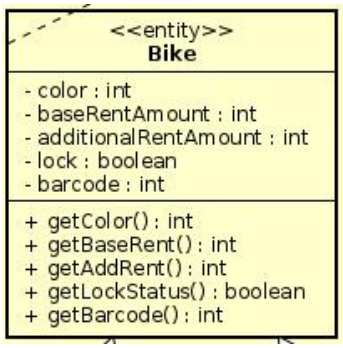
- bike – the bike selected

*Exception*:

**Method**

**State**

### *4.4.3.11 Class "Bike"*




**Attribute**

| # | *Name* | *Data type* | *Default value* | *Description* |
|---|---|---|---|---|
| 1 | color | int | NULL | Color of the bike |
| 2 | baseRentAmount | int | NULL | Base amount of rent for the bike |
| 3 | additionalRentAmount | int | NULL | Additional amount of rent for the bike |

| | | | | |
|---|---|---|---|---|
| 4 | lock | boolean | NULL | Lock status for rented bike |
| 5 | barcode | int | NULL | Barcode of the bike |

**Operation**

| # | *Name* | *Return type* | *Description (purpose)* |
|---|---|---|---|
| 1 | getColor | int | Request selected bike's color |
| 2 | getBaseRent | int | Request selected bike's base rent amount |
| 3 | getAddRent | int | Request selected bike's additional rent amount |
| 4 | getLockStatus | boolean | Request selected bike's lock status |
| 5 | getBarcode | int | Request selected bike's barcode |

*Parameter*:

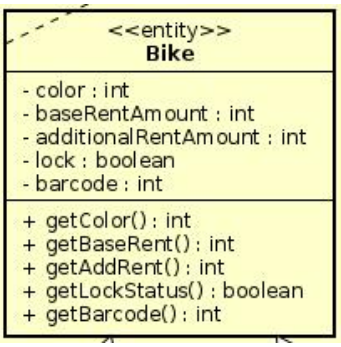
- bike – the bike selected

*Exception*:

**Method**

**State**

### *4.4.3.12 Class "RentingController"*




**Attribute**

None

**Operation**

| # | *Name* | *Return type* | *Description (purpose)* |
|---|---|---|---|
| 1 | lockBike | boolean | Lock the rented bike |
| 2 | unlockBike | boolean | Unlock the rented bike |
| 3 | requestRentBike | boolean | Request to rent a bike in dock |
| 4 | calculateRentingAmount | Long | Calculate the renting amount of the rented bike |
| 5 | getRentedBikeInformation | Bike | Get the information of the rented bike |
| 6 | requestReturnBike | Boolean | Request to return a bike to a dock |
| 7 | checkPaymentStatus | Boolean | Check whether the transaction has been paid or not |
| 8 | getAvailableDock | List<DockStation> | Get available dock for returning bike |

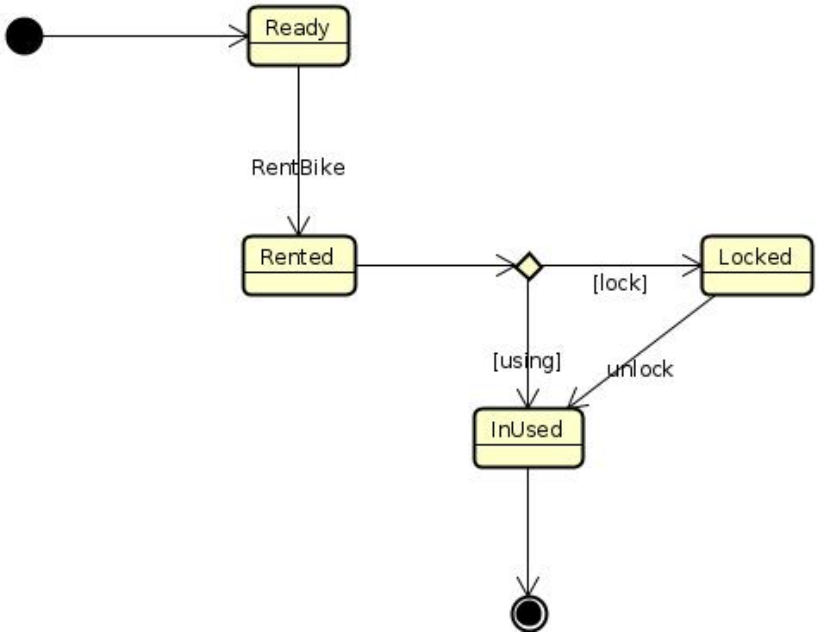
*Parameter*:

- rentalCode:string – the rental code of the rented bike
- Barcode:string - barcode of the bike
- Dock:DockStation - dock station model

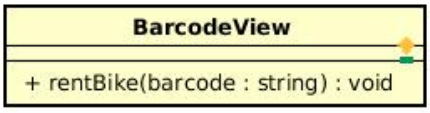
*Exception*:

**Method**

**State**




#### *4.4.3.13 Class "BarcodeView"*




**Attribute**

None

**Operation**

| # | *Name* | *Return type* | *Description (purpose)* |
|---|---|---|---|
| 1 | rentBike | void | Request to rent bike from user |

*Parameter*:

- Barcode:string - barcode of the bike

*Exception*:

**Method**

**State**

#### *4.4.3.14 Class "ReturnBikeView"*


**Attribute**

None

**Operation**

| # | *Name* | *Return type* | *Description (purpose)* |
|---|---|---|---|
| 1 | showAvailableDock | void | Show available dock to user to return bike |
| 2 | returnBike | void | Button to request to return bike |

*Parameter*:

- listDock: List<DockStation> - list of DockStation model
- bike:Bike – the bike selected

*Exception*:

- dockException - show exception if retrieve dock information failed

**Method**

**State**

#### *4.4.3.15 Class "ReturnBikeView"*


**Attribute**

None

**Operation**

| # | *Name* | *Return type* | *Description (purpose)* |
|---|---|---|---|
| 1 | showAvailableDock | void | Show available dock to user to return bike |

| | | | |
|---|---|---|---|
| 2 | returnBike | void | Button to request to return bike |

*Parameter*:

- listDock: List<DockStation> - list of DockStation model
- bike:Bike – the bike selected

*Exception*:

- dockException - show exception if retrieve dock information failed

**Method**

**State**

# 5 Design Considerations

## *5.1 Goals and Guidelines*

### *5.1.1* Goals

- Usability: User Interface Easy-to-use
- Speed Optimization for less-than-5-second User Interaction
- Memory Usage Optimization for better app performance

### *5.1.2* Guidelines

*Coding Convention for Flutter-Dart:*

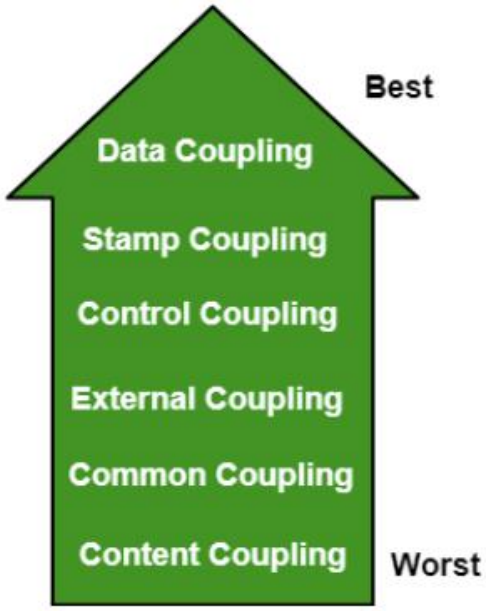
Using Basic Coding Convention of Dart Language. Detailed show at https://dart.dev/guides/language/effective-dart/style

## *5.2 Architectural Strategies*

- *Programming Language: Dart*
- *Framework: Flutter*
- *Database Management System: PostgreSQL*
- *Using Subsystem: Interbank System for Card Management*
- *Error Detection: Using Unit-test and Integration Test*
- *Synchronization: Asynchronization Method using Dart Language*

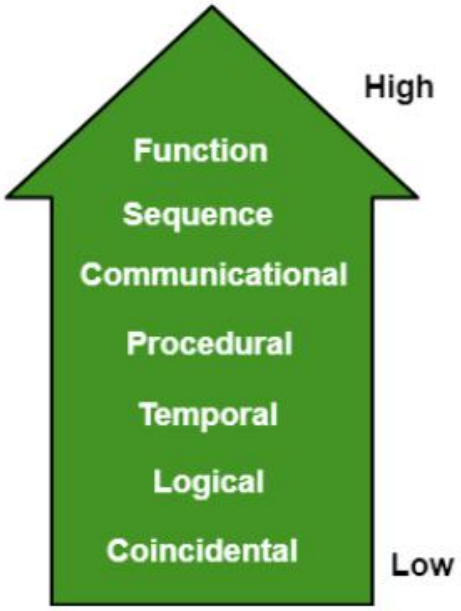
## *5.3 Coupling and Cohesion*

**Coupling:** Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling




Our Project Use **Data Coupling**. In data coupling, the components are independent to each other and communicating through data. Module communications don't contain tramp data. Example - RentingController class only know how to get data from Bike Entity class and use it for totally different method to take it to the View package.

**Cohesion:** Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion




Our Project Use **Functional Cohesion.** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. For example, calculateRentingAmount method in RentingController class perform the calculation task and send it to RentedBikeView. The RentedBikeView class only rendering the returned value.

## *5.4 Design Principles*

SOLID principles are the design principles that enable us manage most of the software design problems.

SOLID Acronym:

- S: Single Responsibility Principle (SRP)
- O: Open closed Principle (OSP)
- L: Liskov substituition Principle (LSP)
- I: Interface Segregation Principle (ISP)
- D: Dependency Inversion Principle (DIP)

1.1 Single Responsibility Principle

" A class should have only one reason to change". Every module or class should have responsibility over a single part of the functionality provided by the software and that responsibility should be entirely encapsulated by the class.

1.2 Liskov Substitution Principle

“Objects in a program should be replaceable with instances of their sub-types without altering the correctness of that program”. If a program module is using a Base class, then the reference to the Base class can be replaced with a Derived class without affecting the functionality of the program module. We can also state that Derived types must be substitutable for their base types.

1.3 Open/Closed Principle

“Software entities should be open for extension, but closed for modification”. The design and writing of the code should be done in a way that new functionality should be added with minimum changes in the existing code. The design should be done in a way to allow the adding of new functionality as new classes, keeping as much as possible existing code unchanged.

1.4 Interface Segregation Principle

“Many client-specific interfaces are better than one general-purpose interface”. We should not enforce clients to implement interfaces that they don't use. Instead of creating one big interface we can break down it to smaller interfaces

1.5 Dependency Inversion Principle

One should “depend upon abstractions, [not] concretions" . Abstractions should not depend on the details whereas the details should depend on abstractions. High-level modules should not depend on low level modules.

## *5.5 Design Patterns*

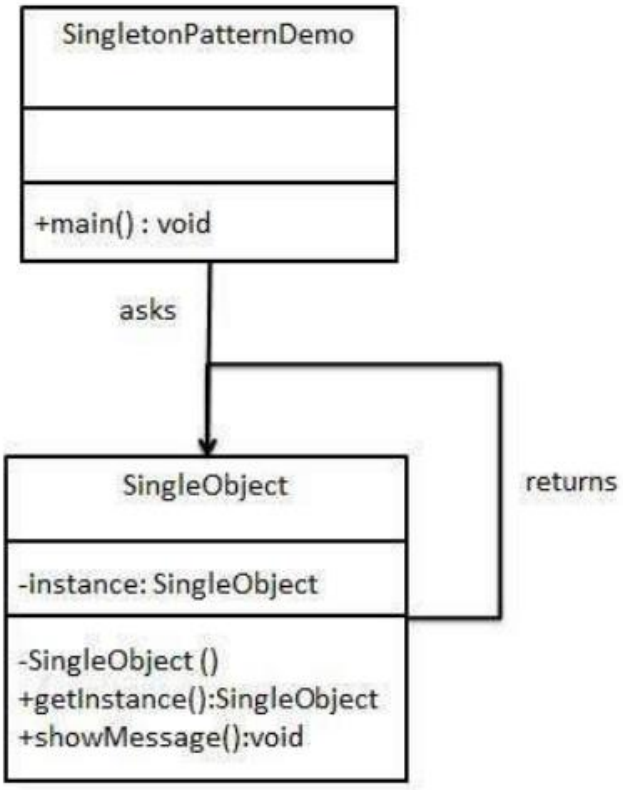
### 5.5.1 Singleton

*Singleton pattern is one of the simplest design patterns in OOP Language. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.*

*This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.*

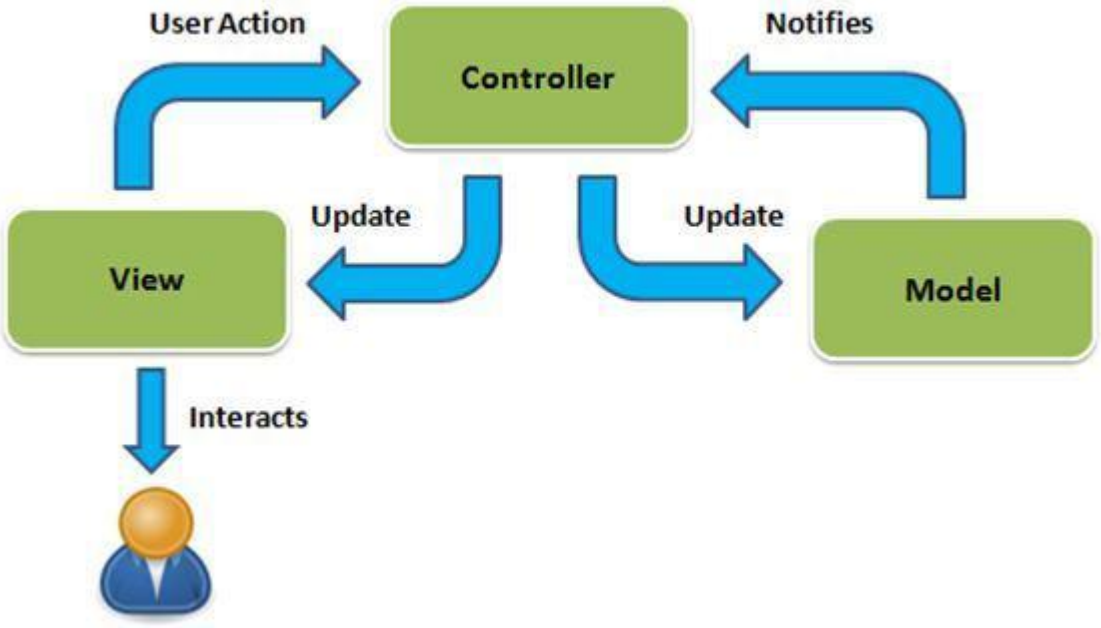
*SingleObject class provides a static method to get its static instance to outside*

*world. SingletonPatternDemo, our demo class will use SingleObject class to get a SingleObject object.*




### 5.5.2 MVC

In our EcoBikeRental System, we also use MVC Pattern as our main Design Pattern Structure. Our general MVC Design Pattern will follow as the below picture:


User Action
Controller
Notifies
Update
Update
View
Model
Interacts