# Design Document for Federalised Bike System

Team Member 1: Iman Abubakar
UUN: s1891419

Team Member 2: Abigail Agyemang Appah
UUN: s1871633

## Introduction

The federalised bike rental system is a system that allows potential customers to find and book from different providers throughout Scotland. This system was proposed by the Scotland Tourism Board to boost the tourism as it will allow visitors to easily find providers that meet their needs. To know more about the system and what it entails please refer to course work 1.

## High Level Description

In this description, we focused mainly on the parts of our design which might not be very clear, and these parts are explained below.

Since the customer will need to provide some rental needs before he/she get quotes, we decided to have a customer requirement class (CustomerReq) to capture these needs of the customer. The system class has a method generateQuotes that will compare quotes with the rental needs and these matching quotes will be presented to the customer via the findQuotes method in the customer class. We also included a HashMap of the bike type and the number which stores the number of bike types requested by the customer for each specific bike.

A customer has an option to book after viewing the quotes. The booking class contains the attribute orderNo, that would allow the system to uniquely identify the customer and bikes rented.

There is method in the system class recordBikeReturn invoked in the provider class to allow the provider to update the bike status upon return.

Our dateRange class has a method overlap which checks if two dates overlap during the booking process.

In the bike class, we decided to add bikeWithPartner and bikeWithDriver Booleans which we will use to check if bike is returned to partner of the original provider and bike is with driver when returning the bike to the provider respectively.

## Assumptions and how their possible solutions are included in our UML class Diagram

In our design, we have provided the following to cater for our assumptions with their possible solutions written in course work one:

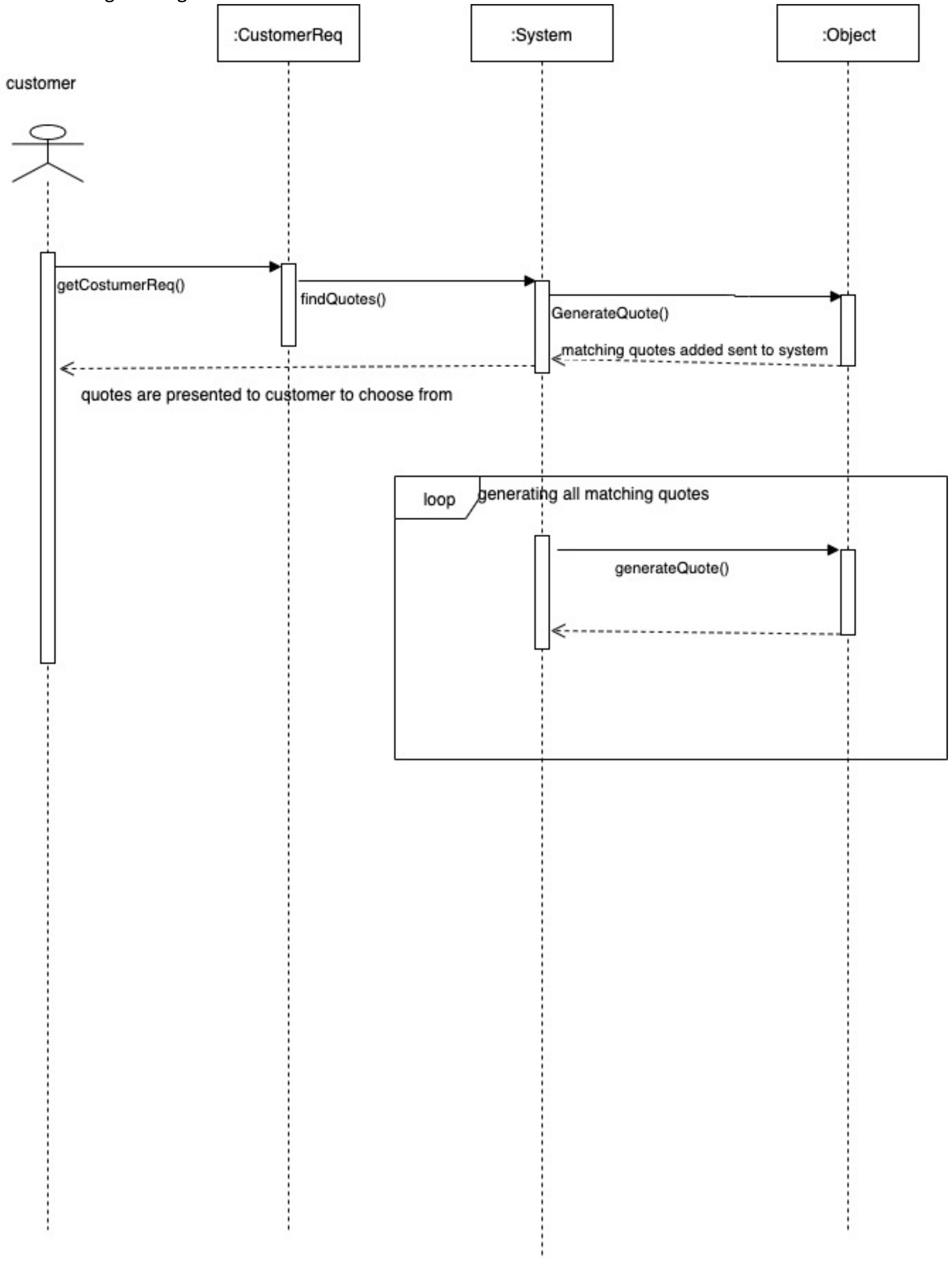
1.The deliveryService class incorporates all the information needed on delivery driver and its services as directed in the coursework two instructions.

2. We provided a method, isNear that takes the locations of the customer and provider and, allows the customer to know if delivery is possible.

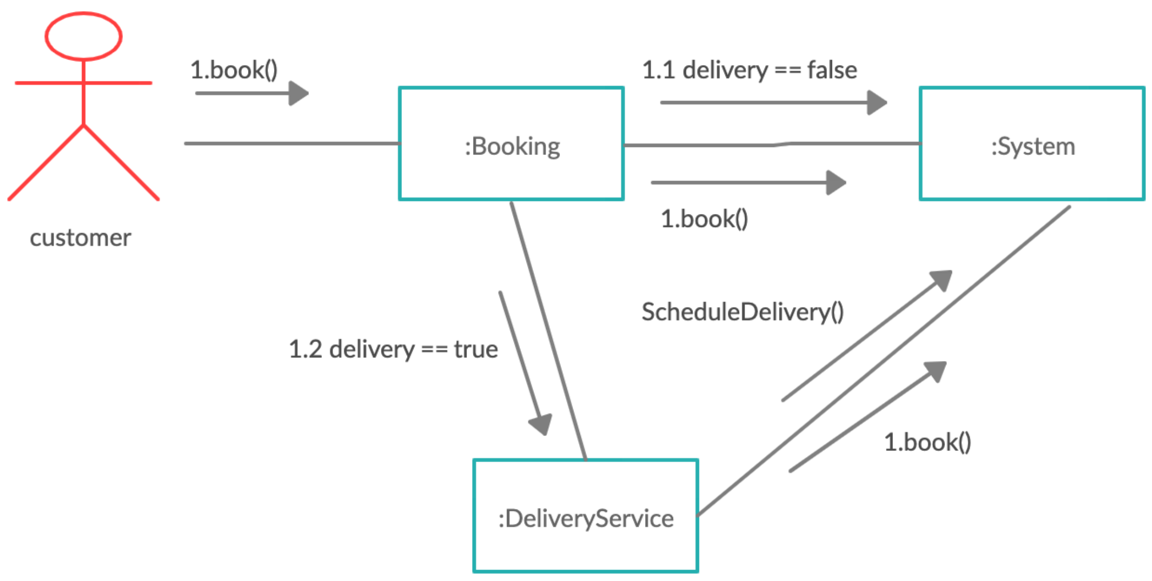
3. We have included a method receivedDeposit, that enables the customer to update their status after they have received their deposit.

4. There is a login class which allows the customer to create an account before getting quote in order to protect their privacy.

## **Sequence Diagram**

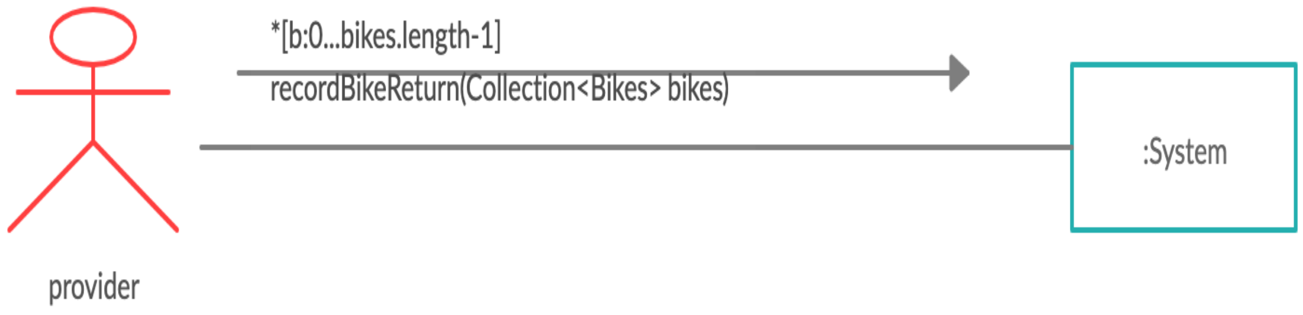

:CustomerReq
:System
:Object
customer
getCostumerReq()
findQuotes()
GenerateQuote()
matching quotes added sent to system
quotes are presented to customer to choose from
loop
generating all matching quotes
generateQuote()

## Communication Diagram for Book Quote



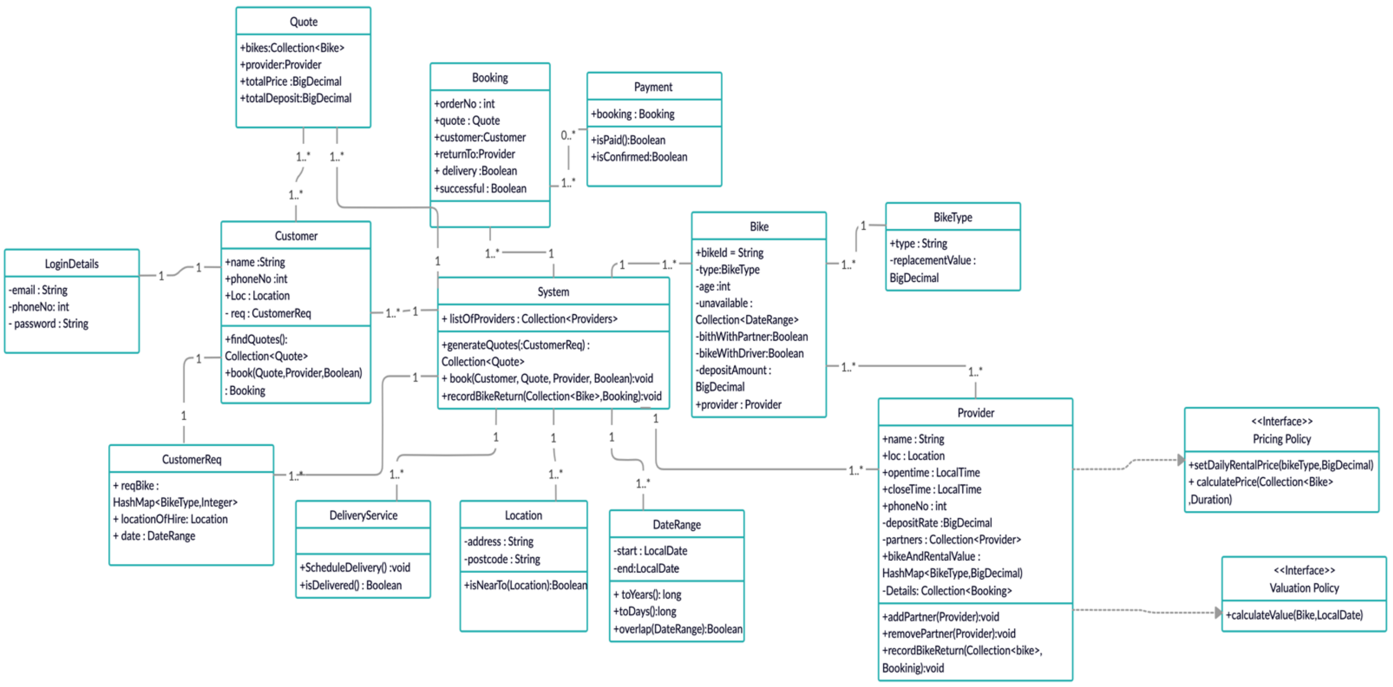
Assumptions made for the communication:
1. the delivery Service is also to make the delivery date.
2. No bike is doubled booked.

## Communication Diagram for return bike to original provider



Software Engineering Coursework 2

# UML class Diagram with Interfaces



## Justification of Good Software Engineering practices

We made sure our design followed the requirements by including all necessary information stated in coursework one.

We made sure our design was consistent by giving all the classes, methods and attributes meaningful name which makes it easier to know what they will do if implemented. We made sure all of our methods and attributes have visibilities and return types.

In our design, we indicated associations between classes and also linked interfaces with appropriate class with correct type of lines.

Overall, we tried our possible best to keep our design simple and clear.

# Self-Assessment

## Q 2.2.1 Static model 22%

• We used the correct UML class diagram notation. 5%

• We split the system design into appropriate classes using the nouns in the system background in coursework one. 3.5%

• We have included most of the necessary attributes and methods for use cases. 3.5%

• We represented associations between the classes. 5%

• We followed good software engineering practices by giving descriptive names to our classes, attributes and methods. 4%

## Q 2.2.2 High-level description 13%

• We focused on clarifying some key components of our design. 8%

• We included the possible solutions suggested for ambiguities in our design. 5%

## Q 2.3.1 UML sequence diagram 11%

• We used the correct UML sequence diagram notation. 3.5%

• We covered class interactions involved in use case. 5%

• We represented an alternative behaviour where appropriate in our design. We are not really sure if that is correct. 2.5%

## Q 2.3.2 UML communication diagram 7%

• We provided a communication diagram for register bike return to original provider use case, but we are not sure if we are missing some components. Although we tried very hard to use the feedback to make corrections. 3%

• We provided a communication diagram for book quote use case, with some assumptions made but we are not sure if we are missing some components.
4%

## Q 2.4 Conformance to requirements 4%

• We made sure our requirements conformed with our design. 4%

## Q 2.5.3 Design extensions 8%

• We specified interfaces for pricing policies and deposit/valuation policies. 2%

• We integrated these interfaces into class diagram. 6%

## Q 2.6 Self-assessment 10%

• We attempted a reflective self-assessment linked to the assessment criteria. 5%

• We justified good software engineering practice. 5%