HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology

Software Design Document

Version 1.3

EcoBikeRental Application

Subject: ITSS Software Management

Group 6

|  |  |  |
| --- | --- | --- |
| No. | Student Name | Student ID |
| 1 | Vu Hong Quang | 20205192 |
| 2 | Vu Xuan Quy | 20205193 |
| 3 | Bui Van Thanh | 20200585 |
| 4 | Nguyen Hai Son | 20205168 |

*Hanoi, 08/2023*

Table of contents

[List of Figures 2](#_Toc37026516)

[List of Tables 4](#_Toc411978688)

[1. Introduction 5](#_Toc1220688390)

[1.1. Objective 5](#_Toc605249322)

[1.2. Scope 6](#_Toc1119527338)

[1.3. Glossary 6](#_Toc546460115)

[2. Overall Description 7](#_Toc1675561228)

[2.1. General Overview 7](#_Toc460779650)

[2.2. Assumptions/Constraints/Risks 8](#_Toc1917248418)

[2.2.1. Assumptions 8](#_Toc760449358)

[2.2.2. Constraints 8](#_Toc161339552)

[· Hardware or software environment 9](#_Toc1629735163)

[· End-user environment 9](#_Toc355348601)

[· Availability or volatility of resources 9](#_Toc1632801809)

[· Standards compliance 9](#_Toc355372728)

[· Interoperability requirements 9](#_Toc1300765267)

[· Interface/protocol requirements 9](#_Toc1128943219)

[· Licensing requirements 9](#_Toc219908715)

[· Data repository and distribution requirements 9](#_Toc204277809)

[· Security requirements (or other such regulations) 9](#_Toc311191135)

[· Memory or other capacity limitations 9](#_Toc1949429568)

[· Performance requirements 9](#_Toc877862163)

[· Network communications 9](#_Toc2081111800)

[· Verification and validation requirements (testing) 9](#_Toc746070139)

[· Other means of addressing quality goals 9](#_Toc20341921)

[· Other requirements described in the Requirements Document 9](#_Toc1370105633)

[2.2.3. Risks 9](#_Toc705037463)

[3. System Architecture and Architecture Design 10](#_Toc1328244436)

[3.1. Architectural Patterns 10](#_Toc2088294570)

[3.2. Interaction Diagrams 11](#_Toc1592384459)

[3.2.1. Communication Diagrams 11](#_Toc859181254)

[3.2.2. Sequence Diagrams 12](#_Toc1049670286)

[3.3. Analysis Class Diagrams 14](#_Toc212844156)

[3.4. Unified Analysis Class Diagram 15](#_Toc1215305844)

[3.5. Security Software Architecture 16](#_Toc1396080108)

[4. Detailed Design 16](#_Toc593641406)

[4.1. User Interface Design 16](#_Toc381423477)

[4.1.1. Screen Configuration Standardization 16](#_Toc380685572)

[4.1.2. Screen Transition Diagrams 16](#_Toc2090675945)

[4.1.3. Screen Specifications 17](#_Toc1298330435)

[4.1.3.1. Main Screen 17](#_Toc932031544)

[4.1.3.2. Dock Screen 17](#_Toc75971326)

[4.1.3.3. Bike Screen 18](#_Toc212493872)

[4.1.3.4. Choose Credit Card Screen 19](#_Toc768281600)

[4.1.3.5. Transaction rent screen 20](#_Toc914746897)

[4.1.3.6. Return bike Screen 20](#_Toc1122483564)

[4.2. Data Modeling 21](#_Toc1385212107)

[4.2.1. Conceptual Data Modeling 21](#_Toc1471945667)

[4.2.2. Database Design 21](#_Toc1026718897)

[4.2.2.1. Database Management System 21](#_Toc990366517)

[4.2.2.2. 21](#_Toc1486313453)

[Database Diagram 22](#_Toc1467013348)

[4.2.2.3. Database Detail Design 22](#_Toc1441554185)

[4.3. Non-Database Management System Files 35](#_Toc2146445217)

[5. Design Considerations 35](#_Toc321758542)

[5.1. Goals and Guidelines 35](#_Toc107700556)

[5.2. Architectural Strategies 36](#_Toc1406969949)

[5.3. Coupling and Cohesion 36](#_Toc1880637847)

[5.3.1. Coupling 36](#_Toc337901564)

[5.3.1.1. Content coupling 36](#_Toc899821798)

[5.3.1.2. Common coupling 36](#_Toc876948225)

[5.3.1.3. Control coupling 37](#_Toc1969053019)

[5.3.1.4. Stamp coupling 37](#_Toc1196474054)

[5.3.1.5. Data coupling 37](#_Toc1299546336)

[5.3.2. Cohesion 38](#_Toc1957562137)

[5.3.2.1. Coincidental cohesion 38](#_Toc1033943391)

[5.3.2.2. Logical cohesion 38](#_Toc1001614297)

[5.3.2.3. Temporal cohesion 38](#_Toc1351224962)

[5.3.2.4. Procedure cohesion 39](#_Toc1454877750)

[5.3.2.5. Communicational cohesion 39](#_Toc207541179)

[5.3.2.6. Sequential cohesion 39](#_Toc1602898897)

[5.3.2.7. Information cohesion 39](#_Toc2046249881)

[5.3.2.8. Functional cohesion 39](#_Toc952069502)

[5.4. Design Principles 40](#_Toc795853539)

[5.5. Design Patterns 40](#_Toc223145569)

# List of Figures

*Figure 1.1: General use case diagram* *11*

*Figure 3.1: Communication Diagram for Rent Bike Use case* *13*

*Figure 3.2: Communication Diagram for Deposit Use case* *13*

*Figure 3.3: Communication Diagram for Update Payment Method Use case* *14*

*Figure 3.4: Communication Diagram for Return Bike Use case* *14*

*Figure 3.5: Communication Diagram for Return Deposit Use case* *15*

*Figure 3.6: Communication Diagram for Pay For Rental Use case* *15*

*Figure 3.7: Sequence Diagram for Rent Bike Use case* *16*

*Figure 3.8: Sequence Diagram for Deposit Use case* *16*

*Figure 3.9: Sequence Diagram for Update Payment Method Use case* *17*

*Figure 3.10: Sequence Diagram for Return Bike Use case* *17*

*Figure 3.11: Sequence Diagram for Return Deposit Use case* *18*

*Figure 3.12: Sequence Diagram for Pay For Rental Use case* *18*

*Figure 3.13: Class Diagram for View Bike Use case* *19*

*Figure 3.14: Class Diagram for View Bike Use case* *19*

*Figure 3.15: Class Diagram for Deposit Use case* *20*

*Figure 3.16: Class Diagram for Return Bike Use case* *20*

*Figure 3.17: Class Diagram for Return Deposit Use case* *21*

*Figure 3.18: Class Diagram for Pay Rental Use case* *21*

*Figure 3.19: Unified Class Diagram for EcoBikeRental Application* *21*

*Figure 4.1: Screen Transition Diagram for EcoBikeRental Application* *23*

*Figure 4.2. ER Diagram for EcoBikeRental Application* *28*

*Figure 4.3. Database Diagram for EcoBikeRental Application* *29*

# List of Tables

*Table 1.1: Terms used in the document* *10*

*Table 4.1. Splash Screen Specification* *24*

*Table 4.2. View Dock Screen Specification* *25*

*Table 4.3. View Bike Screen Specification* *25*

*Table 4.4. Choose Credit Card Screen Specification* *26*

*Table 4.6. Rent A Bike Screen Specification* *26*

*Table 4.7. Return Bike Screen Specification* *26*

*Table 4.8. Customer table design* *29*

*Table 4.9. Administrator table design* *30*

*Table 4.10. Dock table design* *30*

*Table 4.11. Bike table design* *31*

*Table 4.12. Bike In Dock table design* *31*

*Table 4.13. Bike Status table design* *32*

*Table 4.14. Invoice table design* *32*

*Table 4.15. Transaction table design* *33*

*Table 4.16. Rent Bike table design* *33*

*Table 4.17. Credit Card table design* *34*

*Table 4.18. RentBikeController attributes* *39*

*Table 4.19. RentBikeController operations* *40*

*Table 4.20. BikeTracker attributes* *41*

*Table 4.21. BikeTracker operations* *41*

*Table 4.22. PaymentController attributes* *42*

*Table 4.23. PaymentController operations* *43*

*Table 4.24. BikeInformationScreenHandler operations* *45*

*Table 4.25. BikeInformationScreenHandler operations* *45*

# 1. Introduction

## 1.1. Objective

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

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

## 1.2. Scope

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

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

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

## 1.3. Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Administrator | The person who uses EcoBikeRental application system for the purposes of monitoring list of bicycles in the system |
| Admin | as “administrator” |
| Bicycle | The transportation mean to be rent in this application system |
| Bike | as “bicycle” |
| Card number | The ID number of the credit card, printed on the credit card |
| Cardholder name | The name of the owner of the credit card, printed on the credit card |
| Credit card | A card connected to the interbank, used for performing transaction |
| Customer | The person who uses EcoBikeRental application system for the purposes of renting bike |
| Database | Collection of all information monitored by this system |
| Deposit | An amount of money customer has to pay at first in order to rent a bike |
| Dock | A place where bicycles are put |
| Interbank | The organization in charges of performing payment and return deposit transactions in the system |
| Payment | An amount of money customer has to pay to rent a bike, including deposit and rental fee |
| Rent a bike | The action of using a bike in a period of time, with paying deposit and rental fee |
| Rental fee | An amount of money customer has to pay, outside of the deposit, which depends on the rental time |
| Rental time | The time period when the bike is being rented |
| Return a bike | The action of stopping using a bike after having rented |
| Software Requirement Specification | A document that completely describes all of the functions of a proposed system and the constraints under which it must operate. For example, this document. |
| Station | as “dock” |
| Transaction | The action of paying for bike deposit, bike rental or returning deposit |
| User | Customer or Administrator |

*Table 1.1: Terms used in the document*

# 2. Overall Description

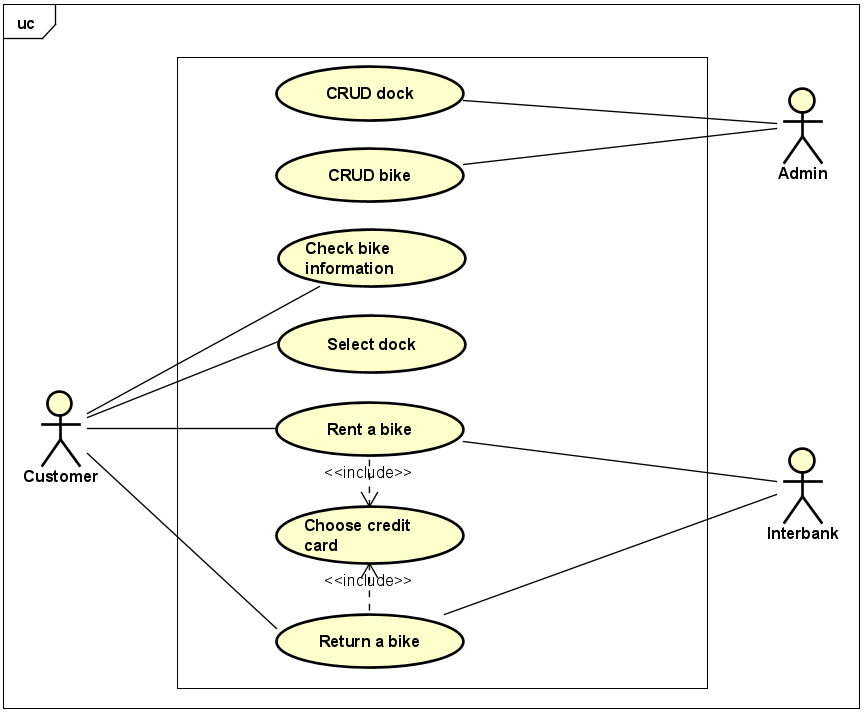
## 2.1. General Overview

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

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

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

|  |  |
| --- | --- |
|  |  |
|  |  |



*Figure 1.1: General use case diagram*

## 2.2. Assumptions/Constraints/Risks

### 2.2.1. Assumptions

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

### 2.2.2. Constraints

##### · *Hardware or software environment*

##### · *End-user environment*

##### · *Availability or volatility of resources*

##### · *Standards compliance*

##### · *Interoperability requirements*

##### · *Interface/protocol requirements*

##### · *Licensing requirements*

##### · *Data repository and distribution requirements*

##### · *Security requirements (or other such regulations)*

##### · *Memory or other capacity limitations*

##### · *Performance requirements*

##### · *Network communications*

##### · *Verification and validation requirements (testing)*

##### · *Other means of addressing quality goals*

##### · *Other requirements described in the Requirements Document*

### 2.2.3. Risks

None

# 3. System Architecture and Architecture Design

## 3.1. Architectural Patterns

*In the EcoBikeRental project, we have chosen to implement the Domain-Driven Design (DDD) architectural pattern due to its suitability for complex business domains and its focus on aligning software development with real-world business requirements. DDD offers several strategic and tactical patterns that enable us to create a well-structured and maintainable application.*

1. *Bounded Contexts: We have divided our application into distinct Bounded Contexts, each representing a specific business area with its own models, language, and rules. For instance, the "Bike" context handles bike-related operations, while the "Card" context manages credit card operations. This separation ensures clear boundaries and prevents conflicts between different parts of the system.*
2. *Entities and Aggregates: Entities represent real-world objects with a distinct identity, while Aggregates are clusters of related entities that are treated as a single unit for data modifications. This helps maintain data consistency and encapsulates complex business logic within aggregates, ensuring that business rules are followed.*
3. *Repositories: We use the Repository pattern to manage data persistence and retrieval. Each repository is responsible for providing an interface to access and manipulate entities within a bounded context. This abstraction shields the domain layer from the specifics of data storage and retrieval.*
4. *Domain Services: For operations that do not naturally belong to a single entity or aggregate, we utilize Domain Services. These services encapsulate domain-specific logic that doesn't fit within the scope of a single entity, promoting cohesive domain behavior.*
5. *Value Objects: Value Objects represent attributes without an identity, and they're immutable. We use them to capture attributes that don't change business meaning. For example, a bike's type, model, and attributes could be modeled as value objects.*
6. *Domain Events: We apply Domain Events to notify the system of significant changes within the domain. These events are used to trigger side effects or inform other parts of the application about relevant domain state changes.*

***Reasons for Choosing DDD:***

1. *Complex Domain Logic: The bike rental domain involves intricate business rules, processes, and relationships. DDD's emphasis on understanding and modeling the domain enables us to capture and implement these complexities effectively.*
2. *Collaborative Development: DDD promotes a shared understanding of the domain between technical and non-technical stakeholders. This aligns development efforts with business needs and improves communication.*
3. *Flexibility and Maintainability: By organizing the application around business concepts, DDD enables easier modifications and enhancements as the business evolves. Changes can be made within bounded contexts without affecting other parts of the system.*
4. *Clear Separation of Concerns: DDD enforces a clear separation between the domain layer and infrastructure concerns. This separation results in cleaner code and better testability.*
5. *Scalability: DDD's focus on modularization and bounded contexts allows for better scalability. Different parts of the system can be scaled independently based on demand.*

*In conclusion, the Domain-Driven Design architectural pattern was chosen for the EcoBikeRental project due to its ability to model complex business domains, align development with real-world requirements, and provide a structured approach to designing and building a maintainable and extensible application.*

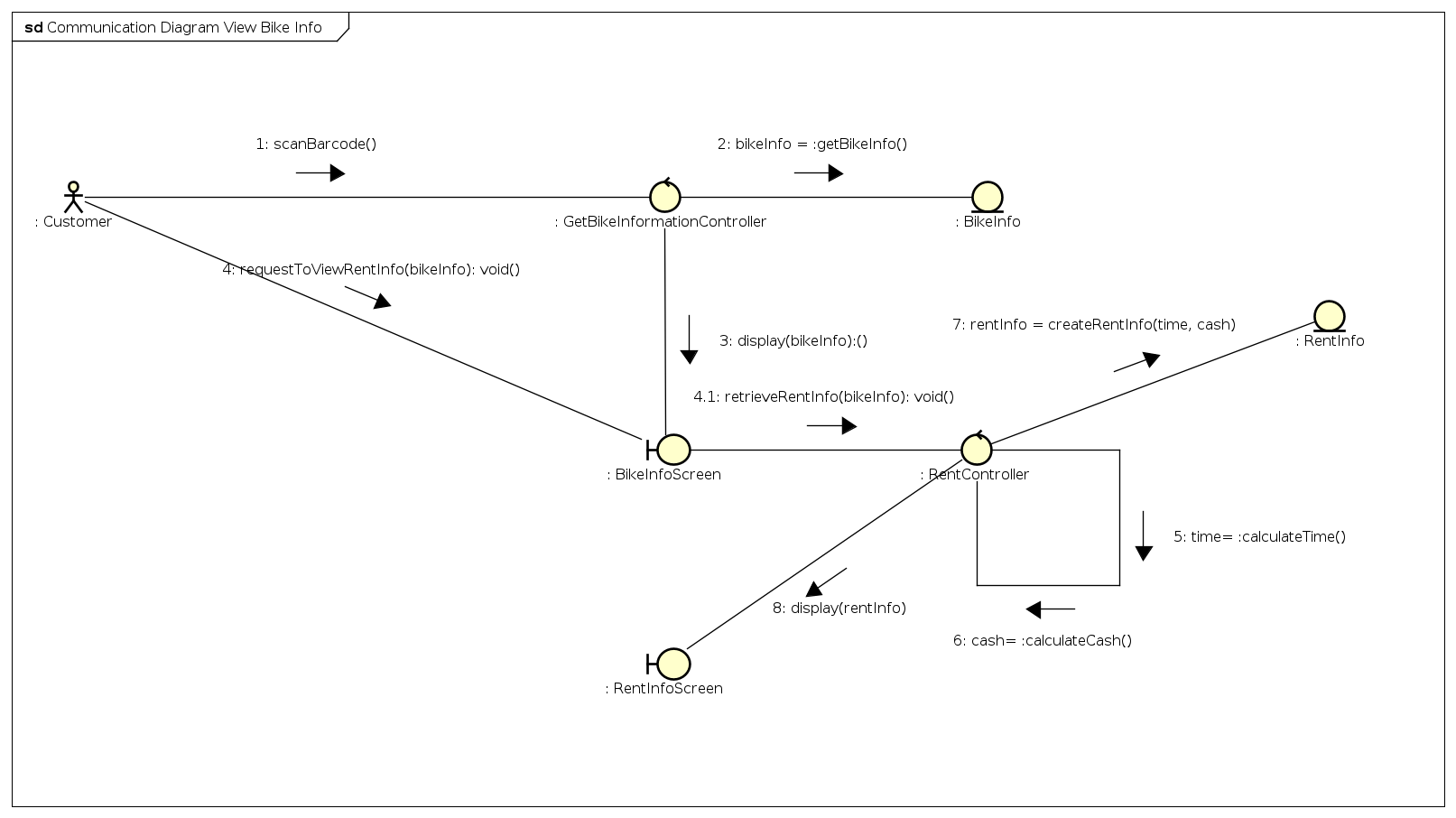
## 3.2. Interaction Diagrams

### 3.2.1. Communication Diagrams

|  |  |
| --- | --- |
|  |  |
|  |  |

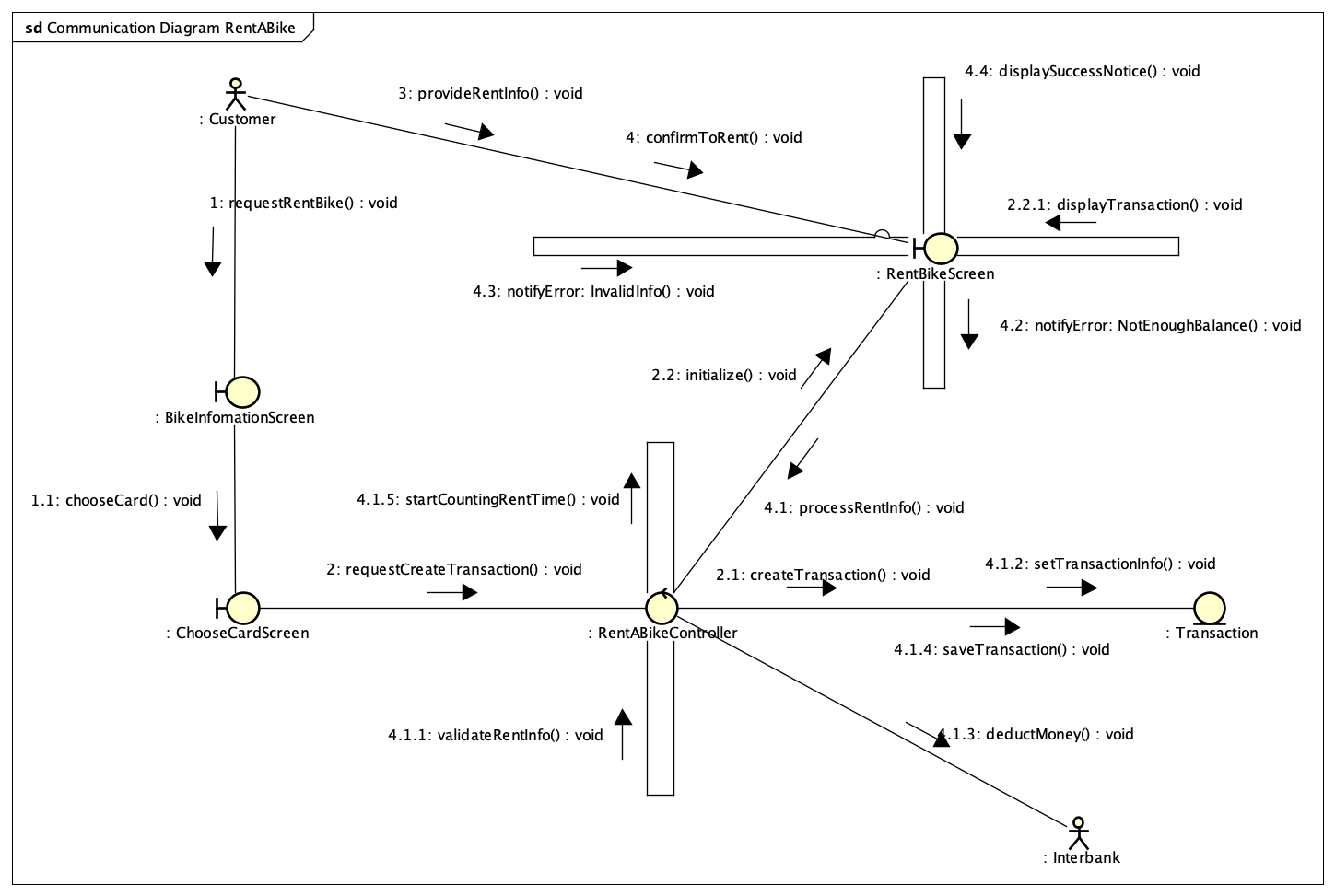
*Figure 3.1: Communication Diagram for Select Dock use case*

*Figure 3.2: Communication Diagram for Check Bike Information use case*



*Figure 3.3: Communication Diagram for Choose Credit Card use case*

*Figure 3.4: Communication Diagram for Rent A Bike use case*



|  |  |
| --- | --- |
|  |  |

*Figure 3.5: Communication Diagram for Return Bike use case*

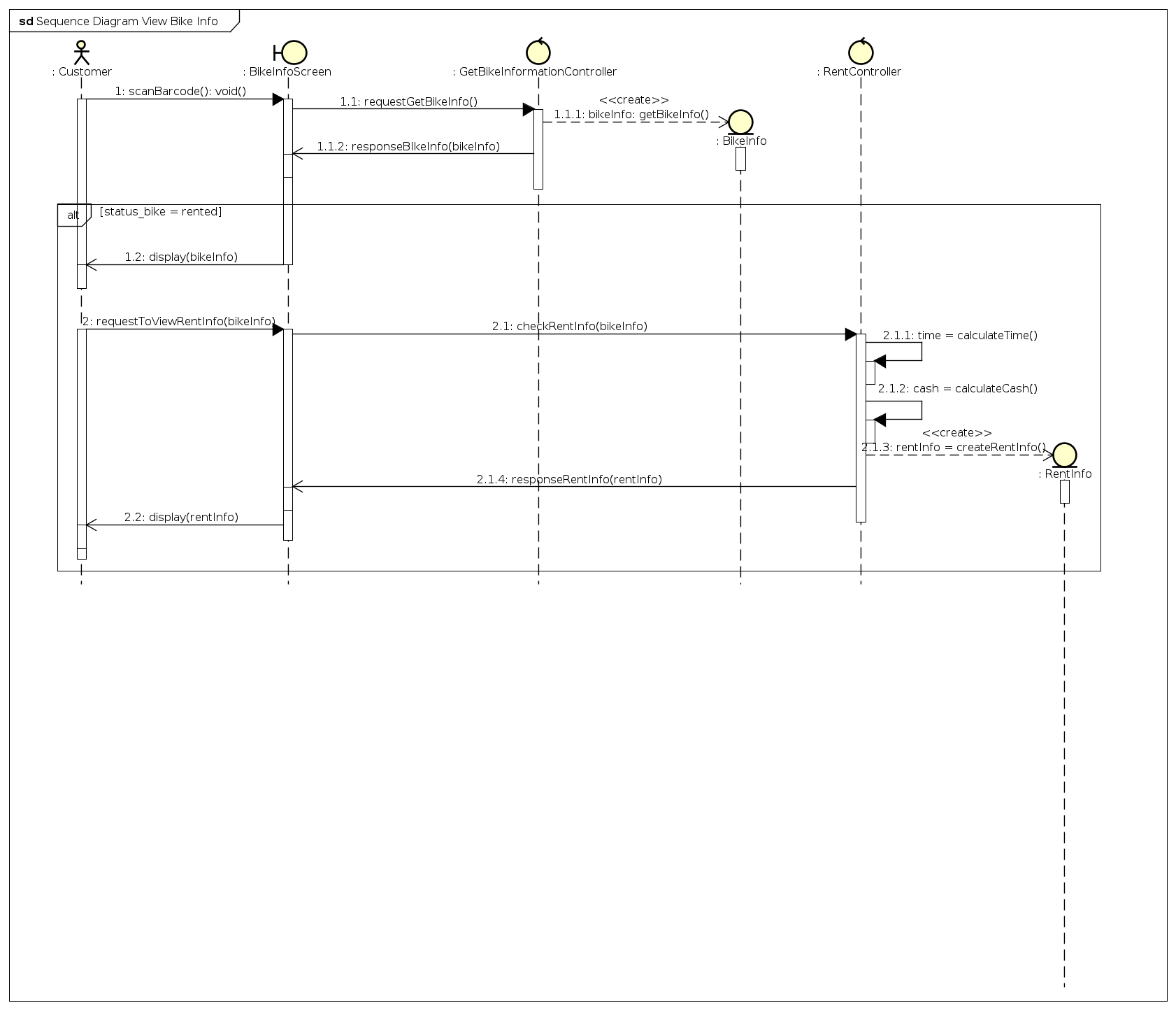
*Figure 3.6: Communication Diagram for Pay For Rental use case*

### 3.2.2. Sequence Diagrams

|  |  |
| --- | --- |
|  |  |
|  |  |

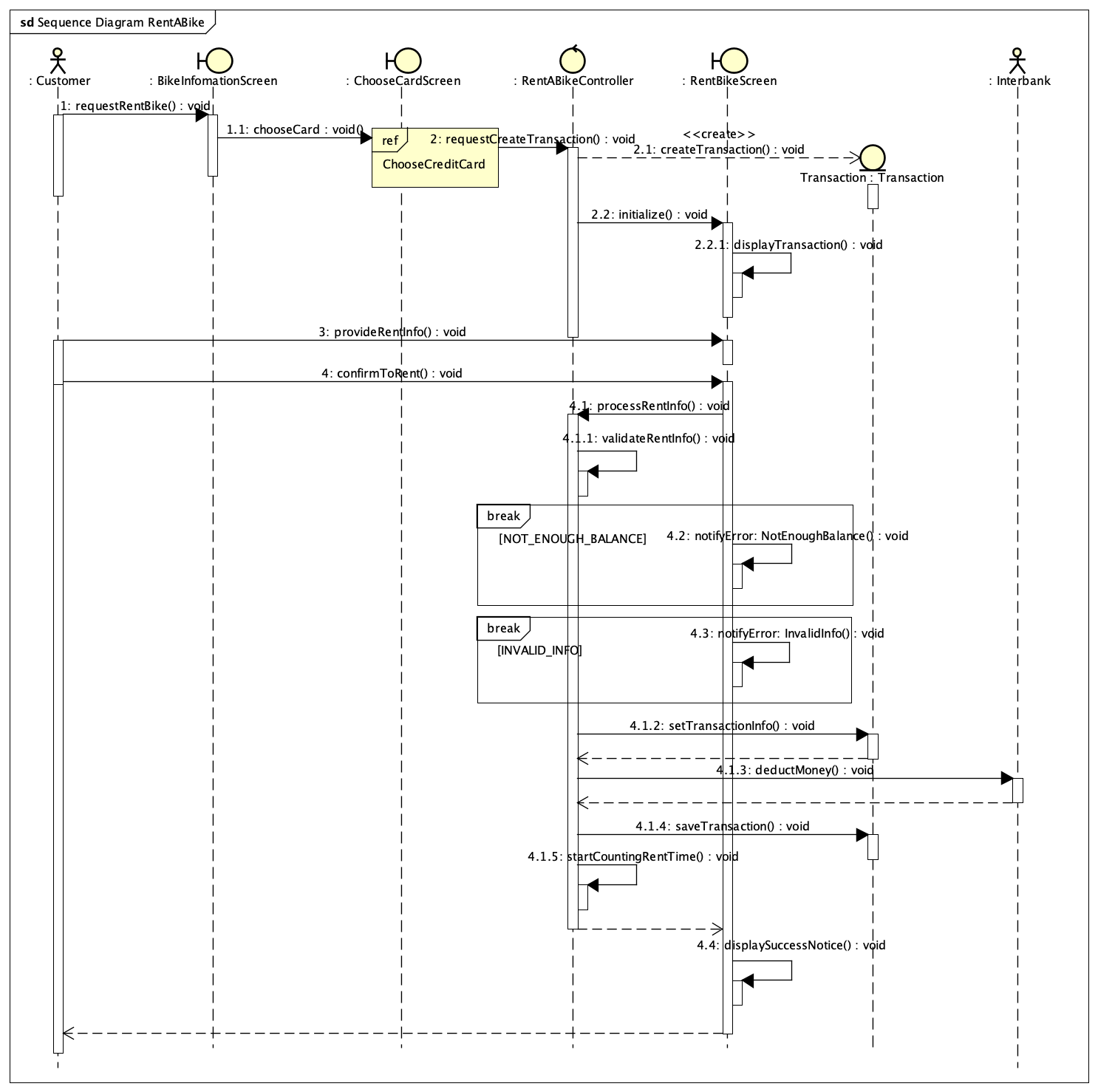
*Figure 3.7: Sequence Diagram for Select Dock use case*

*Figure 3.8: Sequence Diagram for Check Bike Information use case*



*Figure 3.9: Sequence Diagram for Choose Credit Card use case*

*Figure 3.10: Sequence Diagram for Rent A Bike use case*



*Figure 3.11: Sequence Diagram for Return Bike use case*

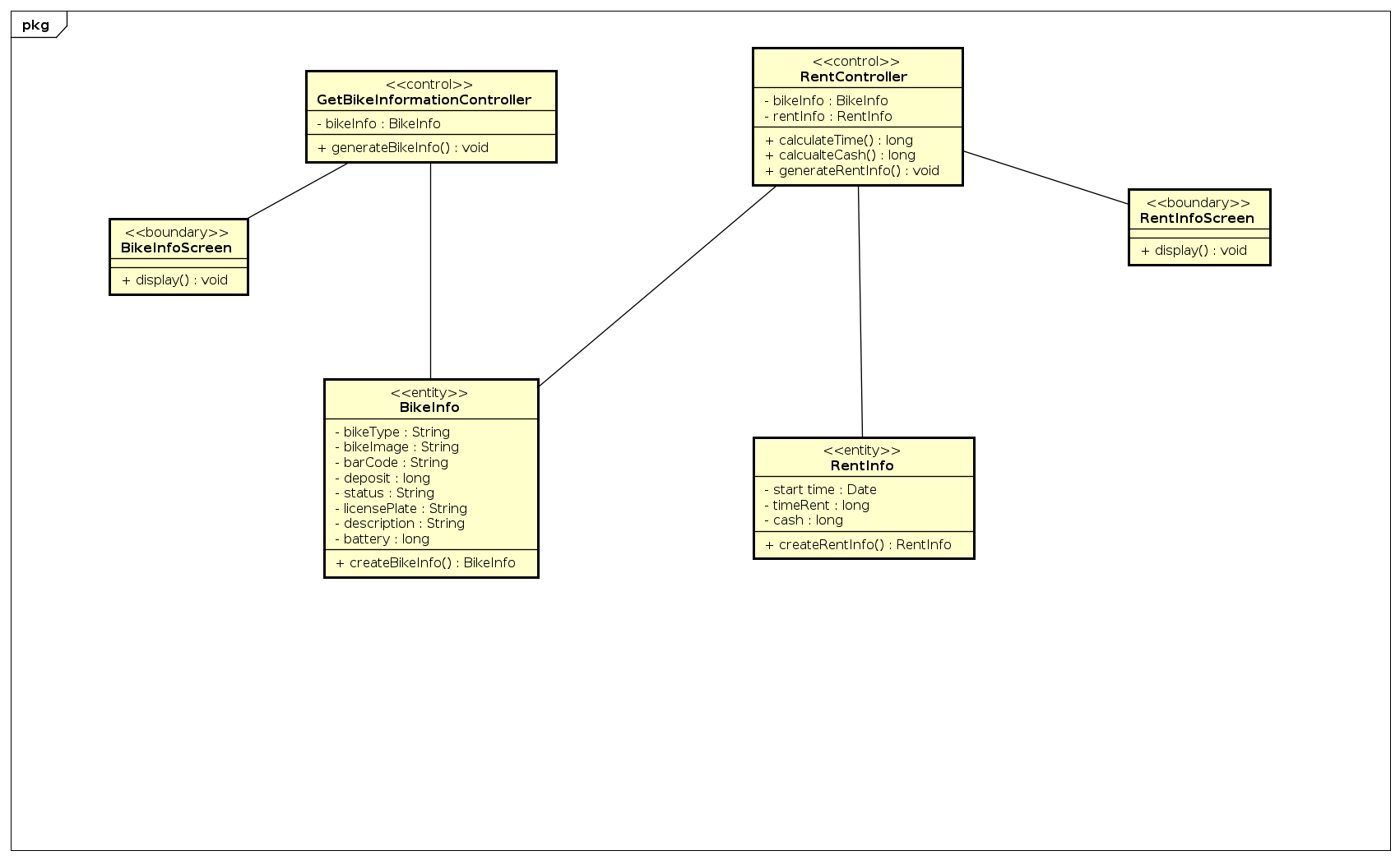
*Figure 3.12: Sequence Diagram for Pay For Rental use case*

## 3.3. Analysis Class Diagrams

|  |  |
| --- | --- |
|  |  |
|  |  |

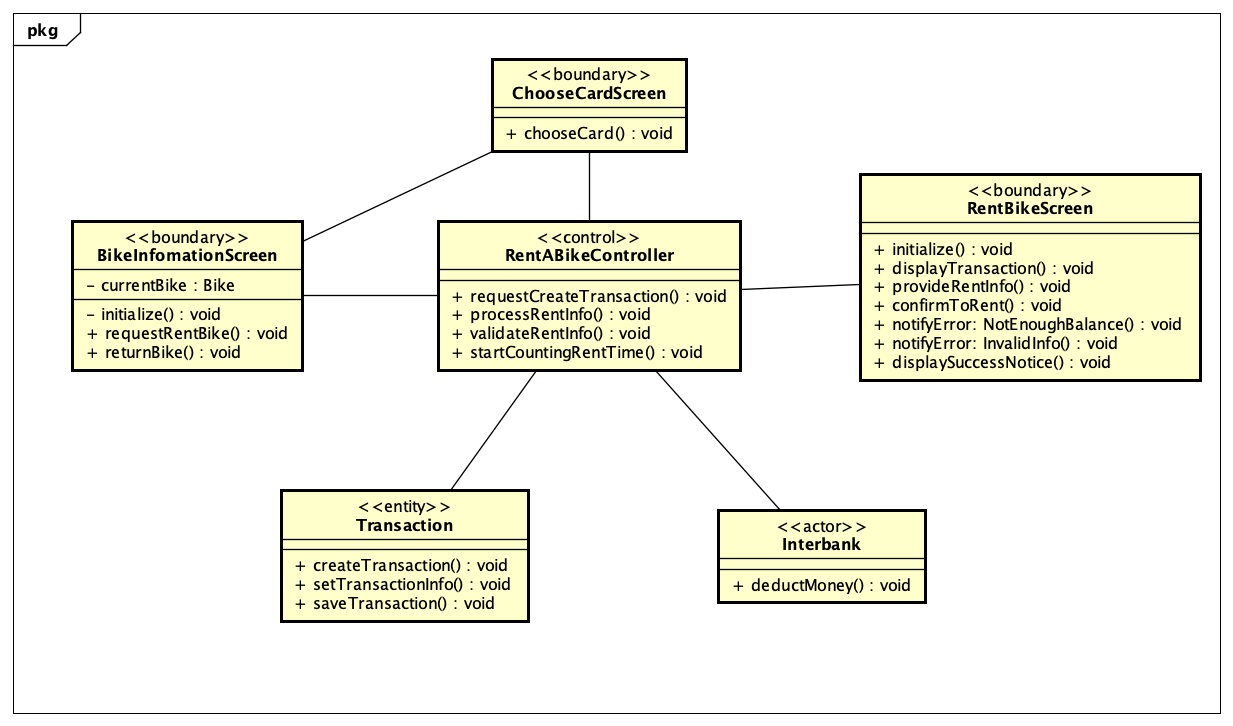
*Figure 3.13: Class Diagram for Select Dock use case*

*Figure 3.14: Class Diagram for Check Bike Information use case*



*Figure 3.15: Class Diagram for Choose Credit Card use case*

*Figure 3.16: Class Diagram for Rent A Bike use case*



*Figure 3.17: Class Diagram for Return Bike use case*

*Figure 3.18: Class Diagram for Pay For Rental use case*

## 3.4. Unified Analysis Class Diagram

|  |  |
| --- | --- |
|  |  |
|  |  |

*Figure 3.19: Unified Class Diagram for EcoBikeRental Application*

## 3.5. Security Software Architecture

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

# 4. Detailed Design

## 4.1. User Interface Design

### 4.1.1. Screen Configuration Standardization

Display

***Screen resolution:*** 1366x768px

***Number of colors supported:*** 16,177,216 colors Screen

***Size:*** 1200 x 600px

***Main background color:*** #e6ebbc (R: 230, G: 235, B: 188)

***Location of buttons:*** Bottom center of the frame

***Logo:*** 100x200 px

***Header logo:*** 100x100 px, located top left of the screen

***Header/Screen title:*** Segoe UI, Bold, 24px, black

***Numbers:*** comma for thousand separation, dot for decimal separation

***Text:*** Segoe UI, size at most 24px

***Frame border (if necessary):*** bounded rectangle, dashed line with width of 3px, color #afc139 (R: 175, G:193, B:57)

### 4.1.2. Screen Transition Diagrams

|  |  |
| --- | --- |
|  |  |
|  |  |

*Figure 4.1: Screen Transition Diagram for EcoBikeRentalApplication*

### 4.1.3. Screen Specifications

#### 4.1.3.1. Main Screen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***EcoBikeRentalSoftware*** | | ***Date of creation*** | ***Approved by*** | ***Reviewed by*** | ***Person in charge*** |
| ***Screen specification*** | ***Main screen*** | *8/2023* |  |  | *Son* |
|  | | ***Control*** | ***Operation*** | ***Function*** | |
| *Header logo* | *Click* | *Return immediately to main screen* | |
| *Search bar* | *Type, select & click* | *Type in information and select search type to search for docks or bikes* | |
| *Main area* | *Initial* | *Display map at current location of users and nearby docks in term. The pins of docks can be clicked to see docks details* | |
|  |  |  |  |  |  |

*Table 4.2. Main Screen Specification*

#### 4.1.3.2. Dock Screen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***EcoBikeRentalSoftware*** | | ***Date of creation*** | ***Approved by*** | ***Reviewed by*** | ***Person in charge*** |
| ***Screen specification*** | ***View Dock screen*** | *8/2023* |  |  | *Son* |
|  |  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Control*** | ***Operation*** | ***Function*** |
| *Logo* | *Click* | *Return to the main screen immediately* |
| *Header* | *Initial* | *Display title of screen* |
| *Dock information* | *Initial* | *Display dock information* |
| *Return bike* | *Click* | *Allow user to start return bike process at the dock* |
| *Bike list* | *Click* | *Display brief details about bikes available in the current dock. Allow choosing each bike to see detailed information* |

*Table 4.3. View Dock Screen Specification*

#### 4.1.3.3. Bike Screen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***EcoBikeRentalSoftware*** | | ***Date of creation*** | ***Approved by*** | ***Reviewed by*** | ***Person in charge*** |
| ***Screen specification*** | ***View bike screen*** | *8/2023* |  |  | *Quy* |
|  | | ***Control*** | ***Operation*** | ***Function*** | |
| *Logo* | *Click* | *Return to the main screen immediately* | |
| *Header* | *Initial* | *Display title of screen* | |
| *Bike information* | *Initial* | *Display bike information* | |
| *Option pane* | *Click* | *Allow customer to perform renting, pause or return bike* | |
|  |  |  |  |  |  |

*Table 4.4. View Bike Screen Specification*

#### 4.1.3.4. Choose Credit Card Screen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***EcoBikeRental Software*** | | ***Date of creation*** | ***Approved by*** | ***Reviewed by*** | ***Person in charge*** |
| ***Screen specification*** | ***Choose Credit Card screen*** | *8/2023* |  |  | *Quang* |
|  | | ***Control*** | ***Operation*** | ***Function*** | |
| *Logo* | *Click* | *Return to the main screen immediately* | |
| *Header* | *Initial* | *Display title of screen* | |
| *Choose card* | *Initial* | *Display form of credit card* | |
| *Button* | *Click* | *Allow customer confirm to check and choose the card* | |
|  |  |  |  |  |  |

*Table 4.5. Payment Method Screen Specification*

#### 4.1.3.5. Transaction rent screen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***EcoBikeRental Software*** | | ***Date of creation*** | ***Approved by*** | ***Reviewed by*** | ***Person in charge*** |
| ***Screen specification*** | ***Transaction screen*** | *8/2023* |  |  | *Quang* |
|  | | ***Control*** | ***Operation*** | ***Function*** | |
| *Logo* | *Click* | *Return to the main screen immediately* | |
| *Header* | *Initial* | *Display title of screen* | |
| *Information of transaction* | *Initial* | *Display information of transaction* | |
| *Button* | *Click* | *Allow customer confirm to rent the bike and deposit* | |
|  |  |  |  |  |  |

*Table 4.6. Deposit Screen Specification*

#### 4.1.3.6. Return bike Screen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***EcoBikeRentalSoftware*** | | ***Date of creation*** | ***Approved by*** | ***Reviewed by*** | ***Person in charge*** |
| ***Screen specification*** | ***Return bike screen*** | *8/2023* |  |  | *Thanh* |
|  | | ***Control*** | ***Operation*** | ***Function*** | |
| *Logo* | *Click* | *Return to the main screen immediately* | |
| *Header* | *Initial* | *Display title of screen* | |
| *Information of return bike* | *Initial* | *Display information of return bike* | |
| *Buttons* | *Click* | *Allow customer to choose to return bike* | |
|  |  |  |  |  |  |

*Table 4.7. Return bike Screen Specification*

## 4.2. Data Modeling

### 4.2.1. Conceptual Data Modeling

|  |  |
| --- | --- |
|  |  |
|  |  |

*Figure 4.2. ER Diagram for EcoBikeRental Application*

### 4.2.2. Database Design

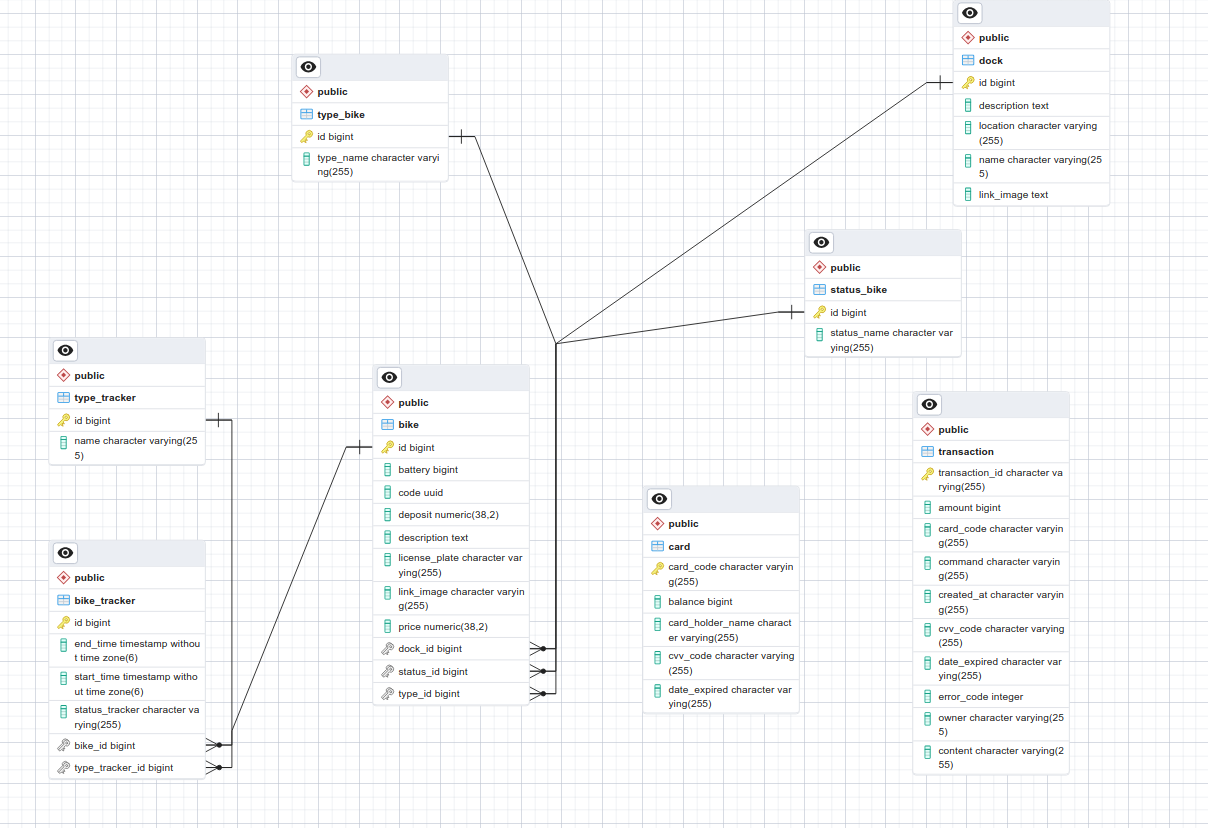
#### 4.2.2.1. Database Management System

Database Management System: Postgresql

#### 4.2.2.2.

|  |  |
| --- | --- |
|  |  |
|  |  |

#### Database Diagram



*Figure 4.3. Database Diagram for EcoBikeRentalApplication*

#### 4.2.2.3. Database Detail Design

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***No.*** | ***PK*** | ***FK*** | ***Name*** | ***Data type*** | ***Mandatory*** | ***Description*** |
| *1* | *x* | *x* | *dock\_id* | *Long* | *x* | *ID of dock* |
| *2* |  |  | *name* | *varchar(256)* | *x* | *Name of the dock* |
| *3* |  |  | *address* | *varchar(256)* | *x* | *Address of the dock* |
| *4* |  |  | *description* | *TEXT* |  | *Description of the dock* |
| *5* |  |  | *Link\_image* | *TEXT* |  | *Image of the dock* |

*Table 4.8. Dock table design*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***No.*** | ***PK*** | ***FK*** | ***Name*** | ***Data type*** | ***Mandatory*** | ***Description*** |
| *1* | x |  | *id* | *Long* | *x* | *Id of bike* |
| *2* |  |  | *bike\_name* | *varchar(256)* | *x* | *Name of the bike* |
| *3* |  | x | *Type\_id* | *Long* | *x* | *TypeId of bike* |
| *4* |  |  | *license\_plate\_code* | *varchar(32)* | *x* | *Code of the license plate of the bike* |
| *5* |  |  | *bike\_image* | *TEXT* |  | *Path to image of the bike* |
| *6* |  |  | *bike\_barcode* | *UUID* | *x* | *Barcode of the bike* |
| *7* |  |  | *bike\_rental\_price* | *float* | *x* | *Price to rent the bike* |
| *8* |  |  | *deposit\_price* | *float* | *x* | *Deposit cost to rent the bike* |
| *9* |  |  | *currency\_unit* | *varchar(3)* | *x* | *Currency unit used to calculate rental fee and deposit fee* |
| *10* |  |  | *description* | *TEXT* | *x* | *Description of bike* |
| *11* |  | x | *Status\_id* | *Long* | *x* | *StatusId of bike* |
| *12* |  | x | *Dock\_id* | *LONG* | *x* | *DockId of bike* |

*Table 4.9. Bike table design*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***No.*** | ***PK*** | ***FK*** | ***Name*** | ***Data type*** | ***Mandatory*** | ***Description*** |
| *1* | x |  | *id* | *Long* | *x* | *Id of the status\_bike* |
| *2* |  |  | *Status\_name* | *String* | *x* | *Name of the status* |

*Table 4.10. Status\_Bike table design*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***No.*** | ***PK*** | ***FK*** | ***Name*** | ***Data type*** | ***Mandatory*** | ***Description*** |
| *1* | x |  | *id* | *Long* | *x* | *Id of the type\_bike* |
| *2* |  |  | *Type\_name* | *VARCHAR(256)* | *x* | *‘free’/’rent’* |

*Table 4.11. TYPE\_BIKE table design*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***No.*** | ***PK*** | ***FK*** | ***Name*** | ***Data type*** | ***Mandatory*** | ***Description*** |
| *1* | x |  | *id* | *Long* | *x* | *Id of the type\_bike* |
| *2* |  |  | *Type\_name* | *VARCHAR(256)* | *x* | *'MINUTE/HOUR/DAY’* |

*Table 4.12. TYPE\_TRACKER table design*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***No.*** | ***PK*** | ***FK*** | ***Name*** | ***Data type*** | ***Mandatory*** | ***Description*** |
| *1* | *x* |  | *invoice\_id* | *int* | *X* | *ID of the invoice* |
| *2* |  | *x* | *transaction\_id* | *int* | *x* | *ID of the transaction* |
| *3* |  | *x* | *customer\_id* | *int* | *x* | *ID of the customer* |

*Table 4.13. Invoice table design*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***No.*** | ***PK*** | ***FK*** | ***Name*** | ***Data type*** | ***Mandatory*** | ***Description*** |
| *1* | *x* |  | *transaction\_id* | *int* | *x* | *ID of transaction* |
| *2* |  |  | *transaction\_amount* | *int* | *x* | *The amount of money for the transaction* |
| *3* |  |  | *transaction\_time* | *DATETIME* | *x* | *Time the transaction is made* |
| *4* |  |  | *transaction\_detail* | *varchar(256)* |  | *The content of the transaction* |
| *5* |  | *x* | *creditcard\_number* | *int* | *x* | *The number of the credit card* |

*Table 4.14. Transaction table design*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***No.*** | ***PK*** | ***FK*** | ***Name*** | ***Data type*** | ***Mandatory*** | ***Description*** |
| *1* | *x* | *x* | *transaction\_id* | *Long* | *X* | *Id of transaction* |
| *2* |  |  | *amount* | *bigint* | *x* | *amount* |
| *3* |  |  | *Card\_code* | *VARCHAR(256)* | *x* |  |
| *4* |  |  | *command* | *VARCHAR(256)* | x |  |
| *5* |  |  | *Create\_at* | *date* | x |  |
| *6* |  |  | *Cvv\_code* | *VARCHAR(256)* | X |  |
| *7* |  |  | *Date\_expired* | *VARCHAR(256)* | X |  |
| *8* |  |  | *Error\_code* | *Interger* |  |  |
| *9* |  |  | *owner* | *VARCHAR(256)* | X |  |
| *10* |  |  | *content* | *VARCHAR(256)* | X |  |

*Table 4.15. Transaction table design*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***No.*** | ***PK*** | ***FK*** | ***Name*** | ***Data type*** | ***Mandatory*** | ***Description*** |
| *1* | *x* | *x* | *card\_code* | *Varchar(256)* | *X* | *code of the credit card* |
| *2* |  |  | *Card\_holder\_name* | *varchar(256)* | *x* | *Name of the cardholder* |
| *3* |  |  | *balance* | *bigint* | *x* | *Balance of the card* |
| *4* |  |  | *Cvv\_code* | *VARCHAR(256)* | *X* |  |
| *5* |  |  | *Date\_expired* | *VARCHAR(256)* | *X* |  |

*Table 4.16. Card table design*

Database script:

CREATE TABLE public.bike (

id bigint NOT NULL,

battery bigint,

code uuid,

deposit numeric(38,2),

description text,

license\_plate character varying(255),

link\_image character varying(255),

price numeric(38,2),

dock\_id bigint,

status\_id bigint,

type\_id bigint

);

ALTER TABLE public.bike OWNER TO postgres;

CREATE SEQUENCE public.bike\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER TABLE public.bike\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.bike\_id\_seq OWNED BY public.bike.id;

CREATE TABLE public.bike\_tracker (

id bigint NOT NULL,

end\_time timestamp(6) without time zone,

start\_time timestamp(6) without time zone,

status\_tracker character varying(255),

bike\_id bigint,

type\_tracker\_id bigint

);

ALTER TABLE public.bike\_tracker OWNER TO postgres;

CREATE SEQUENCE public.bike\_tracker\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER TABLE public.bike\_tracker\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.bike\_tracker\_id\_seq OWNED BY public.bike\_tracker.id;

CREATE TABLE public.card (

card\_code character varying(255) NOT NULL,

balance bigint,

card\_holder\_name character varying(255),

cvv\_code character varying(255),

date\_expired character varying(255)

);

ALTER TABLE public.card OWNER TO postgres;

CREATE TABLE public.dock (

id bigint NOT NULL,

description text,

location character varying(255),

name character varying(255),

link\_image text

);

ALTER TABLE public.dock OWNER TO postgres;

CREATE SEQUENCE public.dock\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER TABLE public.dock\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.dock\_id\_seq OWNED BY public.dock.id;

CREATE TABLE public.status\_bike (

id bigint NOT NULL,

status\_name character varying(255)

);

ALTER TABLE public.status\_bike OWNER TO postgres;

CREATE SEQUENCE public.status\_bike\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER TABLE public.status\_bike\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.status\_bike\_id\_seq OWNED BY public.status\_bike.id;

CREATE TABLE public.transaction (

transaction\_id character varying(255) NOT NULL,

amount bigint,

card\_code character varying(255),

command character varying(255),

created\_at character varying(255),

cvv\_code character varying(255),

date\_expired character varying(255),

error\_code integer,

owner character varying(255),

content character varying(255)

);

ALTER TABLE public.transaction OWNER TO postgres;

CREATE TABLE public.type\_bike (

id bigint NOT NULL,

type\_name character varying(255)

);

ALTER TABLE public.type\_bike OWNER TO postgres;

CREATE SEQUENCE public.type\_bike\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER TABLE public.type\_bike\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.type\_bike\_id\_seq OWNED BY public.type\_bike.id;

CREATE TABLE public.type\_tracker (

id bigint NOT NULL,

name character varying(255)

);

ALTER TABLE public.type\_tracker OWNER TO postgres;

CREATE SEQUENCE public.type\_tracker\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER TABLE public.type\_tracker\_id\_seq OWNER TO postgres;

ALTER SEQUENCE public.type\_tracker\_id\_seq OWNED BY public.type\_tracker.id;

ALTER TABLE ONLY public.bike ALTER COLUMN id SET DEFAULT nextval('public.bike\_id\_seq'::regclass);

ALTER TABLE ONLY public.bike\_tracker ALTER COLUMN id SET DEFAULT nextval('public.bike\_tracker\_id\_seq'::regclass);

ALTER TABLE ONLY public.dock ALTER COLUMN id SET DEFAULT nextval('public.dock\_id\_seq'::regclass);

ALTER TABLE ONLY public.status\_bike ALTER COLUMN id SET DEFAULT nextval('public.status\_bike\_id\_seq'::regclass);

ALTER TABLE ONLY public.type\_bike ALTER COLUMN id SET DEFAULT nextval('public.type\_bike\_id\_seq'::regclass);

ALTER TABLE ONLY public.type\_tracker ALTER COLUMN id SET DEFAULT nextval('public.type\_tracker\_id\_seq'::regclass);

ALTER TABLE ONLY public.bike

ADD CONSTRAINT bike\_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.bike\_tracker

ADD CONSTRAINT bike\_tracker\_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.card

ADD CONSTRAINT card\_pkey PRIMARY KEY (card\_code);

ALTER TABLE ONLY public.dock

ADD CONSTRAINT dock\_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.status\_bike

ADD CONSTRAINT status\_bike\_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.transaction

ADD CONSTRAINT transaction\_pkey PRIMARY KEY (transaction\_id);

ALTER TABLE ONLY public.type\_bike

ADD CONSTRAINT type\_bike\_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.type\_tracker

ADD CONSTRAINT type\_tracker\_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.bike\_tracker

ADD CONSTRAINT uk\_b2o14o6pvno77ink6cpvdfsnc UNIQUE (type\_tracker\_id);

ALTER TABLE ONLY public.bike\_tracker

ADD CONSTRAINT uk\_qo75a48pla9sg7pj30bfqdnao UNIQUE (bike\_id);

ALTER TABLE ONLY public.bike

ADD CONSTRAINT fk66c4v0s40vx60lx2tefn0j9wm FOREIGN KEY (type\_id) REFERENCES public.type\_bike(id);

ALTER TABLE ONLY public.bike

ADD CONSTRAINT fkjuy7f6ixjlyjbjasgs5j9qekh FOREIGN KEY (status\_id) REFERENCES public.status\_bike(id);

ALTER TABLE ONLY public.bike\_tracker

ADD CONSTRAINT fkl3dr0bdciht9vvlqcjlcpt5c1 FOREIGN KEY (type\_tracker\_id) REFERENCES public.type\_tracker(id);

ALTER TABLE ONLY public.bike\_tracker

ADD CONSTRAINT fklhqs317f17k3uqsdjl8xvref7 FOREIGN KEY (bike\_id) REFERENCES public.bike(id);

ALTER TABLE ONLY public.bike

ADD CONSTRAINT fkr0bi7ohl0b17b79g759fesmvr FOREIGN KEY (dock\_id) REFERENCES public.dock(id);

## 4.3. Non-Database Management System Files

# 5. Design Considerations

## 5.1. Goals and Guidelines

Goals:

1. Provide a user-friendly application
2. Provide an eye-catching interface and convenient experience for users
3. The response time for the system is 1 second at normal and 2 seconds during a peak load Guidelines:
4. Obligate the coding convention in Java, and OOP principles.
5. Avoid hard-coding
6. Write comments for codes
7. Structure the doc for maintenance

## 5.2. Architectural Strategies

Our intention is to reuse components

1. Programming Language: Java (backend), Javascript (frontend)
2. Database: Postgresql
3. UML: Astah

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

## 5.3. Coupling and Cohesion

### 5.3.1. Coupling

#### 5.3.1.1. Content coupling

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

#### 5.3.1.2. Common coupling

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

#### 5.3.1.3. Control coupling

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

#### 5.3.1.4. Stamp coupling

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| No module | No | Fix the method to accept only needed arguments instead of the accepting Bike entities as the argument |

#### 5.3.1.5. Data coupling

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

### 5.3.2. Cohesion

#### 5.3.2.1. Coincidental cohesion

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

#### 5.3.2.2. Logical cohesion

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| No module |  | No improvement |

#### 5.3.2.3. Temporal cohesion

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

#### 5.3.2.4. Procedure cohesion

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| No module |  | No improvement |

#### 5.3.2.5. Communicational cohesion

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| No module |  | No improvement |

#### 5.3.2.6. Sequential cohesion

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| No module |  | No improvement |

#### 5.3.2.7. Information cohesion

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| No module |  | No improvement |

#### 5.3.2.8. Functional cohesion

|  |  |  |
| --- | --- | --- |
| Related modules | Description | Improvement |
| Most of the modules |  | No improvement |

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

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

## 5.4. Design Principles

We design simple classes follow SOLID principles that means:

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

## 5.5. Design Patterns

1. Repository Pattern: we are using the Repository pattern in your infrastructure layer to abstract the data access operations for different entities. This helps in encapsulating the data access logic and provides a clean separation between your domain logic and data persistence.
2. Factory Method Pattern: we use the Factory Method pattern to create instances of different bike types or trackers. This would allow us to encapsulate the object creation process and make it more flexible for adding new types in the future.
3. Strategy Pattern: The way you calculate the rental fee based on different factors (hourly rate, bike type) could be improved using the Strategy pattern. We can define different fee calculation strategies as separate classes and switch between them based on the bike type or other conditions.
4. Facade Pattern: As our application grows, we are using the Facade pattern to provide a simplified interface to a complex subsystem of classes. This helps in providing a more user-friendly API to the clients and hides the complexities of the underlying system.
5. DTO (Data Transfer Object) Pattern: We’re using DTOs to transfer data between layers, which is a common practice. This helps in decoupling our internal domain objects from the external representation we expose through APIs.
6. Observer Pattern: We have components in frontend that need to react to changes in the backend. This pattern allows certain components to subscribe to changes and get notified when there are updates.
7. CRQS pattern: stand for Command and Query Responsibility Segregation, a pattern that separates read and update operations for a data store. Implementing CQRS in our application can maximize its performance, scalability, and security. The flexibility created by migrating to CQRS allows a system to better evolve over time and prevents update commands from causing merge conflicts at the domain level.