Bike portal
Analysis and Design Document
Student: Ács Dávid
Group: 30432

Revision History

Date	Version	Description	Author
18/Mar/18	0.1	Housekeeping	Ács Dávid
04/Apr/18	0.2	Architecture and views	Ács Dávid

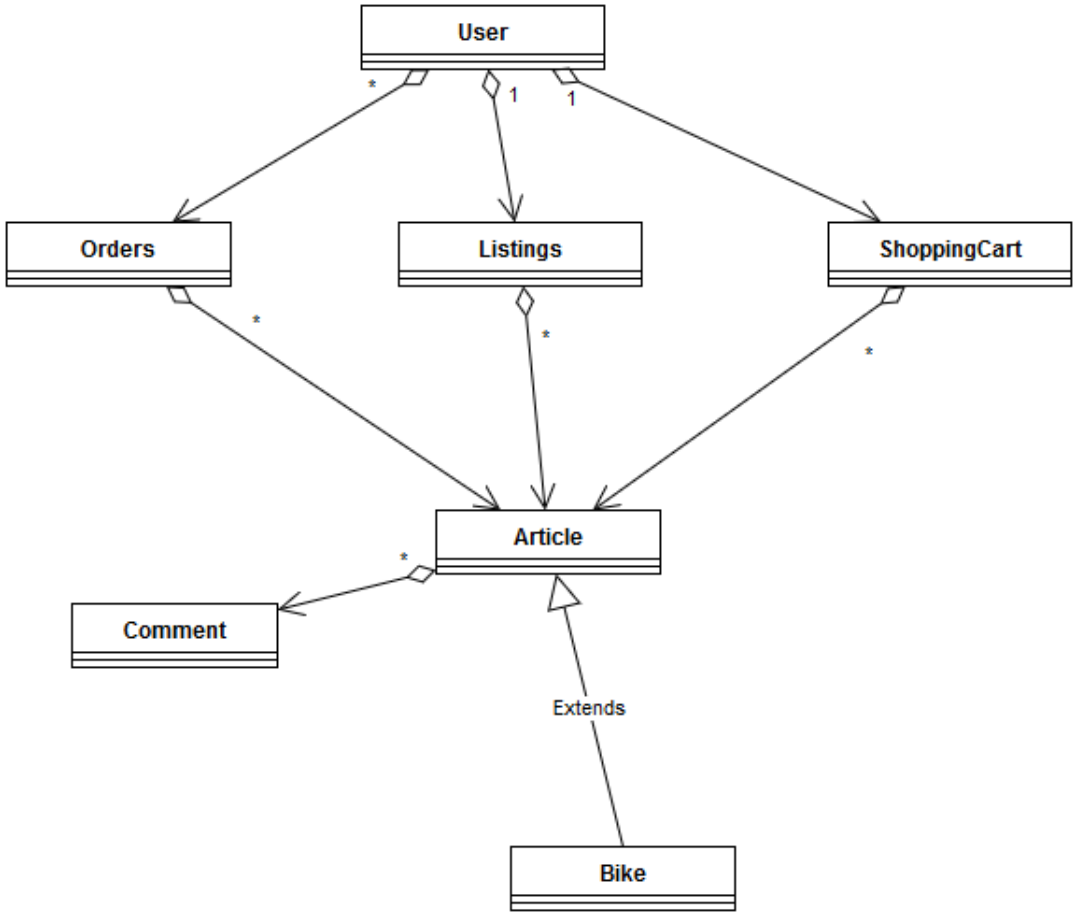
Table of Contents

I. Project Specification

Bike portal is an online Bike and bike parts store. User can buy/sell bikes and related parts online. To buy products, the user must be registered and logged in, and should add the products to the shopping cart. To check out the user can pay by credit card. Users may rent bikes, and the admin may approve these requests. The admin of the portal can delete, modify announcements/listing if they are inappropriate.

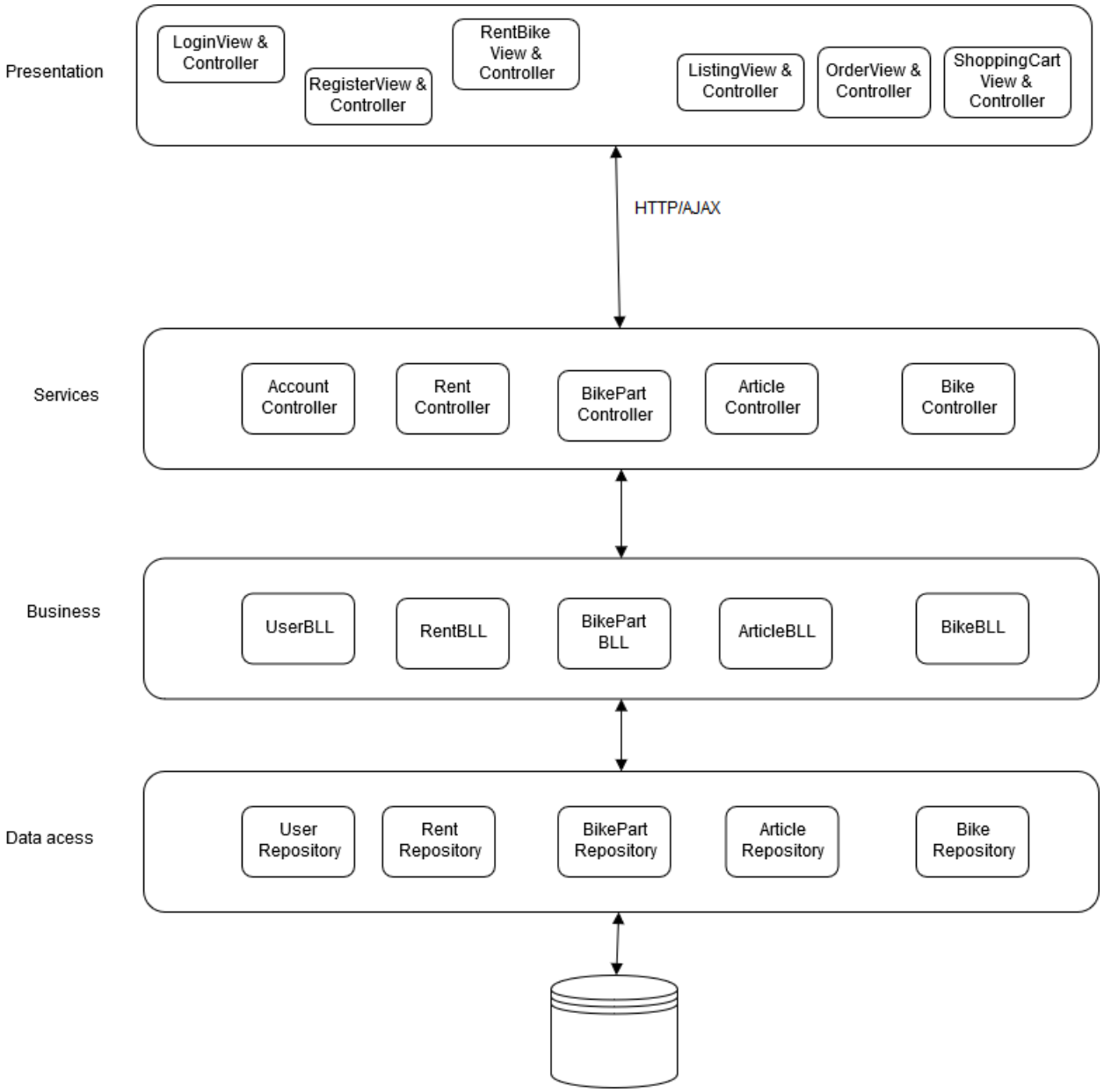
II. Elaboration – Iteration 1.1

1. Domain Model



2. Architectural Design

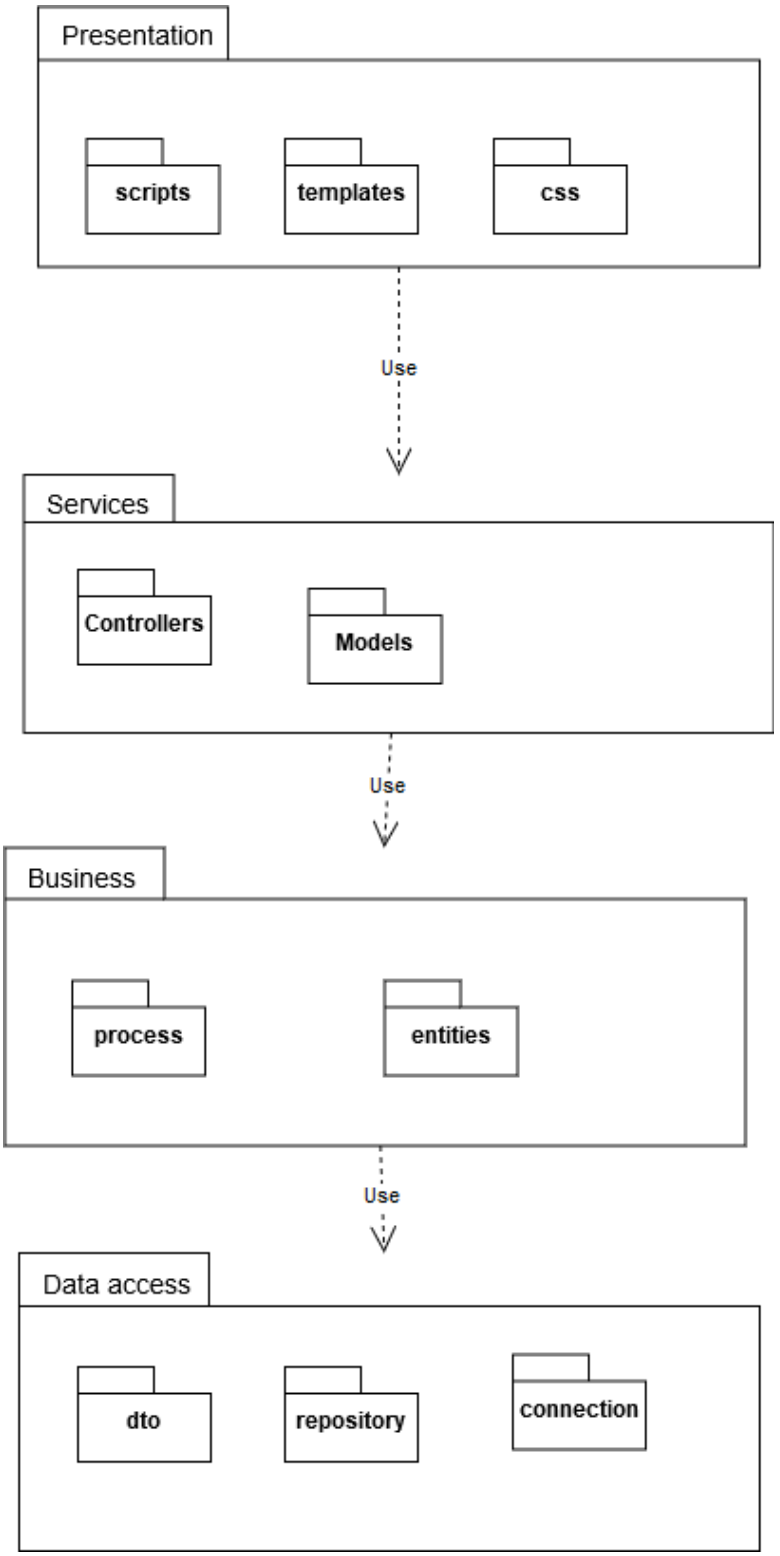
2.1 Conceptual Architecture



Client server architecture is used in this architecture, to separate the server-side application (data and logic) from the client-side (UI). This way the client-side and the server side may evolve independently of each other.

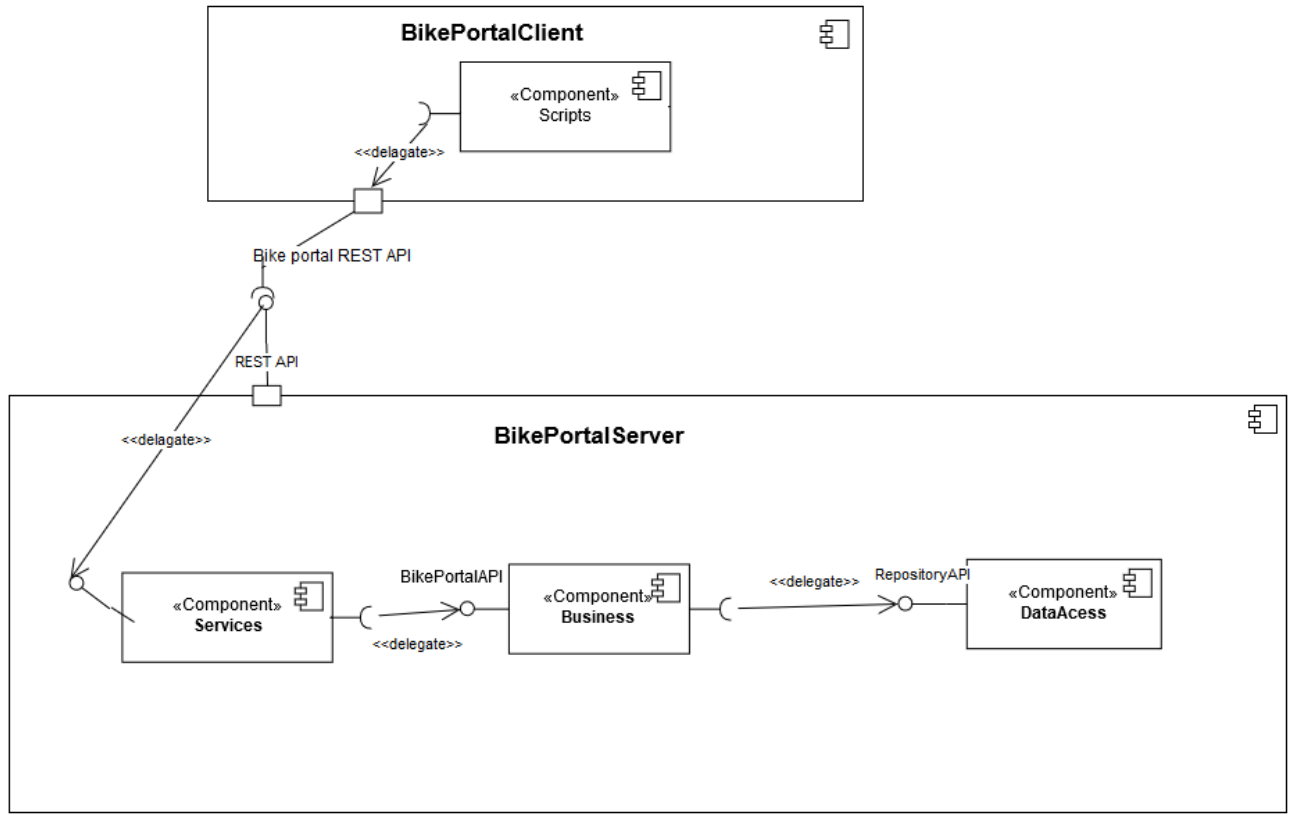
The separation can be observed, where the HTTP/AJAX calls are made over the network.

2.2 Package Design

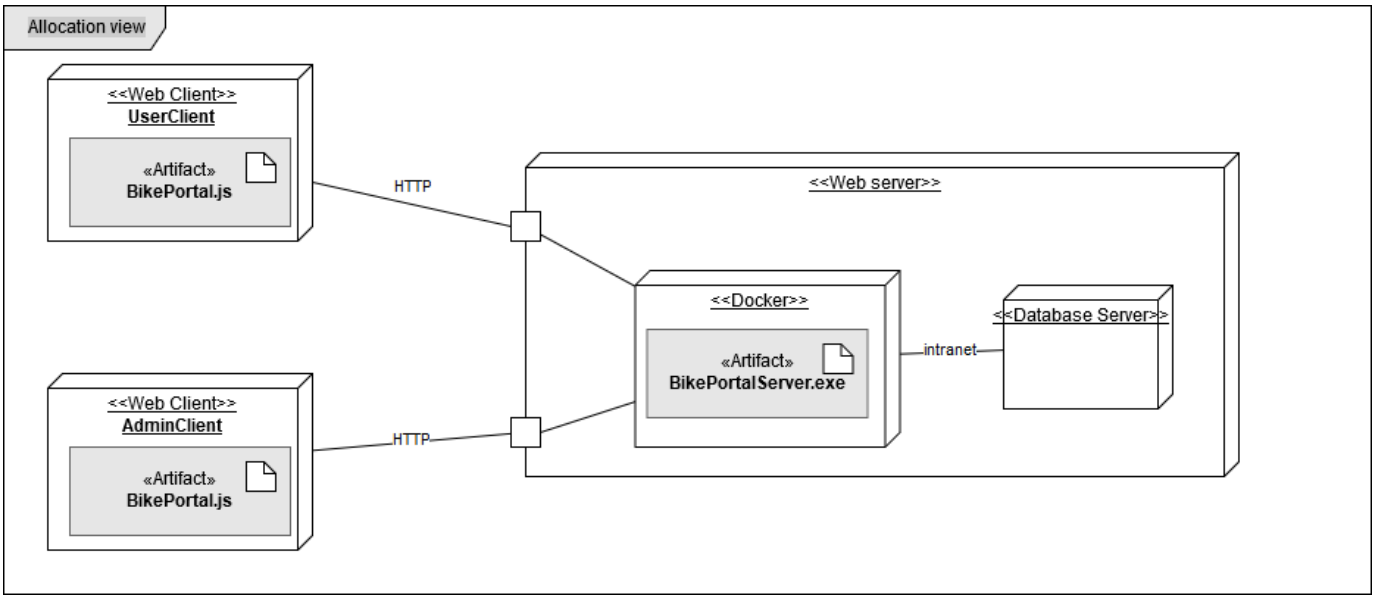


The package diagram of the system represents the layered structure of the application.

2.3 Component and Deployment Diagrams



The component diagram shows the clear delimitation between the server and the client. They can only communicate through HTTP, or as it is shown in the diagram, through the REST API.



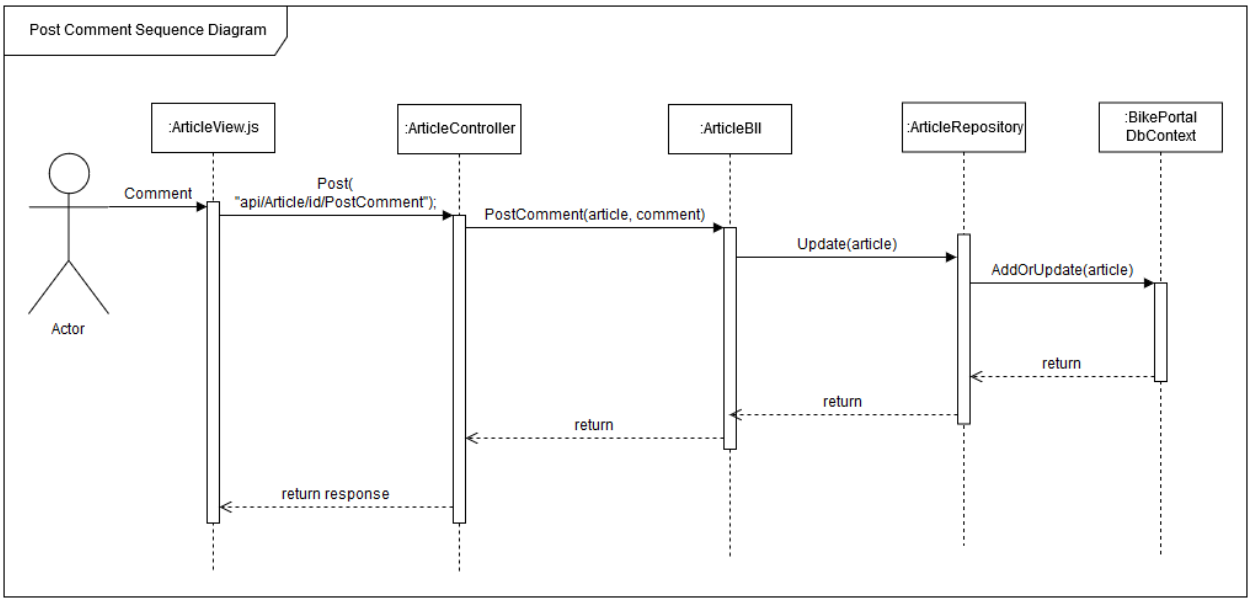
In the allocation view we may notice that there may be more than one client, and they role may differ depending on who is logged in.

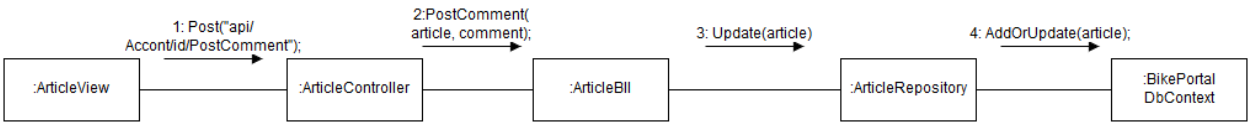
III. Elaboration – Iteration 1.2

1. Design Model

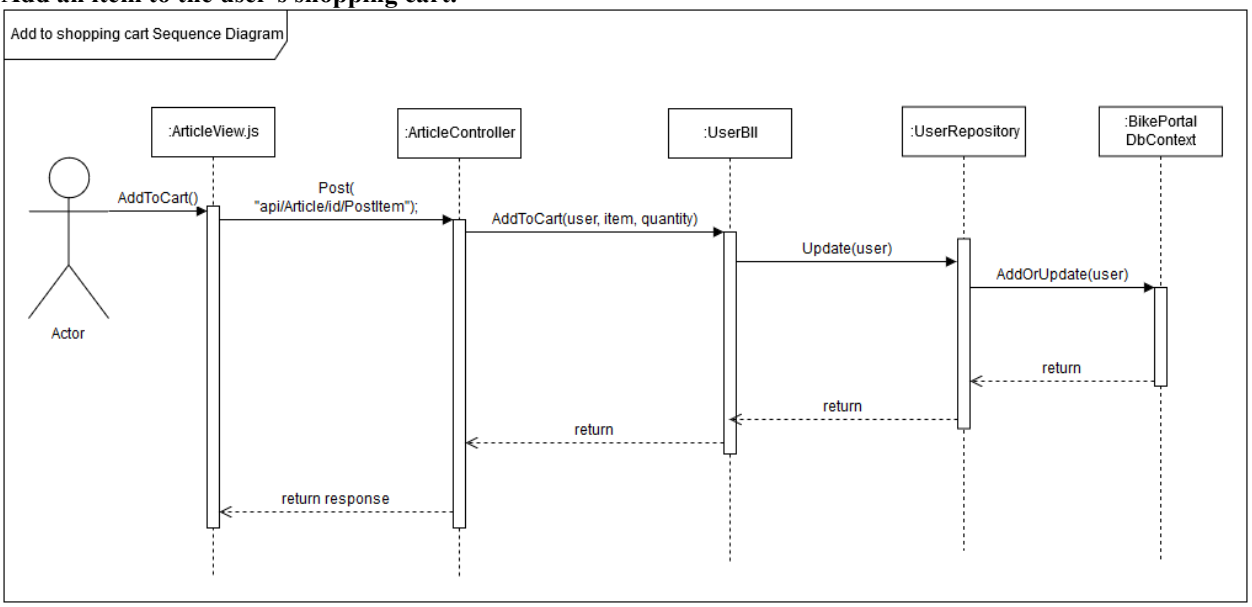
1.1 Dynamic Behavior

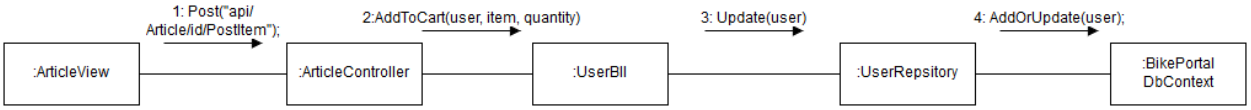
Post a comment to an article.





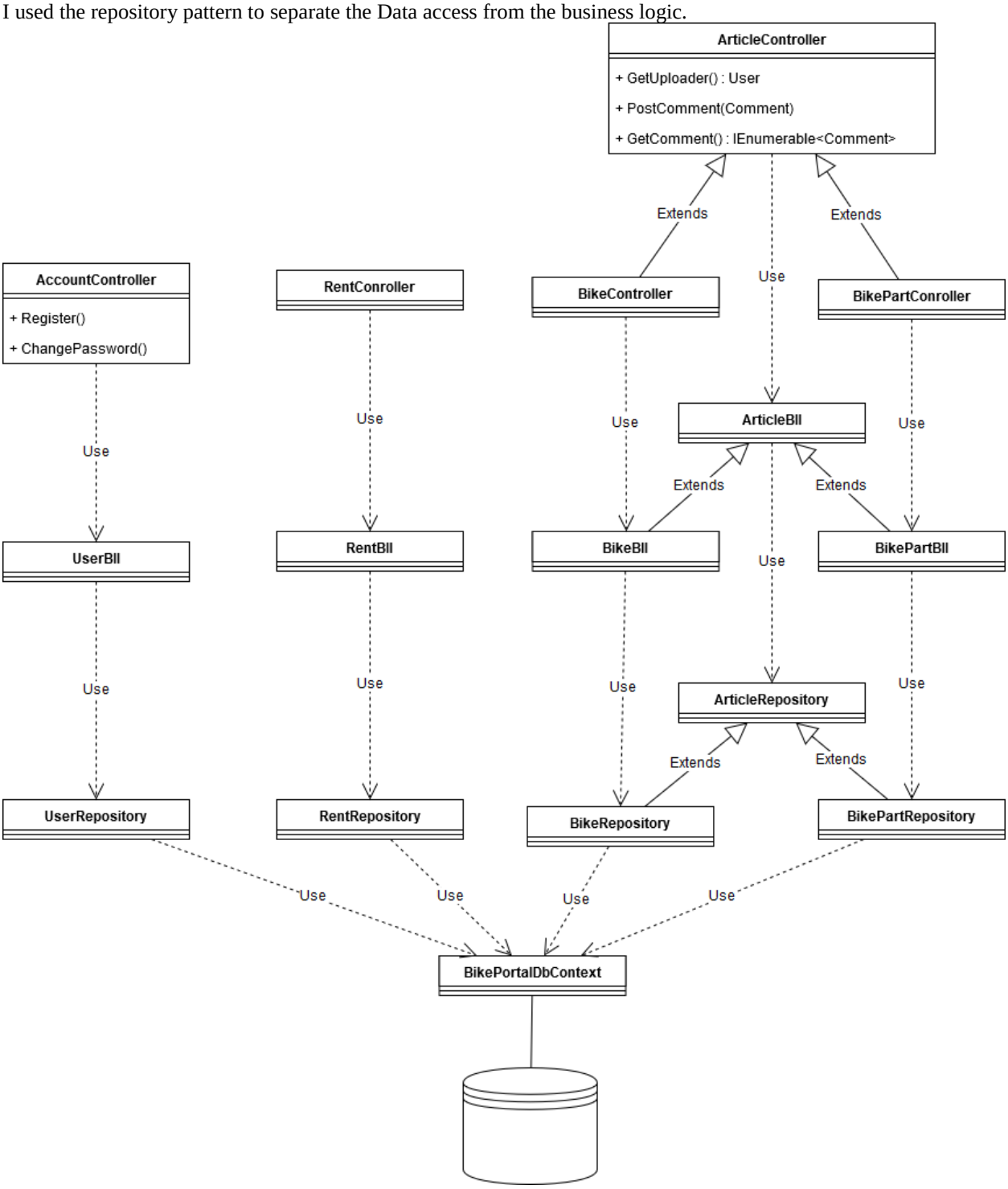
Add an item to the user's shopping cart.





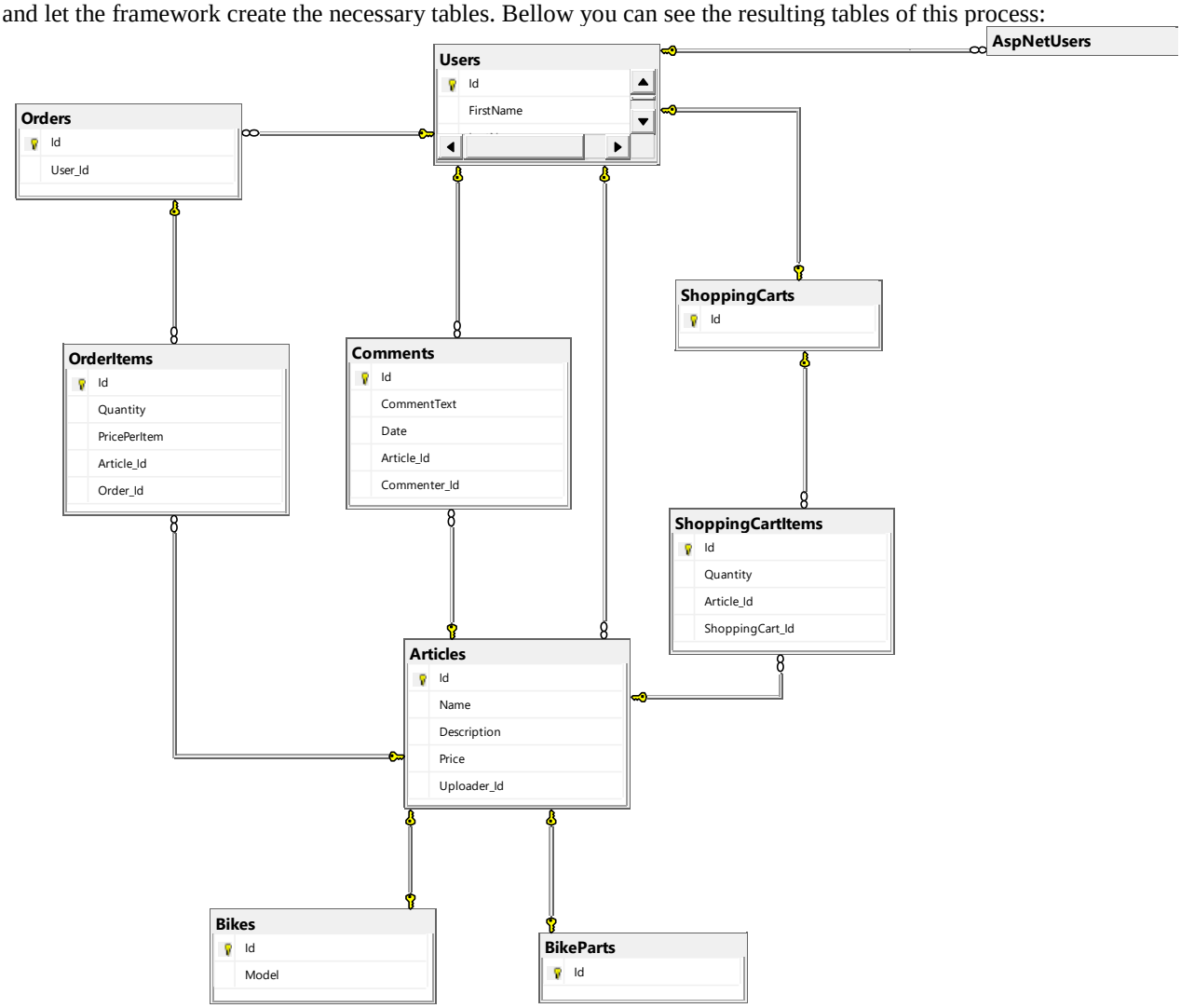
1.2 Class Design

I used the repository pattern to separate the Data access from the business logic.



2. Data Model

The data model closely resembles the domain model. In order to create the data model, first I designed domain classes and let the framework create the necessary tables. Bellow you can see the resulting tables of this process:



3. Unit Testing

For unit testing mocking will be used to separate dependencies from the tested component.

Unit Test scenarios

1. Check that the user can login.
2. Check that users can view articles regardless if they are logged in or not.
3. Check that users can compare two items.
4. Check that users can post comments on articles if they are logged in.
5. Check that users can add articles to their shopping carts.
6. Check that users can buy the items already in their shopping carts.
7. Check that users can put a bike as a listing (sell their bike).
8. Check that users can create a request to rest a bike.
9. Check that administrators can delete inappropriate listings from the website.

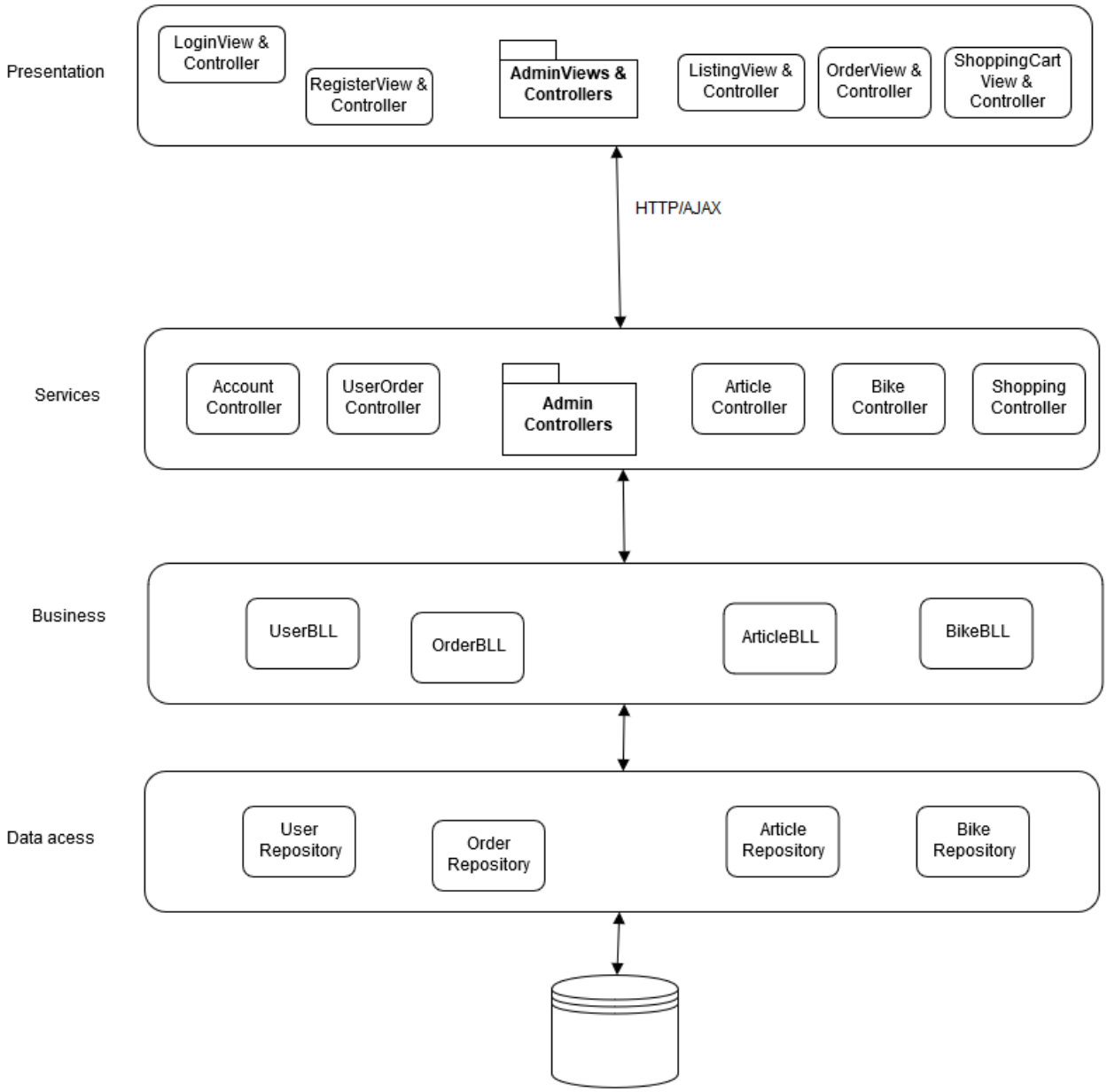
10. Check that administrators can approve requests to rent a bike.

Integration testing will be performed as well to ensure that the individual components fit together in an orderly manner.
System testing will be performed last, to ensure that the system will satisfy all functions and non-functional requirements of the project.

IV. Elaboration – Iteration 2

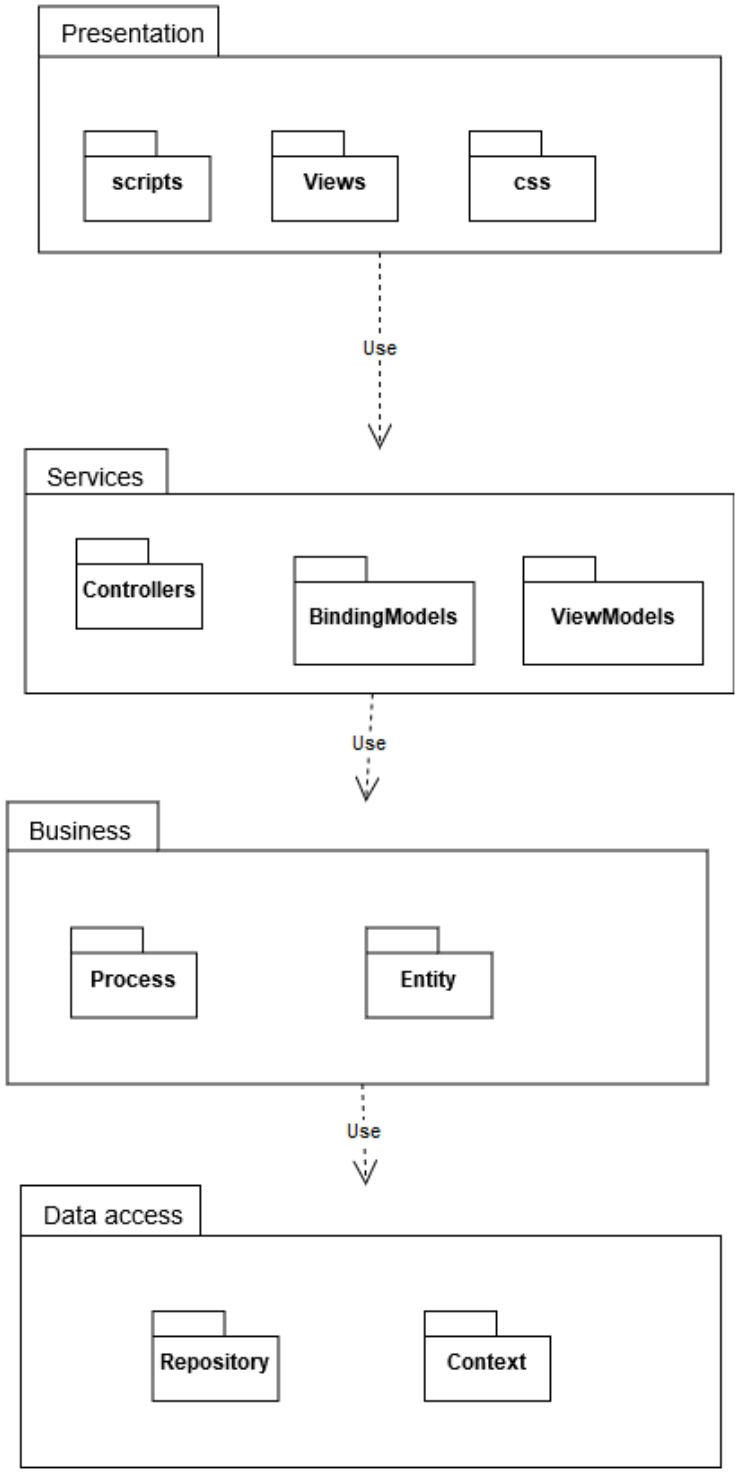
1. Architectural Design Refinement

2.4 Conceptual architecture



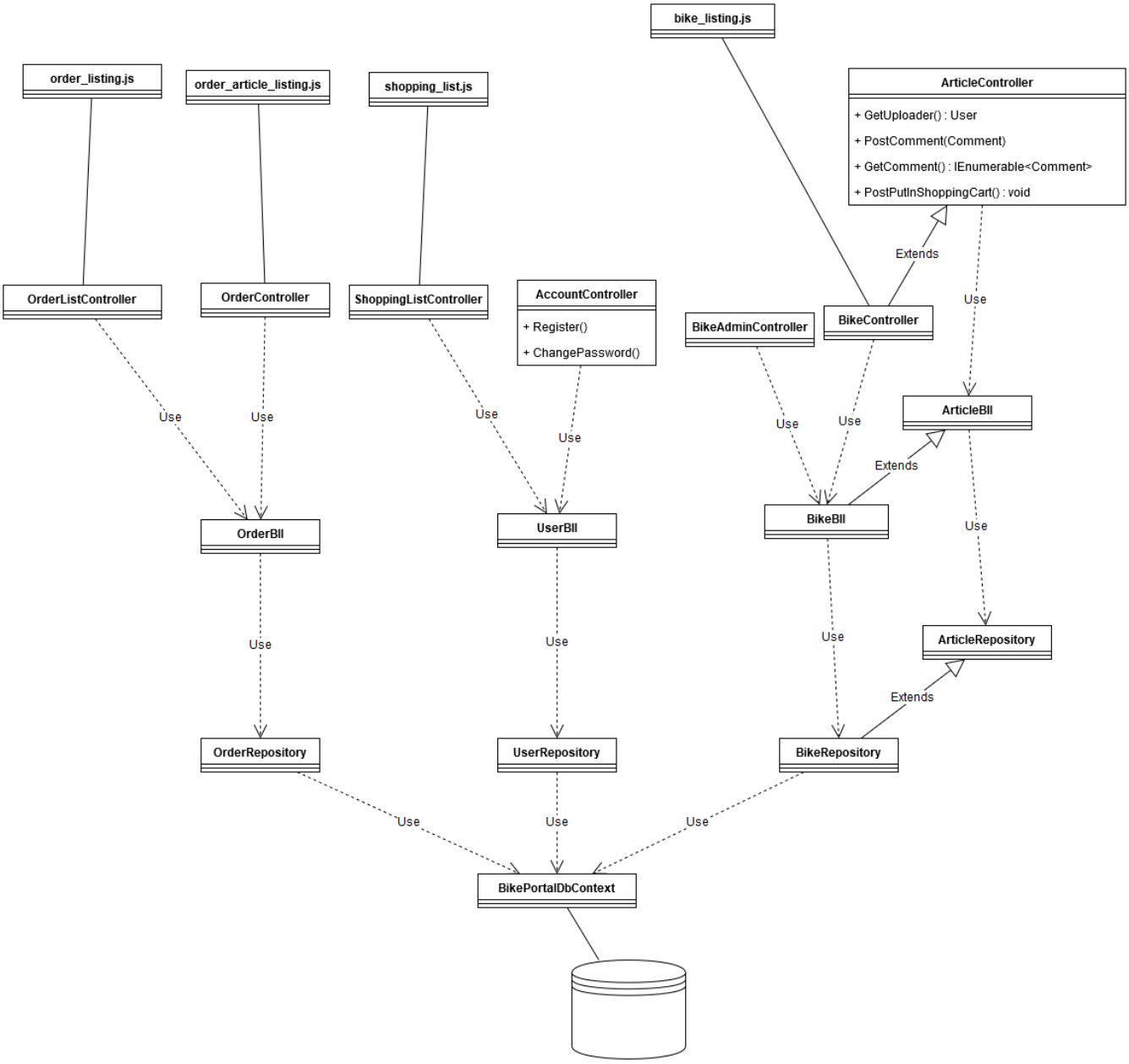
More controllers should use the same business login in the application, in order to create a clear separation of concerns. For example most of the Admin controllers use the same business logic as the User controllers.

2.5 Package design



Only some minor name changes have been made to the package diagram.

2. Design Model Refinement



Some classes are added and removed compared to the other class diagram. For example, and admin controller has been added to represent how Admin controllers will integrate in the system. Most of the javascript files have been added to more clearly identify the relationship between the corresponding controller.

V. Construction and Transition

The system was implemented using C#. Multiple libraries were used, making development easier and safer at the same time. The ASP.NET framework was used in order to implement the webapp.

1. System Testing

Unit tests were written in order to test the application and to increase the confidence in the correct functioning of the system.

2. Future improvements

Multiple features could be added to the product, such as a new type of article, allowing users to rent a bike.
Each of these possible improvements have been accounted for in the design phase and the application should be easily extendable.
More tests could be written to make sure that the application is properly functioning.

VI. Bibliography

ASP.NET:
https://docs.microsoft.com/en-us/aspnet/overview
Inversion of Control with Microsoft Unity:
https://msdn.microsoft.com/en-us/library/dn178470(v=pandp.30).aspx
Conceptual architecture:
http://www.bredemeyer.com/ArchitectingProcess/ConceptualArchitecture.htm
Component diagram:
https://www.ibm.com/developerworks/rational/library/dec04/bell/
Package diagram:
https://www.uml-diagrams.org/package-diagrams/model.html
Layered architecture:
https://msdn.microsoft.com/en-us/library/ee658109.aspx
Deployment diagram:
https://www.tutorialspoint.com/uml/uml_deployment_diagram.htm
http://www.agilemodeling.com/artifacts/deploymentDiagram.htm