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, 12/2020*

# **Table of Contents**

Table of Contents 2

1 Introduction [3](#_2et92p0)

1.1 Objective [3](#_tyjcwt)

1.2 Scope [3](#_3dy6vkm)

1.3 Glossary [3](#_1t3h5sf)

1.4 References [3](#_2s8eyo1)

2 Overall Description [4](#_17dp8vu)

2.1 General Overview [4](#_3rdcrjn)

2.2 Assumptions/Constraints/Risks [4](#_26in1rg)

2.2.1 Assumptions [4](#_lnxbz9)

2.2.2 Constraints [4](#_35nkun2)

2.2.3 Risks [5](#_1ksv4uv)

3 System Architecture and Architecture Design [6](#_44sinio)

3.1 Architectural Patterns [6](#_2jxsxqh)

3.2 Interaction Diagrams [6](#_z337ya)

3.3 Analysis Class Diagrams [6](#_3j2qqm3)

3.4 Unified Analysis Class Diagram [6](#_1y810tw)

3.5 Security Software Architecture [6](#_4i7ojhp)

4 Detailed Design [7](#_2xcytpi)

4.1 User Interface Design [7](#_1ci93xb)

4.1.1 Screen Configuration Standardization [7](#_3whwml4)

4.1.2 Screen Transition Diagrams [7](#_49x2ik5)

4.1.3 Screen Specifications [7](#_2p2csry)

4.2 Data Modeling [7](#_147n2zr)

4.2.1 Conceptual Data Modeling [7](#_3o7alnk)

4.2.2 Database Design [7](#_23ckvvd)

4.3 Non-Database Management System Files [8](#_ihv636)

4.4 Class Design [8](#_32hioqz)

4.4.1 General Class Diagram [8](#_1hmsyys)

4.4.2 Class Diagrams [8](#_41mghml)

4.4.3 Class Design [8](#_vx1227)

5 Design Considerations [10](#_19c6y18)

5.1 Goals and Guidelines [10](#_3tbugp1)

5.2 Architectural Strategies [10](#_28h4qwu)

5.3 Coupling and Cohesion [11](#_37m2jsg)

5.4 Design Principles [11](#_1mrcu09)

5.5 Design Patterns [11](#_46r0co2)

**List of Figures**

No table of figures entries found.

**List of Tables**

No table of figures entries found.

# Introduction

## 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.

## 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.

## 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 |

## 

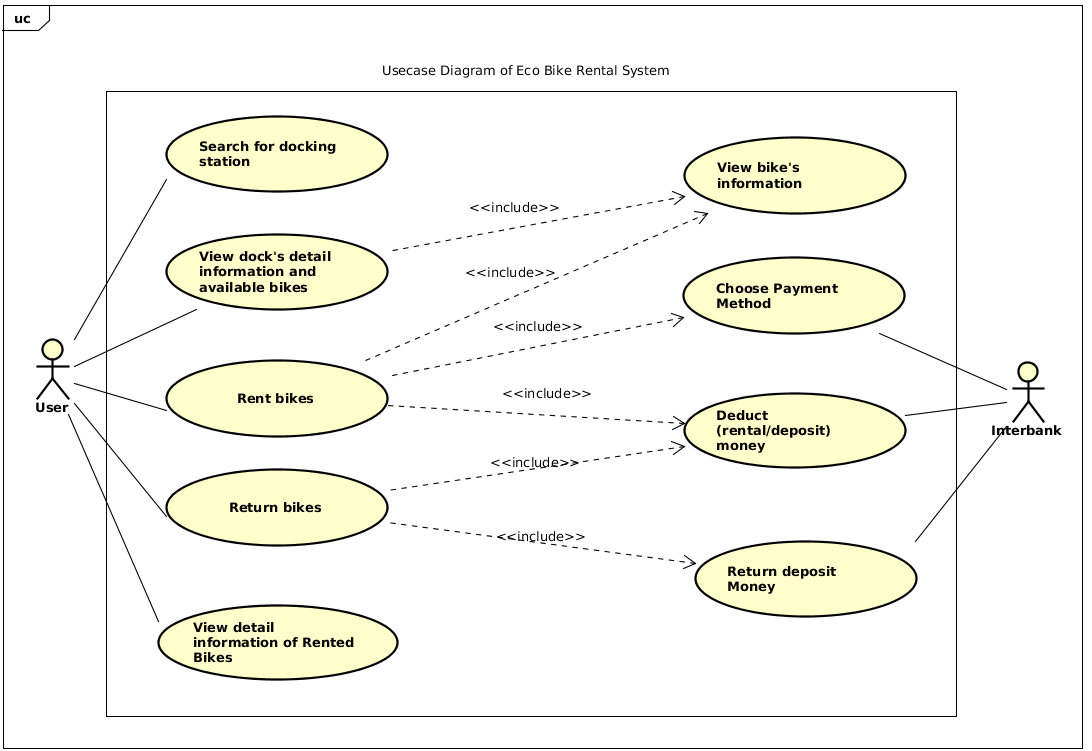
## References

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

# Overall Description

## General Overview

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

**

## Assumptions/Constraints/Risks

### 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.

### 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

### 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

# System Architecture and Architecture Design

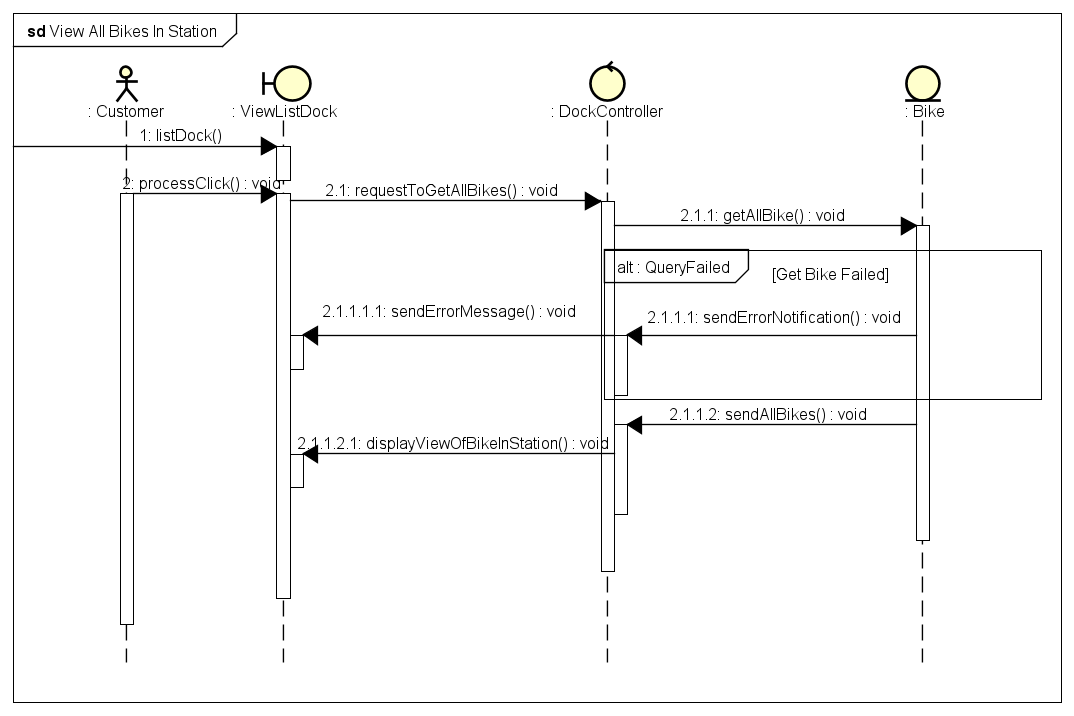
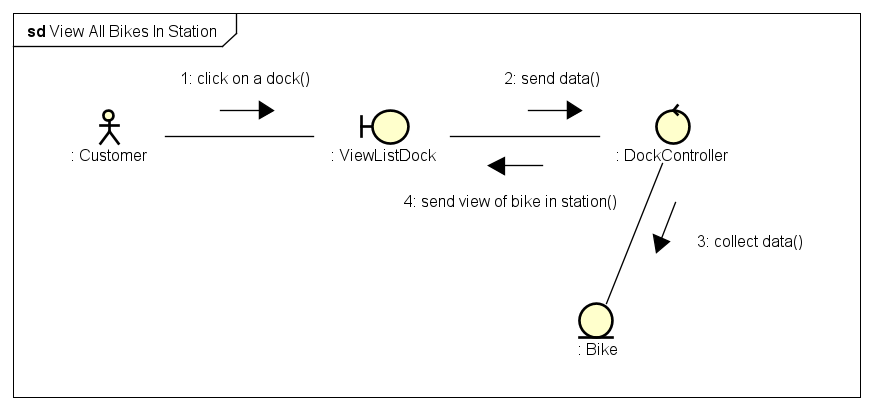
## Architectural Patterns

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.

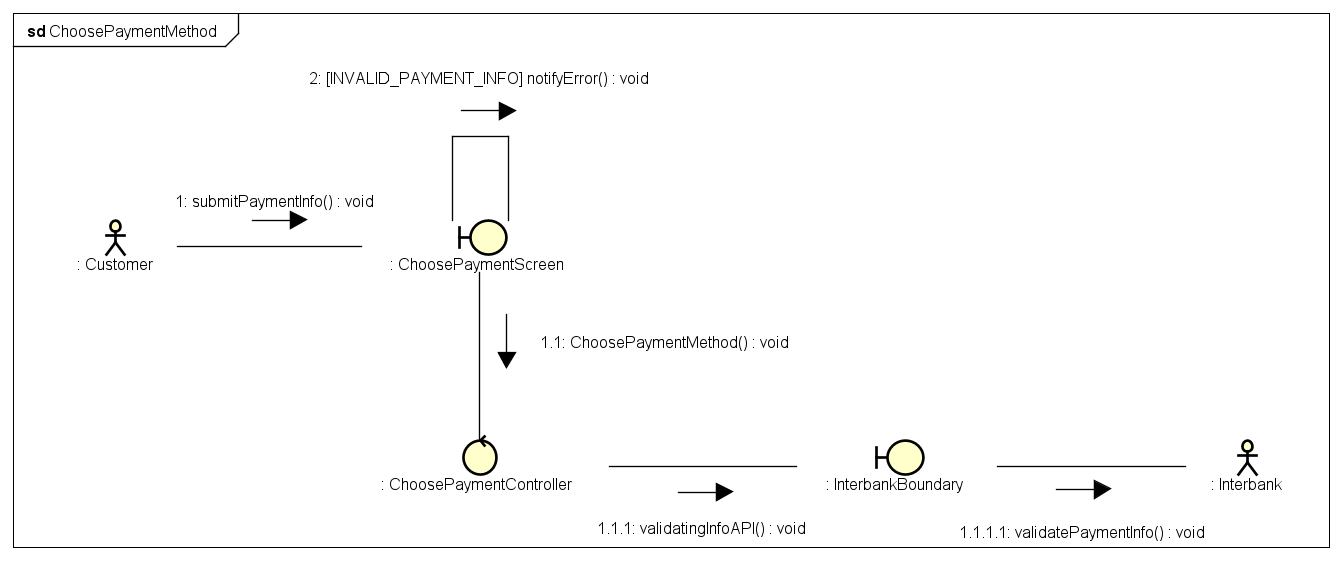
We choose this kind of architectural patterns because it’s simple, light, and can be scalable for this project and also MVC is a very well-known design pattern that developers normally use for this kind of project

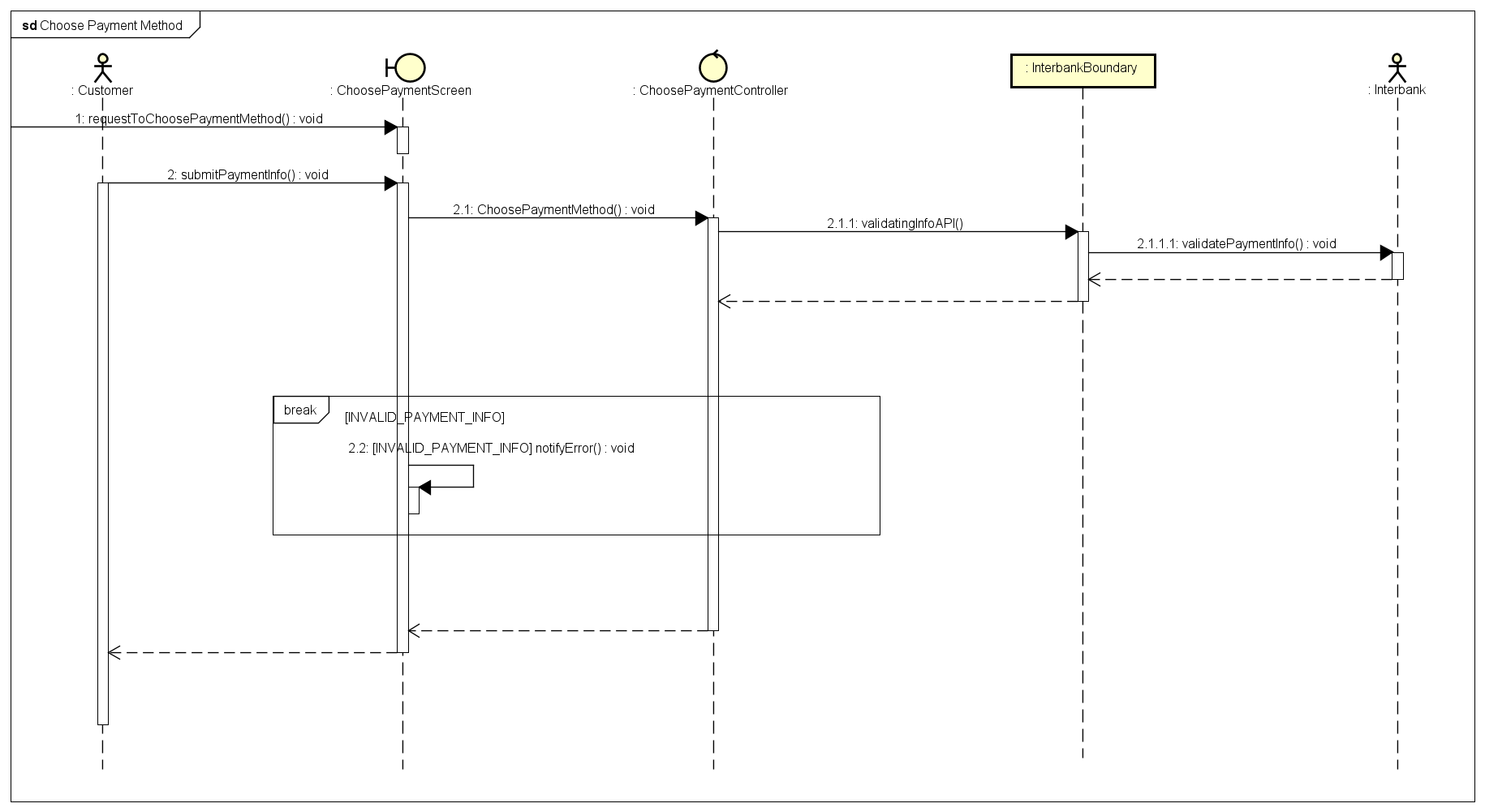
## Interaction Diagrams

*View All Bikes In Station sequence diagram + communication diagram*

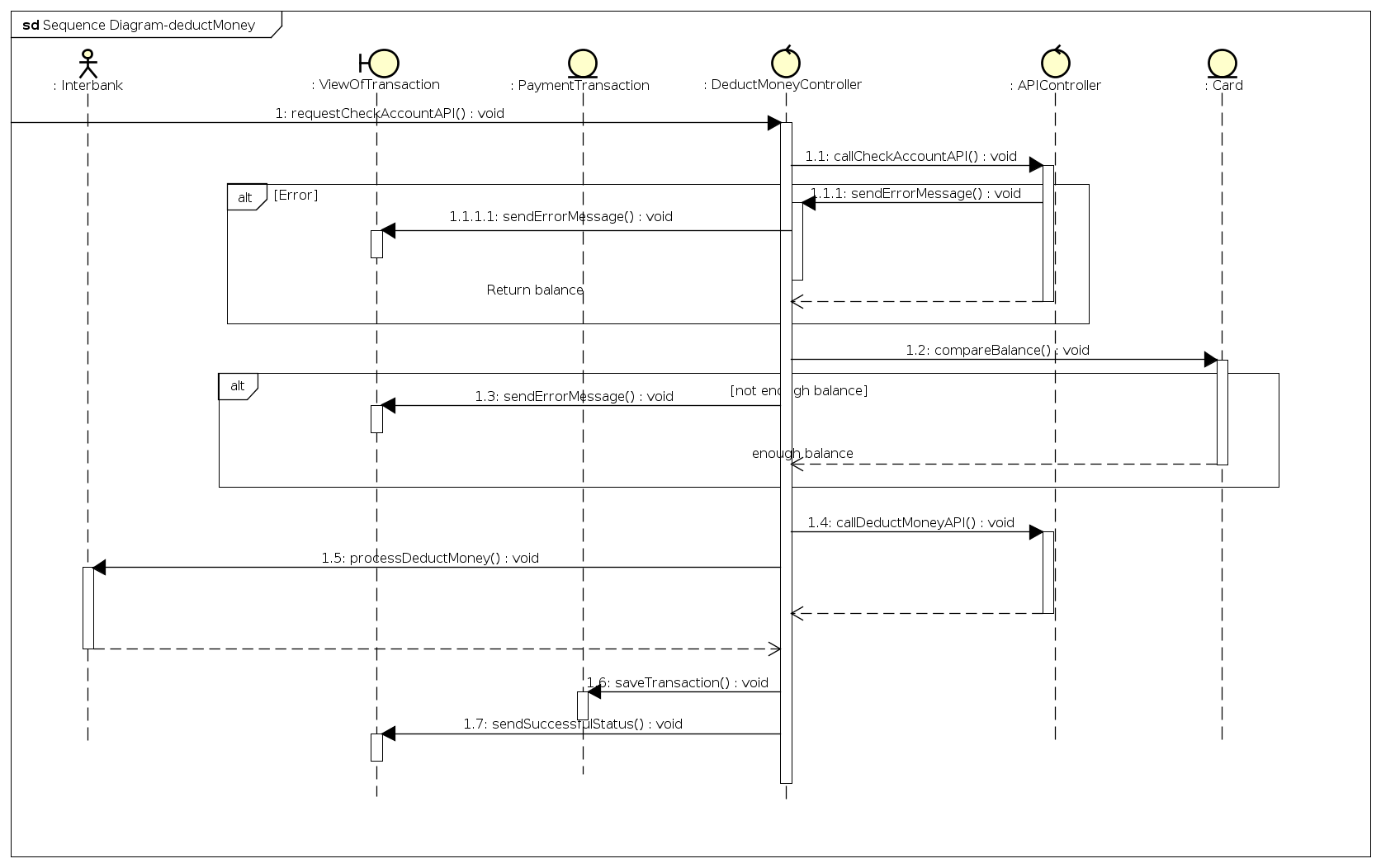
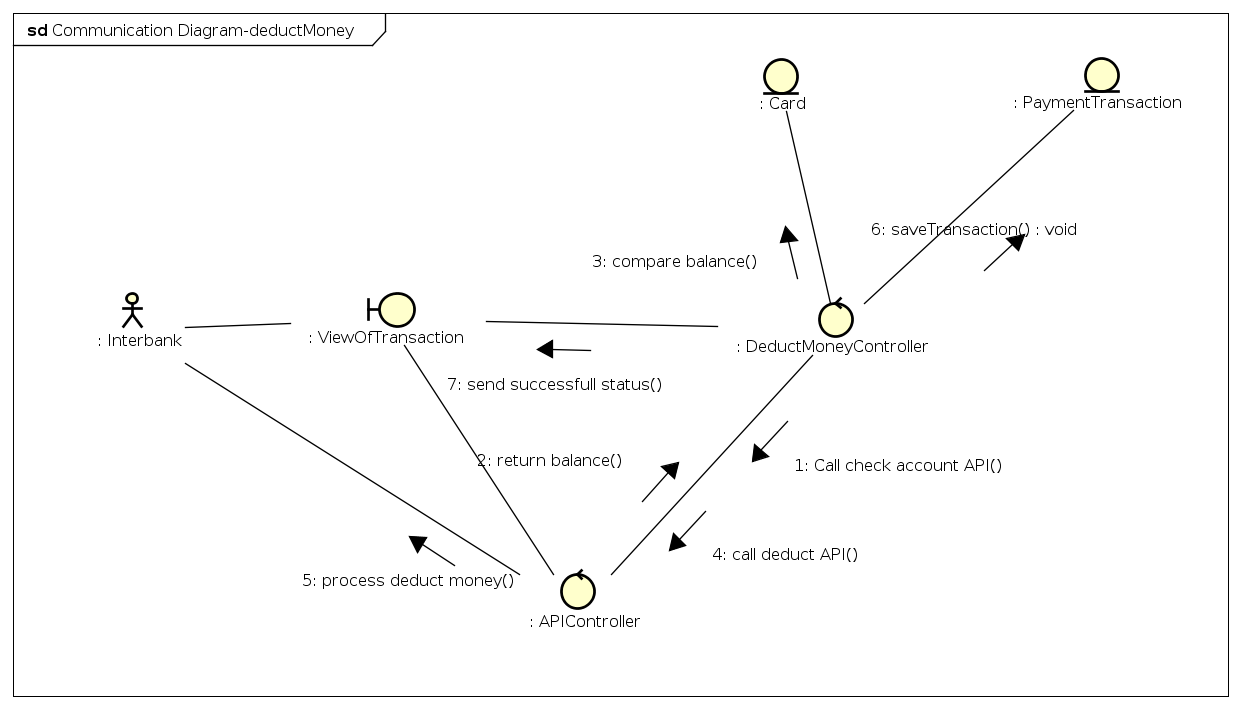


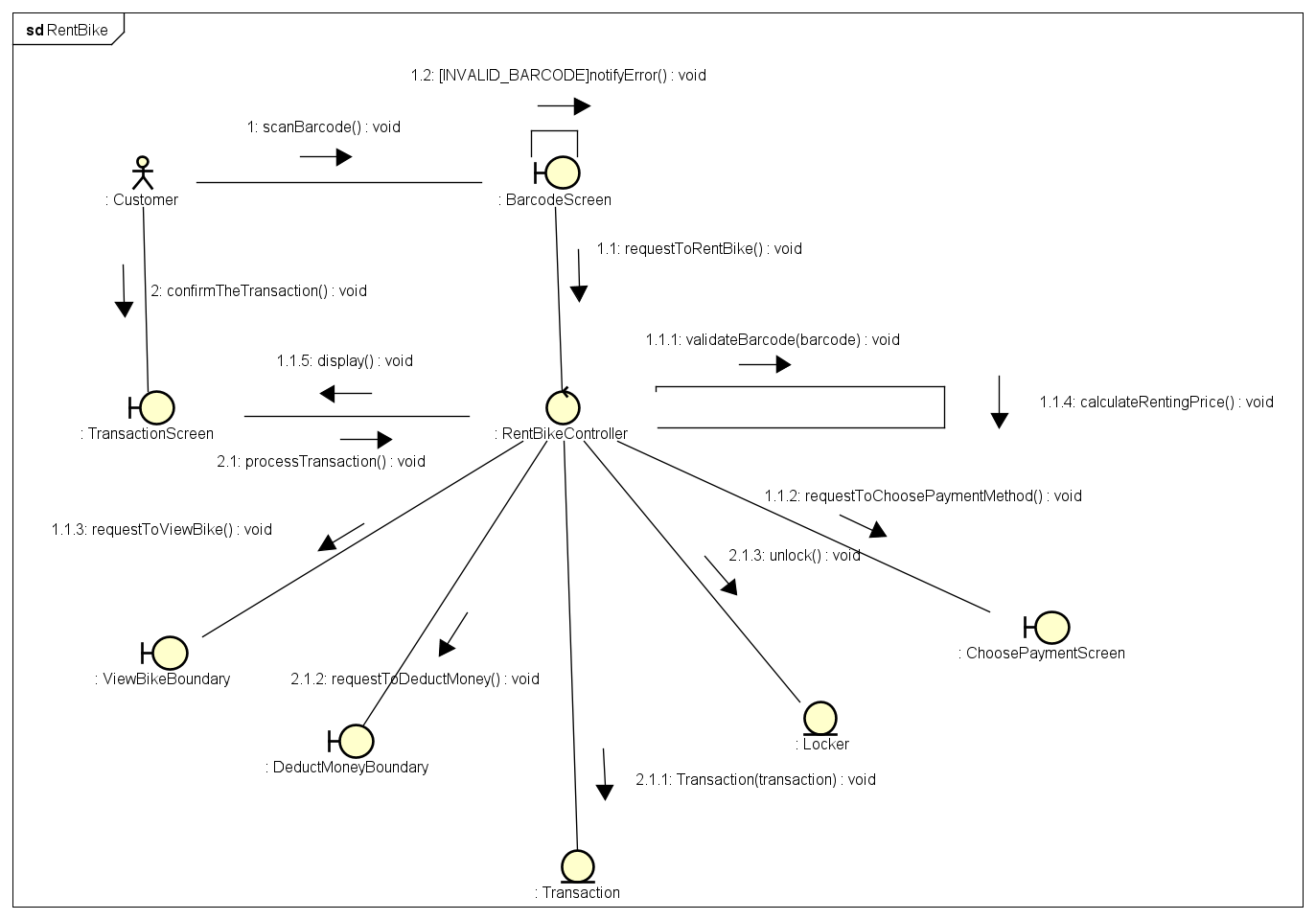
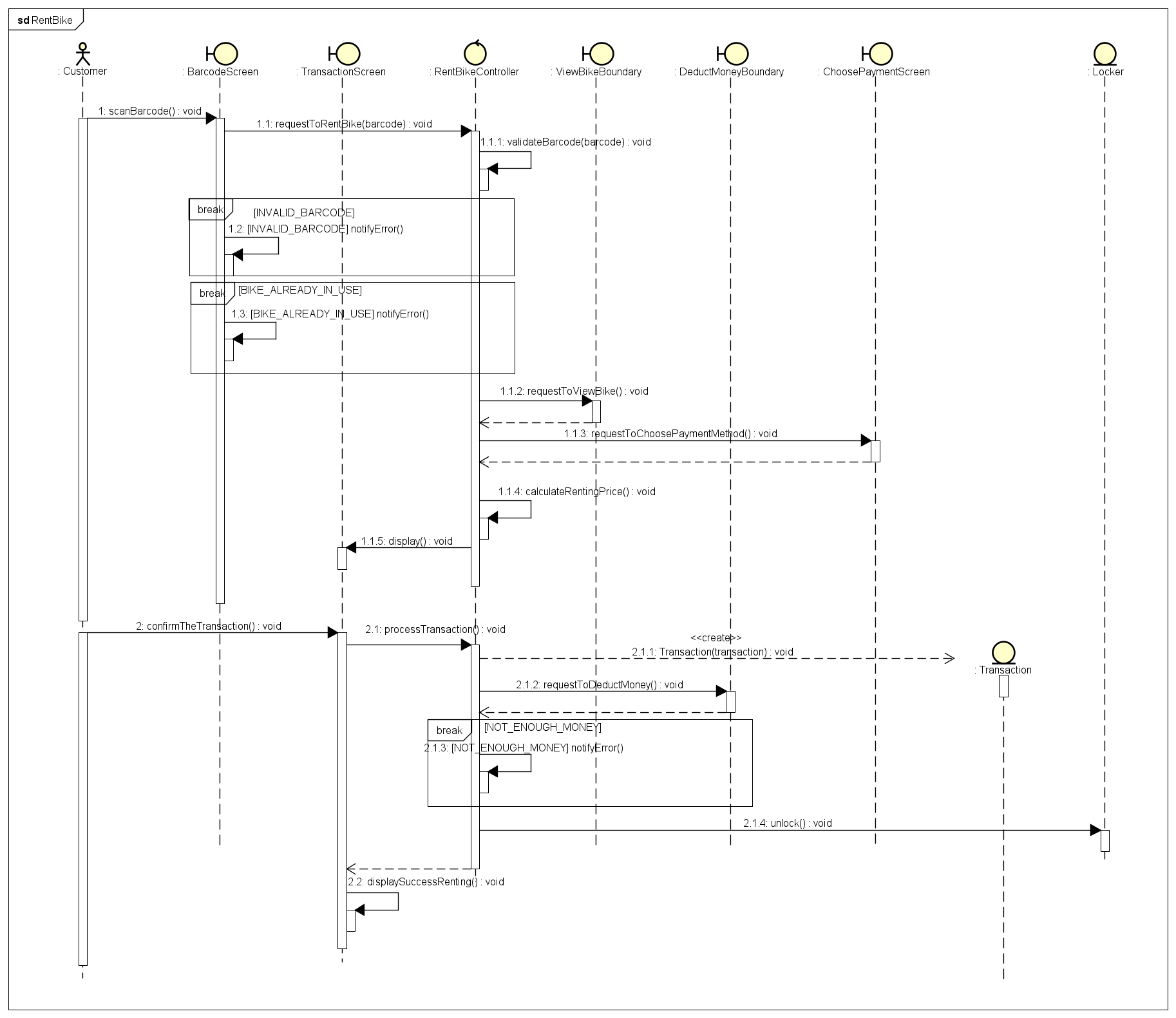
*choosePaymentMethod sequence diagram + communication diagram*



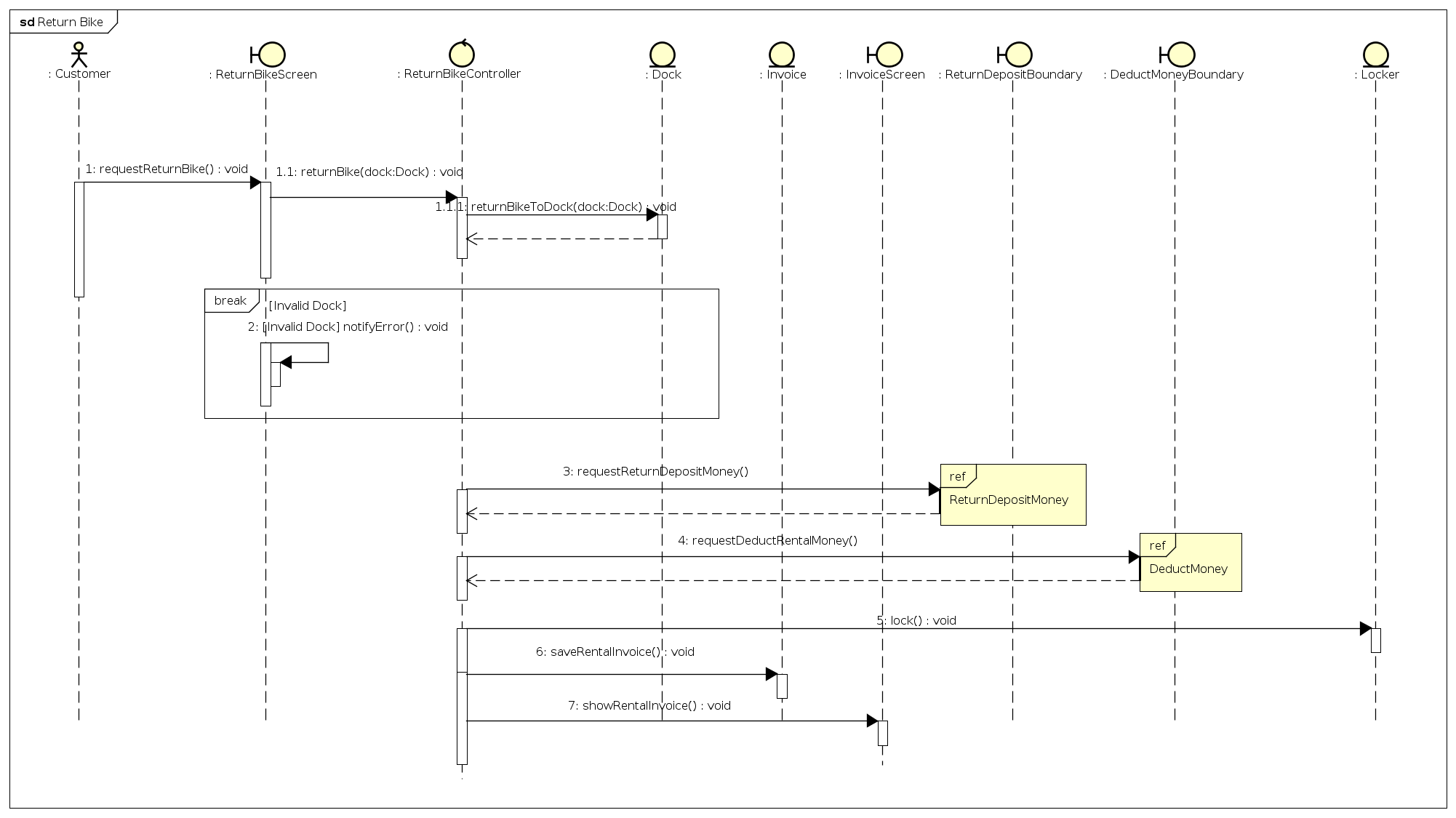
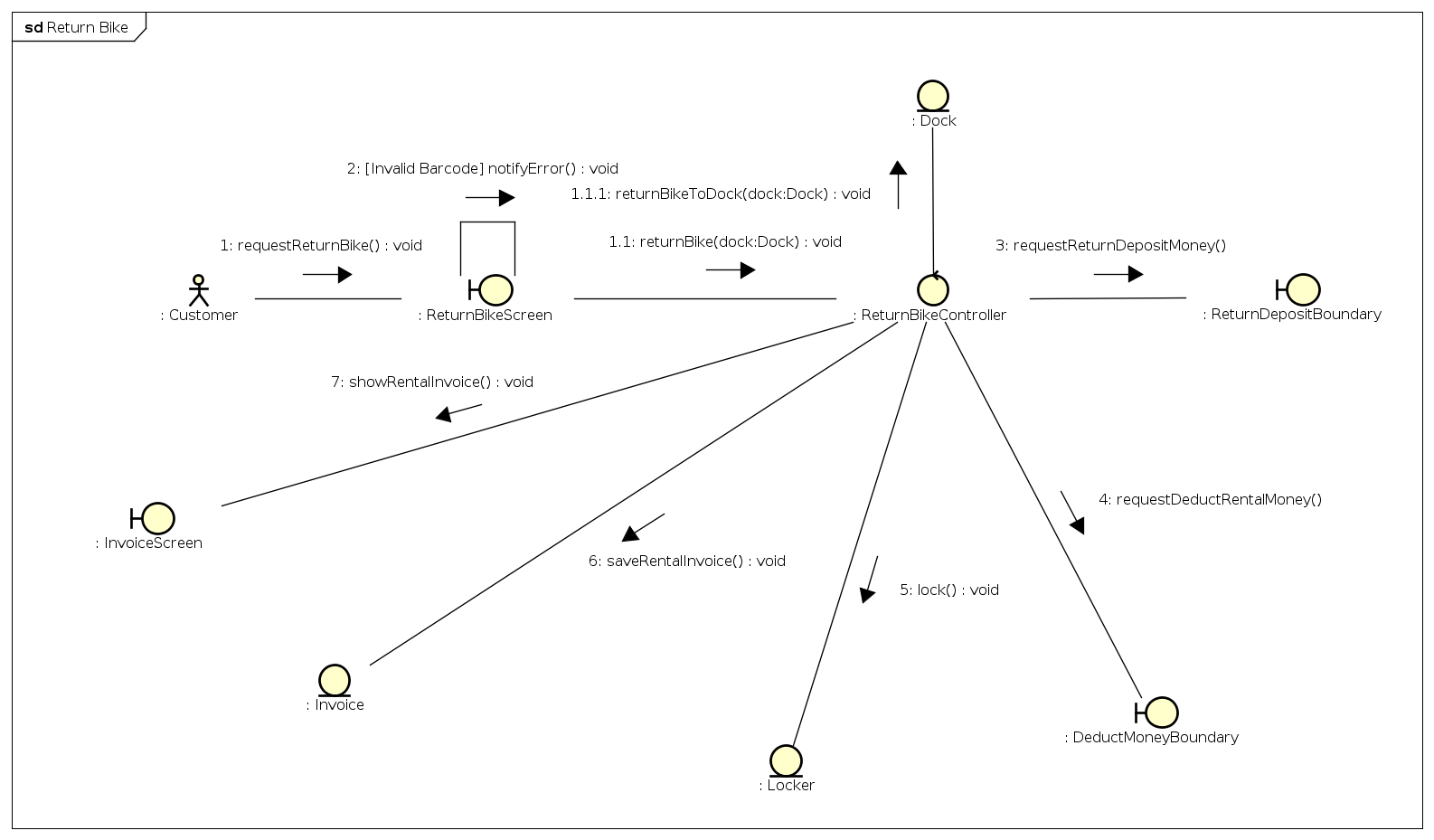


*deductMoney sequence diagram + communication diagram*

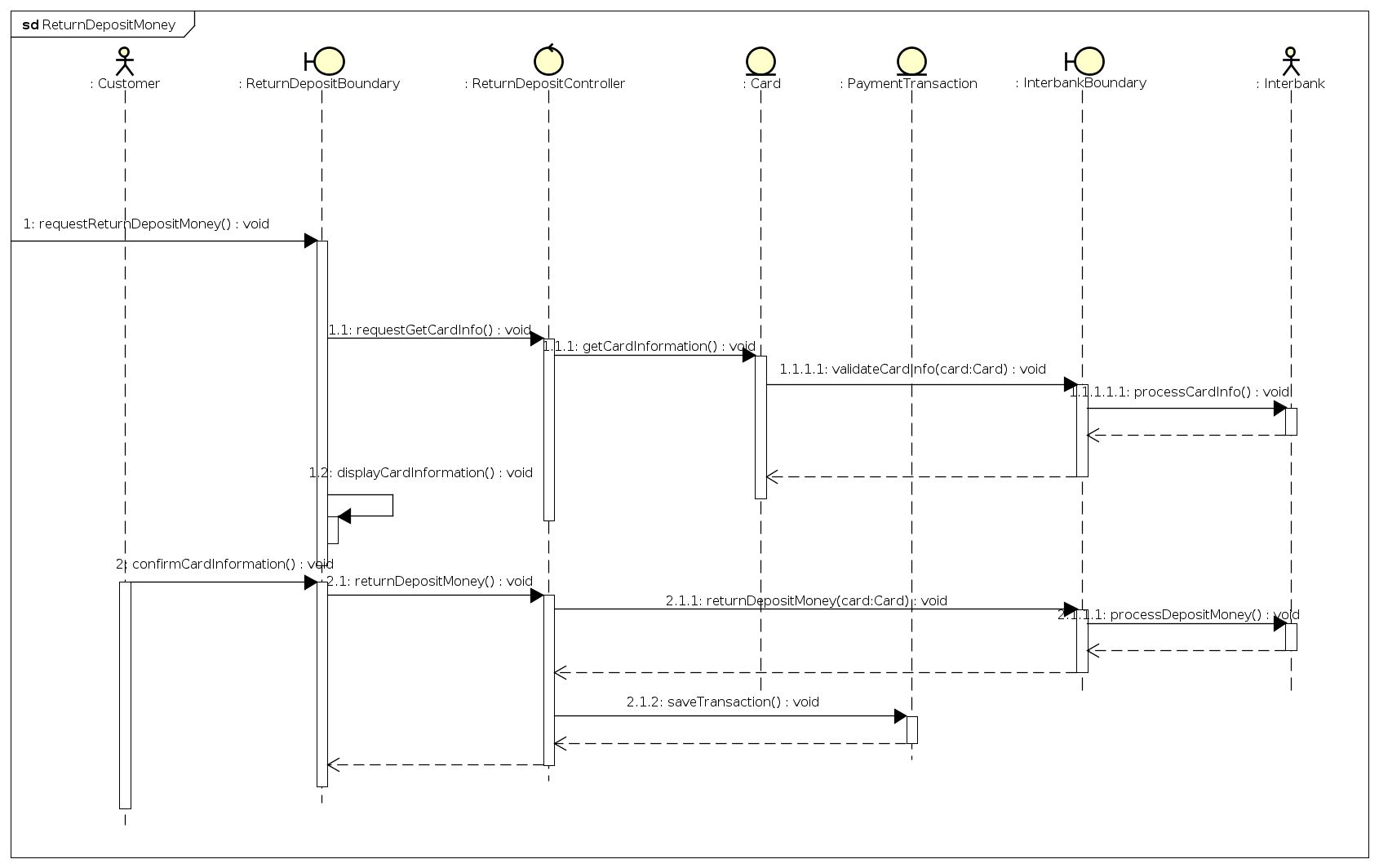
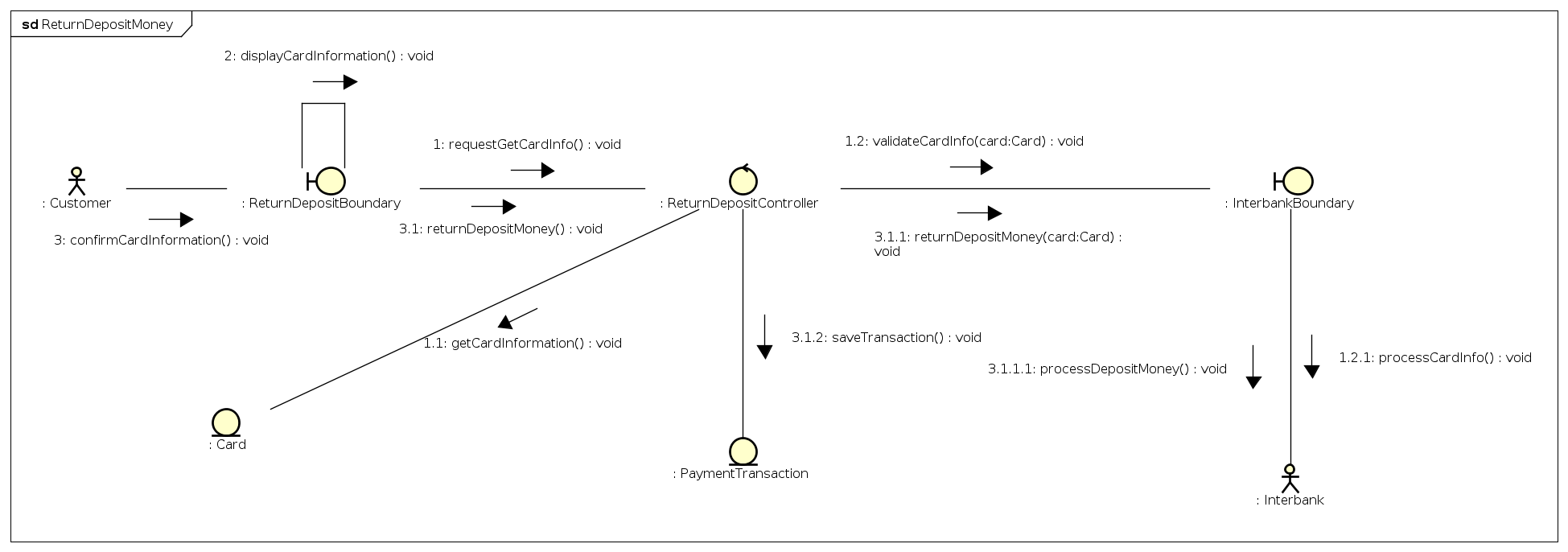
**

*Renting Bikes sequence diagram + communication diagram*

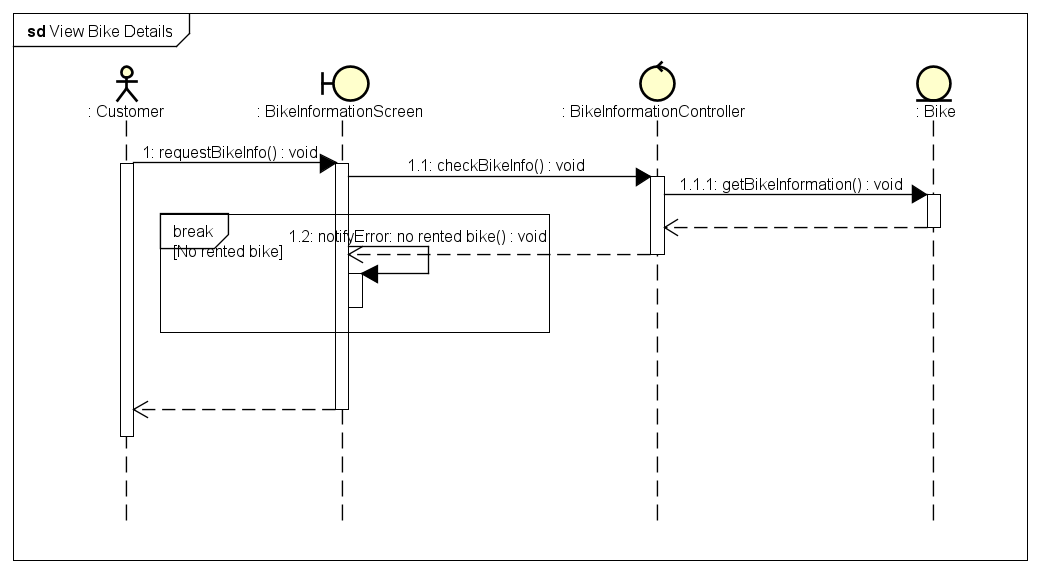
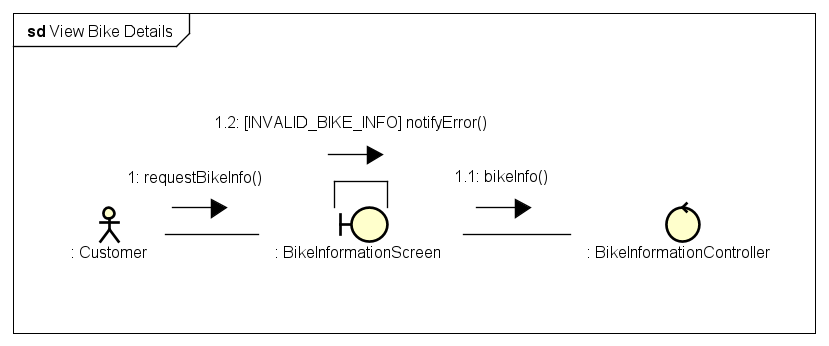
*Return Bikes sequence diagram + communication diagram*

**

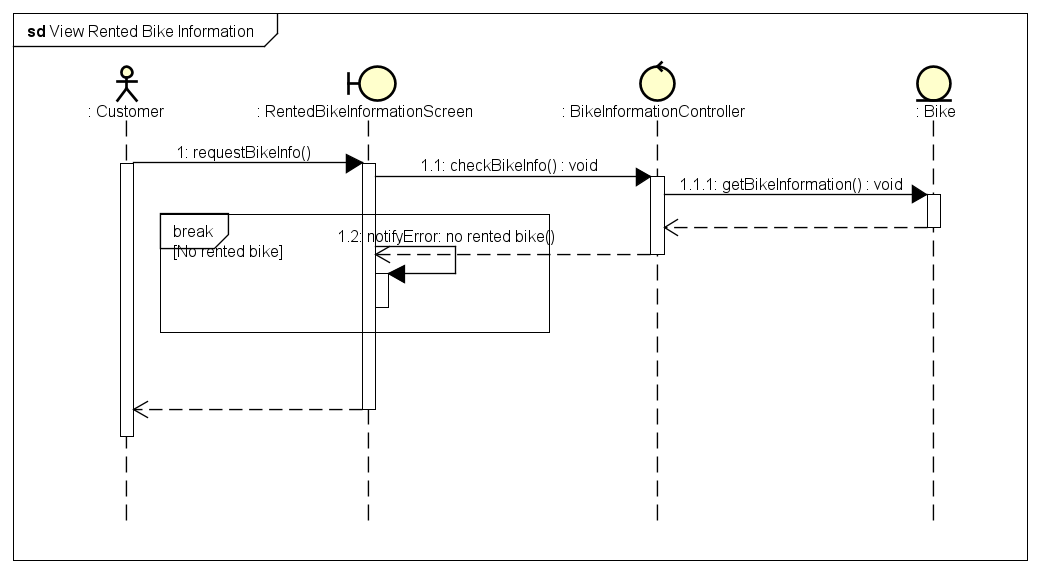
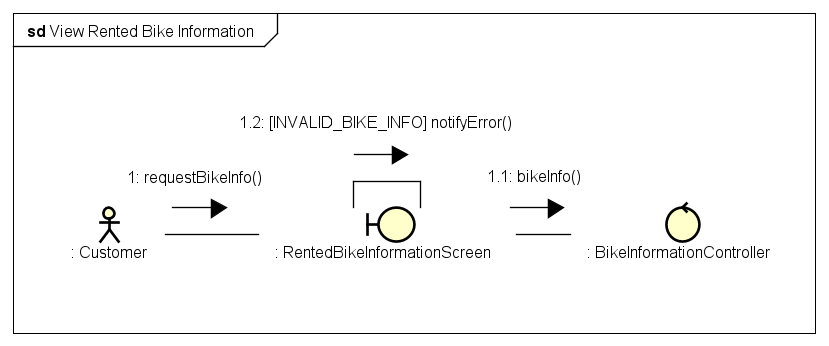
*Return Deposit Money sequence diagram + communication diagram*

**

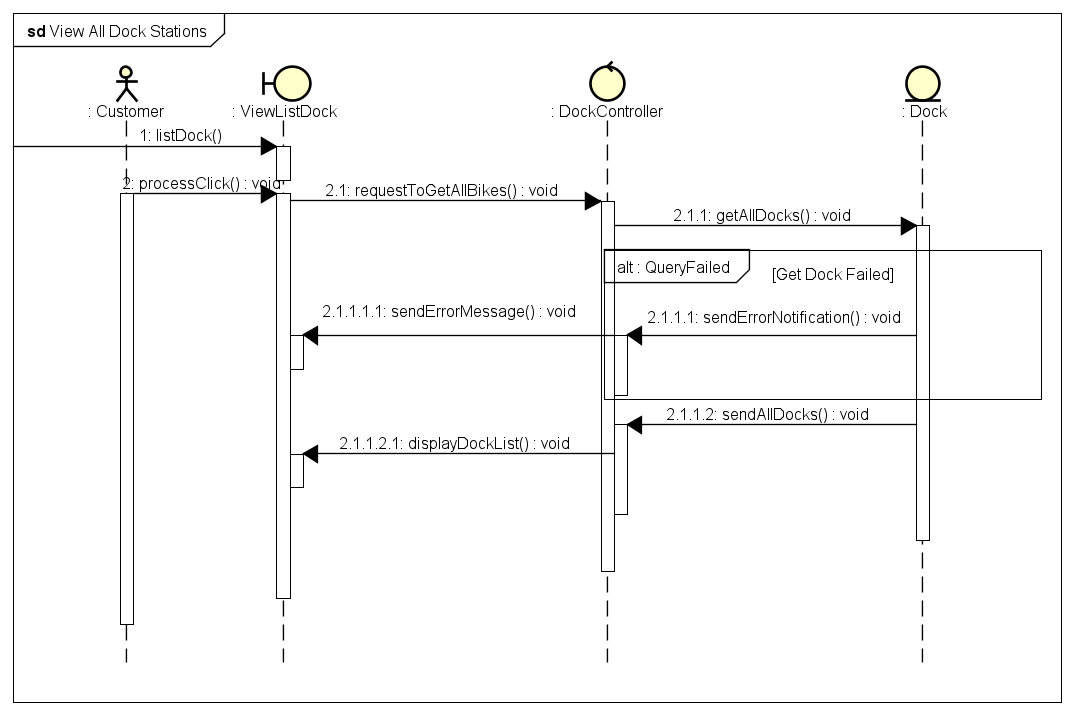
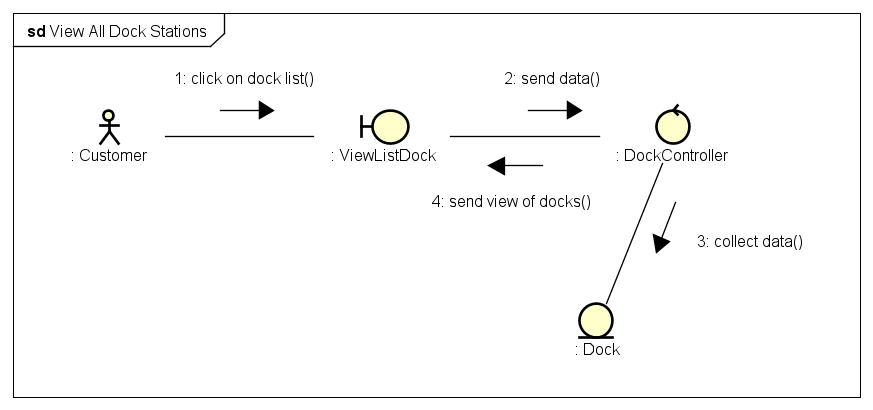
*View Bike Details sequence diagram + communication diagram*

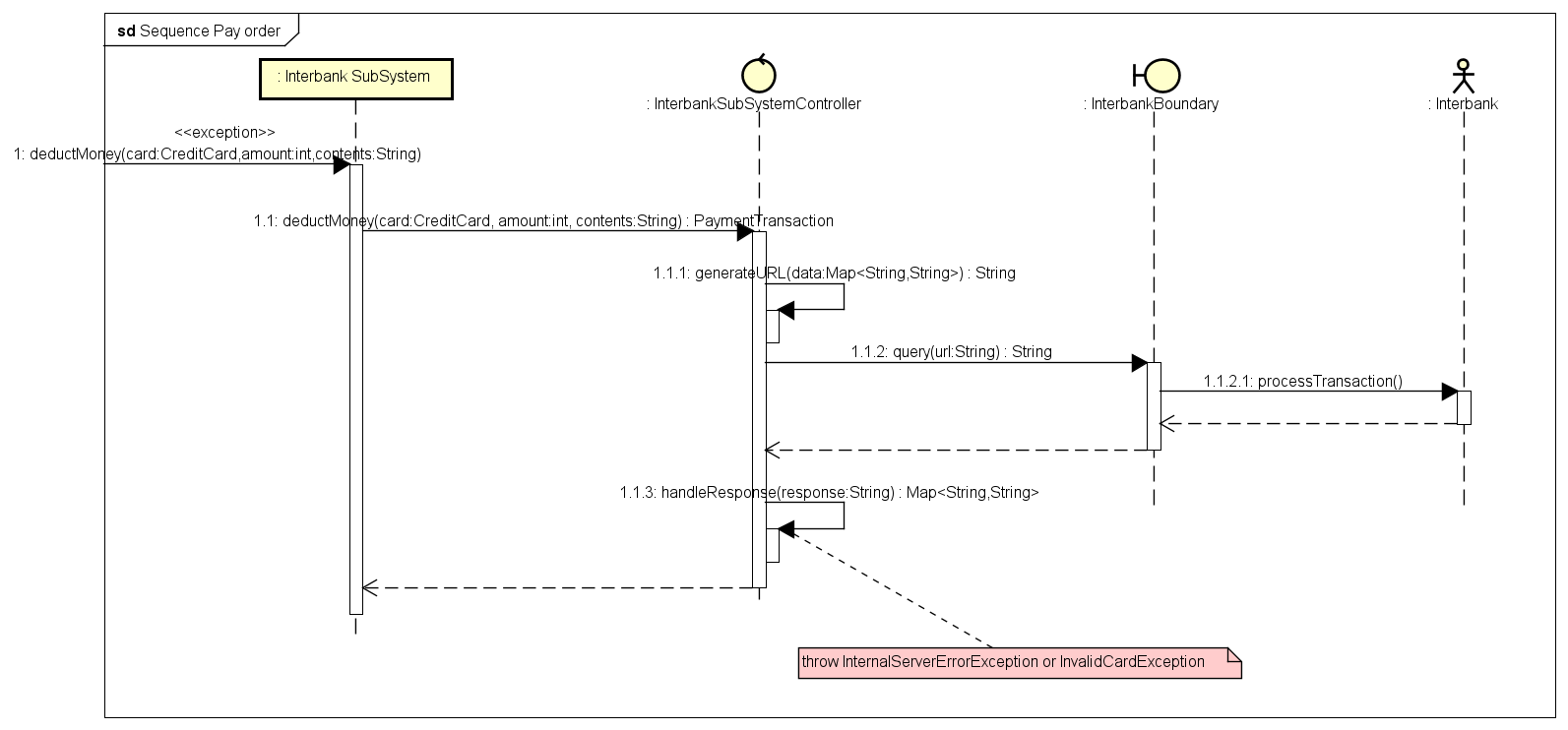
**

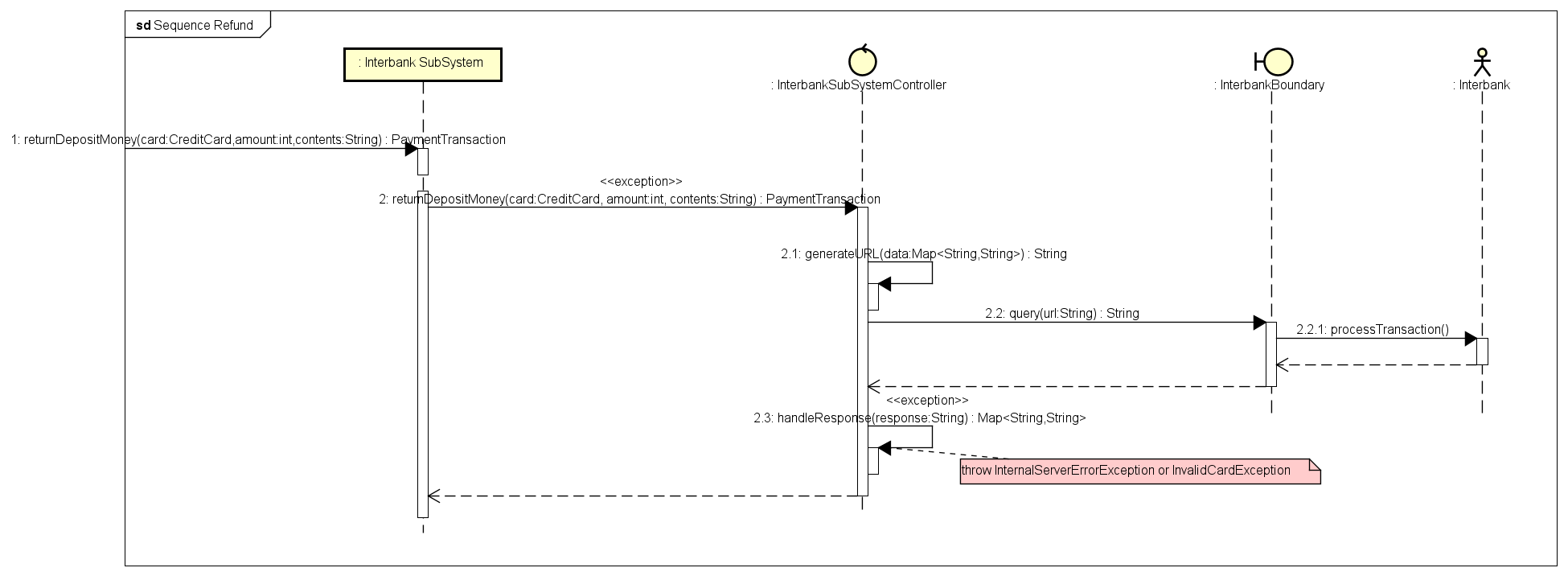
*View Rented Bikes Information sequence diagram + communication diagram*

**

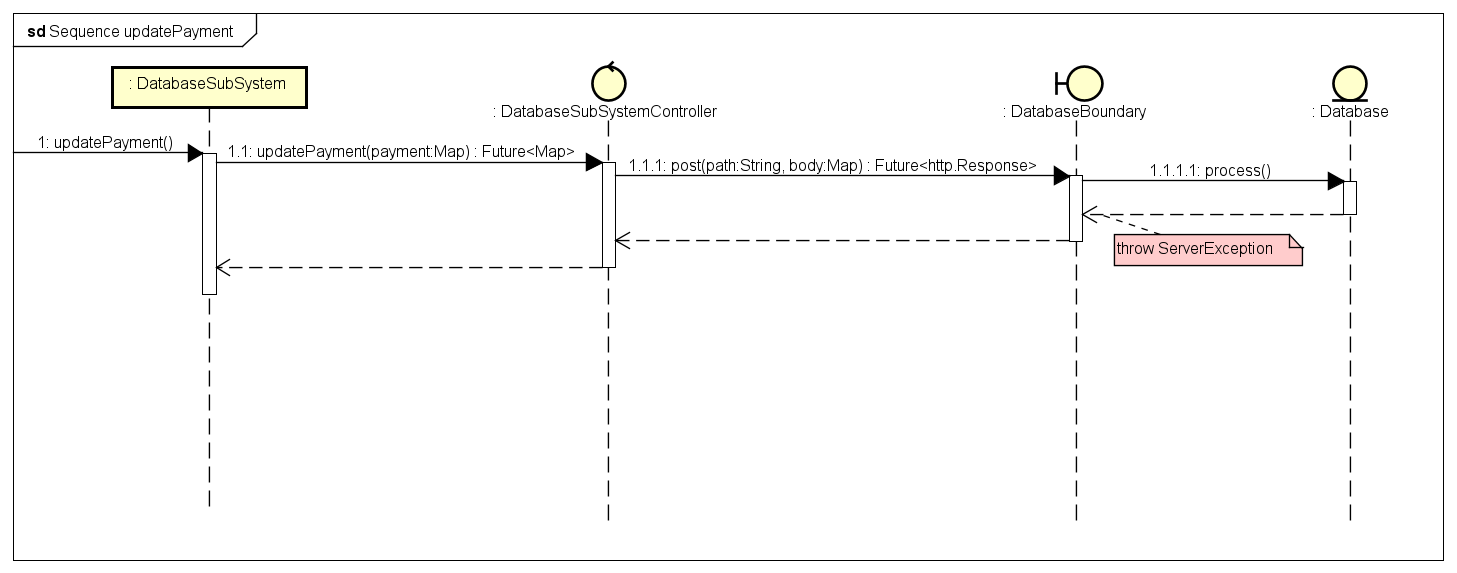
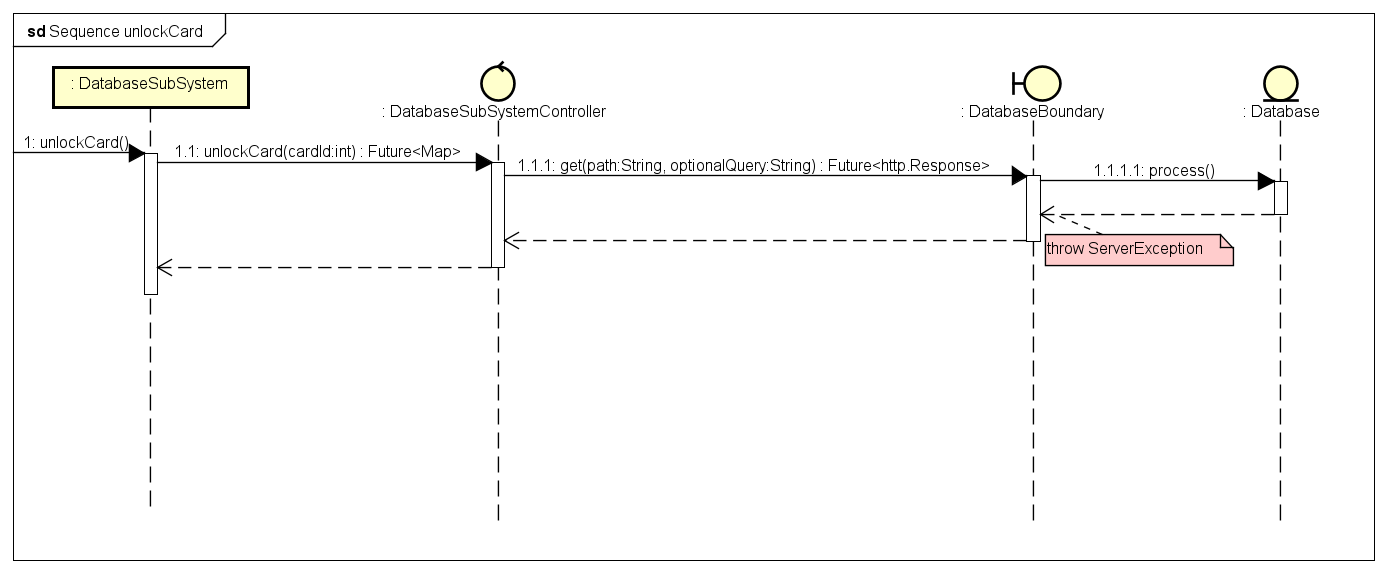
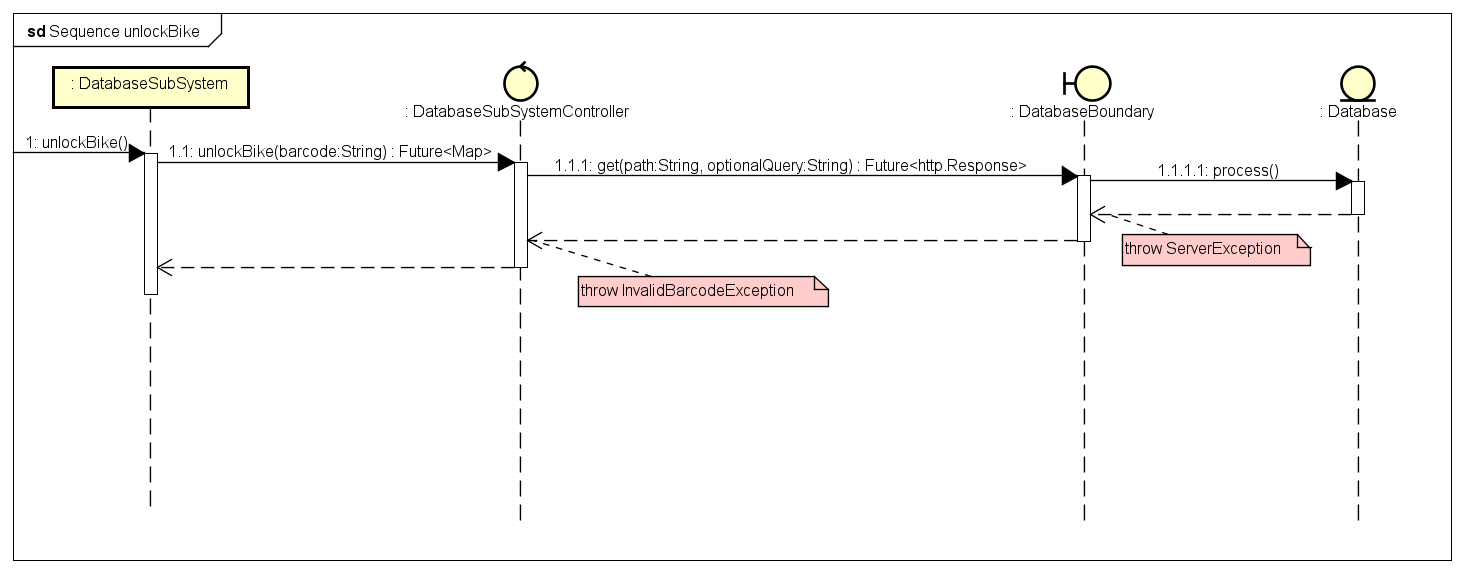
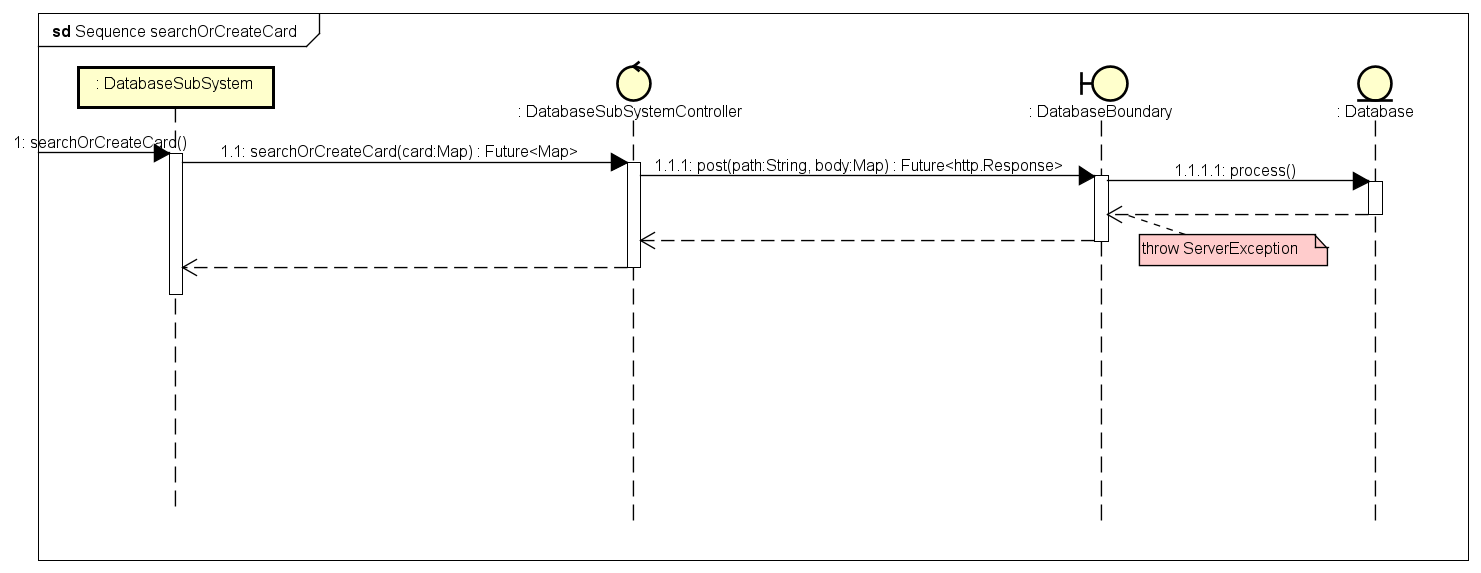
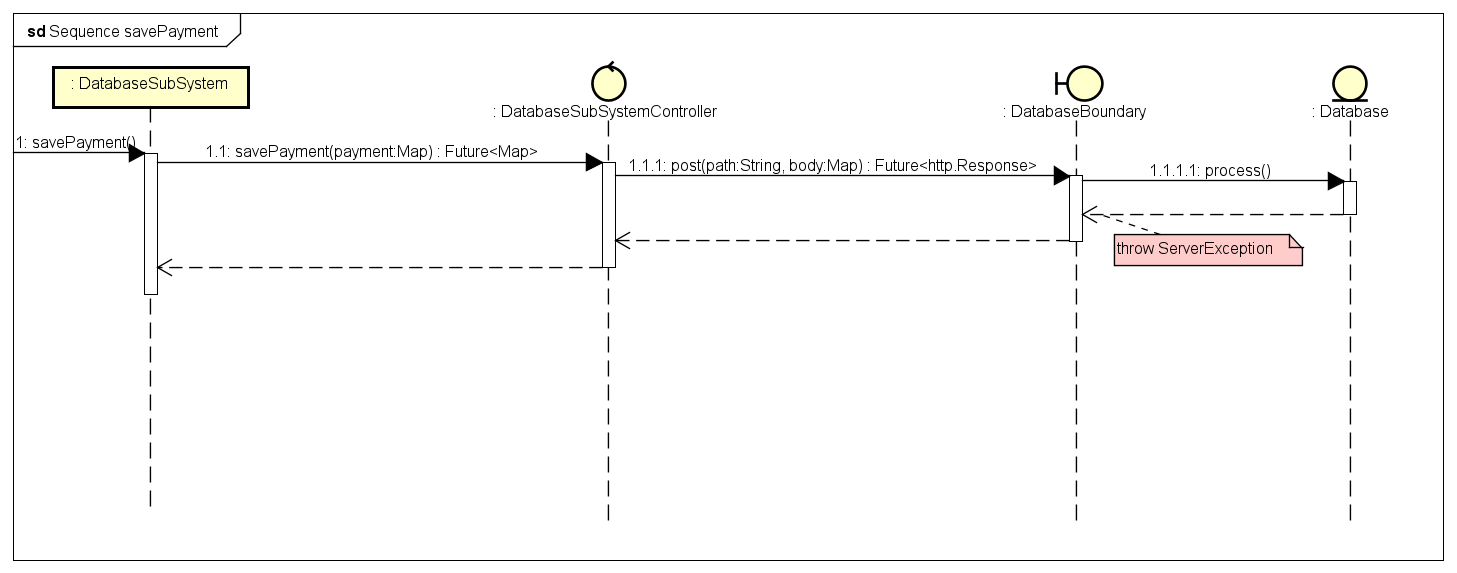
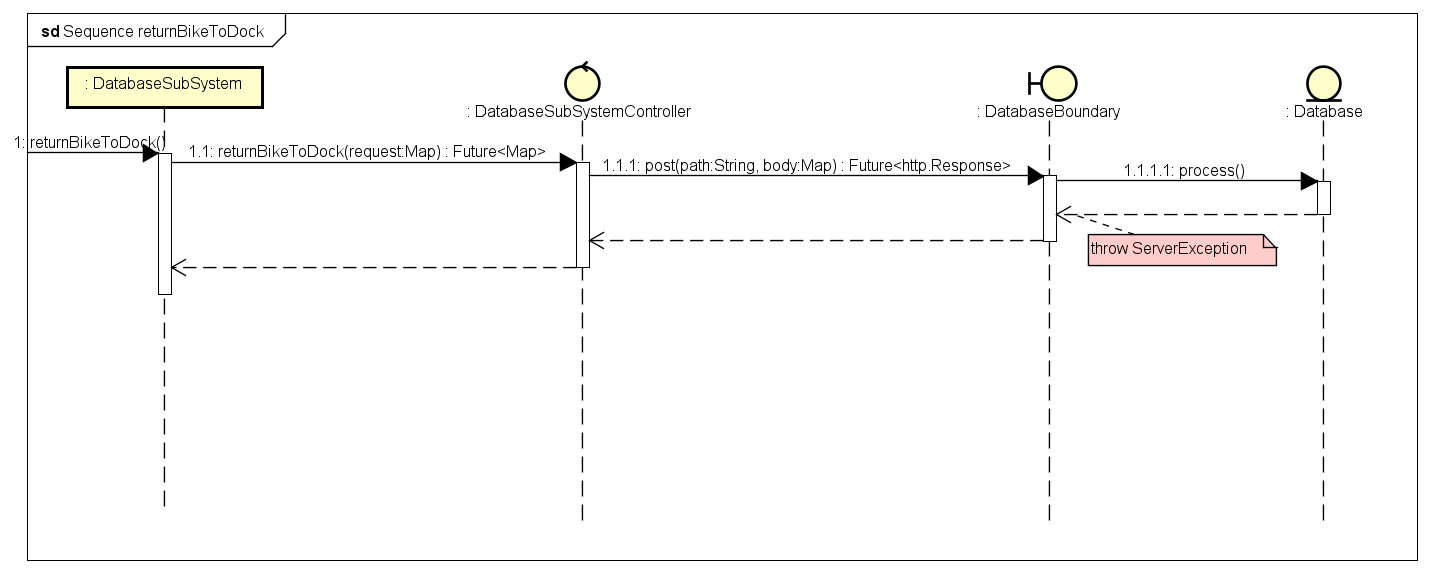
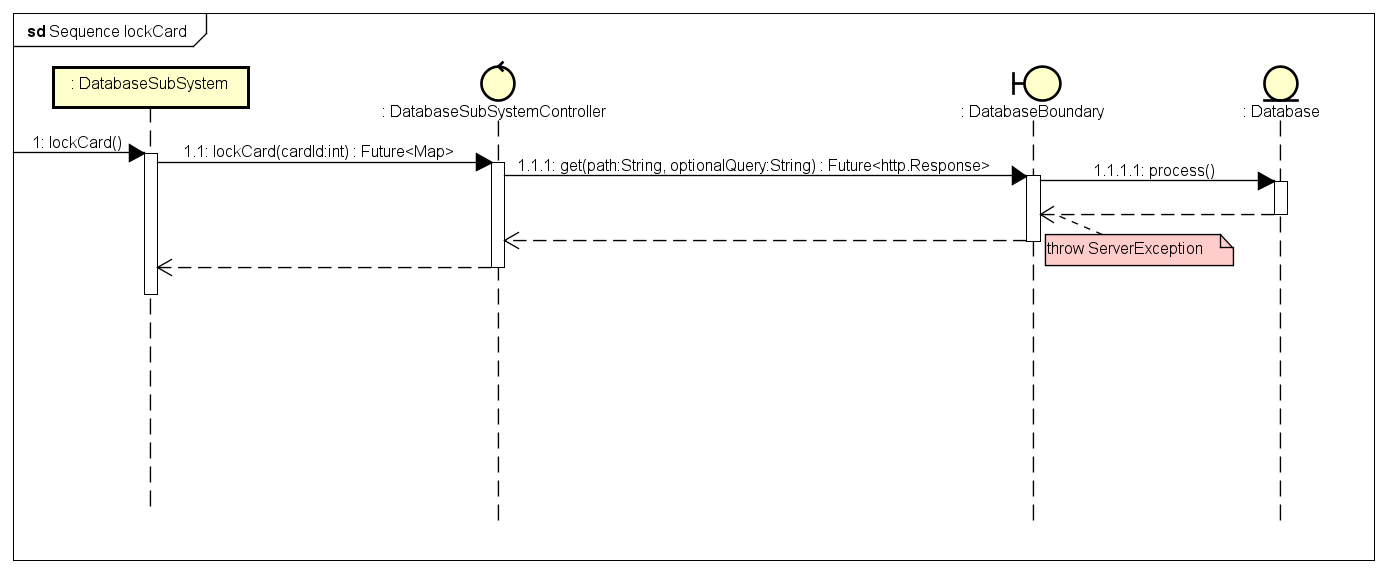
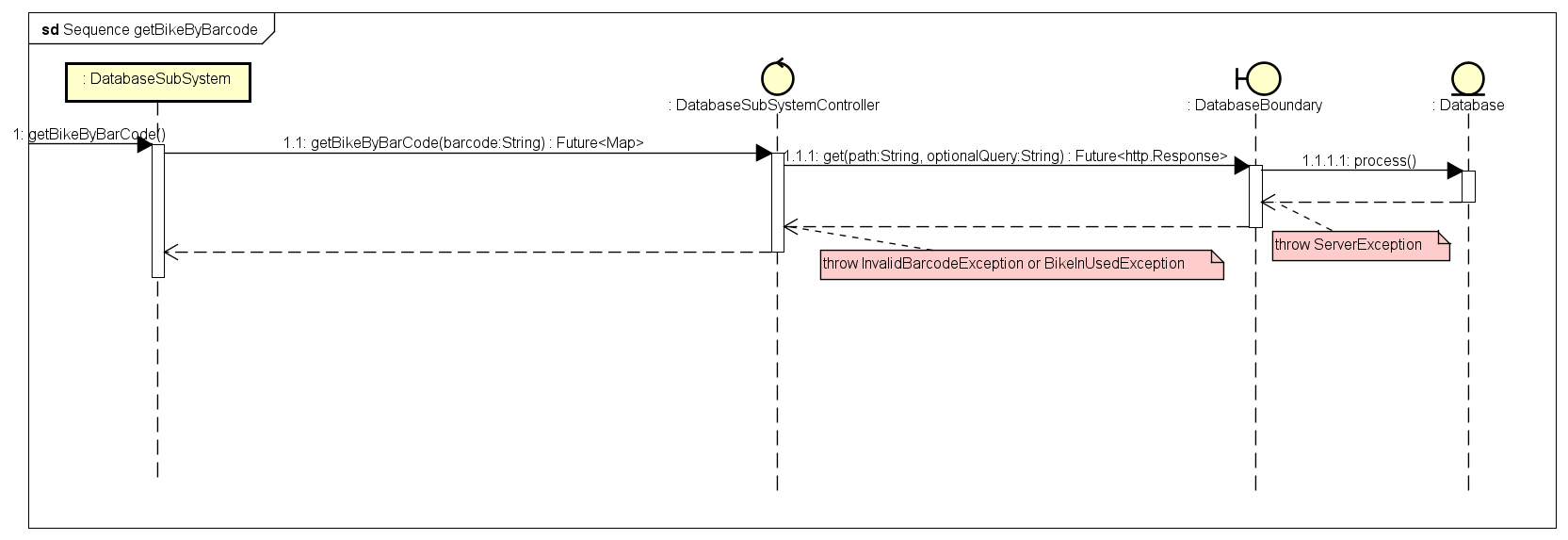
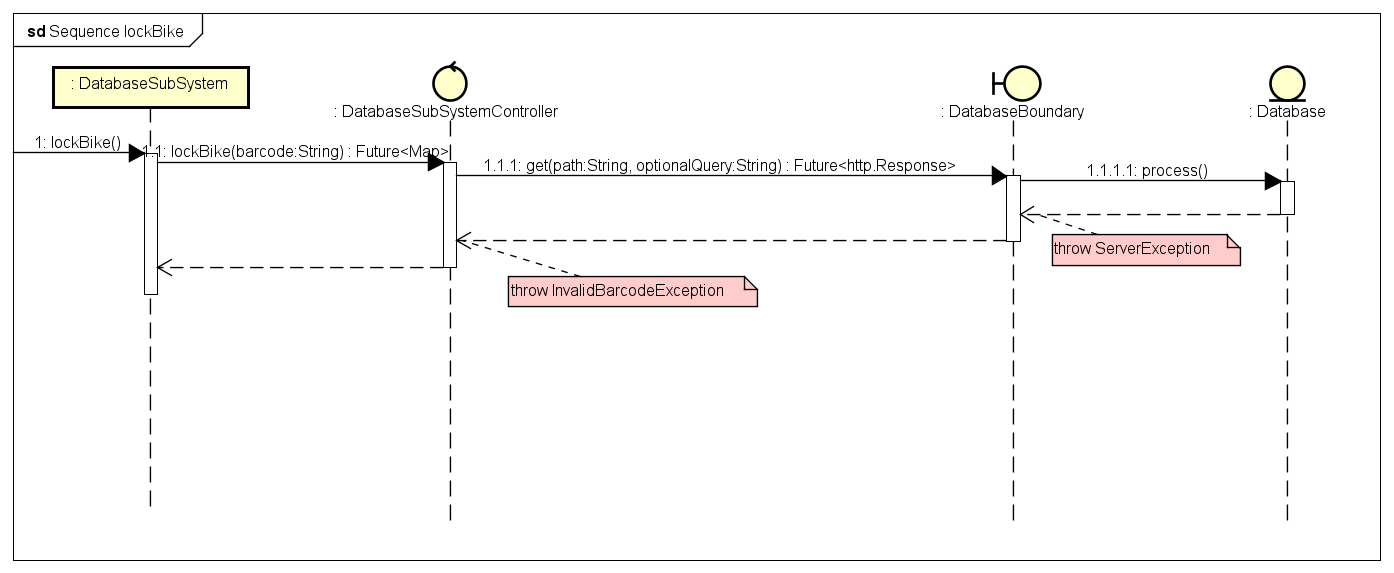
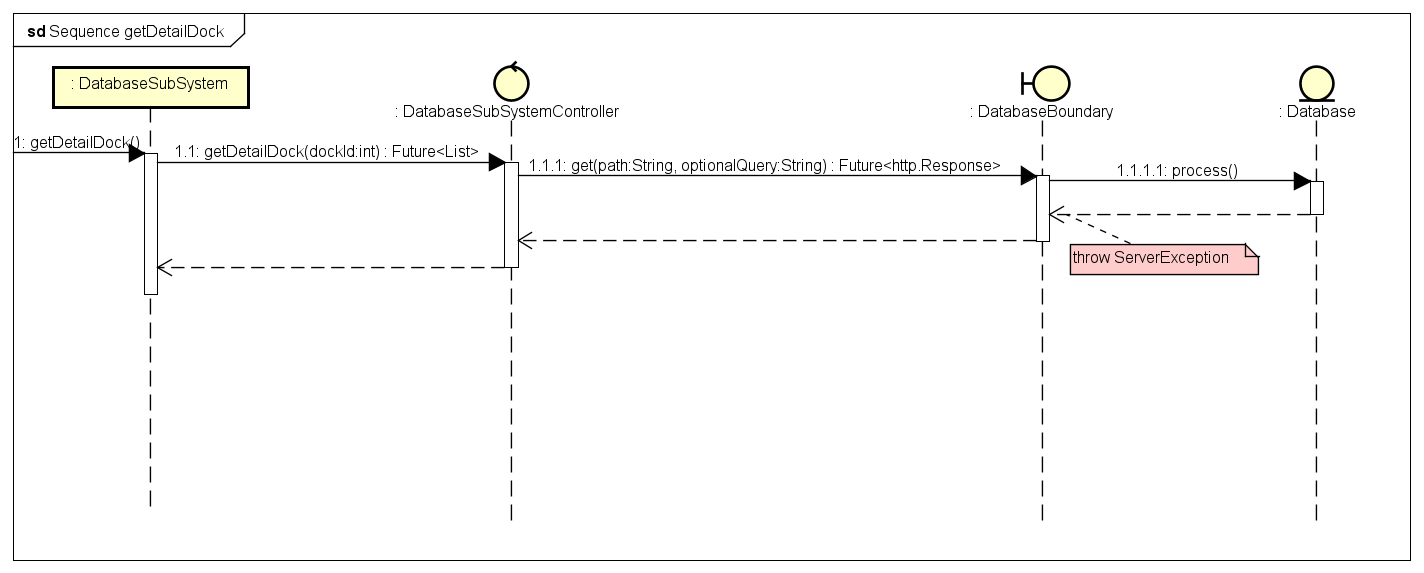
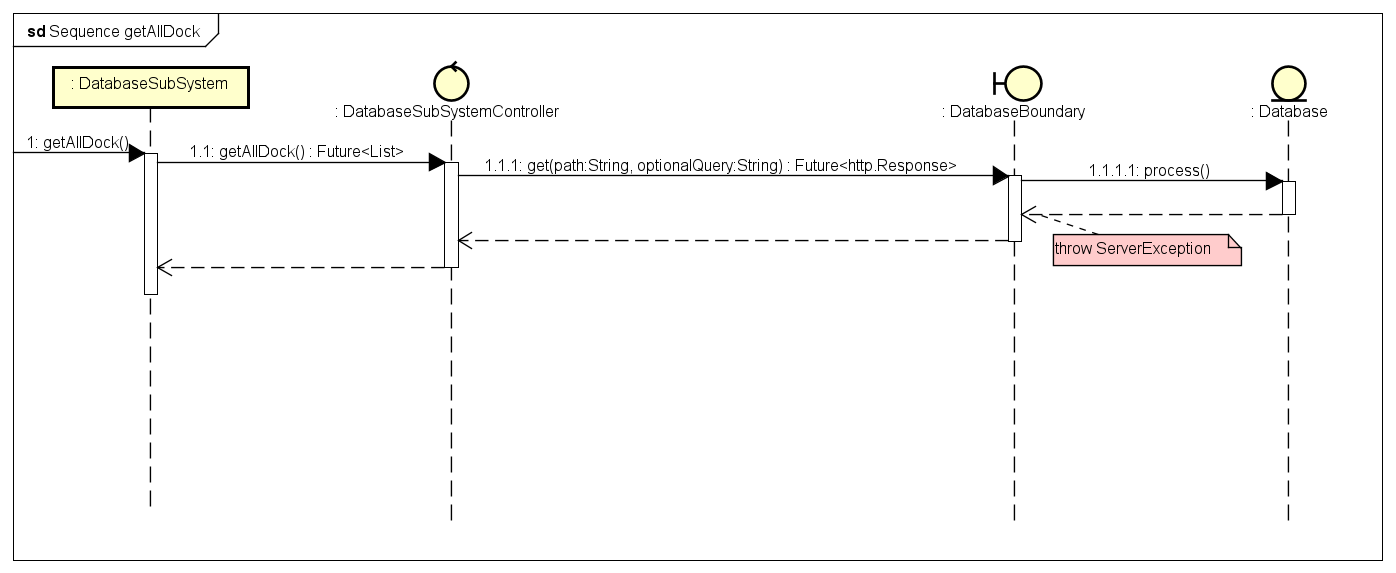
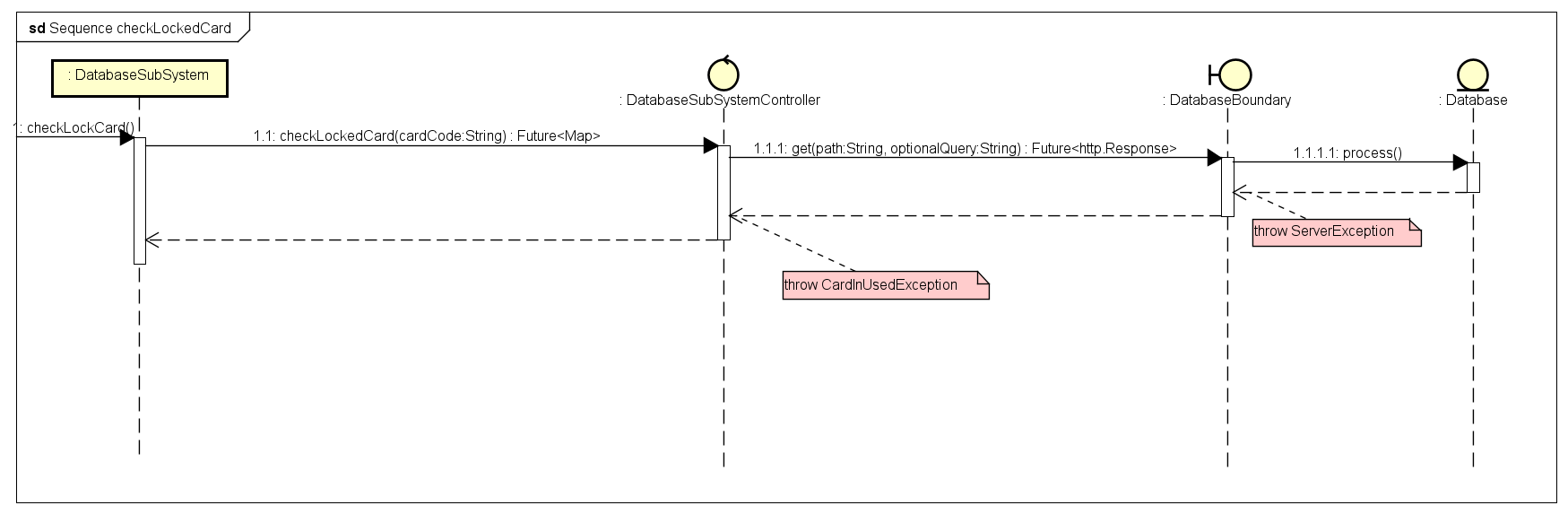
*View All Dock Station sequence diagram + communication diagram*

**

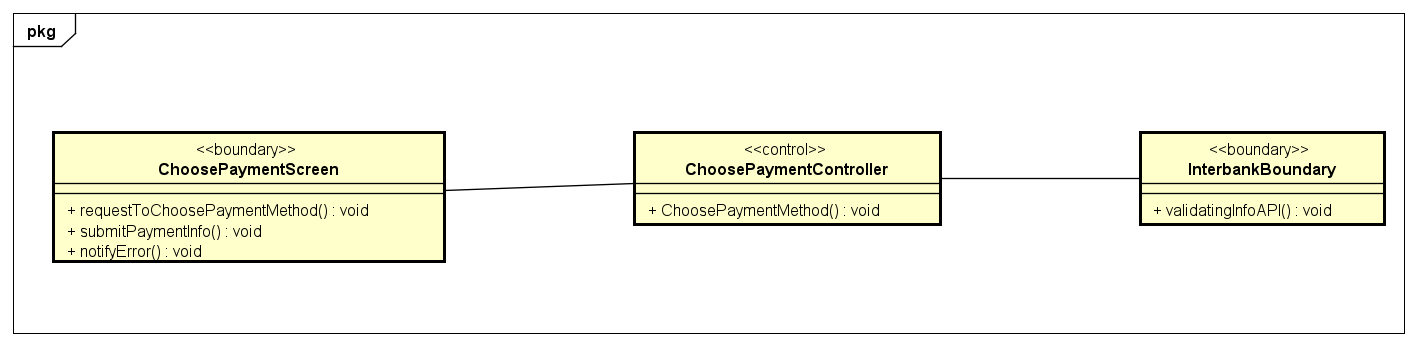
*InterbankSubSystem sequence diagrams*



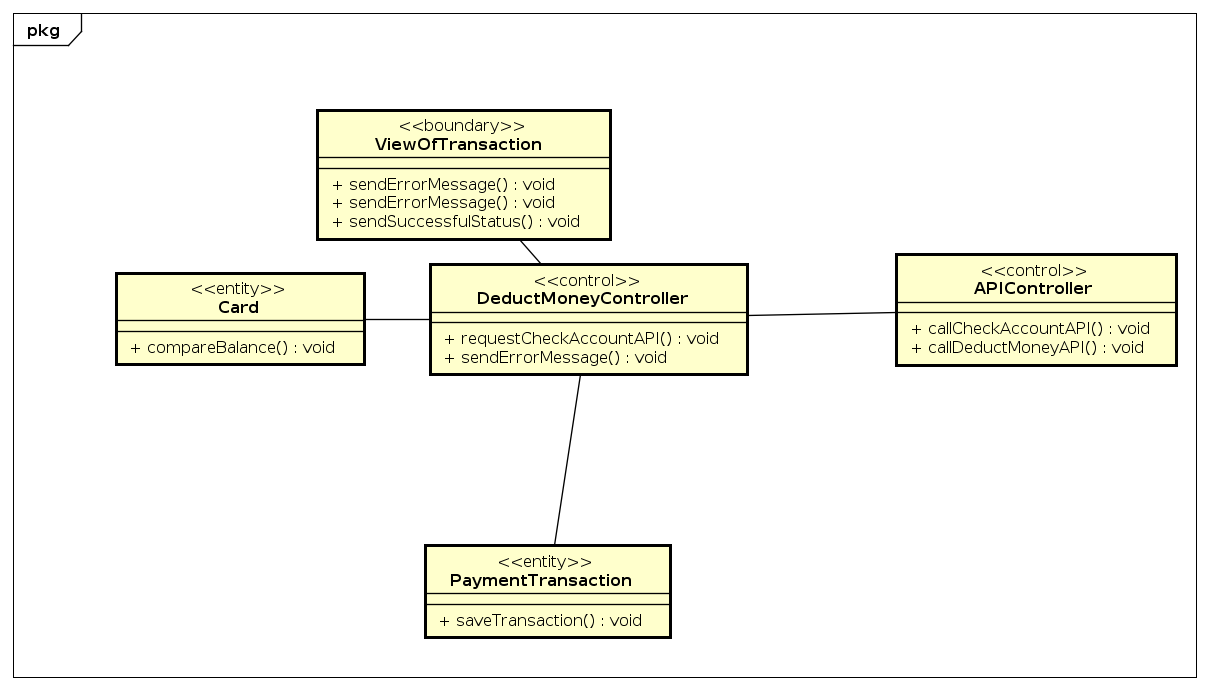
*DatabaseSubSystem sequence diagrams*

**

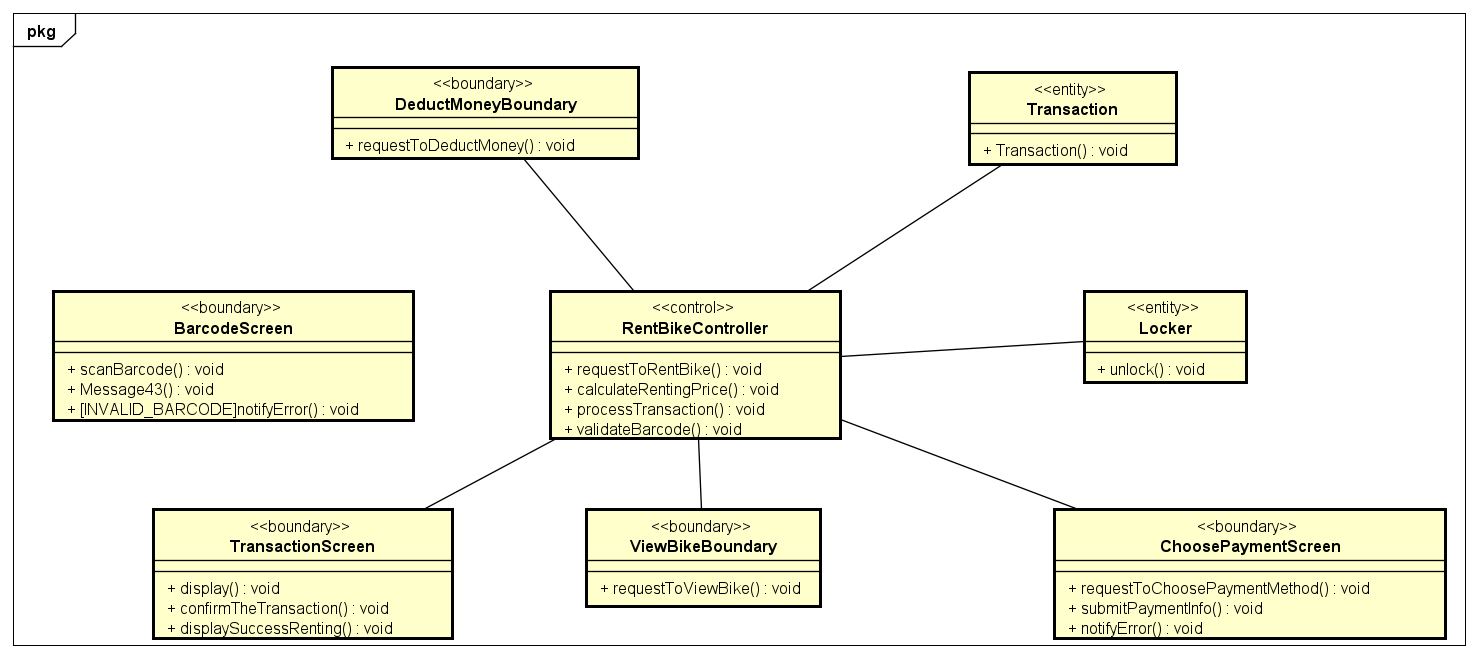
## Analysis Class Diagrams



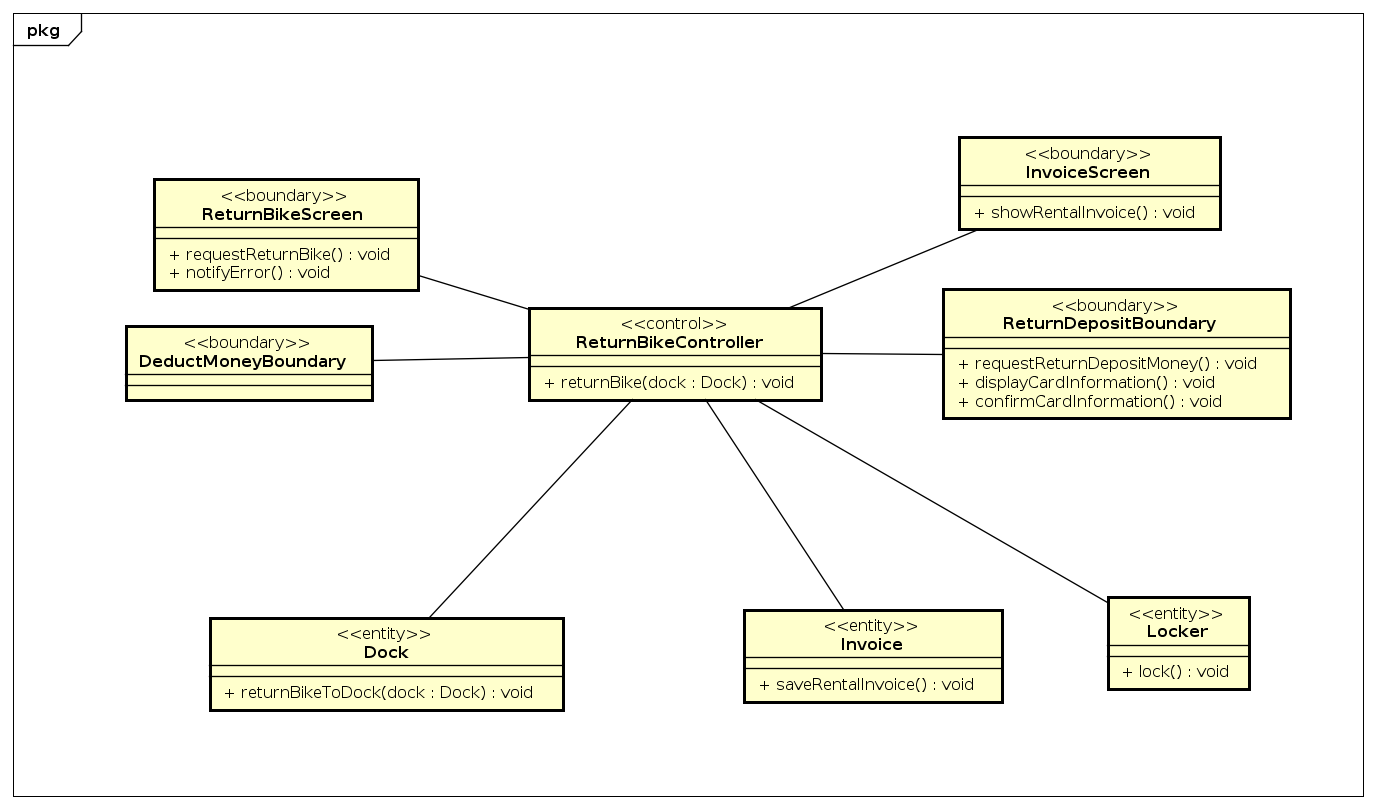
*figure 1: choosePaymentMethod class diagram*

**

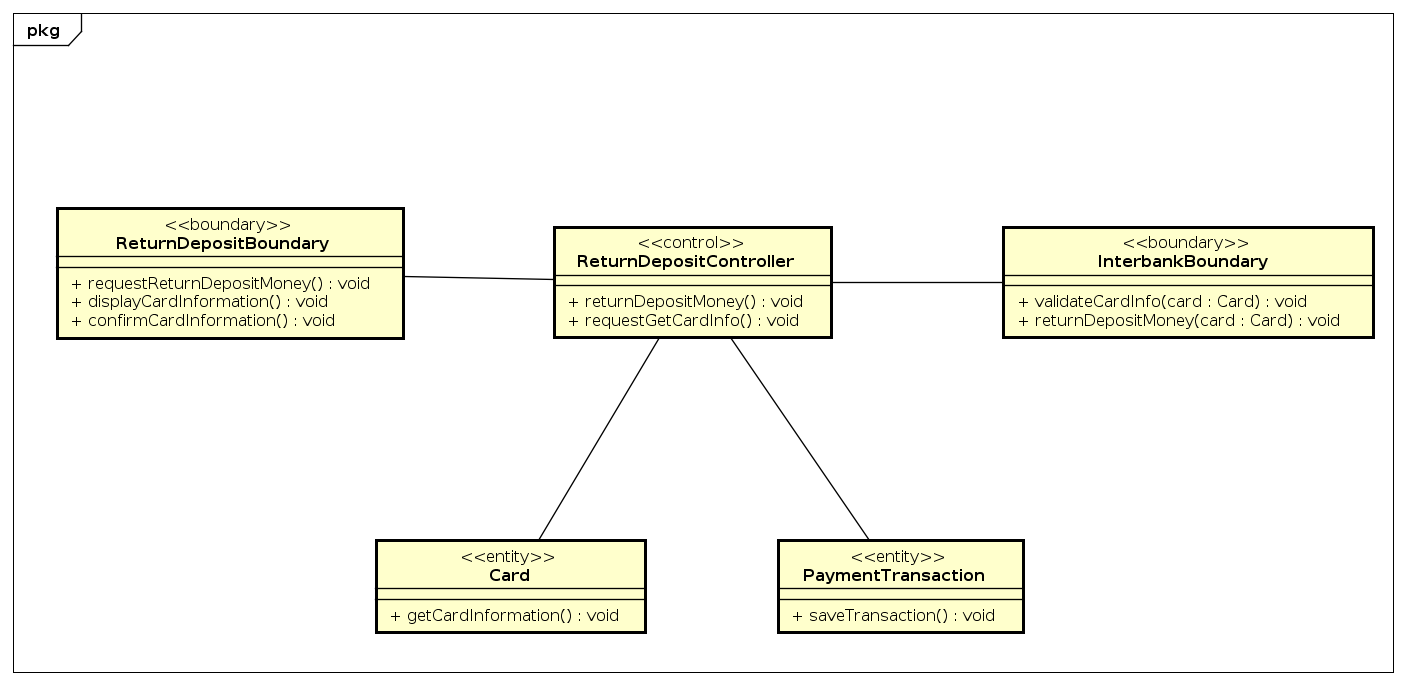
*figure 2: deductMoney class diagram*

**

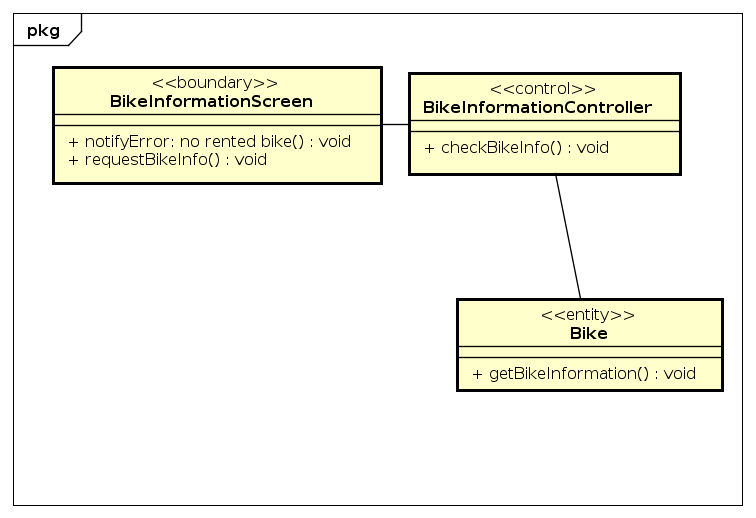
*figure 3: rentBike class diagram*

**

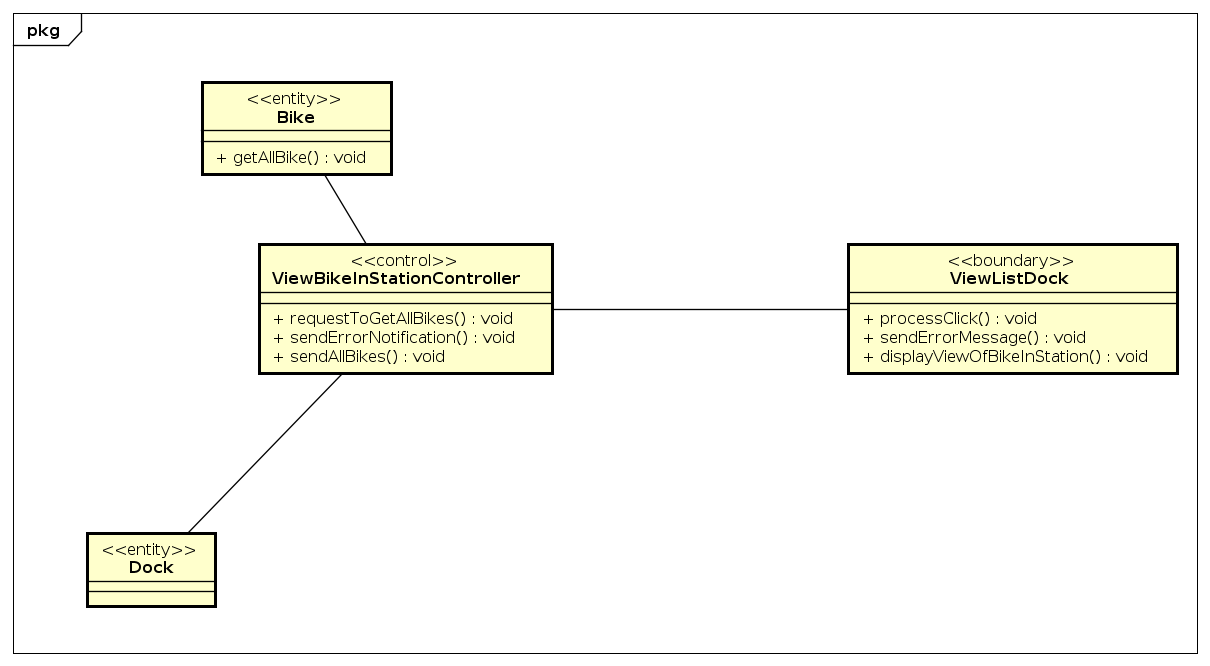
*figure 4: returnBike class diagram*

**

*figure 5: returnDepositMoney class diagram*

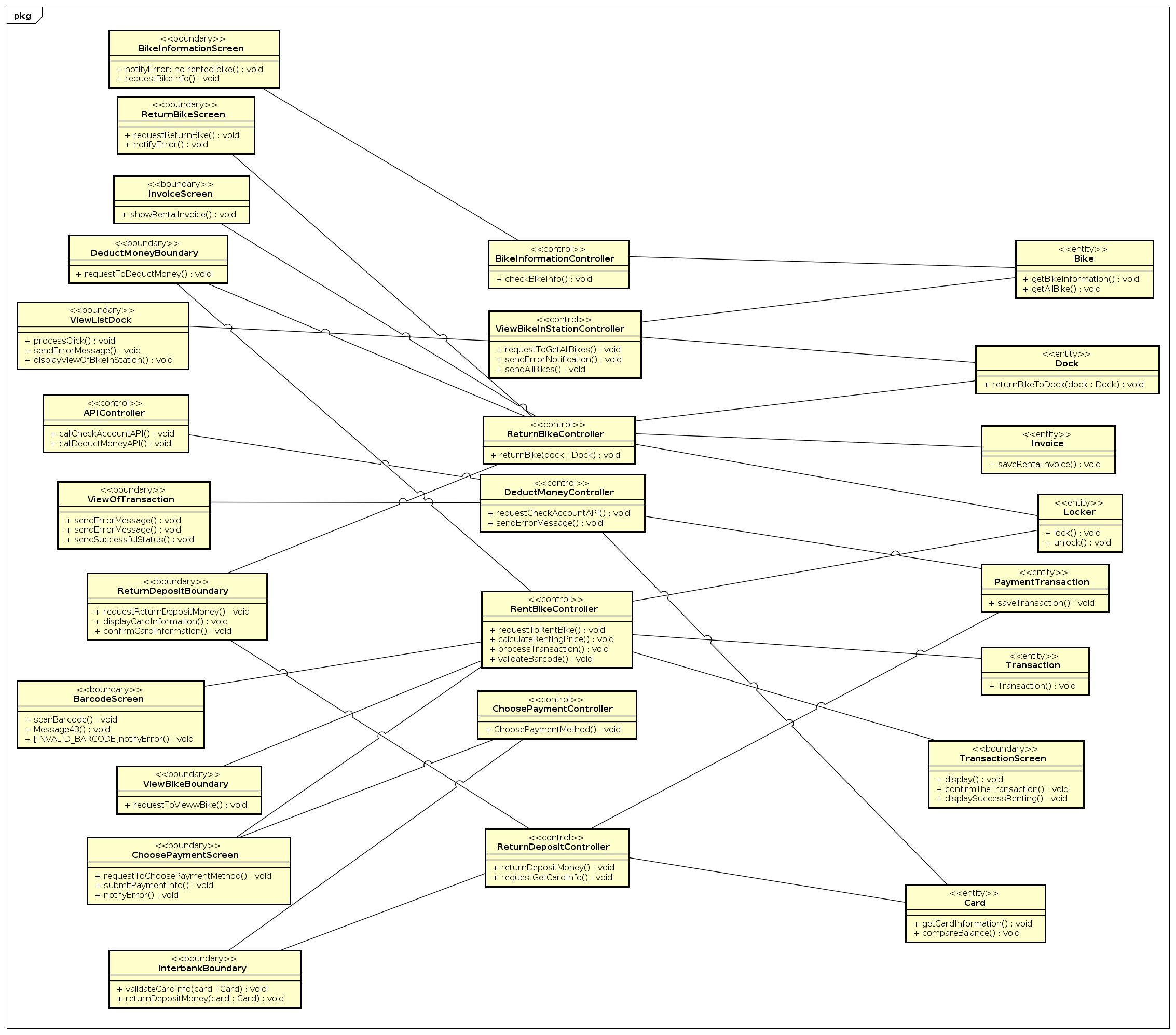
**

*figure 6: viewBikeInformation class diagram*

**

*figure 7: viewBikeInStaion class diagram*

## Unified Analysis Class Diagram



## 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.>*

# Detailed Design

## User Interface Design

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

### Screen Configuration Standardization

#### Screen Configuration Standardizations Display

Number of colors supported: 16,777,216 colors Resolution: 1792 x 828 - Phone Resolution 𝑝𝑖𝑥𝑒𝑙𝑠

#### 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.

#### Control

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

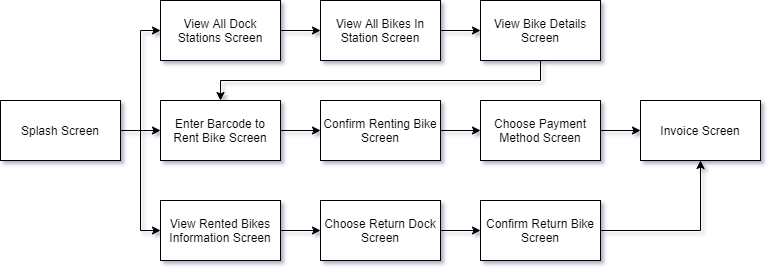
Input check process: Should check if it is empty or not. Next, check if the input is in the correct format or not

Sequence of moving the focus: After the opening screen, the app will start with a splash screen, and then the first screen (home screen) will appear.

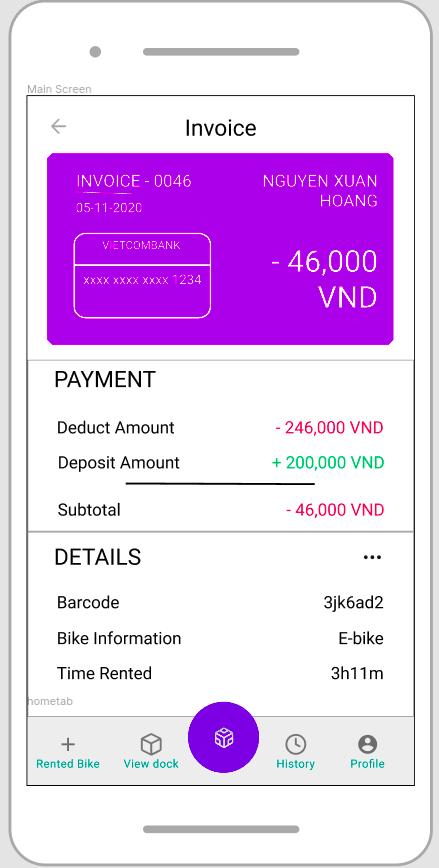
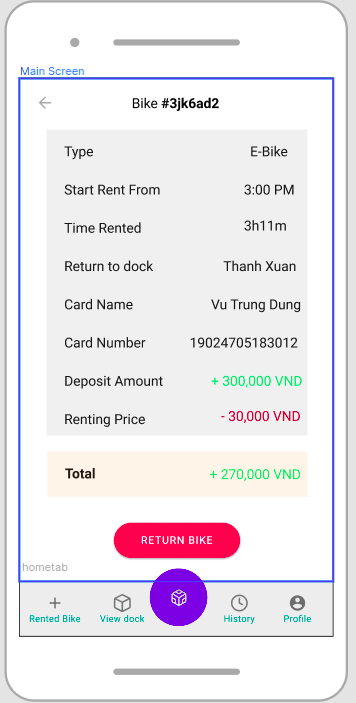
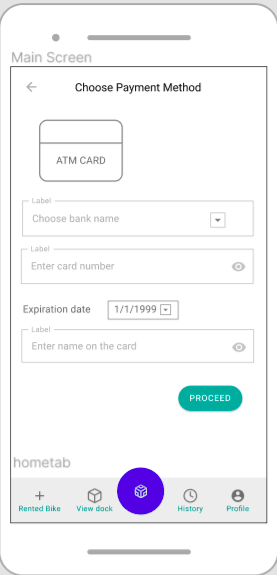
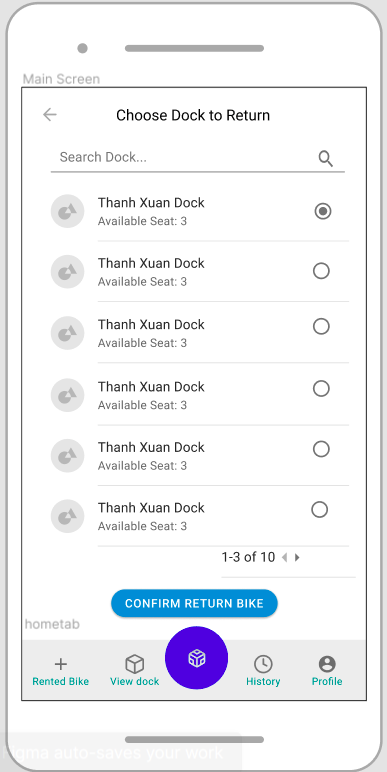
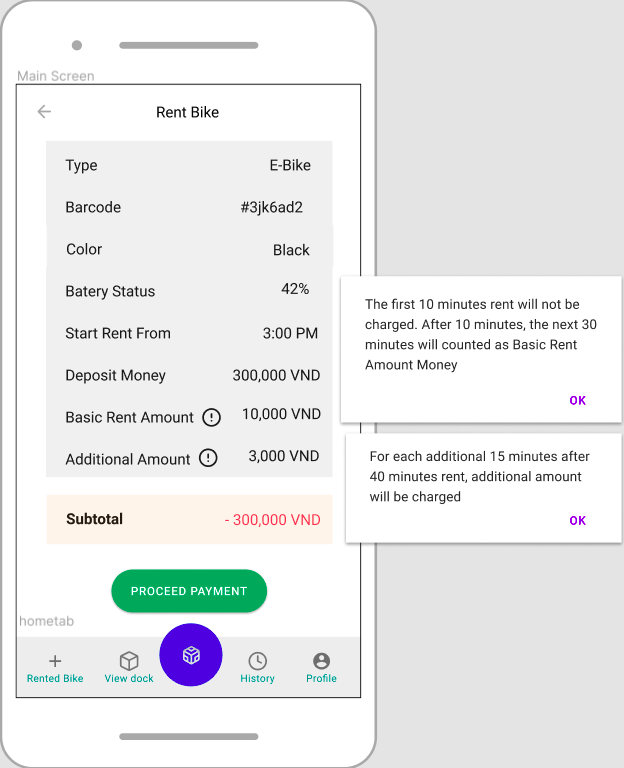
#### Sequences of the system screens:

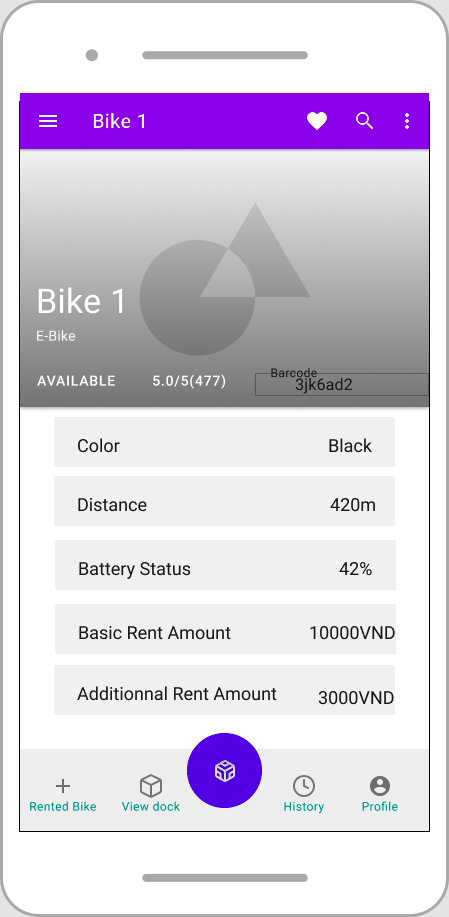
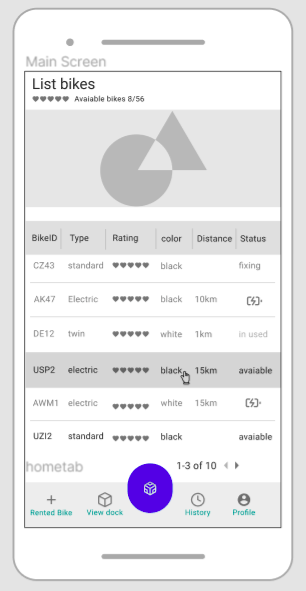
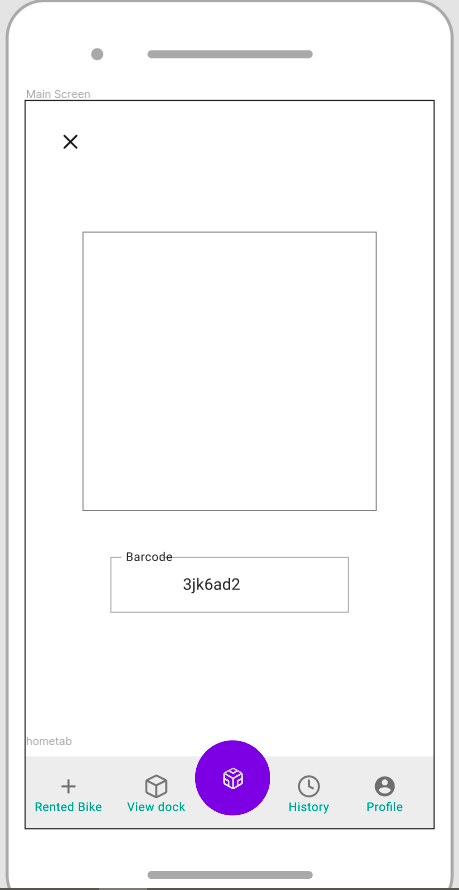
1. Splash Screen
2. View All Dock Stations Screen
3. View All Bikes In Station Screen
4. View Bike Details Screen
5. Enter Barcode to Rent Bike Screen
6. Confirm Renting Bike Screen
7. Choose Payment Method Screen
8. View Rented Bikes Information Screen
9. Return Bike Screen
10. Choose Return Dock Screen
11. Confirm Return Bike Screen
12. Invoice Screen

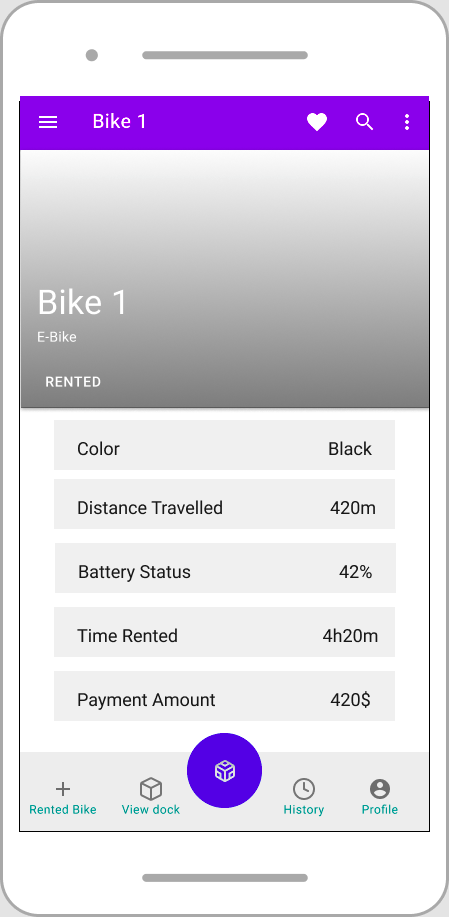
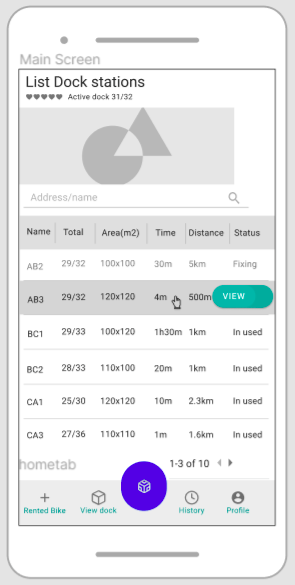
### Screen Transition Diagrams



### Screen Specifications

* *

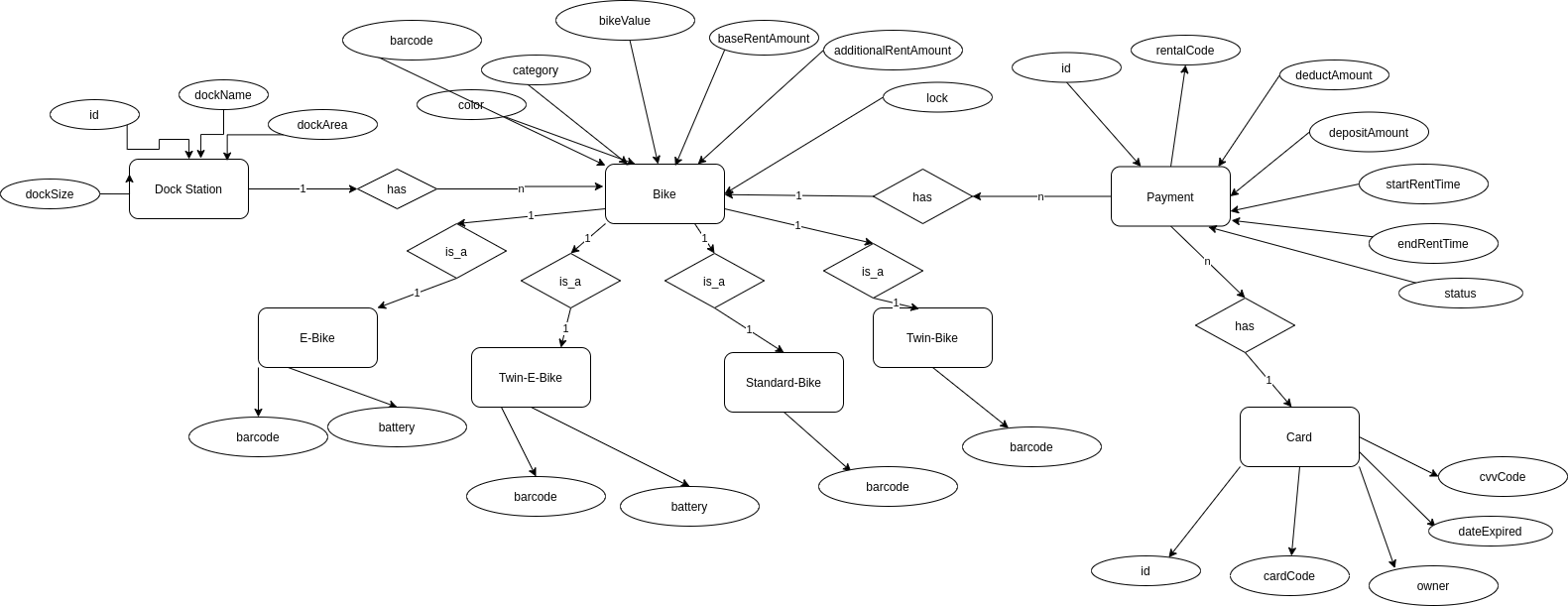
**

**

## Data Modeling

### Conceptual Data Modeling

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

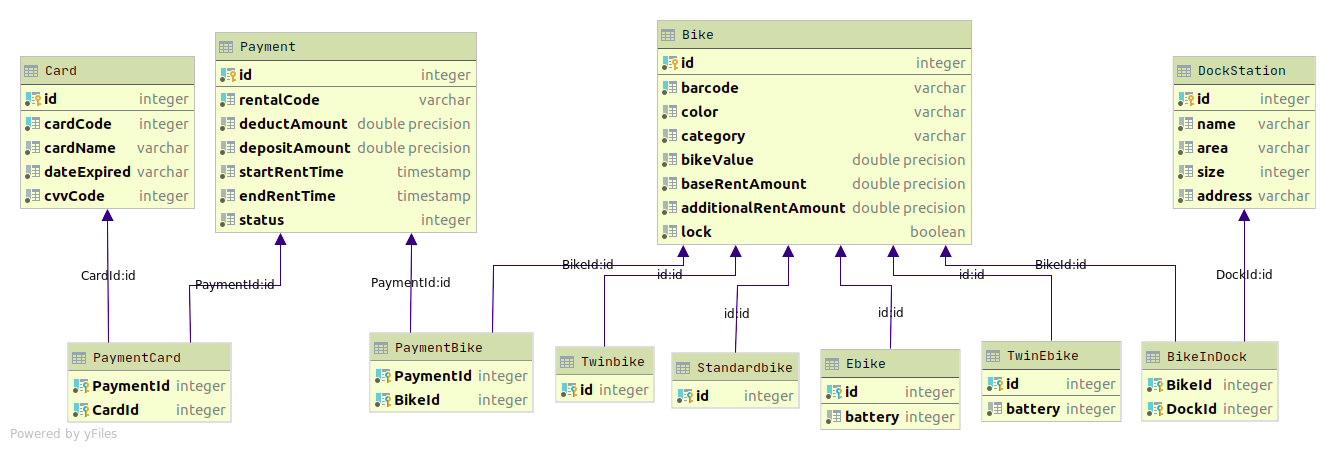
**

### Database Design

#### 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 easier to find a solution when having concern or error.

#### Logical Data Model

**

#### Physical Data Model

• **BikeInfo**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***#*** | *PK* | *FK* | *Column Name* | *Data Type* | *Mandatory* | *Description* |
| **1** | x | x | id | integer | Yes | Payment id |
| **2** |  |  | bikeValue | double | Yes | Bike id |
|  |  |  | baseRentAmount | double | Yes | Base rent amount |
|  |  |  | addRentAmount | double | Yes | Add rent amount |
|  |  |  | saddle | integer | Yes | saddle |
|  |  |  | pedal | integer | Yes | pedal |
|  |  |  | rear | integer | Yes | rear |

• **DockStation**

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

• **Card**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***#*** | *PK* | *FK* | *Column Name* | *Data Type* | *Mandatory* | *Description* |
| 1. | x | x | id | Integer | Yes | ID |
| 2. |  |  | cardCode | VARCHAR | Yes | Card’s Code |
| 3. |  |  | cardName | VARCHAR | Yes | Name of the card’s owner |
| 4. |  |  | cvvCode | Integer | Yes | CVV Code of the Card |
| 5. |  |  | dateExpired | VARCHAR | Yes | Card’s Expiration Date |
|  |  |  |  |  |  |  |

• **Payment**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***#*** | *PK* | *FK* | *Column Name* | *Data Type* | *Mandatory* | *Description* |
| **1** | x |  | id | integer | yes | Payment id |
| **2** |  |  | rentalCode | Integer | yes | Rental code |
| **3** |  |  | rentAmount | double | Yes | Rent amount |
| **4** |  |  | DepositAmount | double | yes | Deposit amount |
| **5** |  |  | startRentTime | TIMESTAMP | yes | Starting rent time |
| **6** |  |  | endRentTime | TIMESTAMP | yes | Ending rent time |
| **7** |  | x | statusId | integer | yes | Id of status transaction |
| **8** |  | x | bikeId | integer |  |  |
| **9** |  | x | cardId | integer |  |  |

• **PaymentStatus**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***#*** | *PK* | *FK* | *Column Name* | *Data Type* | *Mandatory* | *Description* |
| 1. | x | x | id | Integer | Yes | Payment ID |
| 2. | x |  | status | varchar | Yes | Status of payment |

• **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** |  |  | lockBike | Boolean | Yes | Bike is locked or not |
| **6** |  | x | dockId | integer | Yes | Bike’s dock Id |
| **7** |  | x | bikeInfoId | Integer | Yes | Bike’s information ID |

• **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 |

*4.2.2.4 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"."BikeInfo"***

***(***

***id serial PRIMARY KEY not null,***

***bikeValue FLOAT not null,***

***baseRentAmount FLOAT not null,***

***addRentAmount FLOAT not null,***

***saddle INT not null,***

***pedal INT not null,***

***rear INT not null***

***);***

***create table "ecoBikeSystem"."Bike"***

***(***

***id serial not null,***

***barcode VARCHAR not null,***

***color VARCHAR not null,***

***category VARCHAR not null,***

***lockbike BOOLEAN default FALSE not null,***

***"dockId" int not null***

***constraint bikeindock\_dockstation\_id\_fk***

***references "ecoBikeSystem"."DockStation"***

***on update cascade on delete cascade,***

***bikeInfoId int not null***

***constraint bikeInfo\_id\_fk***

***references "ecoBikeSystem"."BikeInfo"***

***on update cascade on delete cascade***

***);***

***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"."Card"***

***(***

***id serial not null,***

***"cardCode" VARCHAR not null,***

***"cardName" VARCHAR not null,***

***"dateExpired" VARCHAR not null,***

***"cvvCode" int not null,***

***lock boolean default false 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"."PaymentStatus"***

***(***

***id int not null***

***constraint paymentStatus\_pk***

***primary key,***

***status VARCHAR NOT NULL***

***);***

***create table "ecoBikeSystem"."Payment"***

***(***

***id serial not null,***

***"rentalCode" VARCHAR not null,***

***"rentAmount" float not null,***

***"depositAmount" float not null,***

***"startRentTime" TIMESTAMP not null,***

***"endRentTime" TIMESTAMP not null,***

***statusId int not null***

***constraint paymentStatus\_payment\_fk***

***references "ecoBikeSystem"."PaymentStatus"***

***on update cascade on delete cascade,***

***"bikeId" int not null***

***constraint paymentbike\_bike\_id\_fk***

***references "ecoBikeSystem"."Bike"***

***on update cascade on delete cascade,***

***"cardId" int not null***

***constraint paymentcard\_card\_id\_fk***

***references "ecoBikeSystem"."Card"***

***on update cascade on delete cascade***

***);***

***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"."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***

***);***

## 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.)*

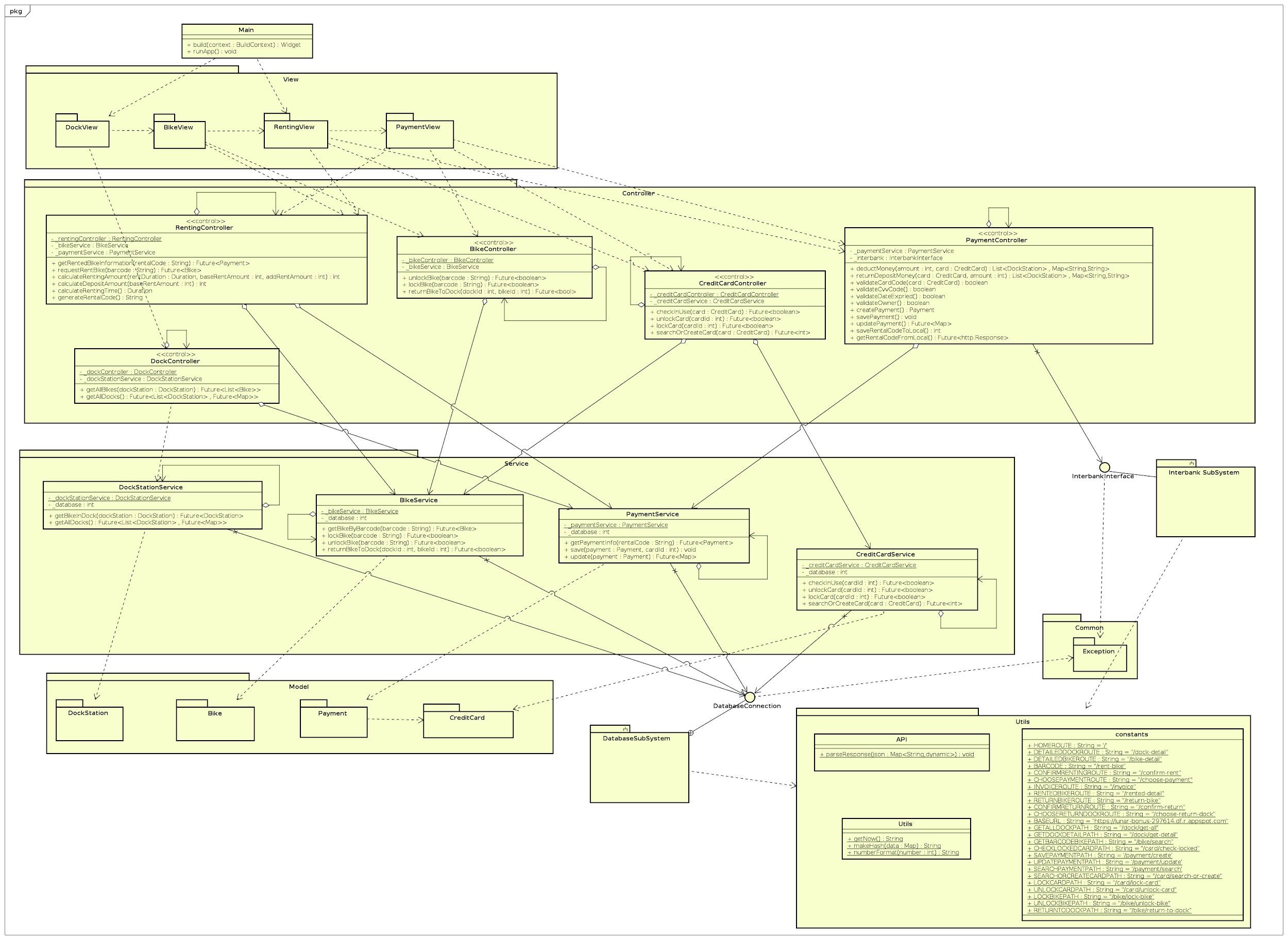
*• Estimate of the file size or volume of data within the file, including overhead resulting from file access methods*

*• 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>*

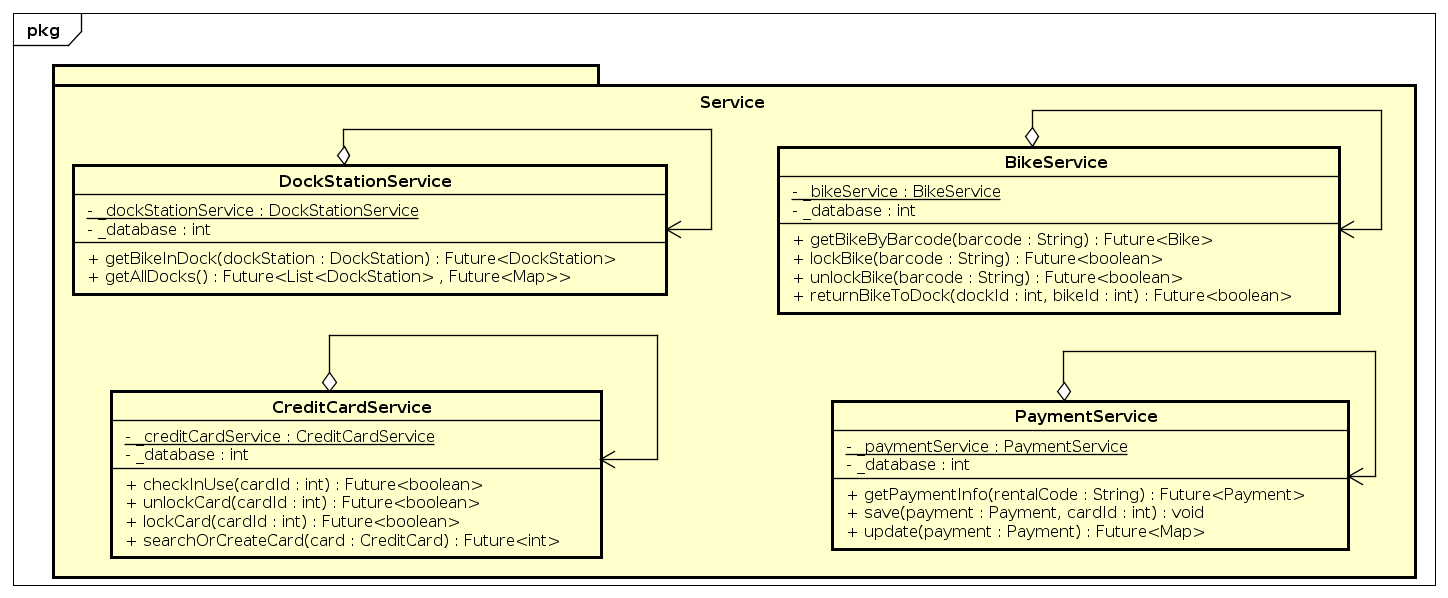
## Class Design

### General Class Diagram

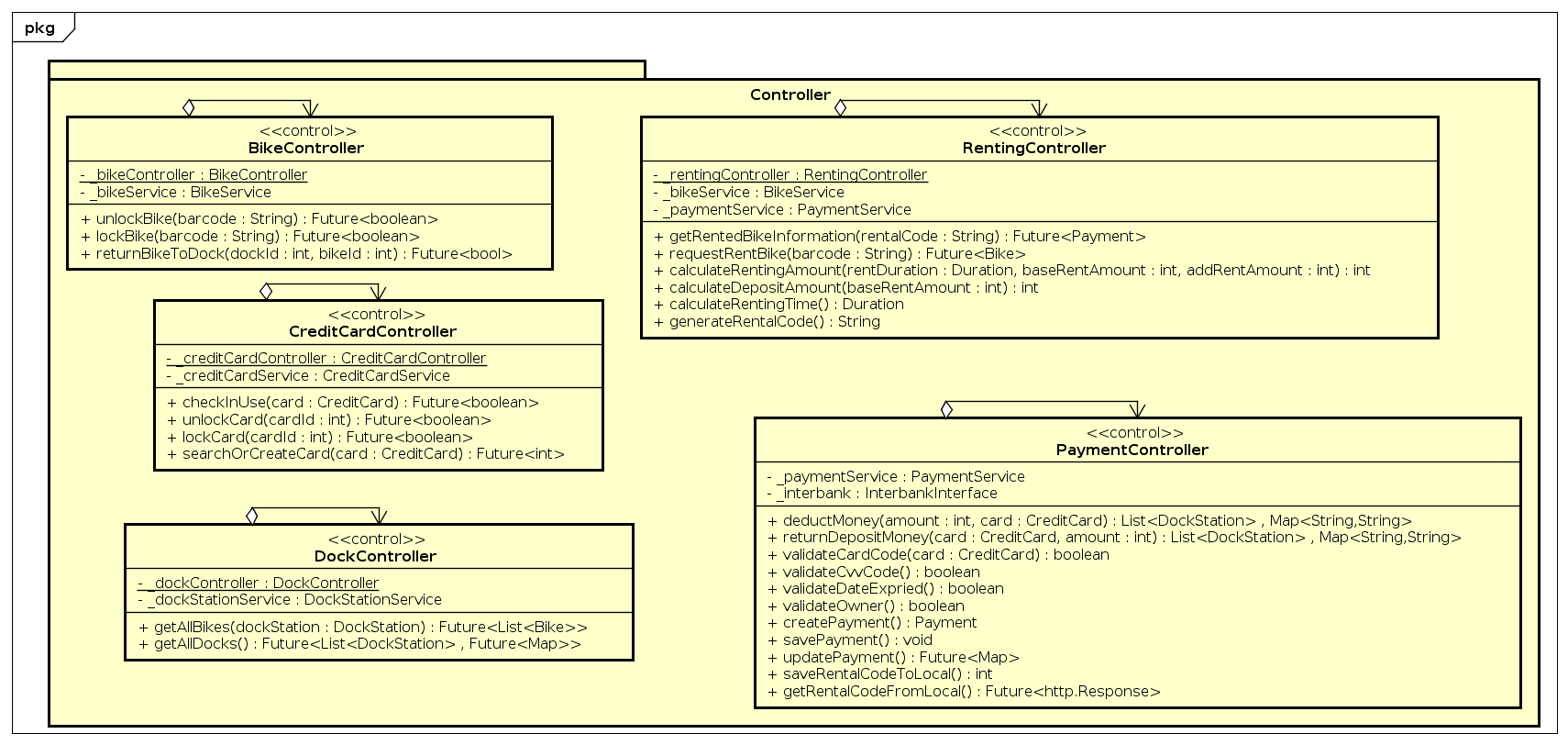


### Class Diagrams

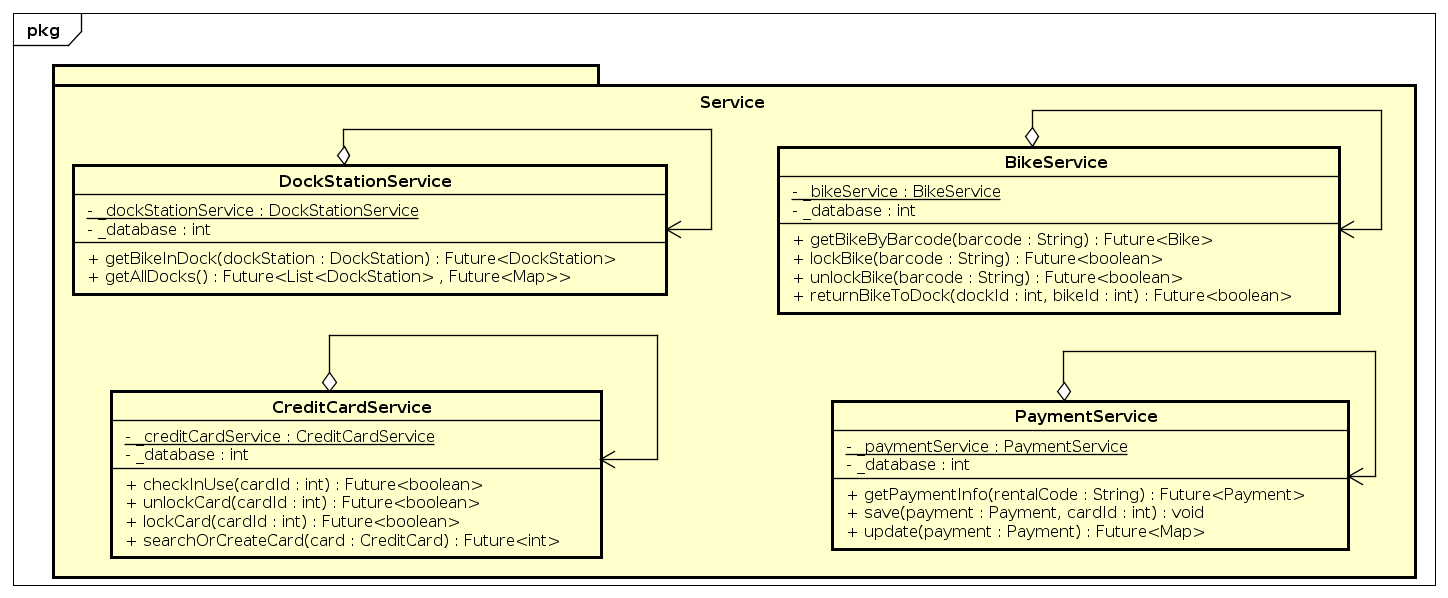
* + - 1. ***Class Diagram for Package View***

******

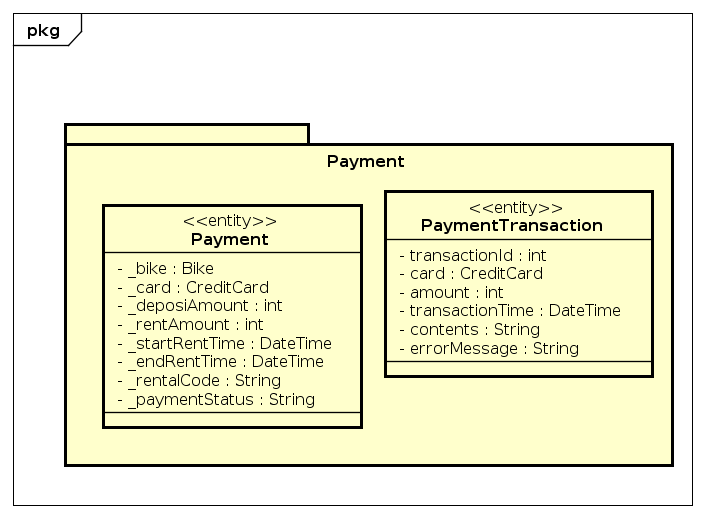
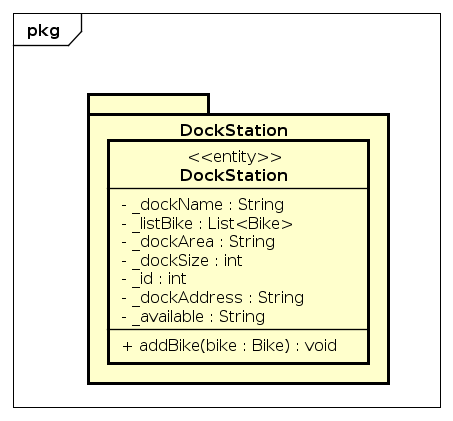
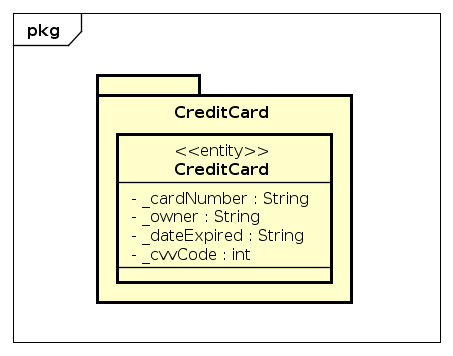
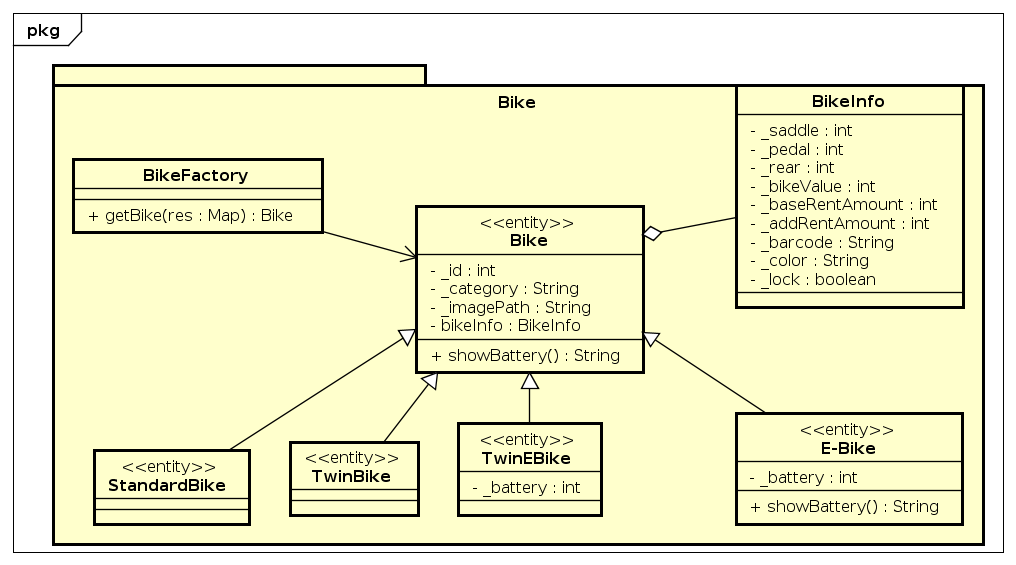
* + - 1. ***Class Diagram for Package Controller***



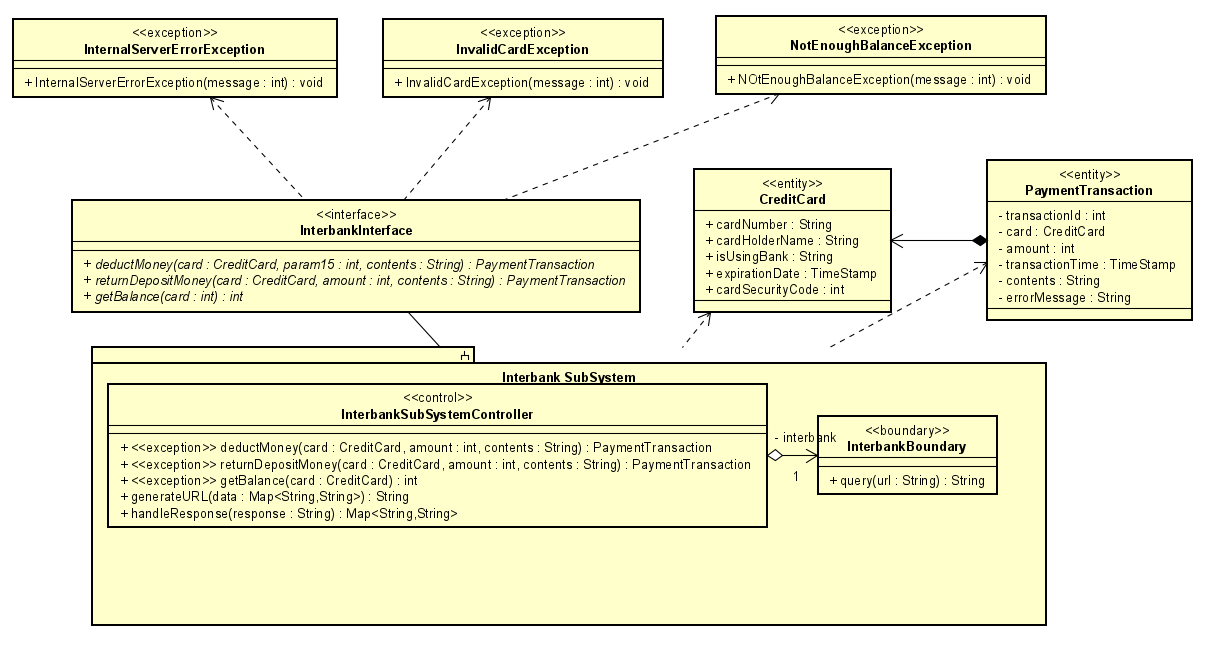
* + - 1. ***Class Diagram for Package Service***

******

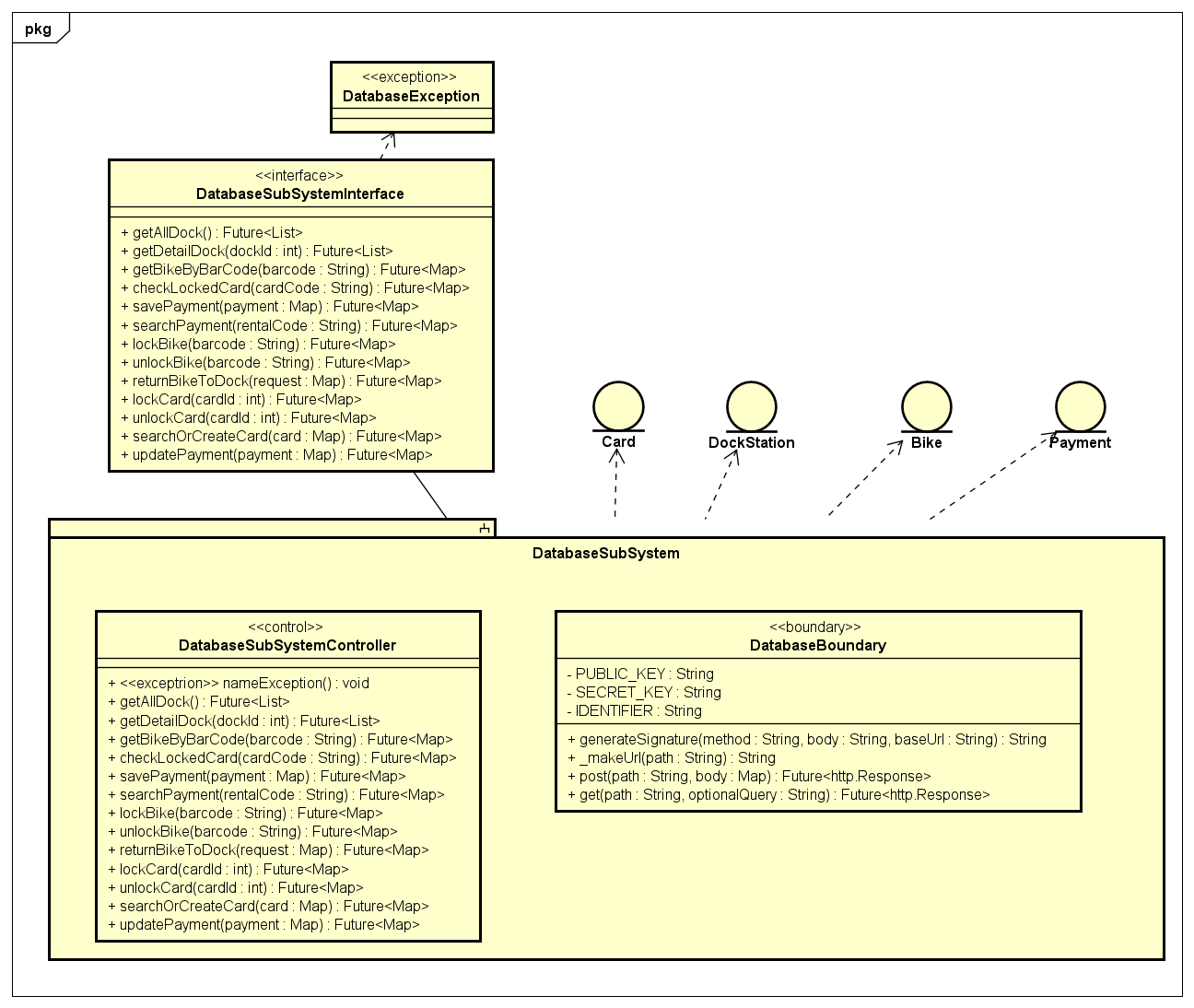
* + - 1. ***Class Diagram for Package Model***



* + - 1. ***Class Diagram for SubSystem Interbank***

****

**4.4.2.4 Class diagram for Database Subsystem**

****

### Class Design

* Class diagram for package View
* All the class in this view package have a function ***build*** to generate UI components following the standard of material design

BikeScreen

RentedBikeScreen

AppBar

BottomBar

Routing

SectionBanner

DetailedDockScreen

ListDockScreen

ChoosePaymentScreen

InvoiceScreen

BarcodeScreen

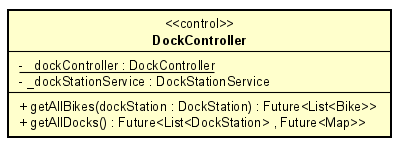
ChooseReturnDockScreen

ConfirmRentedScreen

ConfirmReturnScreen

*Class diagram for package Controller*

#### Class DockController



**Attribute**

None

**Operation**

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Name* | *Return type* | *Description (purpose)* |
| 1 | getAllBikes | List<Bike> | Get all bikes from a dock station |
| 2 | getAllDocks | List<DockStation> | Get all dock station information |

**Parameter**

* DockStation: dockStation - dock station entity

**Exception**

None

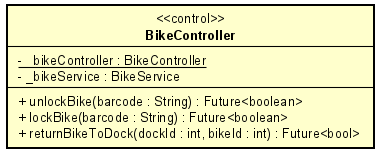
**Method**

None

**State**

None

#### Class BikeController



**Attribute**

None

**Operation**

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Name* | *Return type* | *Description (purpose)* |
| 1 | BikeController | factory | Instantiate this object point to this class |
| 2 | unlockBike | boolean | unlock bike in dock station |
| 3 | lockBike | lock bike | lock bike in dock station |
| 4 | returnBikeToDock | boolean | return bike to dock station |

**Parameter**

barcode: String - barcode of bike

dockId: Integer - dock station ID

bikeId: Integer - bike ID

**Exception**

None

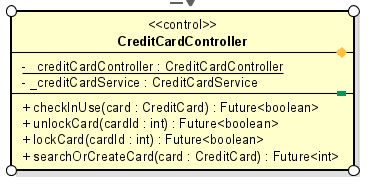
**Method**

None

**State**

None

#### Class CreditCardController



**Attribute**

**Operation**

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Name* | *Return type* | *Description (purpose)* |
| 1 | CreditCardController | factory | Instantiate this object point to this class |
| 2 | unlockCard | boolean | unlock card |
|  | lockCard | boolean | lock card |
| 3 | checkInUse | boolean | check card in use or not |
| 4 | searchOrCreateCard | Int | search for card or create if not exist |

***Parameter***:

card: CreditCard

cardId: Integer

***Exception***:

None

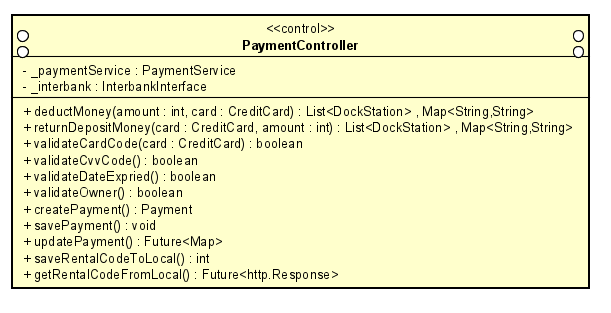
**Method**

None

**State**

None

#### Class PaymentController



**Attribute**

private paymentService: PaymentService

private interbank: InterbankInterface

**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 | savePayment | Void | Save payment information |
| 4 | updatePayment | Map<String,String> | Update payment information |
| 5 | saveRentalCodeToLocal | Void | save information of rental code to local memory |
| 6 | getRentalCodeFromLocal | String | Get rental code from local |

**Parameter**

rentalCode: rental code of renting

payment: payment entity

cardId: Id of card

bike: bike entity

depositMoney: deposit money

start: starting time of renting

end: ending time of renting

card: card entity

**Exception**

None

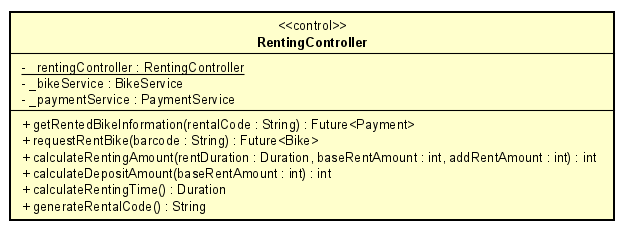
**Method**

None

**State**

None

#### Class RentingController



**Attribute**

private bikeService: BikeService

private paymentService: PaymentService

**Operation**

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Name* | *Return value* | *Description* |
| 1 | requestRentBike | Bike | request renting a bike |
| 2 | getRentedBikeInformation | Payment | get rented bike information |
| 3 | generateRentalCode | String | generate rental code |
| 4 | calculate renting amount | Integer | calculate renting amount |
| 5 | calculate renting time | Duration | calculate renting time |
| 6 | calculate deposit money | int | calculate deposit money |

**Parameter**

barcode: barcode of bike

rentalCode - rental code of renting

rentDuration - duration of renting time

baseRentAmount - base renting amount

addRentAmount - additional renting amount

startTime - starting renting time

endTime - ending renting time

**Exception**

None

**Method**

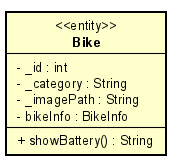
None

**State**

None

*Class diagram for package Model*

#### Class Bike



**Attribute**

private int id - id of bike

private String category - category of bike: Twin, EBike, ...

private String imagePath: path to image: \path\to\image.png

public Bikeinfo bikeInfo: information of bike

**Operation**

None

**Parameter**

None

**Exception**

None

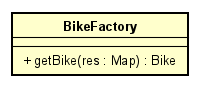
**Method**

None

**State**

None

#### Class BikeFactory



**Attribute**

None

**Operation**

Bike getBike(res: Map) - Getting an instance of bike

**Parameter**

None

**Exception**

None

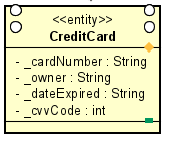
**Method**

None

**State**

None

#### Class CreditCard



**Attribute**

private String cardCode - code of credit card

private String owner - owner’s name

private String cvvCode - security code

private String dateExpired - expiration date

**Operation**

None

**Parameter**

None

**Exception**

*None*

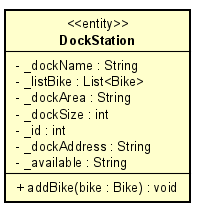
**Method:**

None

**State:**

None

#### Class DockStation



**Attribute**

private int id - id of dock station

private String dockName - dockStation’s name

private String dockArea - area of dock station

private String dockAddress - address of dock station

private String available - available bike in dock station

private int dockSize - maximum bike in dock station

private List<Bike> lstBike - list of bike entity in dock station

**Operation**

None

**Parameter**

None

**Exception**

None

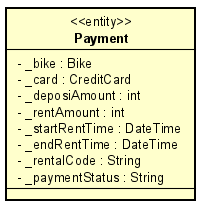
**Method**

None

**State**

None

#### Class Payment



**Attribute**

private Bike bike - bike entity

private CreditCard card - card entity

private int rentAmount - renting Amount

private int depositAmount - deposit amount

private DateTime startRentTime - starting renting time

private DateTime endRentTime - ending renting time

private String paymentStatus - Status of payment

**Operation**

None

**Parameter**

None

**Exception**

None

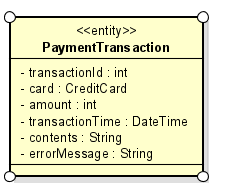
**Method**

None

**State**

None

#### Class PaymentTransaction



**Attribute**

private CreditCard card - card entity

private String command - example: pay, refund, …

private int amount - amount of money

private String createdAt

**Operation**

None

**Parameter**

None

**Exception**

None

**Method**

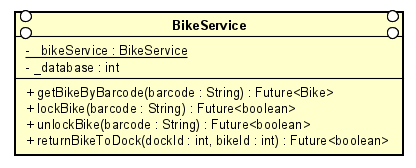
None

**State**

None

Class diagram for package Service

#### Class BikeService



**Attribute**

private BikeService bikeService - static instance of BikeService

private DatabaseConnection database - instance of Database connection

**Operation**

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Name* | *Return type* | *Description (purpose)* |
| 1 | getBikeByBarcode | Bike | get bike by barcode |
| 2 | lockBike | Boolean | lock bike |
| 3 | unlockBike | Boolean | unlock bike |
| 4 | returnBikeToDock | Boolean | return bike to dock station |

**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

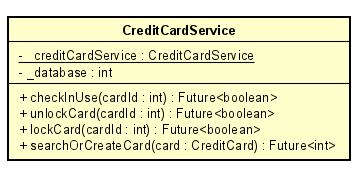
**Method**

None

**State**

None

#### Class CreditCardService



**Attribute**

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Name* | *Return type* | *Description (purpose)* |
| 1 | creditCardService | CreditCardService | Static instance of credit card service |
| 2 | database | DatabaseConnection | Instance of database connection |

**Operation**

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Name* | *Return type* | *Description (purpose)* |
| 1 | checkInUse | boolean | Request all related information for selected bike from database |
| 2 | unlockCard | boolean | Change rented bike’s lock status |
| 3 | lockCard | boolean | lock card by card ID |
| 4 | searchOrCreateCard | Integer | search for card or create if not exist |

**Parameter**

card: card entity

cardid: id of card

**Exception**

None

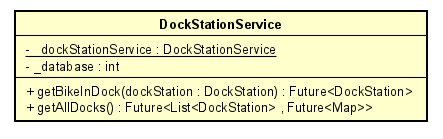
**Method**

None

**State**

None

#### Class DockStationsService



**Attribute**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *Name* | *Data type* | *Default value* | *Description* |
| 1 | dockStationService | DockStationService | Null | Static variable of dock station service |
| 2 | database | DatabaseConnection |  |  |

**Operation**

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Name* | *Return type* | *Description (purpose)* |
| 1 | getBikeInDock | DockStation | Request selected bike’s color |
| 2 | getAllDocks | Map<String,String> | Get all dock station |

**Parameter**

dockStation: dock station entity

**Exception**

None

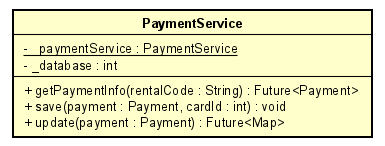
**Method**

None

**State**

None

#### Class PaymentService



**Attribute**

None

**Operation**

|  |  |  |  |
| --- | --- | --- | --- |
| *#* | *Name* | *Return type* | *Description (purpose)* |
| 1 | paymentService | PaymentService | static instance of payment service |
| 2 | database | DatabaseConnection | instance of database connection |

**Parameter**

rentalCode: string – the rental code of the rented bike

barcode: string - barcode of the bike

dock: DockStation - dock station model

**Exception**

None

**Method**

None

**State**

None

# Design Considerations

## Goals and Guidelines

## Goals

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

## Guidelines

## Coding Convention for Flutter-Dart:

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

## Architectural Strategies

*• Programming Language: Dart*

*• Framework: Flutter*

*• Database Management System: PostgreSQL*

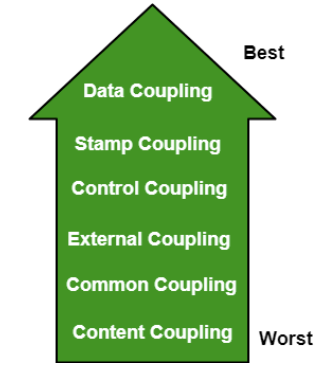
*• Using Subsystem: Interbank System for Card Management, Database System for Getting Data from Database*

*• Error Detection: Using Unit-test and Integration Test*

*• Synchronization: Asynchronization Method using Dart Language*

## Coupling and Cohesion

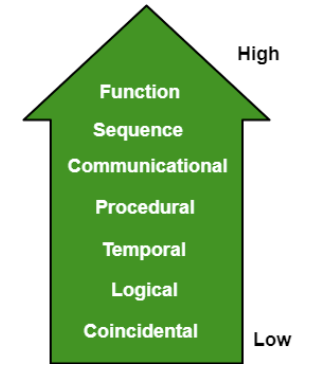
**Coupling:** Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling



Our Project has both **Data Coupling & Control 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 knows how to get data from Bike Entity class and use it for a totally different method to take it to the View package.
* In Control coupling, the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example - In class BikeFactory, we pass the parameter “category” string to know which category will be taken, then return the responding child class Ebike or Standardbike,.....

**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 performs the calculation task and sends it to RentedBikeView. The RentedBikeView class only renders the returned value.

## Design Principles

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

SOLID Acronym:

* + S: Single Responsibility Principle (SRP)
  + O: Open closed Principle (OSP)
  + L: Liskov substitution 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. Liskov Substitution Principle

“Objects in a program should be replaceable with instances of their subtypes 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. 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. 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. 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.

## Design Patterns

* + 1. **Singleton**

*Singleton pattern is one of the simplest design patterns in OOP Language. This type of design pattern comes under the 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 a single object gets created. This class provides a way to access its*

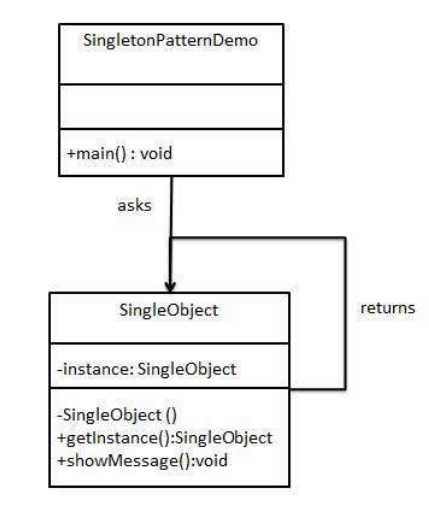
*only object which can be accessed directly without need to instantiate the object of the*

*class.*

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

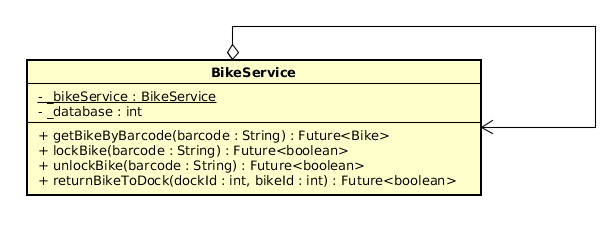
*world. SingletonPatternDemo, our demo class will use SingleObject class to get a*

*SingleObject object.*



In our EcoBikeRental System, we use many Singleton classes, such as class DBConnection() for getting the instance for getting data from database, or some service class for initiating business logic for getting databases and assigning to a model.

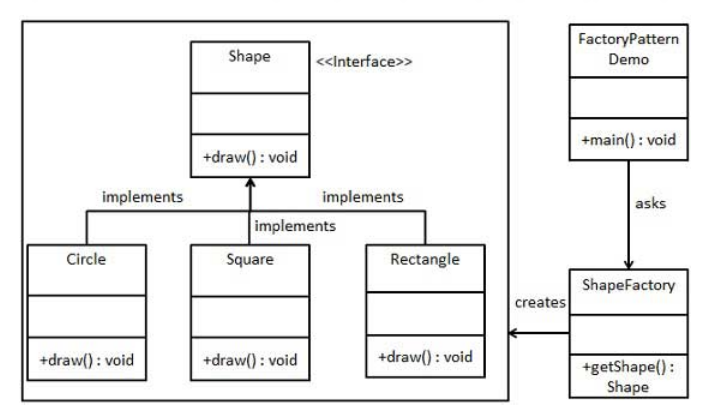
Example:



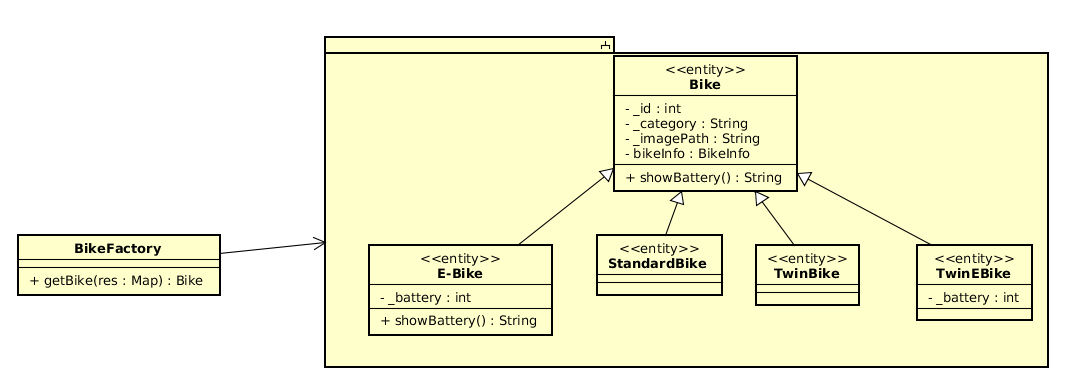
* + 1. **Factory Pattern**

*Factory pattern is one of the most used design patterns. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.*

*In Factory pattern, we create objects without exposing the creation logic to the client and refer to newly created objects using a common interface.*

**

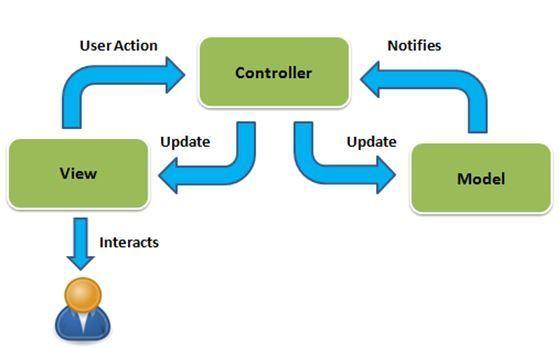
Example:



Here, Our BikeFactory class uses getBike() static class to get instances from different Bike Implementation from the “category” string obtained from the database.

* + 1. **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:

****