Software Engineering

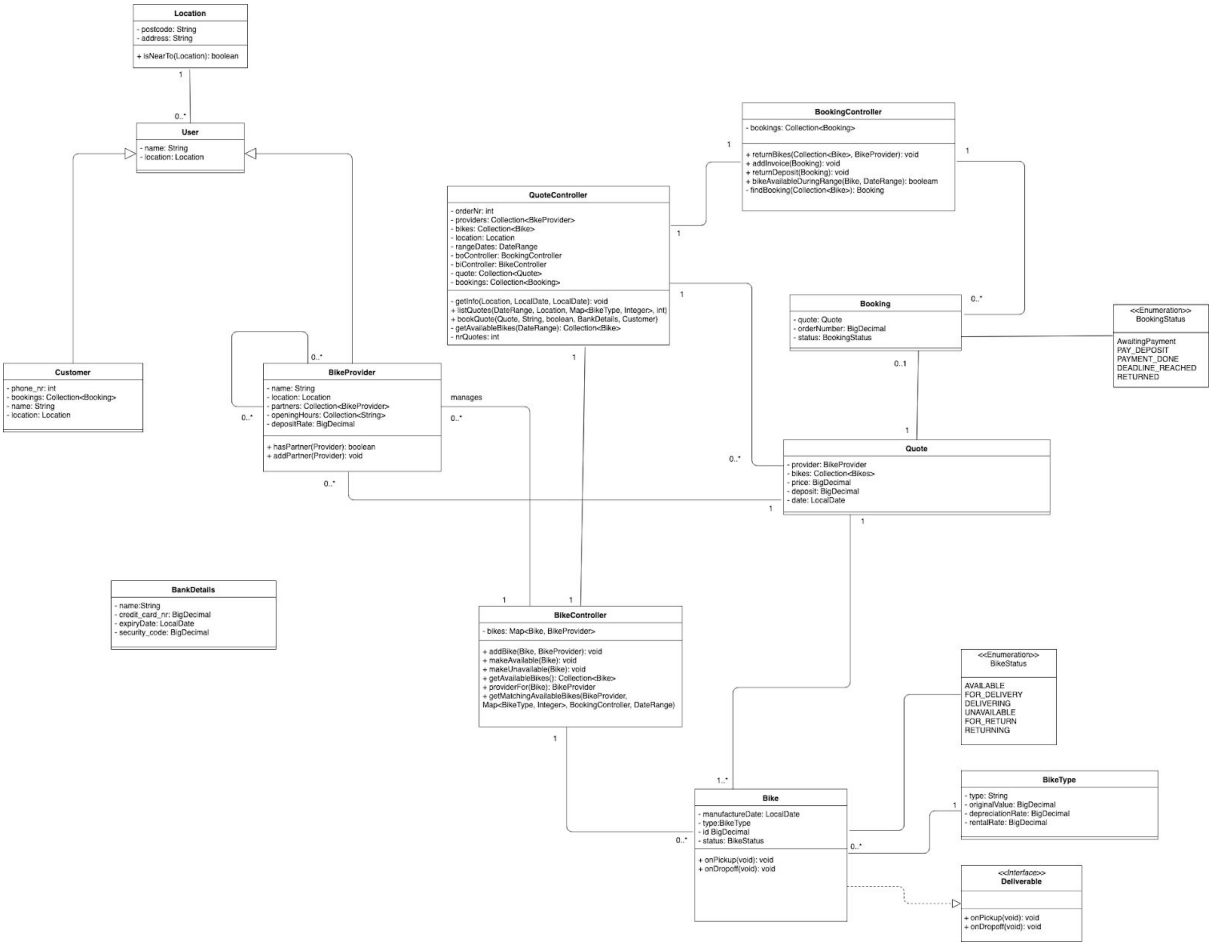
Design

Orges Skura - s1813106

Sami El-Daher - s1830962

Static Models

UML Class Model



High Level Description

The design we have produced has two main controller classes, which coordinate the separate parts of the system. This was done to allow an individual user to query multiple providers, without having to provide a reference to the providers within the Customer class. In contrast to distributing the logic between the classes more having these controllers allows the system to interact with the external resources more easily too.

As the Customer and BikeProvider classes both represent physical entities, and actors in the system, they have lots of properties that are common to both of them. Instead of making these two totally distinct classes we have chosen to add the User class, which is inherited by both the Customer and BikeProvider classes, to store the data expected from a person or corporation that interacts with the system. This reduces repetition within the code, and will encapsulate the shared dynamic nature of these entities.

We have also separated the logic of managing the bike stock into a separate BikeController class, instead of keeping it placing it inside the BikeProvider class. This is useful as it decreases coupling between the Bike and BikeProvider classes, while maintaining the high cohesion between a BikeProvider and the physical providers that exist in the real world.

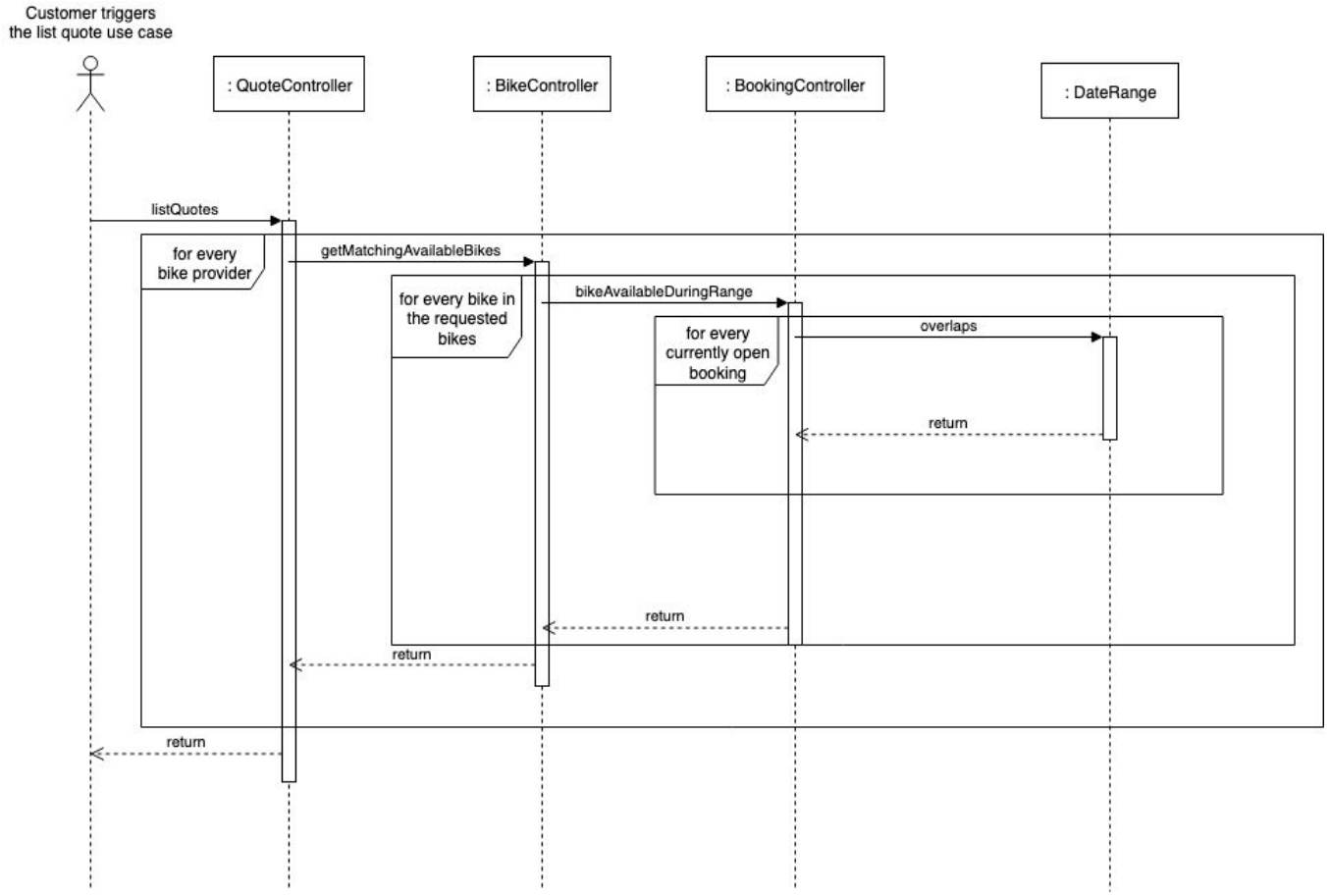
Some of the ambiguities of the requirements, namely those involved with the transfer of money between the parties, has been resolved nicely by the use of the external payment system. The external system has allowed the design to appropriately give these transfers a more transactional quality, for example paying, returning, and transferring a deposit can now be broken down into those distinct parts and can have little influence over each other.

There are also some decisions we have had to make in specific use cases, which include:

- Bike shops that are distinct in the real world will have a distinct Bike Provider object even if they are, for example, two different shops owned by the same company.
- The process providing quotes to users will have a strict search range.
- Partnerships can be triggered by any one bike provider to another

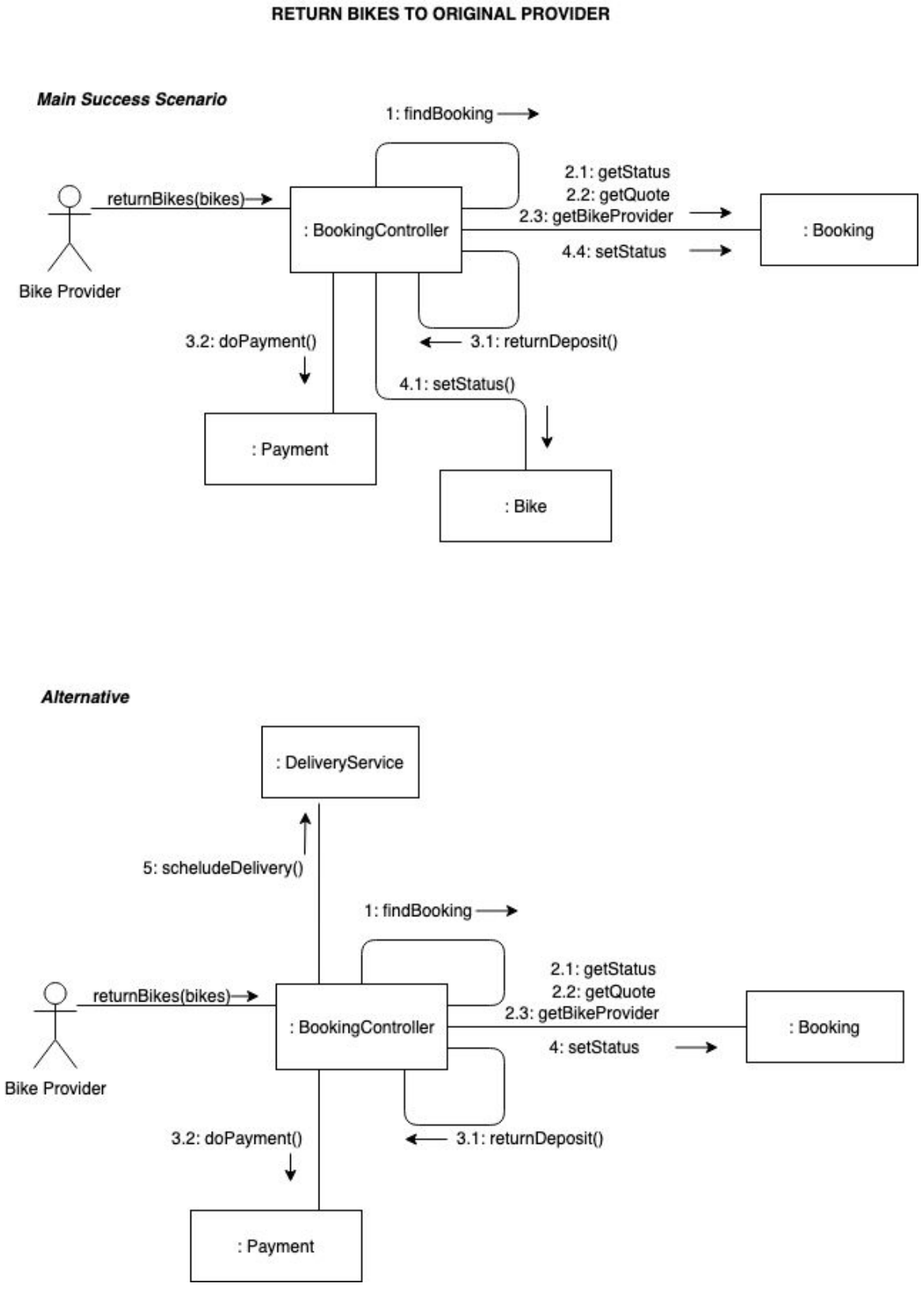
Dynamic Models

UML Sequence Diagram

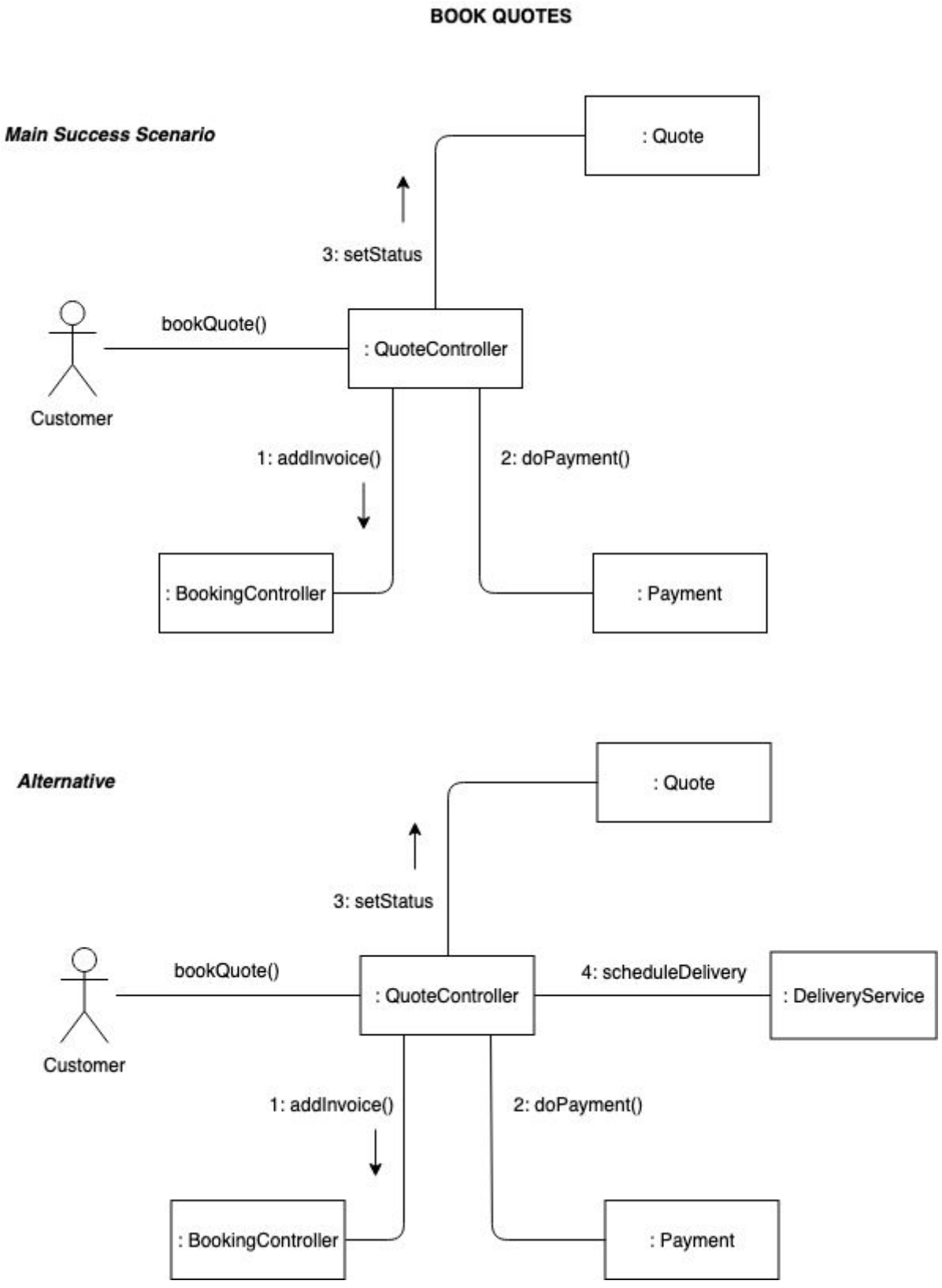


UML Communication Diagram

Return Bikes to Original Provider



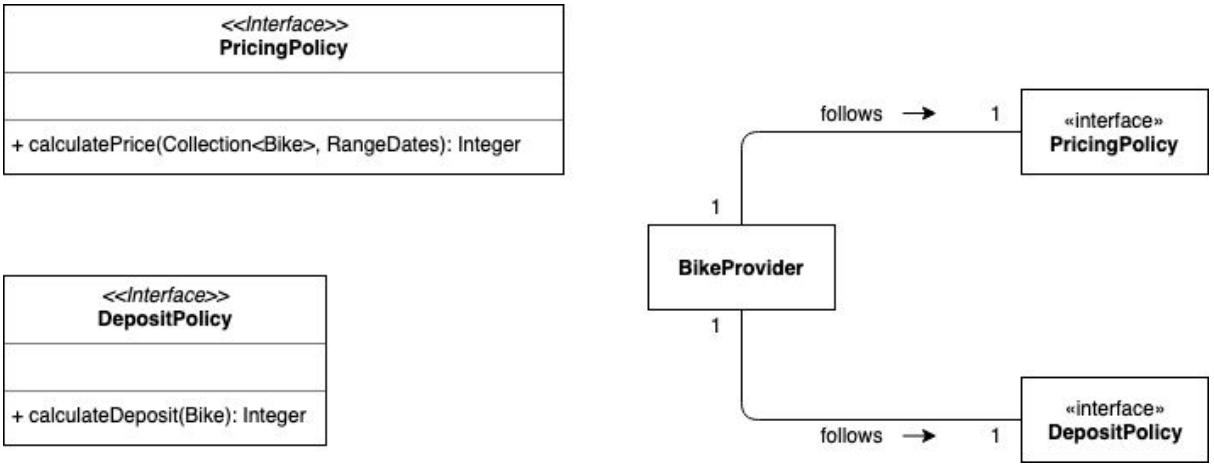
Book Quotes


BOOK QUOTES
Main Success Scenario
: Quote
3: setStatus
bookQuote()
: QuoteController
Customer
1: addInvoice()
2: doPayment()
: BookingController
: Payment
Alternative
: Quote
3: setStatus
bookQuote()
: QuoteController
4: scheduleDelivery
: DeliveryService
Customer
1: addInvoice()
2: doPayment()
: BookingController
: Payment

Conformance to Requirements

This design encompasses all of the desired system state discovered in the requirements, as well as providing means to achieve all of the use cases. This is shown in the attributes of the classes created, and either in the methods or, for more complex use cases, the diagrams that have been provided. Some of the non-functional requirements have also been met, but for others it will be unclear until the system has been implemented, for example performance constraints.

Modified UML Class Diagrams



Self Assessment

Static Model - 22/25%

- **Make correct use of UML class diagram notation** - 5/5% - The correct UML notation was used including classes,enumerations,attributes and methods.
- **Split the system design into appropriate classes** - 4/5% -I have split the systems into almost the right amount of classes but maybe one of the classes has too many methods associated with it.
- **Include necessary attributes and methods for use cases** - 5/5% -The attributes are on point as i have extracted everything from the system description and also from coursework 1.
- **Represent associations between classes** - 5/5% - The associations between classes are on point and represent accurately how each object interacts within the association
- **Follow good software engineering practices** - 3/5% -Most of the UML class diagram is accurate regarding software engineering practices , but one of the problems may be the presence of a class containing 5 methods.

High-Level Description - 14/15%

- **Describe/clarify key components of design** - 10/10% - the high level design details many of the components of the design, giving justifications for them rooted in software engineering practices
- **Discuss design choices/resolution of ambiguities** - 4/5% - many of the decisions we have made in the design have been described, but as most of them were made due to an ambiguity in the original description of the system requirements our chosen solutions may be a little arbitrary

UML Sequence Diagram - 20/20%

- **Correctly use UML sequence diagram notation** - 5/5% - we have used the correct notation for the sequence diagram
- **Cover class interactions involved in use case** - 10/10% - all invocations between have been shown in the diagram
- **Represent optional, alternative, and iterative behaviour where appropriate 5/5%** - we have shown where iterative and alternative behaviour were necessary.

UML Communication Diagram - 13/15%

- **Communication diagram for *return bikes to original provider*** - 7/8% - we have provided models for both scenarios for the use case, in a good amount of detail
- **Communication diagram for *book quote* use case** - 6/7% - we have given a communication diagram for the *book quote* use case

Conformance to requirements - 5/5%

- **Ensure conformance to requirements and discuss issues** - 5/5% - we describe how the system as design will meet most of the requirements set, and also discuss those requirements which we are currently unable to say we have met

Design Extensions - 9/10%

- **Specify interfaces for pricing policies and deposit/valuation policies** - 3/3% - two new interfaces have been provided which, when implemented by a contractor, will provide the desired functionality
- **Integrate interfaces into class diagrams** - 6/7% - high level detail showing how the interfaces will be used in the system is shown, but there may be ambiguities in the way that they will affect the function calls

Self-Assessment - 9/10%

- **Attempt a reflective self-assessment linked to the assessment criteria** - 5/5% - this self-assessment has described the level to which we have achieved the criteria, as well as discussing aspects of our solution which have not quite met them
- **Justification of good software engineering practice** - 4/5% - we have designed a system which follows good software engineering practice, as we have described how our components maintain a high cohesion with their real life analogues.