The University of Melbourne
School of Computing and Information Systems SWEN90007
Software Design and Architecture. Semester 2, 2019

Project Part 3: Feature B

Online Mobile Phone Store Management system

Student ID: 671499,
686141

Student Name: Yixiong Ding,
Ruifeng Luo

Heroku Web Application: https://onlinemobilephonestore.herokuapp.com

GitHub Repo: https://github.com/YixiongDing/Online-Mobile-Phone-Store

Table of Contents

1. Introduction

In Feature B, Customer is able to:

- Register a new account and create new orders of any mobile phones;
  (Create Operation)

- View all mobile phones and their corresponding information in the store;
  (Read Operation)

- Update account profile;
  (Update Operation)

- Cancel orders
  (Delete Operation)

After logged in, Customer can log out at any time.

2. Test Cases

2.1 Register new account

1. Click on the ‘Hello! Please Login’ button on the navigation bar.
2. Click on the ‘Register’ button.
3. Fill the required information.
4. Click on the ‘Submit’ button and direct to register success page.

2.2 Login

1. Click on the ‘Hello! Please Login’ button on the navigation bar.
2. Use registered username and password to login into the system as the customer, wrong usernames or passwords will direct to the login fail page.
3. Login successfully will direct to the ‘Dashboard’ page.

2.3 View home page and all mobile phones

1. Click on the ‘Home’ or ‘All Mobile Phones’ button on the navigation bar.

2. The system directs to the corresponding page and display mobile phones.

2.4 View mobile phone details

1. When at the ‘Home’ page, click on the ‘Check More’ button of any mobile phone, or at ‘All Mobile Phones’ page, click on the ‘View Detail’ button of any mobile phone.
2. The system directs to the corresponding mobile phone detail page:
   a. If logged in as customer, showing the ‘Buy Now’ button;
   b. If not logged in or logged in as administrator, showing ‘Login to purchase this mobile phone’.

2.5 Place order of mobile phone

1. When at the ‘Mobile Phone Detail’ page, click on the ‘Buy Now’ button.
2. The system will direct to the ‘Order Confirmation’ page:
   a. Click on the ‘Confirm’ button, the system will place the order and direct to the ‘Manage My Order’ page;
   b. Click on the ‘Cancel’ button, the system will direct to the ‘All Mobile Phones’ page.

2.6 Update account profile

1. Click on the ‘Dashboard’ button on the navigation bar.
2. Click on the ‘Update’ button under the ‘Update My Profile’ section.
3. The system will direct to the ‘Update My Profile’ page.
4. Update the information and click on the ‘Submit’ button.
5. The profile will be updated and system will direct to the ‘Dashboard’ page.

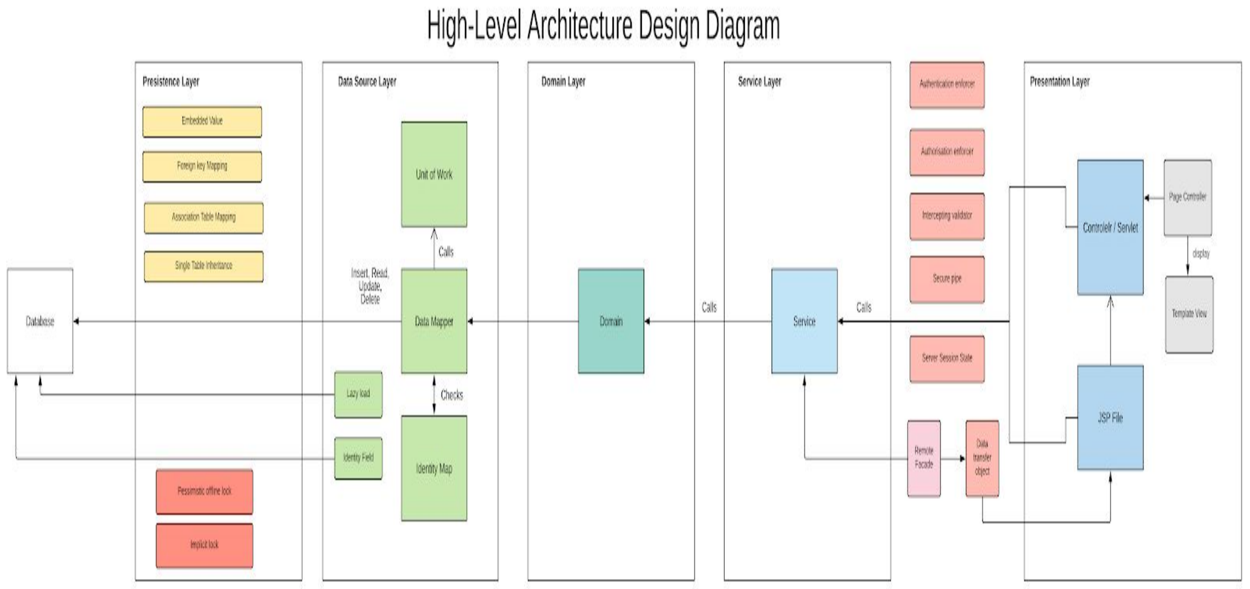
2.7 Cancel orders

1. Click on the ‘Dashboard’ button on the navigation bar.
2. Click on the ‘Manage’ button under the ‘Manage My Orders’ section.
3. The system will direct to the ‘Manage My Order’ page.
4. Click on the ‘Cancel’ button of any order, the system will delete the order.

2.8 Logout

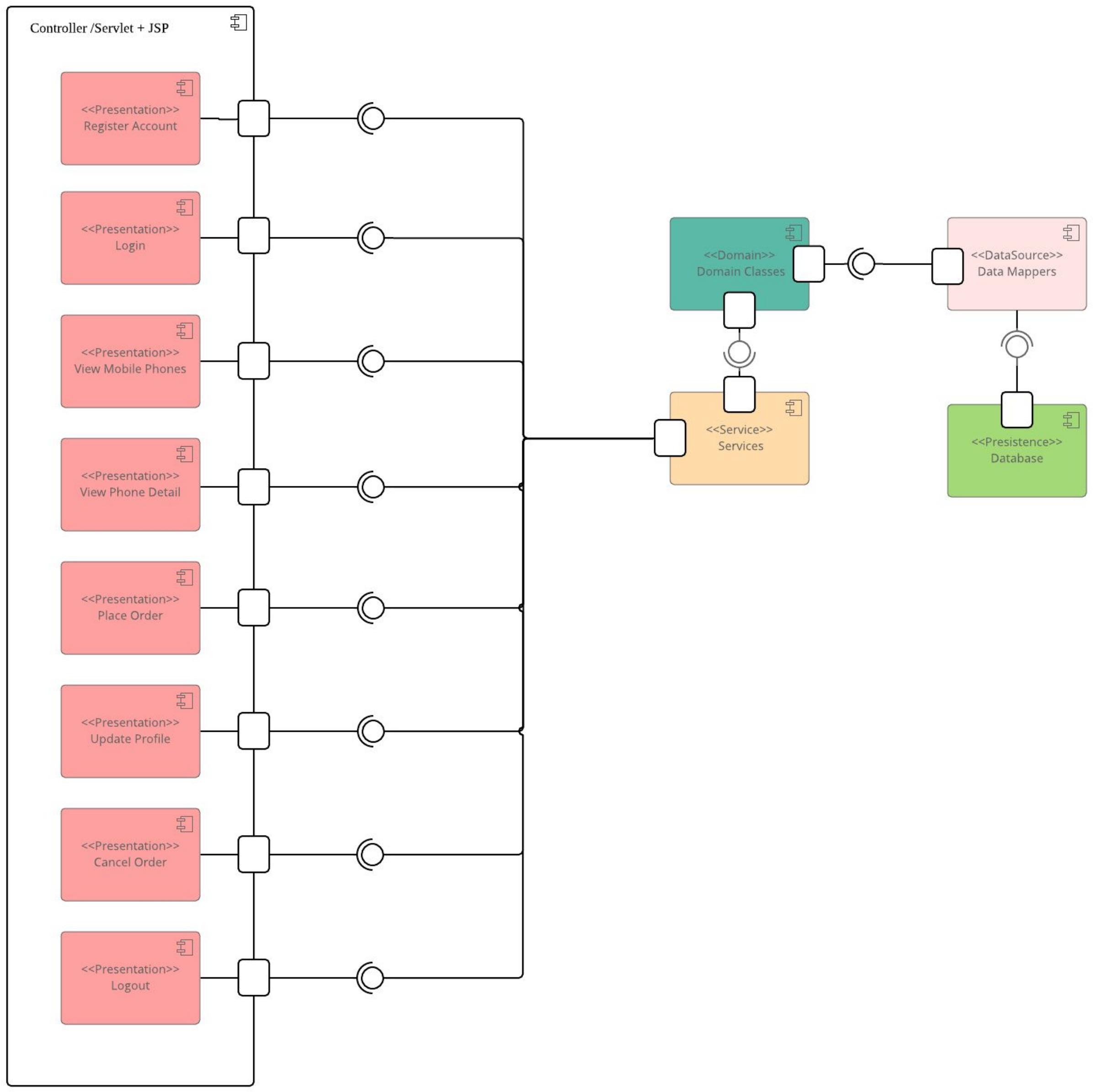
1. Click on the ‘Logout’ button on the navigation bar.
2. The system will log out and direct to the ‘Home’ page.

3. High level architecture



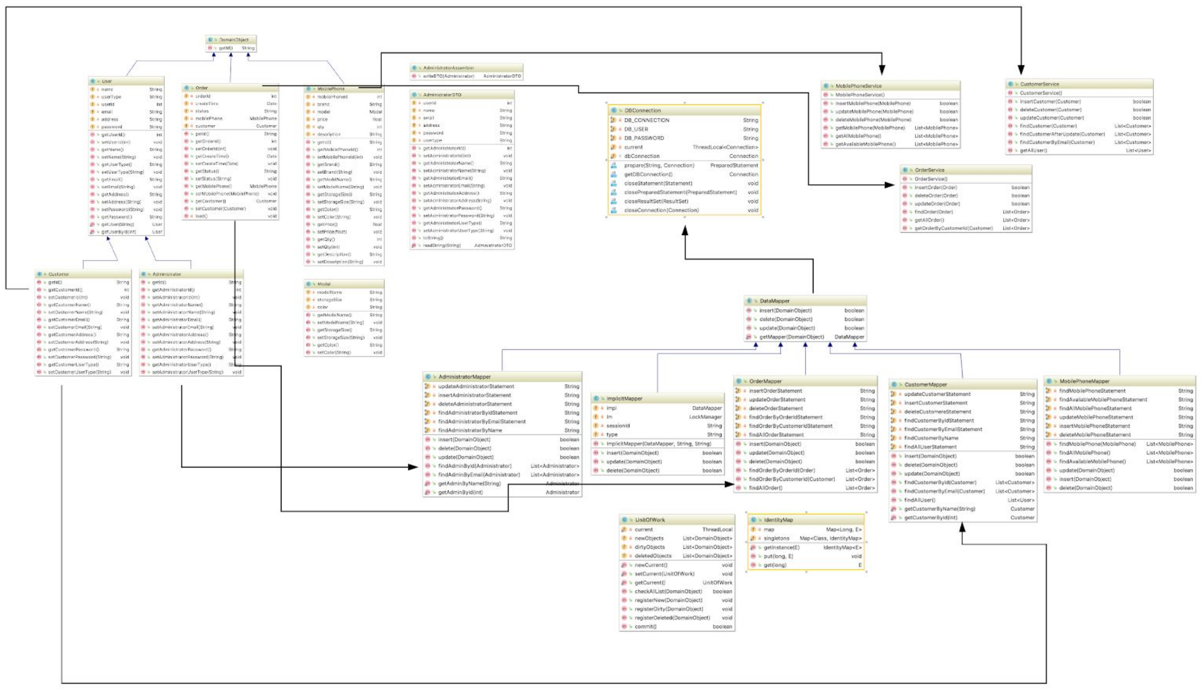
Layer	Responsibilities
Presentation layer	Handling interactions between the user and the system, and interpreting and displaying information sent back from the system.
Service layer	Implementing the application-specific logic, and delegates the domain-specific logic to the domain layer.
Domain logic layer	Implementing the logic of the system by interpreting commands received from the presentation layer.
Data source layer	Handling of communication with the systems that contain the data, such as databases.
Persistence layer	The layer is to manage the actual reading and writing of data to persistent storage.

4. Component Diagram


Controller /Servlet + JSP
<<Presentation>> Register Account
<<Presentation>> Login
<<Presentation>> View Mobile Phones
<<Presentation>> View Phone Detail
<<Presentation>> Place Order
<<Presentation>> Update Profile
<<Presentation>> Cancel Order
<<Presentation>> Logout
<<Domain>> Domain Classes
<<DataSource>> Data Mappers
<<Service>> Services
<<Presistence>> Database

As in the component diagram above, the services layer handle requests from presentation layer and implement those logics by calling domain classes. And the data mappers are handling retrieve data from database based on the requests from domain logics.

5. Domain class diagram

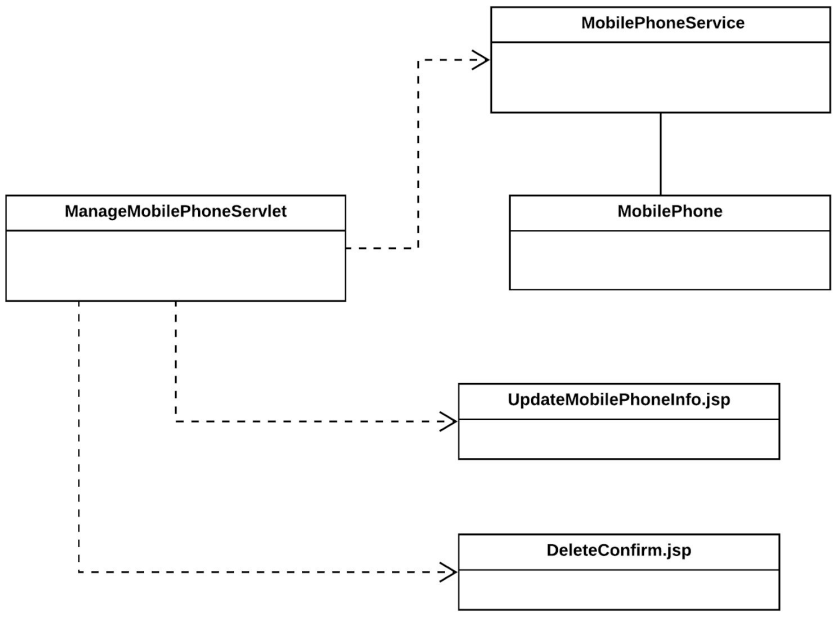


6. Design pattern

6.1 Presentation layer

6.1.1 Page Controller Pattern

In the page controller design pattern, the controller is responsible for handling logic corresponding to user's action and displaying the view. More precisely, the controller is responsible for invoking the appropriate methods in the domain logic layer based on user's actions, and displaying view with those information. In our design diagram below,
The servlet, 'ManageMobilePhoneControllerServlet', is controller. It is responsible for taking requests and invoking appropriate methods in 'MobilePhoneService' based on request. The class 'MobilePhoneService' is associated with class 'MobilePhone' in this diagram. Then 'ManageMobilePhoneServlet' display view, 'UpdateMobilePhoneInfo.jsp' and 'DeleteConfirm.jsp', using information from 'MobilePhoneService'.



- Comparing with page controller pattern, **Front controller** handles all requests that come for a resource in an application using a single handler, and then dispatched to the appropriate commands for the type of request. But it has more complex logic than page controller.

6.1.2 Template View Pattern

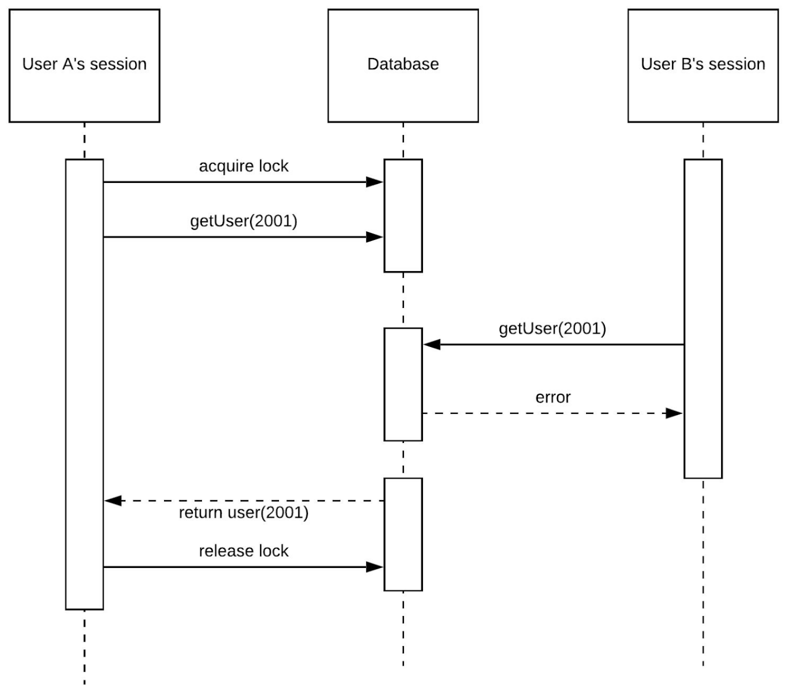
In the template view design pattern, the static HTML part acts as template in the view, and it embedded programming logic inside HTML. In our system, ‘ViewMobilePhone.jsp’ implement the pattern by embedding programming logic of class ‘MobilePhone’ in the static HTML.

- Comparing to Template View pattern, **Transform View** plays a role of transforming domain data into HTML. However, the implementation of Transform View is more difficult than template view.
- Comparing to Template View pattern, **Two Step View** transform domain data into HTML using two steps, forming logic page based on domain data, then rendering the logic page into HTML. However, the implementation of pattern is more difficult.

6.2 Concurrency

6.2.1 Pessimistic offline lock

The Pessimistic offline lock pattern prevents conflicts between concurrent business transactions and it allows only one transaction to access data at a time. It forces a business transaction to acquire a lock on a piece of data before it starts to use it. At the end, it releases the lock when it complete the business transaction. In our application, we implement Exclusive write lock, which require process to acquire the lock if the process are writing data to a table.



As shown in the graph above, user must acquire lock before the user is able to modify the data in database. When the user acquire lock on the data in the database, the other user is not able to make any changes to the data. Other users can only read the data. However, Exclusive write lock does not guarantee consistent reads.

- Comparing to the Pessimistic offline lock, **Optimistic offline lock** are going to validate that the changes about to be committed by one session don't conflict with the changes of another session. However, Optimistic offline lock is assuming conflicts is very low, which indicates many records but relatively few users, or very few updates. In this case, our enterprise system are going to handle hundreds of thousands of users' operation, and it cannot guarantee consistency.

6.2.2 Implicit lock

The Implicit lock pattern is allow application to handle acquiring locks and releasing locks implicitly. The implicit lock are placing the code to acquire and release locks in a mapper layer, which sits between the domain logic code and the data-source code for accessing the database.

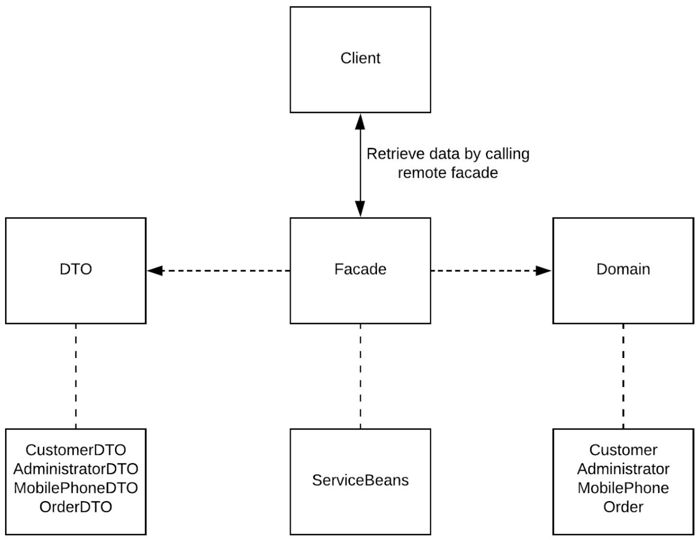
6.3 Distribution

6.3.1 Data transfer object

The Data transfer object pattern is to encapsulate data and send it from one subsystem of an application to another. Data Transfer Objects are used to transfer data between the Service Layer and the Presentation Layer. The main benefit of using the pattern is to reduce the amount of data that needs to be sent over the network in distributed applications. By doing so, it would not transmit the big sized object with code that is not necessary for the server, will consume data, and slow down the transfer.
In our system, data transfer objects are simple objects that do not contain any business logic or methods implementation. The objects only contain parameters and getter/setter methods. Assembler is to implement the assembly of the data transfer object.

Remote facade

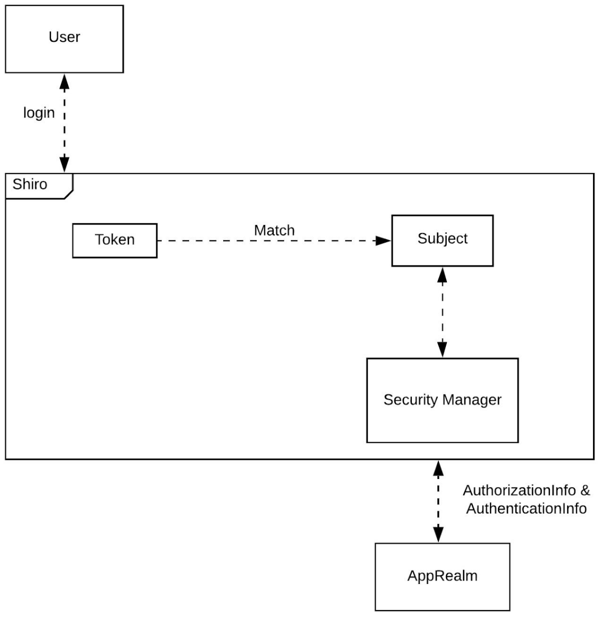
The remote facade provides bulk access operations that allow remote clients to transfer large quantities of data in one method call. The facade packages several remote methods calls into a single call, which will only require one networked message passed and therefore it should take less time to invoke.
In our system, the data transfer object is used with the remote facade. As shown in the graph below, remote facade calls Mapper classes to retrieve data and use assemblers to assemble data into the data transfer object.



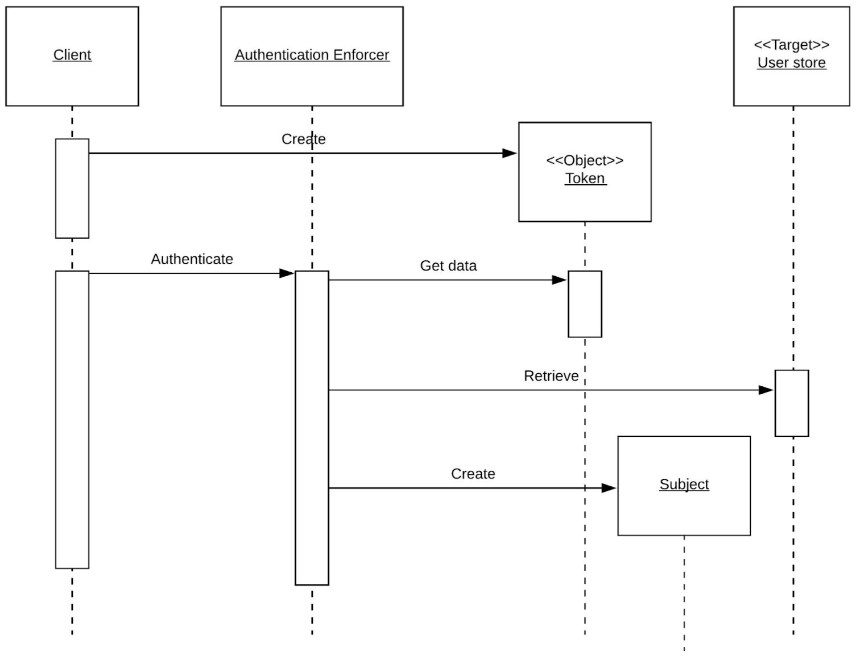
6.4 Security

6.4.1 Authentication enforcer

This Authentication enforcer pattern is a centralized authentication system that performs authentication of all users by authenticate information provided by the user in the presentation layer. And the pattern encapsulates the details of the authentication mechanism. It verifies whether the user exists in the system and the user is who they are. It authenticated username and password provided by logging users and lookup in the database.
And we are using an external library, Apache Shiro, to implement authentication.



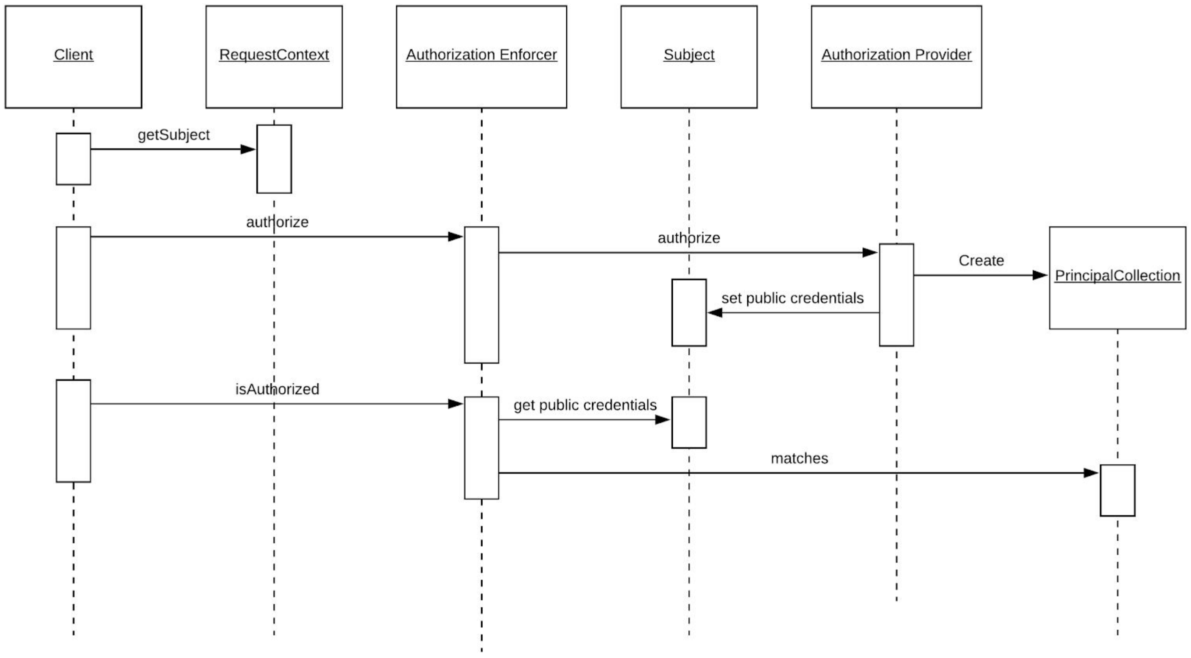
In the system, it creates a token using a username and password when the user logs in the system. The information in the token is authenticated by comparing it to the subject provided by the Shiro security manager. The manager system retrieves the authentication information from the database through AppRealm class if the user existed. Otherwise, the authentication failed and return an exception.



According to the diagram, Client firstly send request creating token with username and password. The username and password from the token is then authenticated by retrieving information from user store. If the user has been authenticated, authentication enforcer will create Subject for the user logging into the system. Otherwise, Authentication Enforcer send exception.

6.4.2 Authorization enforcer

The Authorization enforcer pattern is a centralized access controller, which handles authorizations of users' actions. Based on the different roles of users, end-users should not be able to access all parts of application. The application must authorize that a user can perform function In our system, this design pattern takes control of access to a particular function based on administrator role or customer role. As in the graph below, a user creates a token when the user logs through web UI. Then the Shiro security Manager system creates subject to verify the user's information by calling to AppRealm class. For example, if the user wants to perform a particular action and match to corresponding roles in the system, such as the customer role or administrator role, then the user would have access to parts of functions in the system.

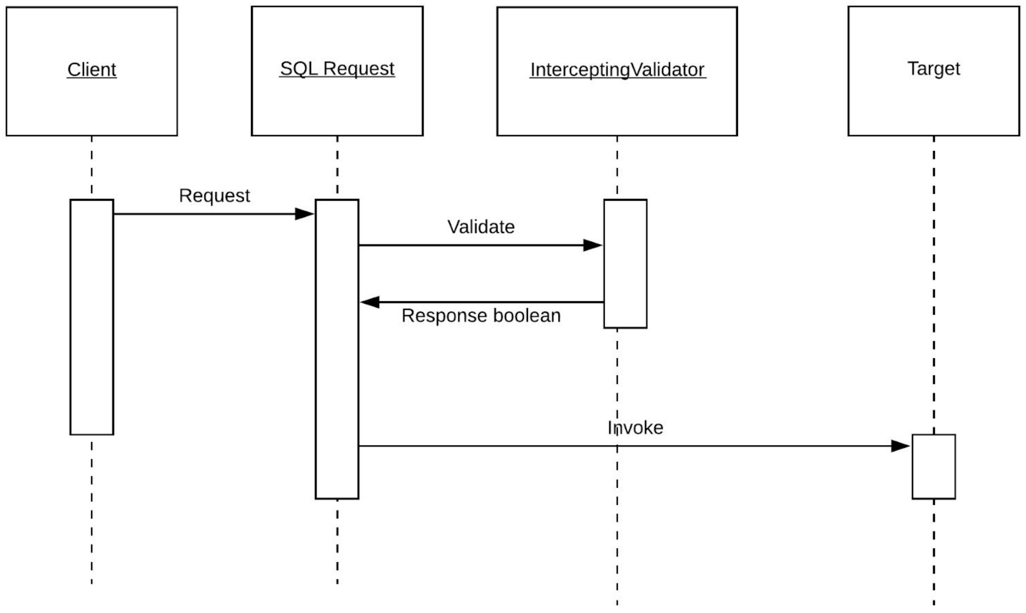


According to the graph, Client send request creating Subject and the Subject is associated with the client. For authorization of the user, it create PrincipalCollection through Authorization

Enforcer, the PrincipalCollection then assign permission/credentials to the user's Subject. The Authorization Enforcer checks permission/credentials before user is able to log into the application.

6.4.3 Intercepting validator

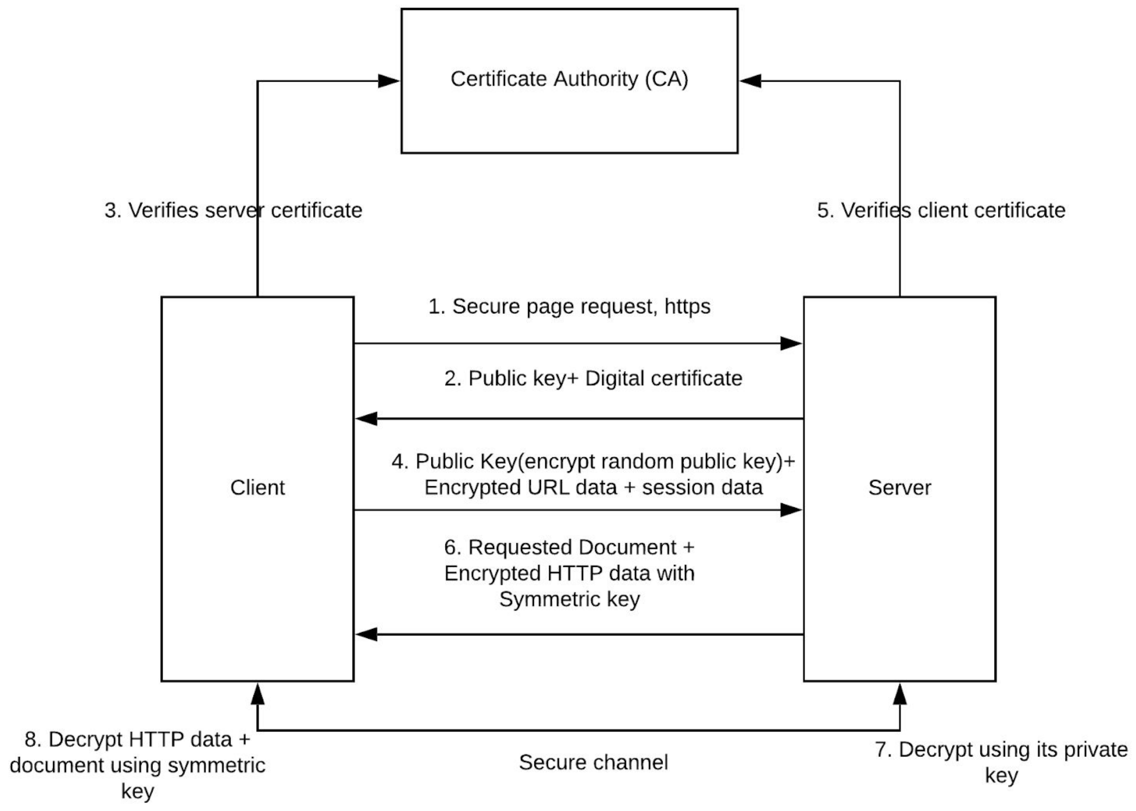
The intercepting validator pattern is to prevent user from gaining unauthorized access to the data of other user's information. All input information will be cleansed and validated before its use in the system. In our system, the intercepting validator is performed on the server-side, and it provides filters that applied to inputs.
Without the intercepting validator, a user may provide inputs including bogus data or malicious code, such as SQL query command. It would expose other user's information stored in the database to the public.



According to the graph, the SQL query statements requested from user are all validated before user is able to retrieve data from target resources.

6.4.4 Secure pipe

This pattern is to ensure requests and data sent over a network are secure and encrypted. In our application deployed on Heroku cloud platform, It has been implemented using web-server-based Transport Layer Security (TSL) and Secure Sockets Layer (SSL). It is using asymmetric cryptography for data exchange. It is implemented by asking the Heroku certificate from a web server and encrypt data using the public key in the certificate every time when the client wants to send data over the network to the server.

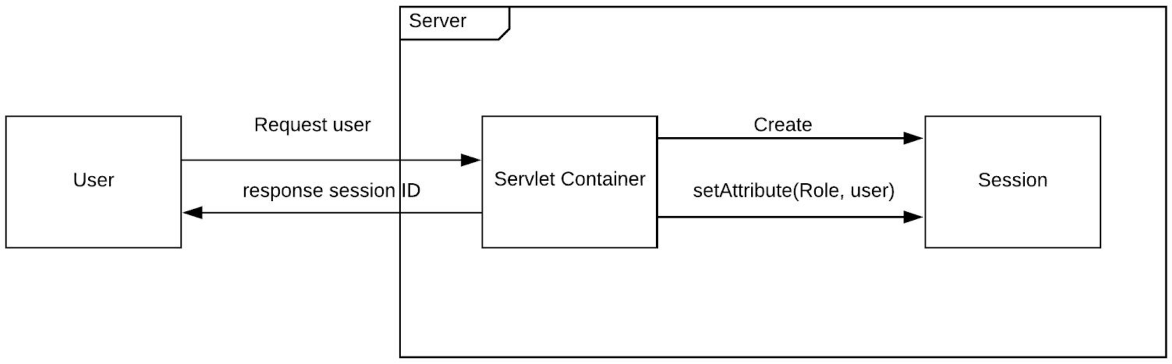

Certificate Authority (CA)
3. Verifies server certificate
5. Verifies client certificate
1. Secure page request, https
2. Public key+ Digital certificate
Client
4. Public Key(encrypt random public key)+ Encrypted URL data + session data
Server
6. Requested Document + Encrypted HTTP data with Symmetric key
8. Decrypt HTTP data + document using symmetric key
Secure channel
7. Decrypt using its private key

As shown in the graph above, the client verifies the server's digital certificate from a certificate authority before encrypting data using the public key. And server decrypts the encrypted data using its private key. By doing so, it establishes a secure communication channel using the public and the private key.

6.5 Sessions

6.5.1 Server Session State

The server session state pattern is to maintain the session state in the server memory. In our application, we implement session state using session management from the external library, Shiro. The session management system defaults to the HttpSession implementation, which creates a session on an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user. As shown in the graph below, when a user logs into the system, the Shiro security management system authenticates, authorizes the user and then creates a session for the user. In the logout controller, the logout() method invalidates the user session and logs the user out.

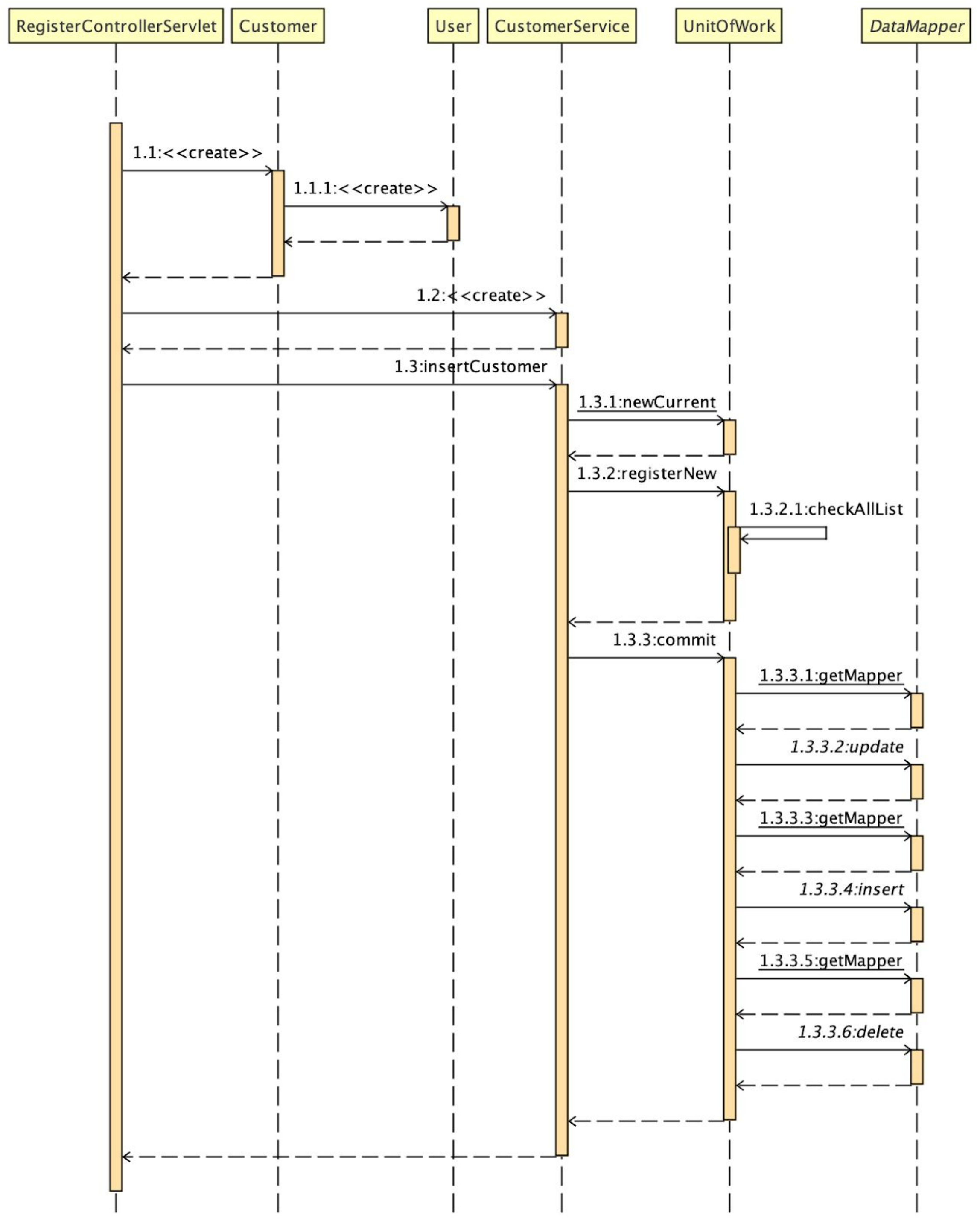


Comparing with Server Session State, **Client Session State** maintains session state in the client, and it does not scale well for large amounts of data and data may be tampered with as data sent across a network.

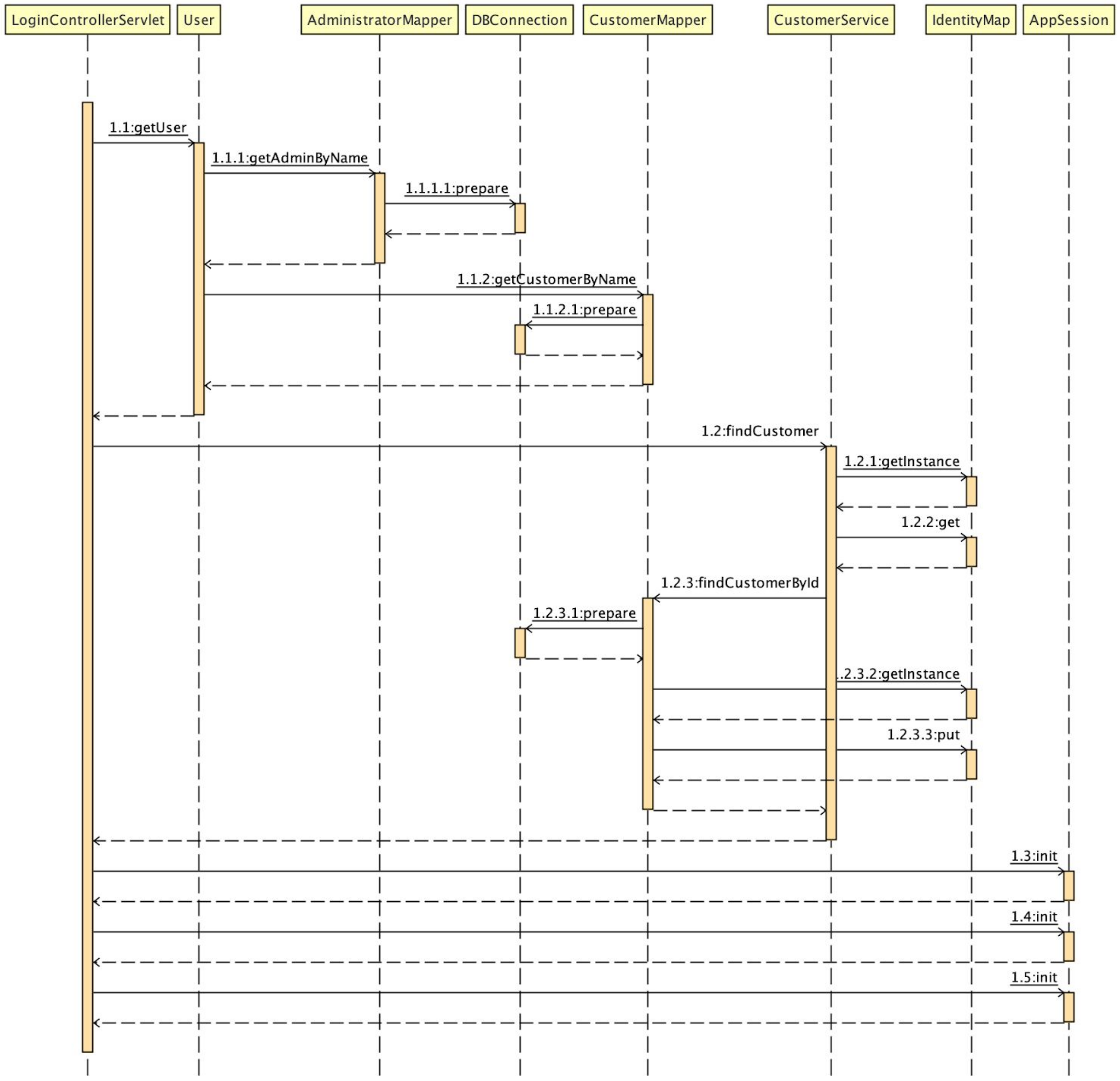
Comparing with Server Session State, **Database Session State** maintains session state in the database. It would slow down performance due to having to pull data from the database with each request and consume more resources in the database.

7. Interaction Diagram

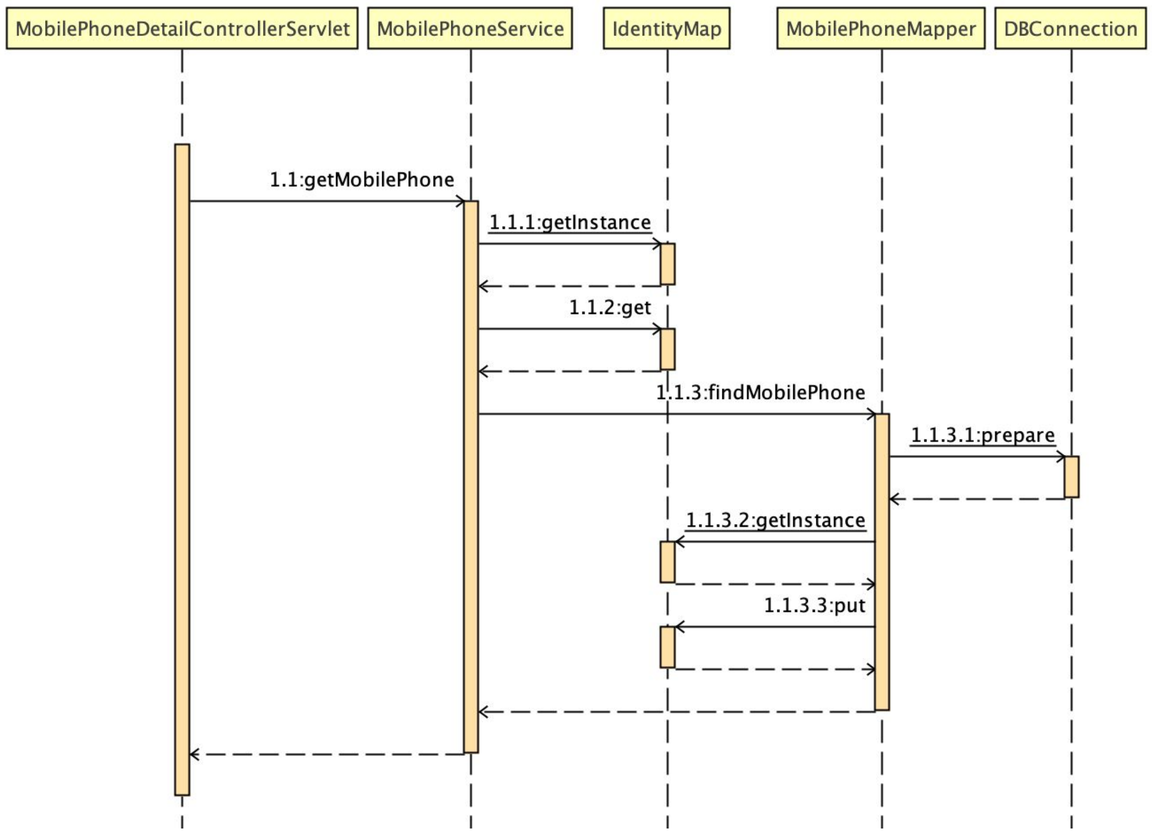
7.1 Register new account



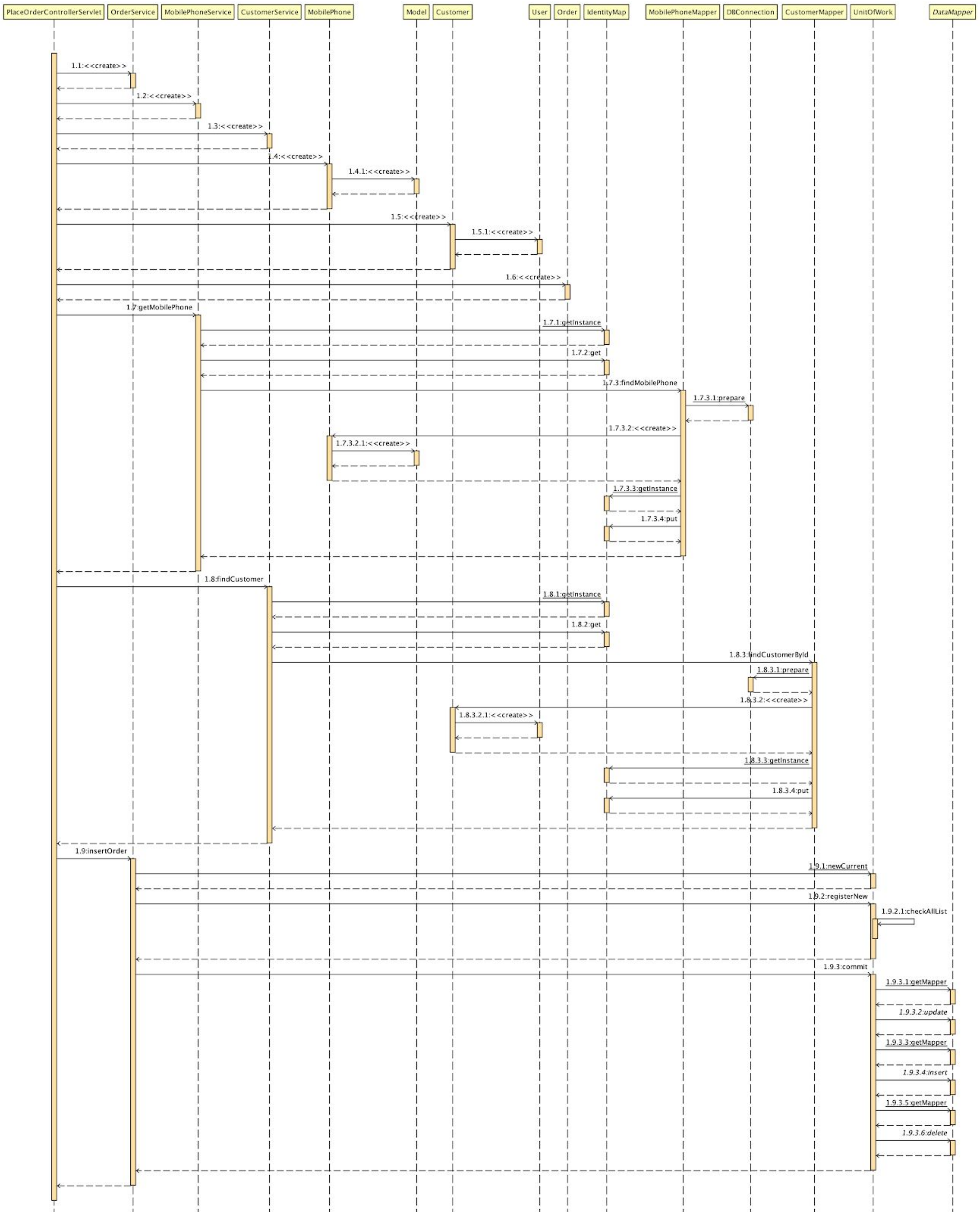
7.2 Login


LoginControllerServlet
User
AdministratorMapper
DBConnection
CustomerMapper
CustomerService
IdentityMap
AppSession
1.1:getUser
1.1.1:getAdminByName
1.1.1.1:prepare
1.1.2:getCustomerByName
1.1.2.1:prepare
1.2:findCustomer
1.2.1:getInstance
1.2.2:get
1.2.3:findCustomerById
1.2.3.1:prepare
.2.3.2:getInstance
1.2.3.3:put
1.3:init
1.4:init
1.5:init

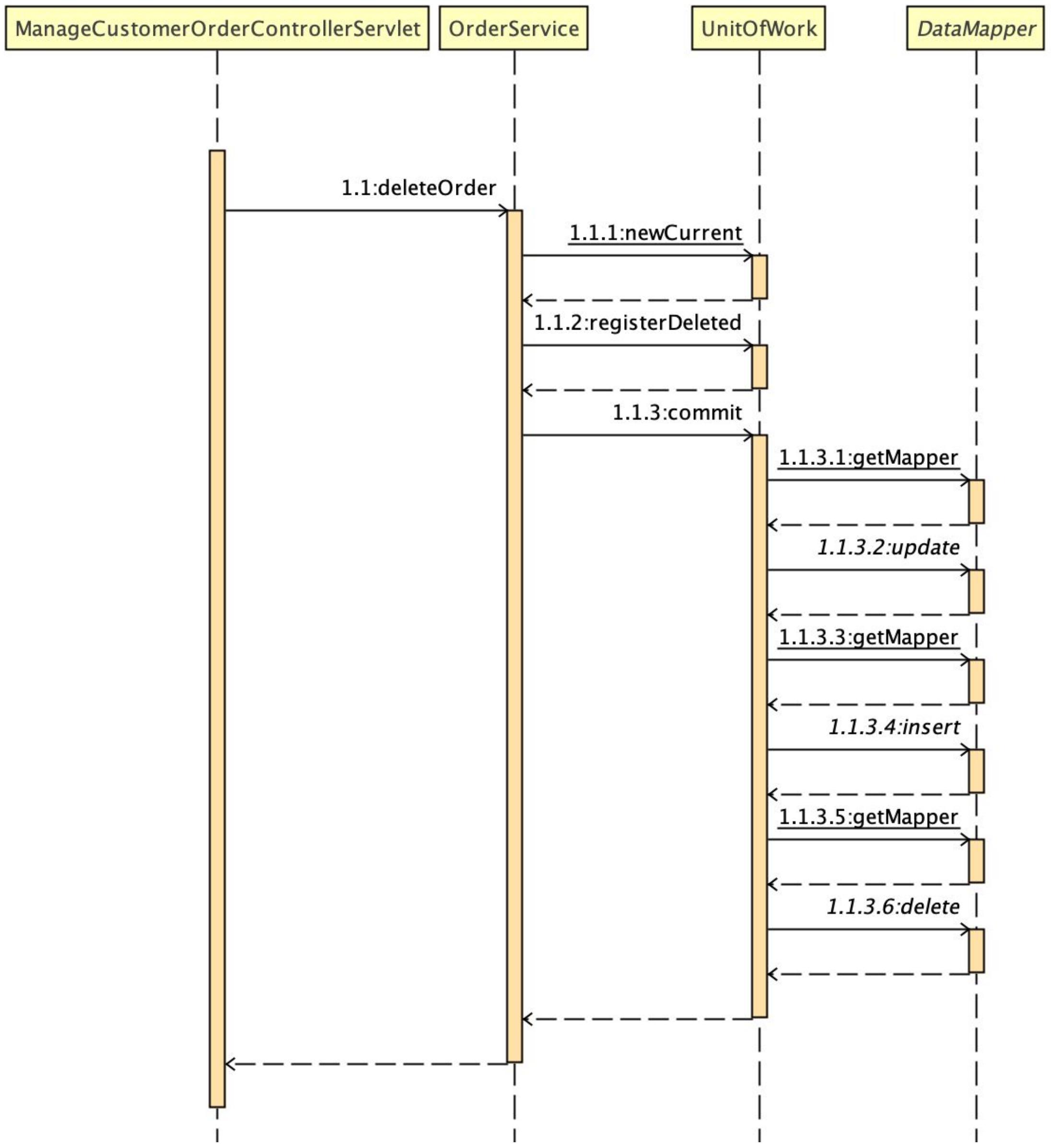
7.3 View mobile phone details



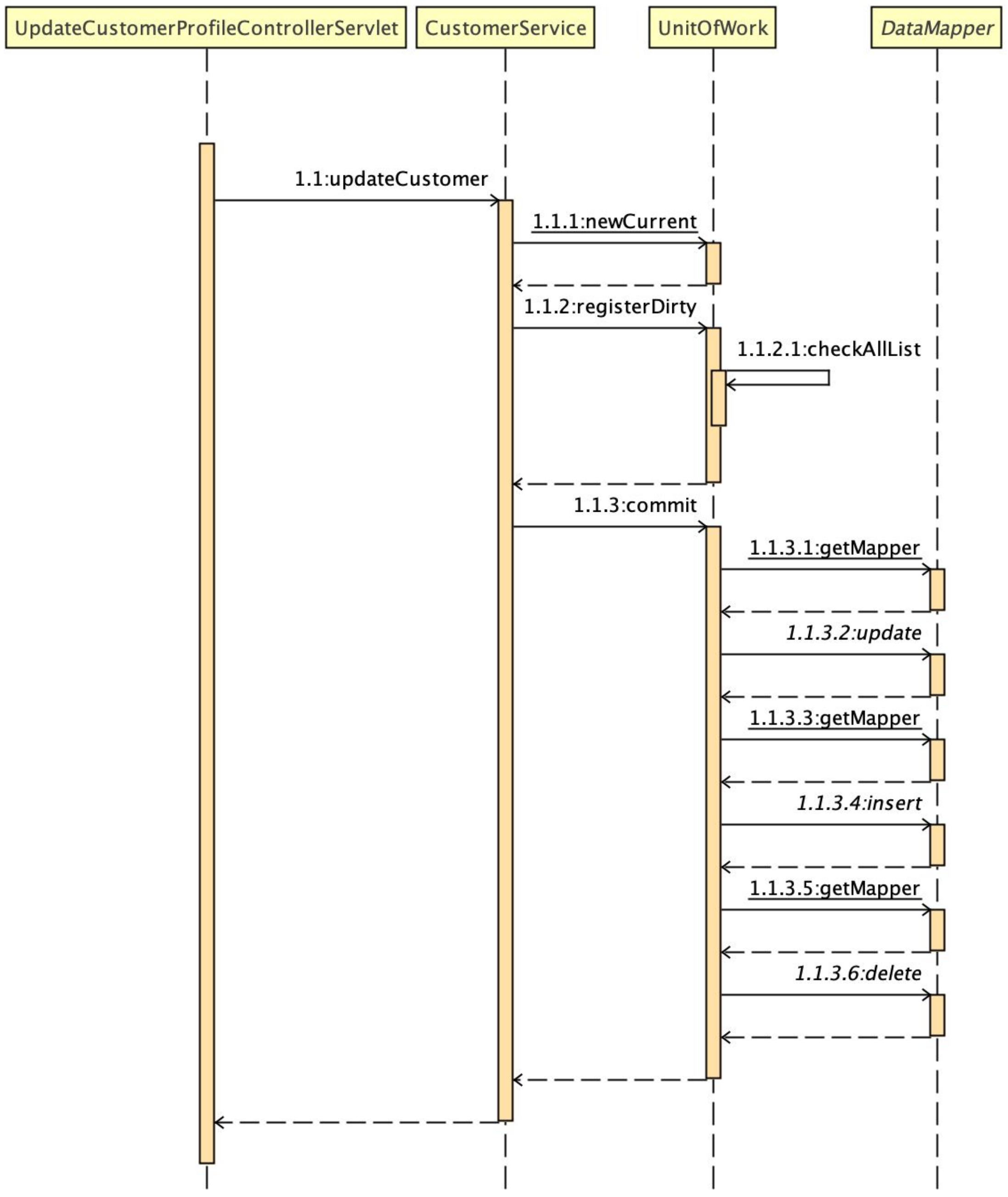
7.4 Place order



7.5 Cancel orders



7.6 Update Profile


UpdateCustomerProfileControllerServlet
CustomerService
UnitOfWork
DataMapper
1.1:updateCustomer
1.1.1:newCurrent
1.1.2:registerDirty
1.1.2.1:checkAllList
1.1.3:commit
1.1.3.1:getMapper
1.1.3.2:update
1.1.3.3:getMapper
1.1.3.4:insert
1.1.3.5:getMapper
1.1.3.6:delete