| **HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY**  **SCHOOL OF INFORMATION TECHNOLOGY AND COMMUNICATION**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***    **Software Design Document**  **Group 17**  **EcoBikeRental** |
| --- |
| | **Instructor:** | **Nguyễn Thị Thu Trang** | | --- | --- | | **Student:** | **Nguyễn Doãn Hoàng Thư - 20184310**  **Nguyễn Tấn Tiến - 20205172**  **Nguyễn Văn Tiến - 20205195**  **Vũ Đức Toản - 20205173** | |  |  |   **Hanoi, July 2023** |

1. **Introduction**
   1. **Objective**

This SDD is written for the purpose of giving the audience a clear view about the design of the software. The document’s intended audience is stakeholder and software designer/developer.

* 1. **Scope**

Product name: **EcoBikeRental Software**

Explain:

The software is for users to rent and return bikes automatically. EcoBikeRental is a 24/7 platform-independent system which allows novice users to use without any training. Users must have an account to use the system. The software allows users to directly choose a bike to rent and choose any bike station to return the bike, pay & view bill, and show detailed information about the station and bike.

Application:

Today, the demand for bicycles is higher than ever. Cycling is not only environmentally friendly, but also a very effective way to exercise. The main downside to this need is that not everyone has a bike or intends to buy one. How about renting a public bike for a relatively cheap price? Presenting our software. EcoBikeRental offers a fast and convenient way to rent a bike. Help to reduce the number of workers, save costs and time, very convenient. It is very easy to use. With many stations and bicycles available, it fulfills the demand for bicycle rental, especially in the Eco Park Township.

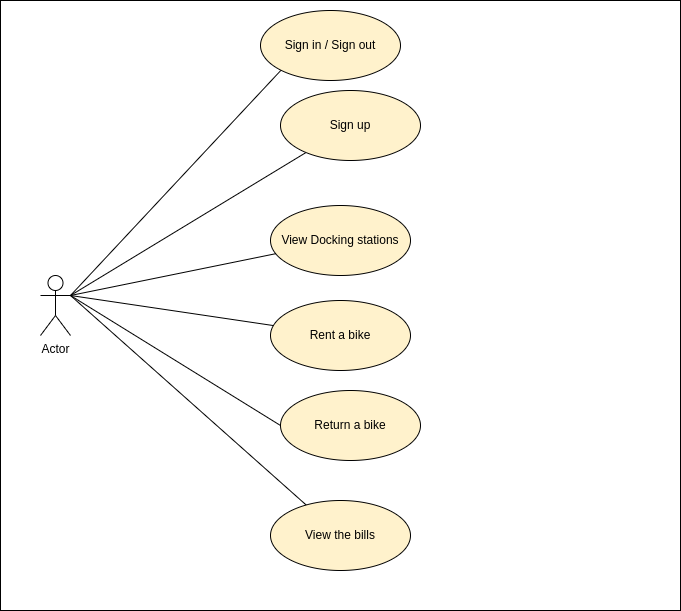
* 1. **Glossary**

We assume that the reader of this document has relatively good base knowledge about computer/software in general. Still, the document will be written in a general-audience-friendly way that most readers can understand. Scholarly terms, if any, in this document will be briefly explained after it has been used.

1. **Overall Description**
   1. **General Overview**

System-wise, we have several features that make the app look like an e-commerce website/software: An interface to interact with the user; user request by clicking on the interface, then the request is handled by the system controller; we have a database (remote) to store any kind of data; any request or modification regarding the data to be queried against the database; Database changes are then reflected in the user interface (user interface). As you can see there are three main components in the system: user interface, controller and data model. We have chosen this software as a web application. We have chosen MVC architecture as our design approach. The design architecture helps to separate the different components and organize the code base better.

Here is general use-case diagram to help you understand the core of our design:



1. **System Architecture and Architecture Design**
   1. **Architecture Design steps**

1. Find out software components -> use cases

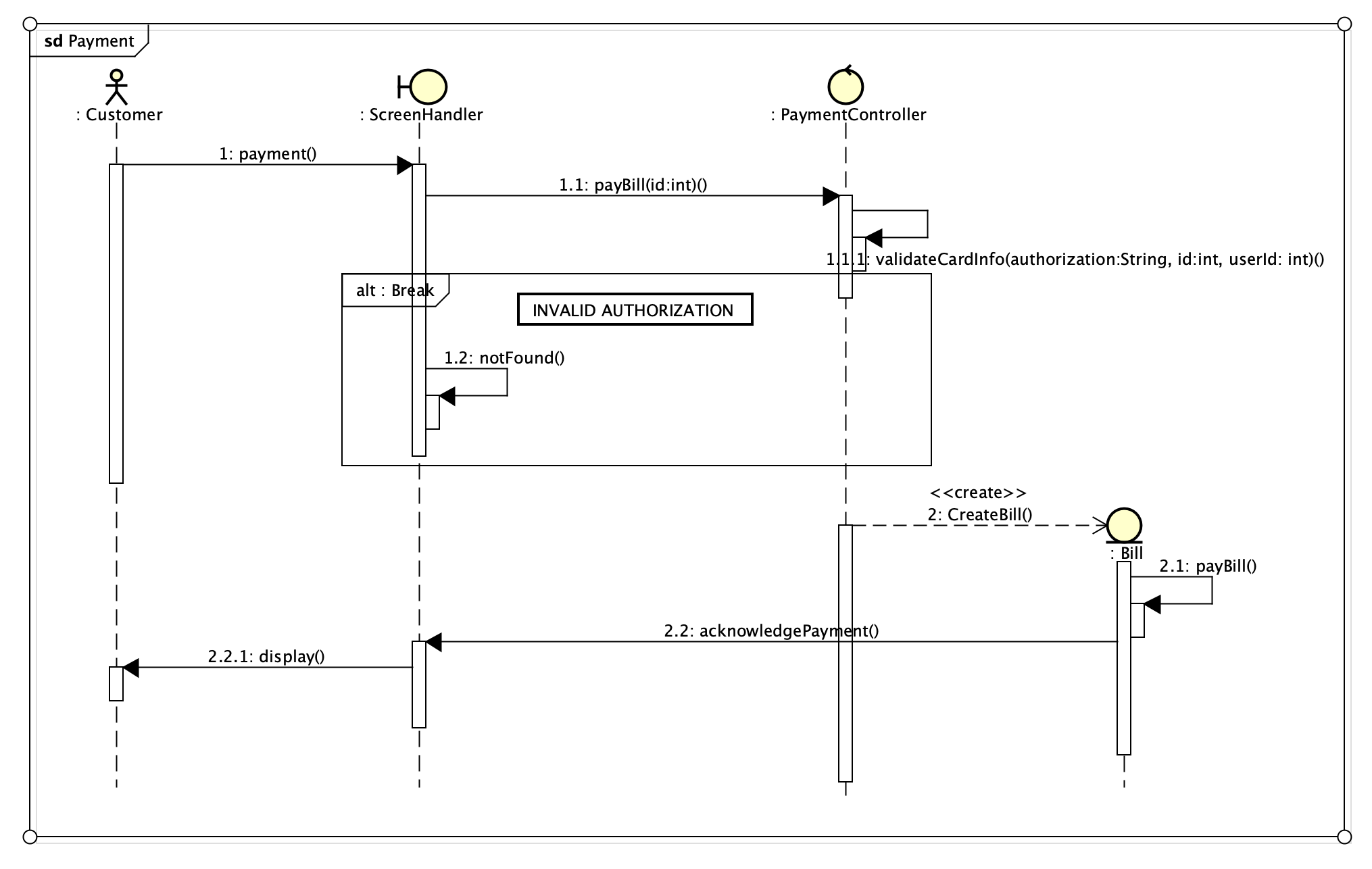
2. Find out Interaction between use cases

3. Find out Relationship between use cases

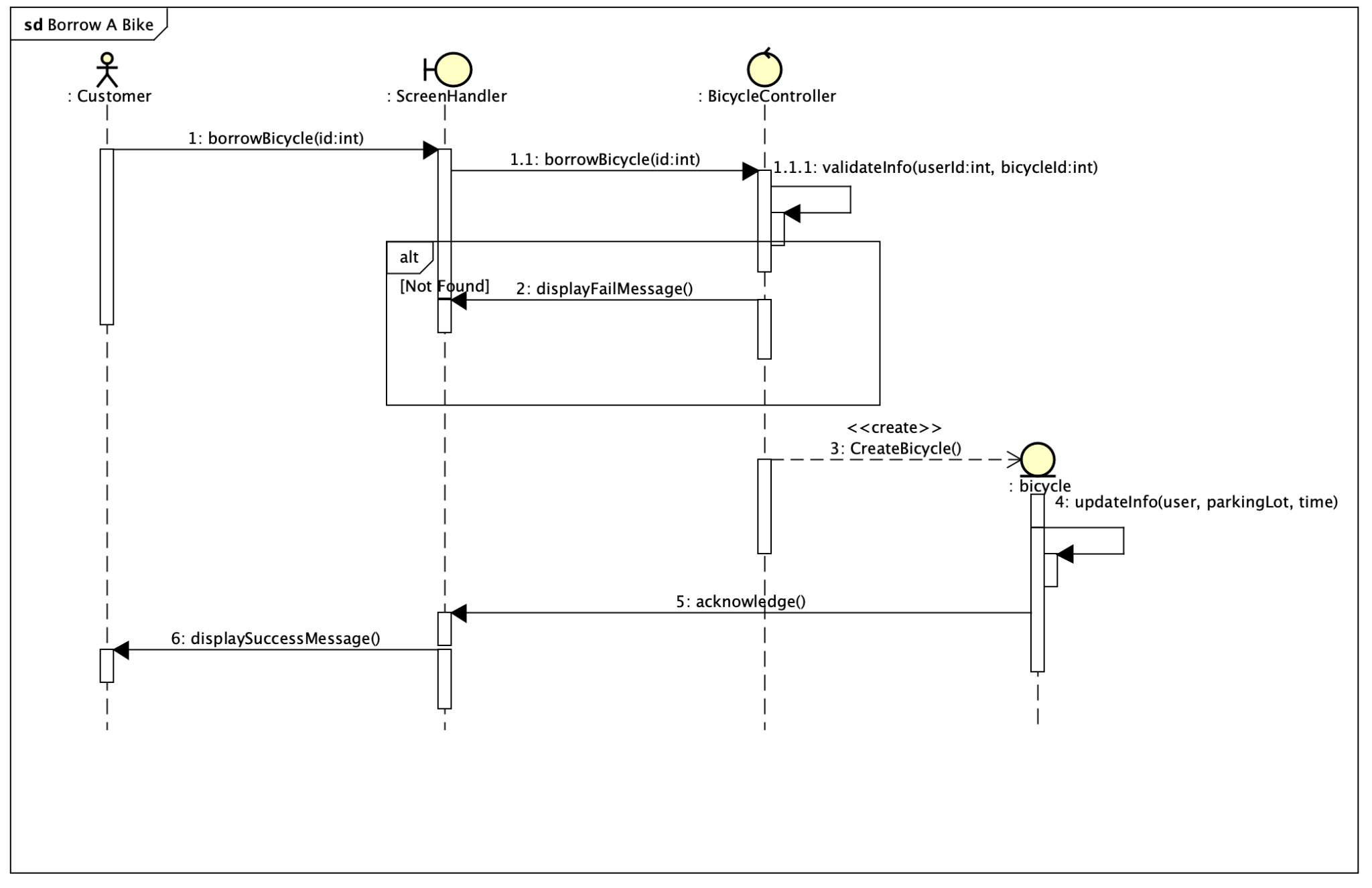
4. Draw UML Diagram includes: interaction diagram and analysis class diagram

* 1. **Interaction Diagram**

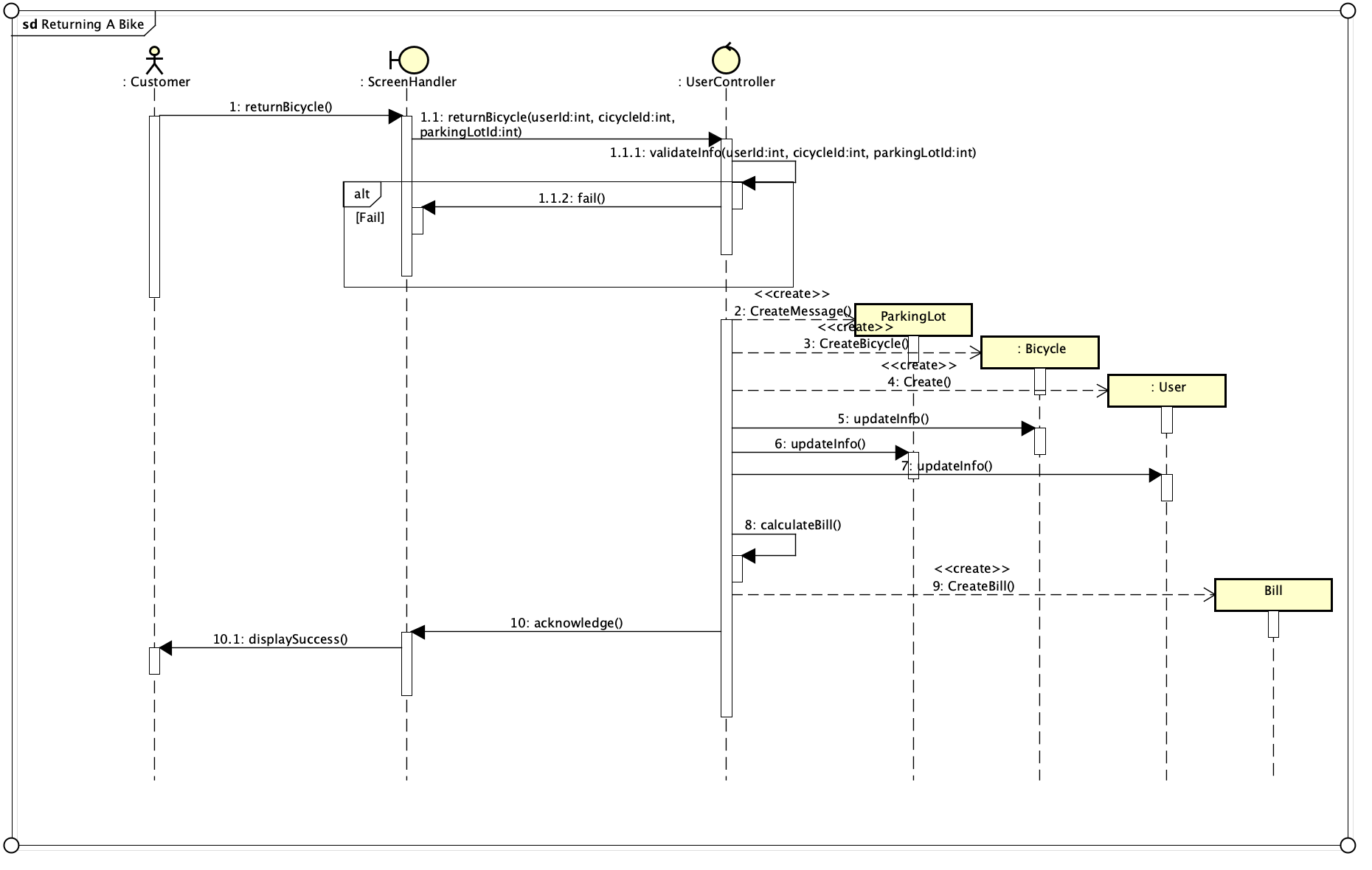
**Sequence Diagram for Payment:**



**Sequence Diagram for Borrow a Bike:**



**Sequence Diagram for Returning A Bike:**

****

1. **Detail Design**
   1. **User Interface Design**

**4.1.1 Screen Configuration Standardization**

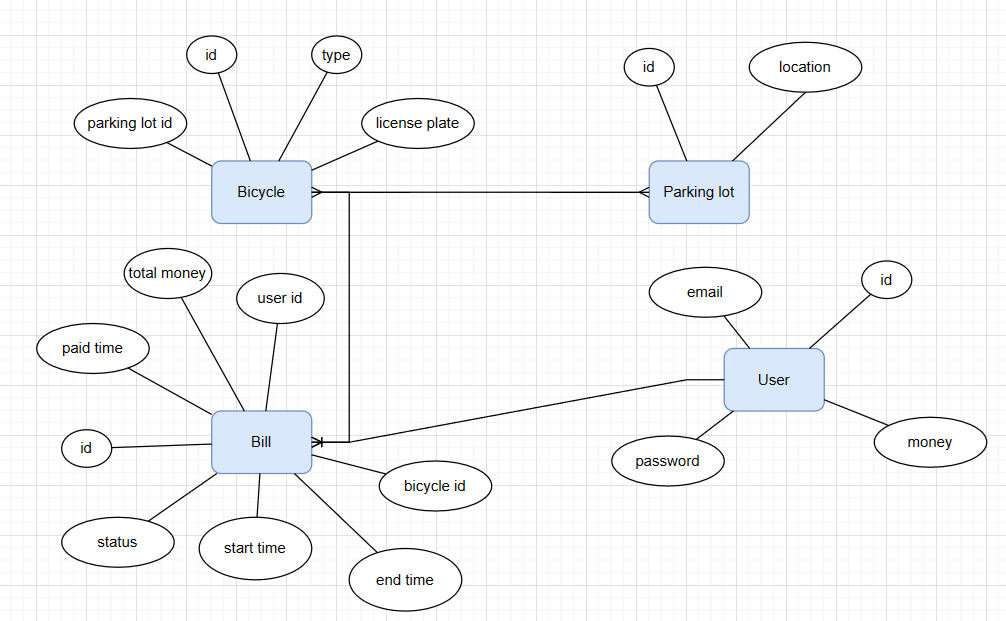
**Display**

Number of colors supported: 16,777,216

Resolution: 1080x1920 default

**Screen**

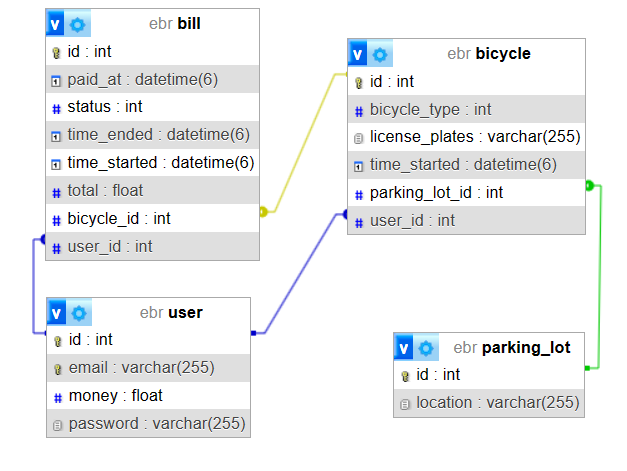
* Header for navigation sticky on top
* User’ account balance section
* Bill list and payment section
* List of Docking station include list of bike in the docking station section
  1. **Data Modeling**
     1. **Conceptual data modeling**



**4.2.2 Database Design**

**4.2.2.1 Database Management System**

**4.2.2.2 Logical Data Model**



Script database:

SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";

START TRANSACTION;

SET time\_zone = "+00:00";

CREATE DATABASE IF NOT EXISTS `ebr` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci;

USE `ebr`;

CREATE TABLE `bicycle` (

`id` int NOT NULL,

`bicycle\_type` int DEFAULT NULL,

`license\_plates` varchar(255) DEFAULT NULL,

`time\_started` datetime(6) DEFAULT NULL,

`parking\_lot\_id` int DEFAULT NULL,

`user\_id` int DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

REATE TABLE `bill` (

`id` int NOT NULL,

`paid\_at` datetime(6) DEFAULT NULL,

`status` int DEFAULT NULL,

`time\_ended` datetime(6) DEFAULT NULL,

`time\_started` datetime(6) DEFAULT NULL,

`total` float DEFAULT NULL,

`bicycle\_id` int DEFAULT NULL,

`user\_id` int DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `parking\_lot` (

`id` int NOT NULL,

`location` varchar(255) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `user` (

`id` int NOT NULL,

`email` varchar(255) NOT NULL,

`password` varchar(255) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

ALTER TABLE `bicycle`

ADD PRIMARY KEY (`id`),

ADD KEY `FKcjvg9ri9ihmrpvr2uub6wdkc6` (`parking\_lot\_id`),

ADD KEY `FKq70m7a425umwobugdyt6371hw` (`user\_id`);

ALTER TABLE `bill`

ADD PRIMARY KEY (`id`),

ADD KEY `FK1p6npys7pwnq4ulg3rgj9k228` (`bicycle\_id`),

ADD KEY `FKqhq5aolak9ku5x5mx11cpjad9` (`user\_id`);

ALTER TABLE `parking\_lot`

ADD PRIMARY KEY (`id`);

ALTER TABLE `user`

ADD PRIMARY KEY (`id`),

ADD UNIQUE KEY `UK\_ob8kqyqqgmefl0aco34akdtpe` (`email`);

ALTER TABLE `bicycle`

MODIFY `id` int NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=11;

ALTER TABLE `bill`

MODIFY `id` int NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=2;

ALTER TABLE `parking\_lot`

MODIFY `id` int NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=3;

ALTER TABLE `user`

MODIFY `id` int NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=2;

ALTER TABLE `bicycle`

ADD CONSTRAINT `FKcjvg9ri9ihmrpvr2uub6wdkc6` FOREIGN KEY (`parking\_lot\_id`) REFERENCES `parking\_lot` (`id`),

ADD CONSTRAINT `FKq70m7a425umwobugdyt6371hw` FOREIGN KEY (`user\_id`) REFERENCES `user` (`id`);

ALTER TABLE `bill`

ADD CONSTRAINT `FK1p6npys7pwnq4ulg3rgj9k228` FOREIGN KEY (`bicycle\_id`) REFERENCES `bicycle` (`id`),

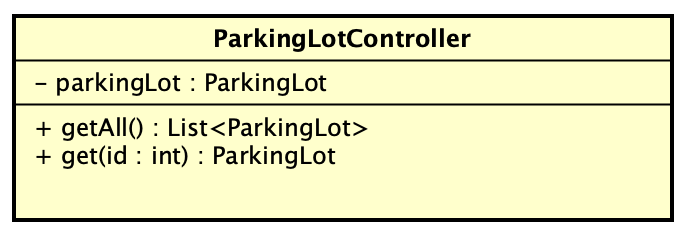
ADD CONSTRAINT `FKqhq5aolak9ku5x5mx11cpjad9` FOREIGN KEY (`user\_id`) REFERENCES `user` (`id`);

COMMIT;

**4.3 Class Design**

**4.3.1 Class Design**

**4.3.1.1 Class “ParkingLotController”**



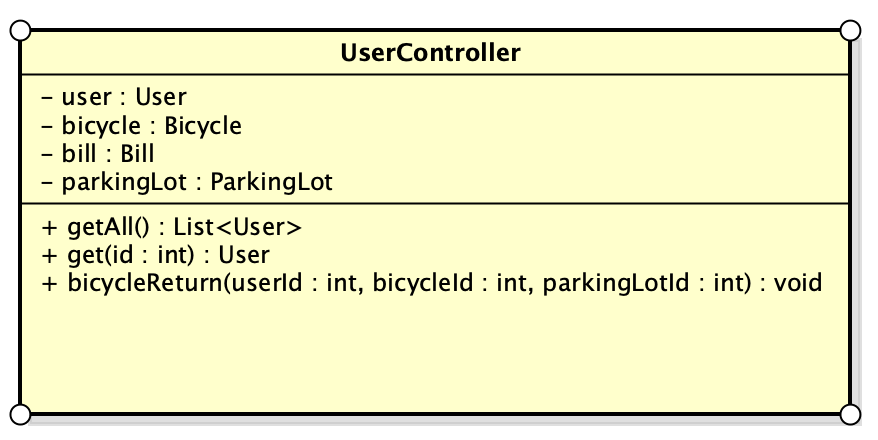
**Attribute**

| # | Name | Data Type | Default Value | Description |
| --- | --- | --- | --- | --- |
| 1 | parkingLot | ParkingLot | NULL | The parking lot used in the rent /return process |

**Operation**

| # | Name | Return type | Description (Purpose) |
| --- | --- | --- | --- |
| 1 | getAll() | A list of Parking Lot | Get all the parking lot in the database |
| 2 | get(int: id) | A parking lot | Get a specific parking lot matching the provided ID |

**4.3.1.2 Class “UserController”**



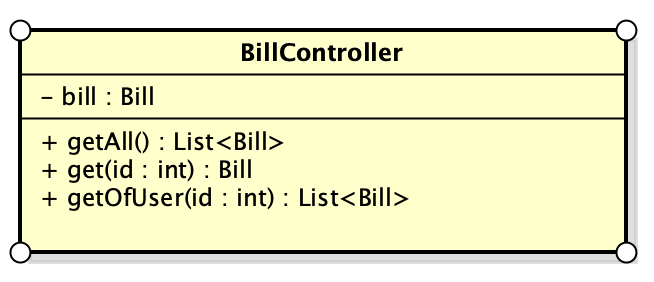
**Attribute**

| # | Name | Data Type | Default Value | Description |
| --- | --- | --- | --- | --- |
| 1 | parkingLot | ParkingLot | NULL | The parking lot used in the rent /return process |
| 2 | user | User | NULL | The user used in the return a bike operation |
| 3 | bill | Bill | NULL | The bill used in the return bike operation |
| 4 | bicycle | Bicycle | NULLL | The bicycle used in the return bike operation |

**Operation**

| # | Name | Return type | Description (Purpose) |
| --- | --- | --- | --- |
| 1 | getAll() | A list of User | Get all the users in the database |
| 2 | get(id: int) | A user | Get a specific user matching the provided ID |
| 3 | bicycleReturn(userId : int, bicycleId : int, parkingLotId : int) | void | Return a rented bicycle to a Parking Lot |

**4.3.1.3 Class “BillController”**



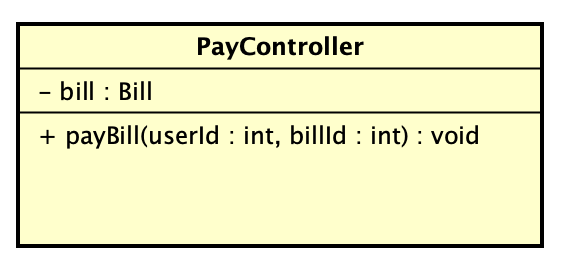
**Attribute**

| 1 | bill | Bill | NULL | The bill used in the return bike operation |
| --- | --- | --- | --- | --- |

**Operation**

| # | Name | Return type | Description (Purpose) |
| --- | --- | --- | --- |
| 1 | getAll() | A list of Bill | Get all the bills in the database |
| 2 | get(id: int) | A bill | Get a specific bill matching the provided ID |
| 3 | getOfUser(id: int) | A list of Bill | Return a list of bills belonging to an user |

**4.3.1.4 Class “PayController”**

****

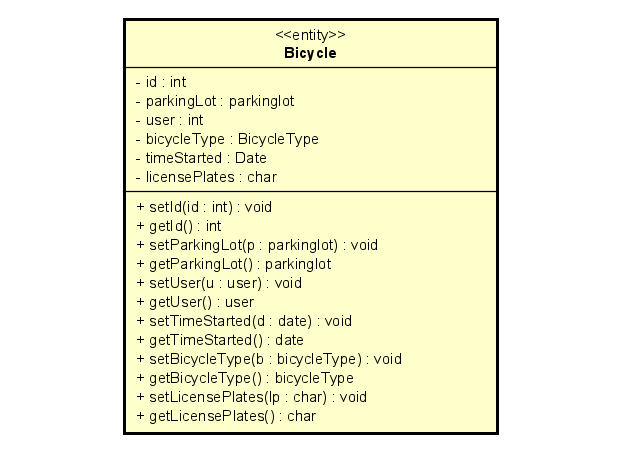
**Attribute**

| 1 | bill | Bill | NULL | The bill used in the return bike operation |
| --- | --- | --- | --- | --- |

**Operation**

| # | Name | Return type | Description (Purpose) |
| --- | --- | --- | --- |
| 1 | payBill(userId : int, billId : int) | NULL | Find the bill and the corresponding user. Turn the status of the bill to “PAID” |

**4.3.1.5 Class “Bicycle”**



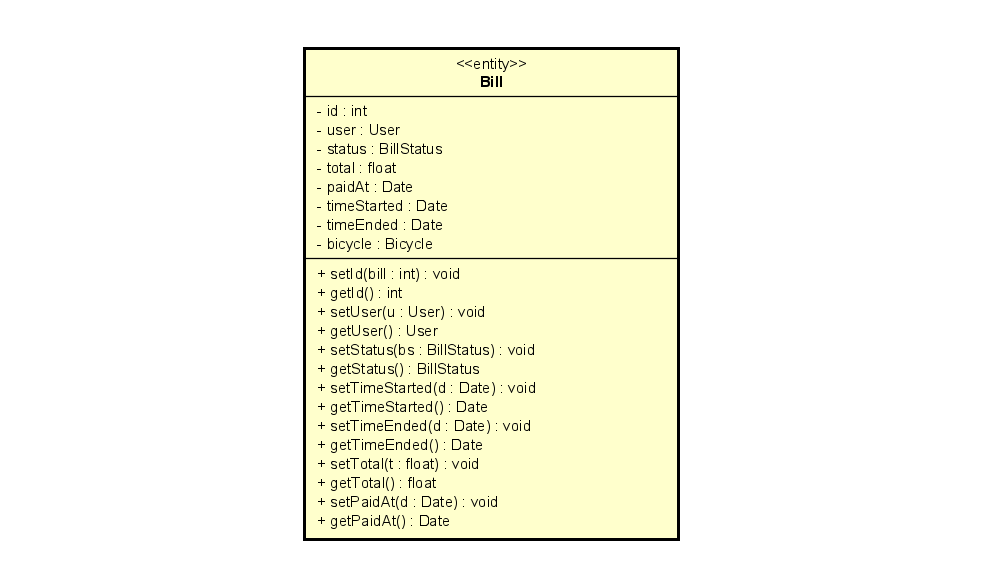
**Attribute**

| # | Name | Datatype | Default value | Description |
| --- | --- | --- | --- | --- |
| 1 | id | int |  | Identify |
| 2 | parkingLot | parkinglot |  | Parking station of this bicycle |
| 3 | user | int |  | User rent bicycle |
| 4 | bicycleType | Bicycle Type |  | Type of bicycle |
| 5 | timeStarted | Date |  | Start time when rent bicycle |
| 6 | licensePlates | char |  | License plate of bicycle |

**Operation**

| # | Name | Return type | Description(purpose) |
| --- | --- | --- | --- |
| 1 | setId(id : int) | void |  |
| 2 | getId() | id |  |
| 3 | setParkingLot(p : parkinglot) | void |  |
| 4 | getParkingLot() | parkinglot |  |
| 5 | setUser(u : user) | void |  |
| 6 | getUser() | user |  |
| 7 | setTimeStarted(d : date) | void |  |
| 8 | getTimeStarted() | date |  |
| 9 | setBicycleType(b : bicycleType) | void |  |
| 10 | getBicycleType() | bicycleType |  |
| 11 | setLicensePlates(lp : char) | void |  |
| 12 | getLicensePlates() | char |  |

**4.3.1.6 Class “Bill”**



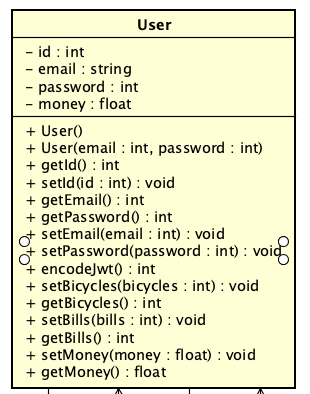
**Attribute**

| # | Name | Data type | Default value | Description |
| --- | --- | --- | --- | --- |
| 1 | id | int |  |  |
| 2 | user | User |  |  |
| 3 | status | BillStatus |  |  |
| 4 | total | float |  |  |
| 5 | paidAt | Date |  | Time bill's paid |
| 6 | timeStarted | Date |  | Time borrow bike |
| 7 | timeEnded | Date |  | Time return bike |
| 8 | bicycle | Bicycle |  |  |

**Operation**

| # | Name | Return type | Description |
| --- | --- | --- | --- |
| 1 | setId | void |  |
| 2 | getId | int |  |
| 3 | setUser | void |  |
| 4 | getUser | User |  |
| 5 | setStatus | void |  |
| 6 | getStatus | BillStatus |  |
| 7 | setTimeStarted | void |  |
| 8 | getTimeStarted | Date |  |
| 9 | setTimeEnded | void |  |
| 10 | getTimeEnded | Date |  |
| 11 | setPaidAt | void |  |
| 12 | getPaidAt | Date |  |
| 13 | getTotal | float |  |

**4.3.2.7 Class User**

****

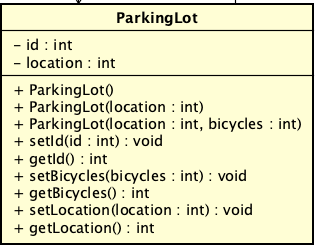
**Attribute**

| # | Name | Data type | Default value | Description |
| --- | --- | --- | --- | --- |
| 1 | id | int |  | The user's full name. |
| 2 | email | string |  | Unique user identifier. |
| 3 | password | int |  | Hashed user password. |
| 4 | money | float |  | User's account balance. |

**Operation**

| # | Name | Return type | Description |
| --- | --- | --- | --- |
| 1 | + User() | void |  |
| 2 | + User(email : int, password : int) | void |  |
| 3 | + getId() : int | int | Returns the user's unique ID. |
| 4 | + setId(id : int) : void | int | Sets the user's unique ID. |
| 5 | + getEmail() : int | int | Returns the user's email address. |
| 6 | + getPassword() : int | int | Returns the user's hashed password. |
| 7 | + setEmail(email : int) : void | void | Sets the user's email address. |
| 8 | + setPassword(password : int) : void | void | Sets the user's password. |
| 9 | + encodeJwt() : int | int | Encodes and returns a JWT (JSON Web Token) for the user. |
| 10 | + setBicycles(bicycles : int) : void | void | Sets the number of bicycles associated with the user. |
| 11 | + getBicycles() : int | int | Returns the number of bicycles associated with the user |
| 12 | + setBills(bills : int) : void | void | Sets the number of bills associated with the user |
| 13 | + getBills() : int | int | Returns the number of bills associated with the user. |
| 14 | + setMoney(money : float) : void | void | Sets the user's account balance. |
| 15 | + getMoney() : float | float | Returns the user's account balance. |

**4.3.2.2 Class “ParkingLot”**

****

**Attribute**

| # | Name | Data type | Default value | Description |
| --- | --- | --- | --- | --- |
| 1 | id | int |  |  |
| 2 | location | int |  | Location id |

**Operation**

| # | Name | Return type | Description |
| --- | --- | --- | --- |
| 1 | setId | void | Sets the unique ID of the parking lot |
| 2 | getId | int | Returns the unique ID of the parking lot. |
| 3 | setBicycle | void | Sets the bicycle or bicycles currently parked at the lot. |
| 4 | getBicycle | List<Bicycle> | Returns the bicycle or bicycles currently parked at the lot. |
| 5 | setLocation | void | Sets the geographical location or address of the parking lot. |
| 6 | getLocation | Location | Returns the geographical location or address of the parking lot. |

1. **Design and considerations**

**5.1 Goal and Guidelines**

Goal:

* Deliver a seamless and friendly user interface

Guidelines:

* Observe java convention in coding, OOP principles
* Avoid hash code

**5.2 Architectural Strategies**

Our design decisions relied on the SOLID principles in maximizing cohesion and minimizing coupling between components. We used the following technology:

* Programming Language: java, html
* Database: MySQL
* Sever: docker

**5.3 Coupling and Cohesion**

In our software design, we detect that there is still room for improvement. We have tried our best to resolve these problems, decreasing Coupling level and increasing Cohesion level. However, due to the lack of time, parts of the code might still be at fault.

We will try to resolve these issues according to your suggestion as soon as possible.

**5.4 Design Principles**

Our design decision focus on reusing components, unified system following the SOLID principles:

* Singularity: A class should have only one job, one responsibility
* Open & Close: Software entities are open for extension but close for modification
* Liskov Substitution Principle
* Interface Segregation Principle: We use specific interfaces if necessary instead of using general purpose interfaces
* Dependency Inversion Principle: We put all classes with the same properties into one package to manage them more easily.