SYSTEM DOCUMENT

Assignment4

Wave Team

June 17, 2016

Supervised By Dr. Liu Yan

Contests

1 Introduction

1.1 Purpose

These days as B2C business is increasing rapidly, the growth of logistics business is also remarkable. Facing such kind of condition, our cross-platform system specifically renders services to an independent logistics company accepting orders from e-business companies as well as individuals. Aiming at improving the efficiency of filed personnel and customer satisfaction of the logistics company, the system mainly considers 10 scenarios. From these scenarios, a use case modeling describing the detailed requirements is developed in the previous software requirements specification.

In this system document, the whole model of our Sloth Express system is specified, including brief requirements, analyses and the design model. It will explain the purpose and scope of the project, refining the system architecture both in logical and physical view, as well as specify what are subsystems, platforms and the frameworks of the system. We have also incorporated design mechanisms and detailed realizations in this document. This document is intended for both the stakeholders and the developers of the system. In addition, this document displays the final mock-ups and prototypes.

1.2 Definition

As Jobs has ever said, “People don't know what I really want at all, until your products are in their eyes”. This project is specially designed for an independent logistics companies like UPS. The business scope is limited within Jiangsu, Zhejiang and Shanghai. The logistics company only serves personal customers and e-business companies with cash-on-delivery express or normal express. The system focuses on logistics service without regard to other business.

Our system model focuses on architectural styles, analysis and design patterns. There are several common patterns from which to be chosen. In order to generate the architecture, we need sufficient considerations and a series of mechanisms to help. Class diagrams, sequence diagrams and communication diagrams will organize with subsystems, interfaces, use case realizations and implements of mechanisms to develop this model. They are all provided in this document.

1.3 System Overview

Architecture Our architecture is extended from MVC (Model-view-controller) architecture including 6 layers: Presentation, Transport, Controller, Service, Entity Domain and Data Persistence. The Presentation Layer contains presentation and

application and Transport Layer provides APIs among users, third-party systems and its own system. Controller Layer assigns tasks and does some simple jobs while the Services Layer finishes the majority of jobs. Entity Domains store entities and relationships among those entities. As for the enormous data, they are stored in the database, which are modified and maintained in Data Persistence Layer.
Considerations are mainly divided into two parts: business and technique, on which building such architecture depends. The business part is responsible for business requirements, business scope and costs. As for the technique part, it is responsible for general techniques, software environment and hardware devices.
For mechanism, four mechanisms are employed: Persistency, Communication, Legacy Interface and Security.
Technically, platforms and frameworks are offered in this document, and it shows the installation topology.

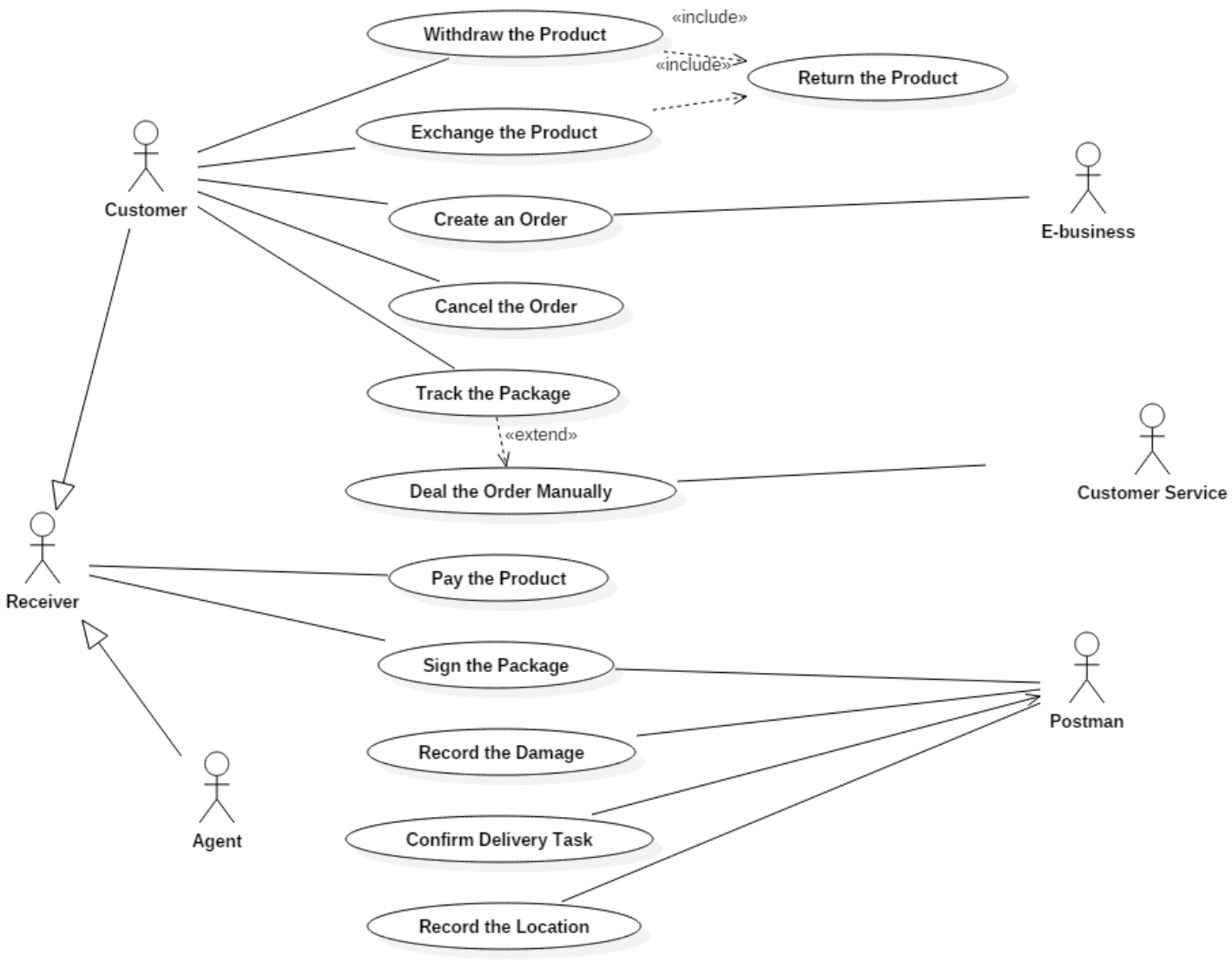
Analysis Key abstractions are generated based on the architecture and the mechanism. With stereotypes and relationships supplemented, the initial global class diagrams display the principal classes of the system. All of the classes are divided into three portions. The first part contains entity classes, corresponding to Service Layer, Domain Entity Layer and Data Persistence Layer. The second part named controllers corresponds to Controller Layer and the last part called boundaries corresponds to Presentation Layer.
To analyze use cases in details, we adopt a sequence diagram and a communication diagram to illustrate the realization of a particular use case.

Design To realize the system, we employs subsystems, packages and independent classes. They are generated according to the architecture and patterns we selected. The documents lists all the subsystems and corresponding interfaces. To explain them concretely, an integrated subsystem will be presented. As for concrete design of mechanisms and use case realizations, each of them reveals two typical parts. Particular SDK, protocols, languages, platforms, parsers, DBMS and other techniques have been determined.
Based on the design, we draw the final mock-ups and codes the two mechanisms mentioned above to show how them work.

2 Requirements

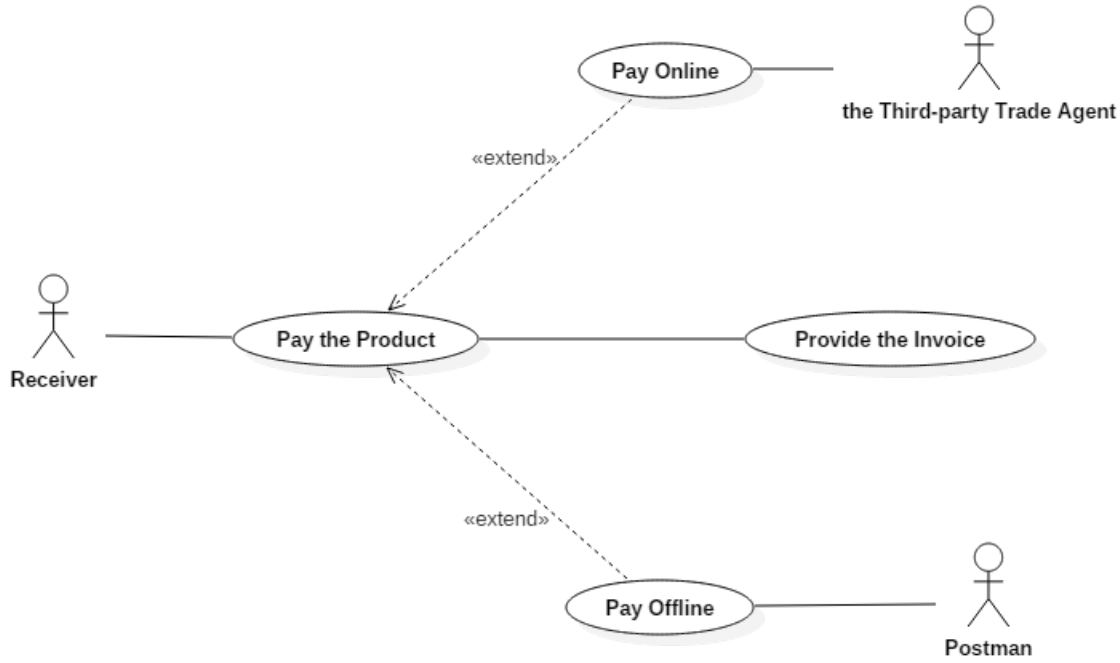
2.1 Use Case Modeling

2.1.1 Global Use Cases

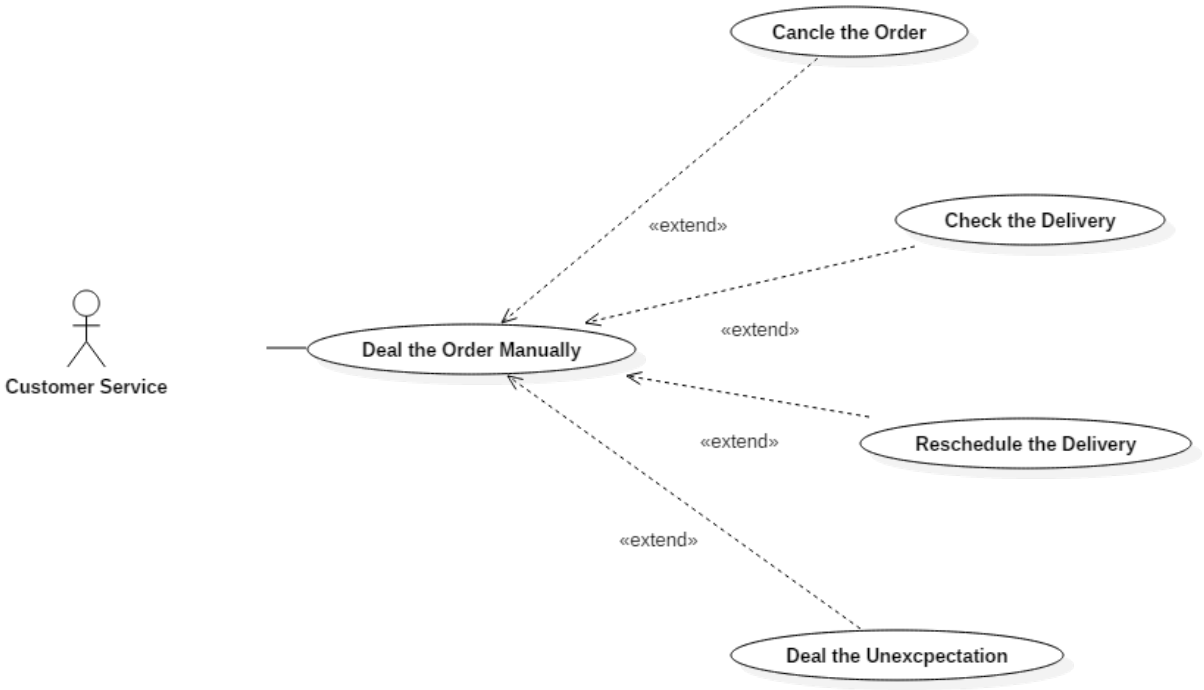


2.1.2 Sub Use Cases

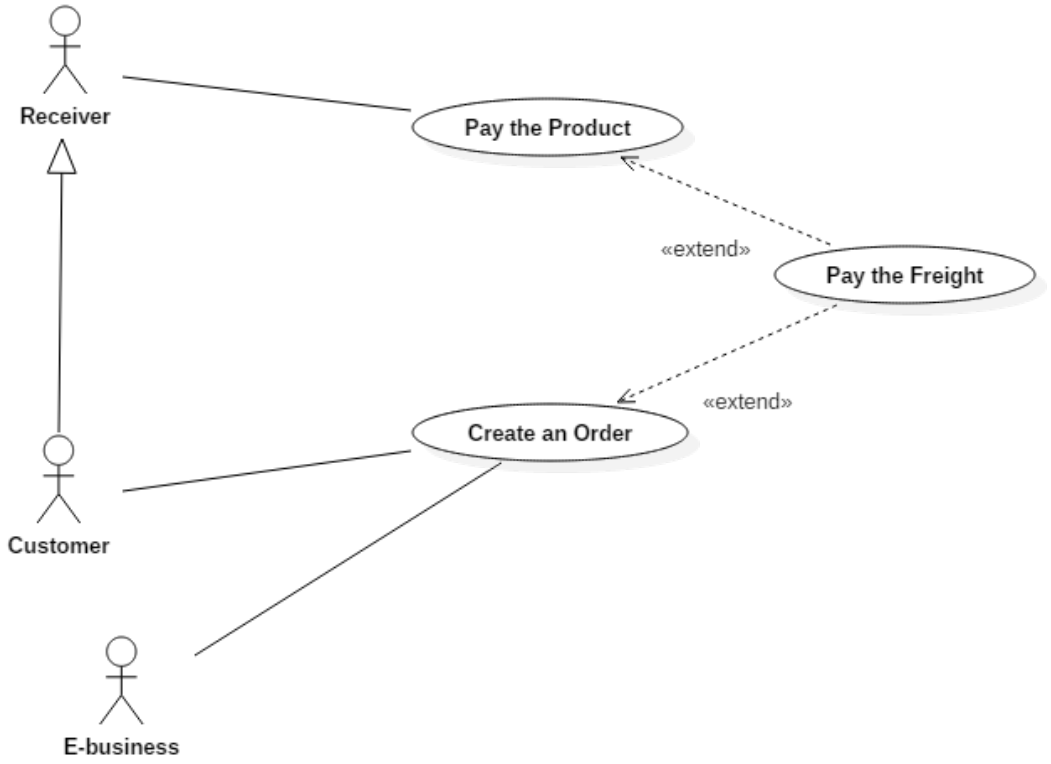
Payment View



Logistics Company View



Freight View



2.2 Supplementary Specification

2.2.1 Security

The system should avoid the database being attacked and data being taken advantage of by the wicked.

- **Access and Data Integrity**

1. The authorization of access to the system of postmen, customers and customer servers should be classified and announced clearly. With certain authorization, different users have limited access to data and operation.
2. The server should use anti-virus software.
3. Firewalls and network protection are necessary, and they should be updated in time.
4. The atomic processes in the database will ensure the accuracy of the database.

- **Encryption**

1. The session should not be transmitted in DNS.
2. All texts and messages should be transformed into a code which is unreadable.
3. Two keys are used to identify a certain user. One public key is used for encryption and anther private key is used for decryption. The key is a completely random mix of

letters.
4. The session will record the activity of the customer, and if the customer has no operation for 5 minutes, he or she will log out the system automatically.
5. After customers log out the system, all the private information (cookies) will be cleaned.

- **Digital Certificates**

1. We use digital certificates as a replacement of user names and passwords, for example, SSL Certificates. It will be used automatically with the permission of users.
2. The IP address or location where users login the system will be recorded and when the account is used beyond their regular locations, the user will get alarm.

- **Digital Signatures**

1. Users should login the system with a password. Our system will test its complexity. If it is too simple, the system will remind the users to complicate it. That involves cryptography.
2. We use a message digest to ensure the integrality of the data.
3. If necessary, we can extend our fingerprint system to login system.

2.2.2 Performance

- The information of the package, including the real-time position, Order-ID, the postman etc., should be checked by customers in 3 seconds with at most 0.1% error rate.
- The payment should be confirmed in 2 seconds by the system from the moment when the third party trade agent sends the message or the postmen report the payment.
- The order created by customers should be processed in 15 minutes.
- The orders obtained from e-business should be processed every hour (about 5,000 orders).
- Information of the delivery such as the phone number, the address, the receiver and others should be updated and checked by postman in 1 min.
- This system allows the e-business to create batch orders which can be sent at regular time.
- The estimate of delivery time should be accurate with the max uncertainty in 2 days.
- The expectation should be sent to custom service in 2 min from the time a postman reports it.
- This system's unavailable time should be controlled in 20 minutes in a year.
- To offer the best user experience, a content delivery network should be used by this system.

2.2.3 Data Storage and Computing

- To store a huge amount of data, distributed database should be used. And it should use Homogeneous DDBMS.
- Considering that there may be an enormous number of visitors and inquires at the same time, the system must implement cloud computing service.
- The system can support as many as 1500 times of visits per second.
- There must be a copy of the database, including device entity, software, data and even employees, in order to prevent some unpredictable disasters.
- If the database is destroyed, the copy should be enabled in 3 hours.
- The data can be in English, Chinese, Japanese, French and Korean.

2.2.4 Track the Package

- In order to track the package, the GIS system should be applied, with the help of the GPS system. The system get geographic information from a third party system, and get the position of postmen who deliver the package through the system of postmen. And this system should match both kinds of the information and show it to users of the system.
- The system for postmen should upload the position of the postman automatically every 2 hours, through 3G, 4G or WLAN net.
- If the locations of postmen are missing for 4 hours, the system should inform the custom servers, and custom servers will contact with postmen.

2.2.5 Maintenance

- The distributed database should be maintained by the employees of our own company including the employees of the stand-by database every day when the visit traffic is not heavy.
- The software for custom service, customer and postmen and the system itself should be maintained by our employees.
- The geographic information source should be multiple, in case that one of the sources is unavailable.
- The engineers from the company offered DBMS will maintain our system every year.
- An integrated scheme to deal accidents, for example the crash of database, is necessary.

2.2.6 Others

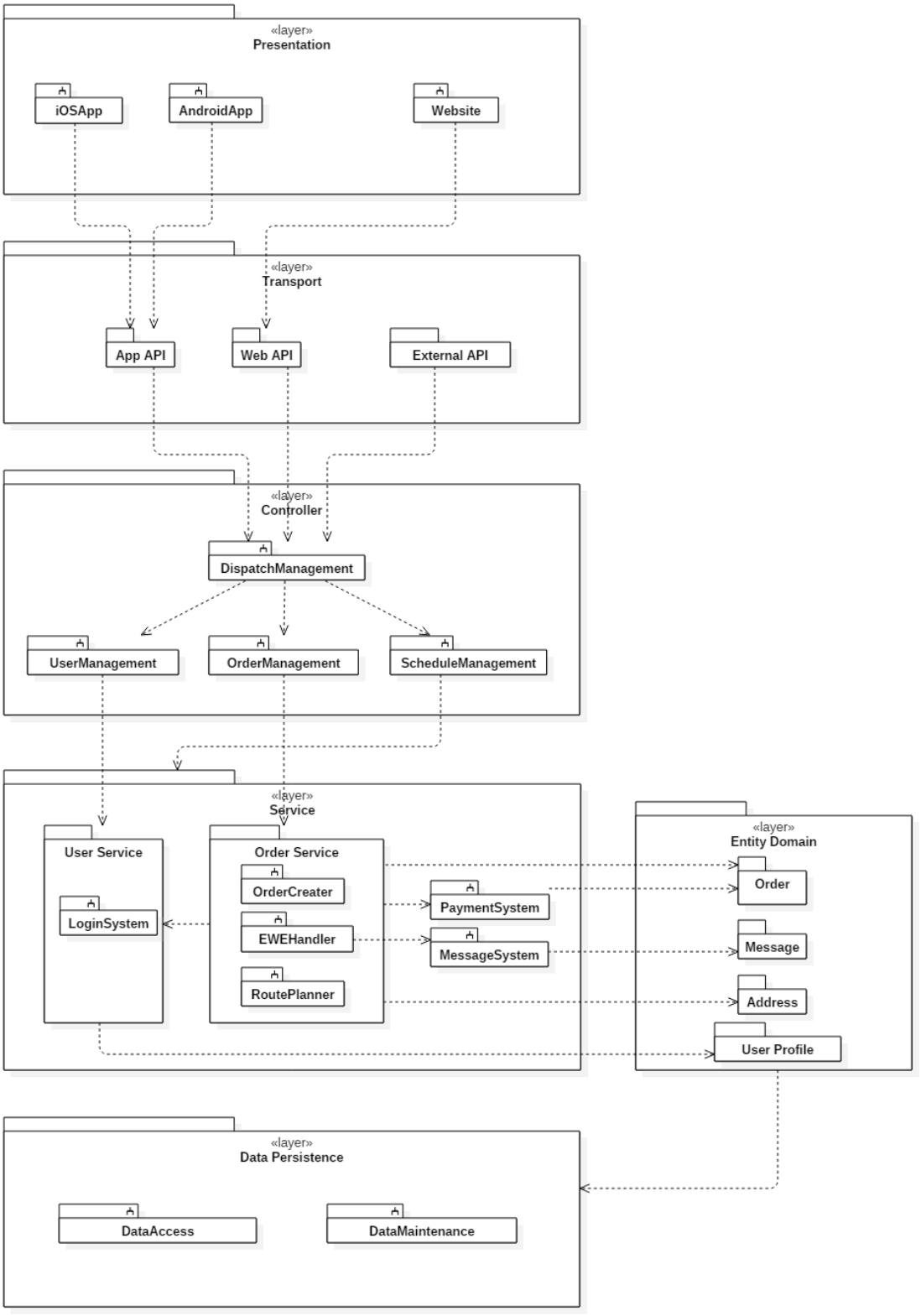
- The architectures of the postman APP and the customer APP are B/S and C/S, but

that of custom service is C/S for safety.

- Our system can be used in iOS and Android on mobile devices and in a normal browser on PC (Windows/OS/Linux).
- Anticipated development time is two months.

3 Analysis

3.1 Architecture



3.1.1 System Structure—Platform-dependent Architecture

The architecture have 6 layers, which are extended from the 5-tier architecture. We have taken extensibility, code reusability, security, flexibility into consideration.

The first layer is **Presentation Layer**. This layer provides a user interface to translate the logistics information to user friendly representation. It contains Web front-end, Android App and IOS App, serving as users' entry of the system. It transfers the information the transport layer and can call the API, provided by the transport layer. The layer in our system consists of three subsystems, iOS App, Android App and Website. As for the platforms consideration, the website will take advantage of Bootstrap framework, using the language of html, css and javascript. The IOS and Android Apps will use the Xamarin platform.

The second layer is **Transport Layer**. This layer works as a coordinator of our system. It provides some Application Programming Interface (API) to the Presentation Layer, including Web API, App API and external API. The external API provides interfaces to some third-party services. It plays a role of delivering the request from Presentation Layer to Controller Layer. As a coordinator, it will also connects those external system to our internal module. This layer is controlled by the controller layer. We have designed protocols and different frameworks to implement those APIs, including Spring Web and Xamarin framework.

The third layer is **Controller Layer**. This layer contains a major controller, named as dispatch management. And this major management controls user management, order management and schedule management. In our system, the dispatch management will choose the correct controller to handle the requests. The corresponding management controller will response to the request and deliver messages to Service Layer. This layer is one of the part of MVC (model-view-controller) framework. We are going to use Spring web framework to implement this part.

The fourth layer is **Service Layer**. This layer provides concrete services for the user, containing user service and order service. It is the most significant part we have refined in the design model. We have designed the details of this service layer, according to different requests and platforms to be used. Each of the service contains several subsystems which connect with entity domains and transfer information. The user service includes login system providing service for users. The order service includes create the order, handle the EWE (Exception/Withdraw/Exchange) and route planning services, which will connect with payment and message subsystems. These two subsystems have relationships with entity classes and fetch information from Entity Domain Layer.

The fifth layer is **Entity Domain Layer**. This Entity Domain layer contains 4 packages,

including User Profile, Message, Order and Address. This layer contains the concrete information stored in our system. It provides services for the service layer by helping it get concrete information. We separate these information into this independent layer, so that there will be less data redundancy and the efficiency of reading or writing will be improved. This layer is also related with the Data Persistence layer. It need the service and support of the Data Persistence layer.

The last layer is **Data Persistence Layer**. This layer define the data model of our system, including data access and data maintenance. In this layer, we will take advantage of different database and handle different type of data efficiently (e.g. structural and non-structural data). We have also provided mechanisms to implement the data maintenance. We will use Oracle DBMS and the SQL language. For the data access, we will choose the Hibernate and NHibernate framework to help.

3.1.2 Consideration

I. Business

Business requirements

We mainly provide services to E-business companies, and we also accept orders from individual customers. And our system provides services for normal users, e-business, postmen and customers attendants.

Business scale

The order quantity is expected to be 5,000 a single day, so our system must be able to withstand large amounts of data flow and high concurrent access volume. What's more, some critical nodes in our system must be able to transfer a large amount of data in real time.

Costs control

In order to control the costs, the order creation and delivery scheduling tasks are automatically controlled by system, but we leave the package delivering and exception handling to manually operation.

II. Technique

General technique

We use MVC architecture to build our system which uses a non-real-time connection (HTTP / HTTPS) or a real-time connection (socket) to achieve communication. And the user interface is rendered by the template engine.

Running Environment

We will use servers whose operating system are based on Linux, CentOS or RedHat And the database is likely to be MySql. As for the client customers, they can use web,

android app and iOS app. Finally, we need GIS and GPS for address collection.

Hardware environment

Our customer and postman use mobile devices or iPad to log in our system.
And after thinking over users' quantity and reliability, we choose enterprise servers to run our system and the server needs to be of high-performance Communication.

3.1.3 Mechanism

Persistency Mechanism

Volume: The volume of database.
Content & Redundancy: The content and redundancy of data such as address.
Duration: The duration of access time.
Access frequency (insert, update, delete, select): The frequency of accessing.
Reliability: The reliability of both system and database.
Exception Management: The capacity for dealing with exception.

Communication Mechanism

Message Routing: The routing and protocol to transfer messages.
Message Content: The type and content of messages.
Synchronicity: A batch of operations happen in the same time.
Consistency: The consistency of data in different ends.
Persistency: Persistence communication between clients and servers.

Legacy Interface Mechanism

Latency Time: The time interval between the stimulation and response.
Element: The elements in the layout of the front-end.
Template Engine: The primary processing element of the system.
Local Data: The data stored in customers' and postmen's devices.

Security Mechanism

Data Granularity: The extent to which the data is composed of distinguishable pieces.
Encapsulation: To restrict access to some of the object's components
Authorization: Users' limited access to data and functions.
Encryption: To encode messages and information so that only authorized parties can read it.

3.1.4 Implementations of Platforms and Frameworks

I. Bootstrap for Website

Language: HTML、CSS、JAVASCRIPT
Instructions:

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web. And it is compatible with the latest versions of the Google Chrome, Firefox, Internet Explorer, Opera, and Safari browsers, although some of these browsers are not supported on all platforms.
Since version 2.0 it also supports responsive web design. This means the layout of web pages adjusts dynamically, taking into account the characteristics of the device used (desktop, tablet, mobile phone).
Starting with version 3.0, Bootstrap adopted a mobile-first design philosophy, emphasizing responsive design by default.
The version 4.0 alpha release added Sass and Flexbox support.
Bootstrap is open source and available on GitHub. Developers are encouraged to participate in the project and make their own contributions to the platform.

II. Spring for MVC framework of application

Language: JAVA
Instructions:
The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.
The Spring Web MVC framework is designed around a DispatcherServlet that handles all the HTTP requests and responses.

III. Matters about Data

Database:

We will use Oracle DBMS and its corresponding **language**: SQL.

Data access:

- Hibernate

Language: JAVA
Instructions:
As an Object/Relational Mapping (ORM) framework, Hibernate encapsulates JDBC. It sets programmers free from writing complicated JDBC code to access data. With Hibernate, developers can more easily write applications whose data outlives the application process. Hibernate enables developers to develop persistent classes following natural Object-oriented idioms including inheritance, polymorphism, association, composition, and the Java collections framework. On the other side, it can store object-oriented data into relational database such as Oracle.
The performance of Hibernate is great. It supports lazy initialization, numerous fetching strategies and optimistic locking with automatic versioning and time stamping.

Hibernate is well known for its excellent stability and quality, proven by the acceptance and use by tens of thousands of Java developers.

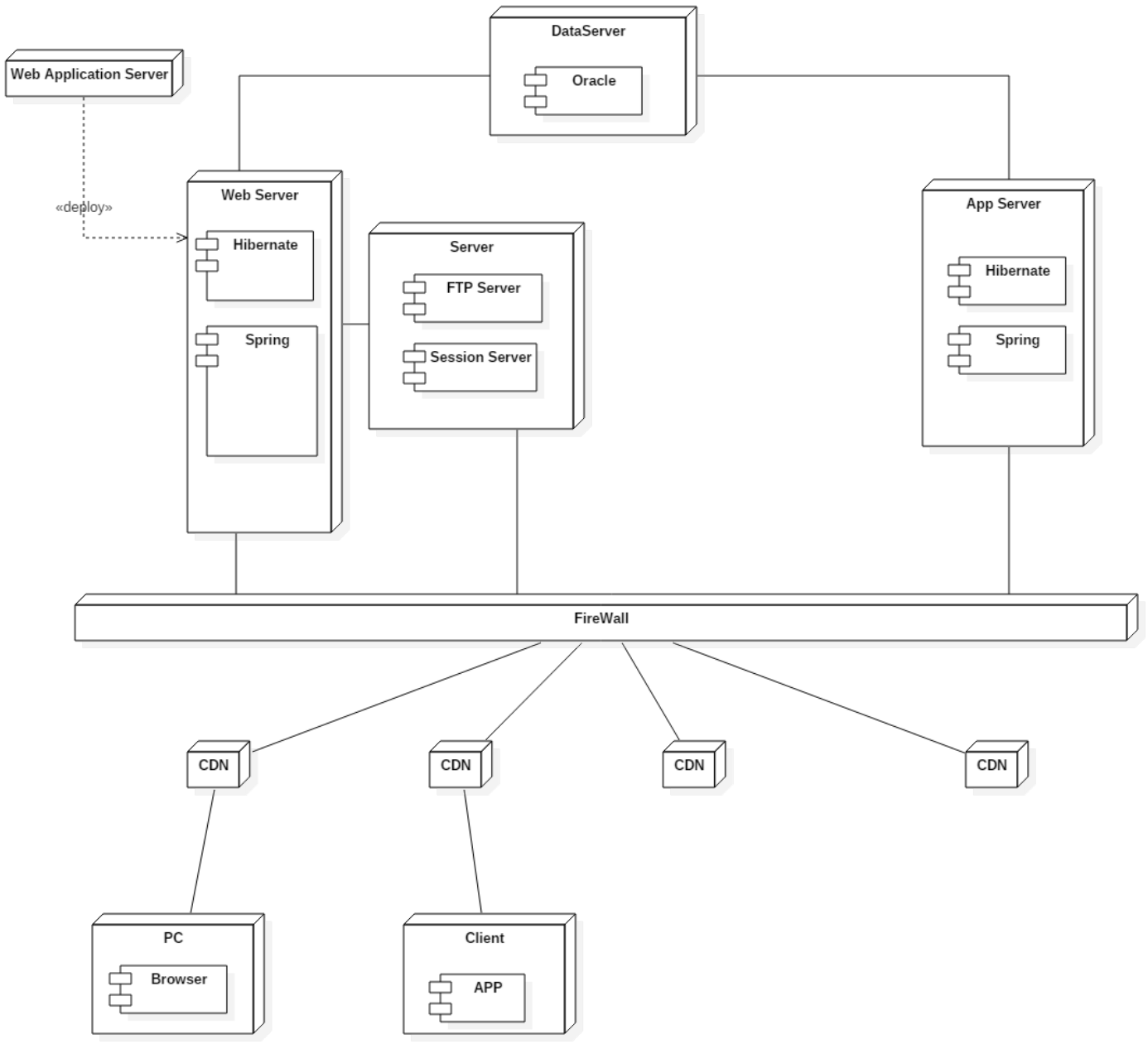
- NHibernate

Language: C# (.Net)

Instructions:

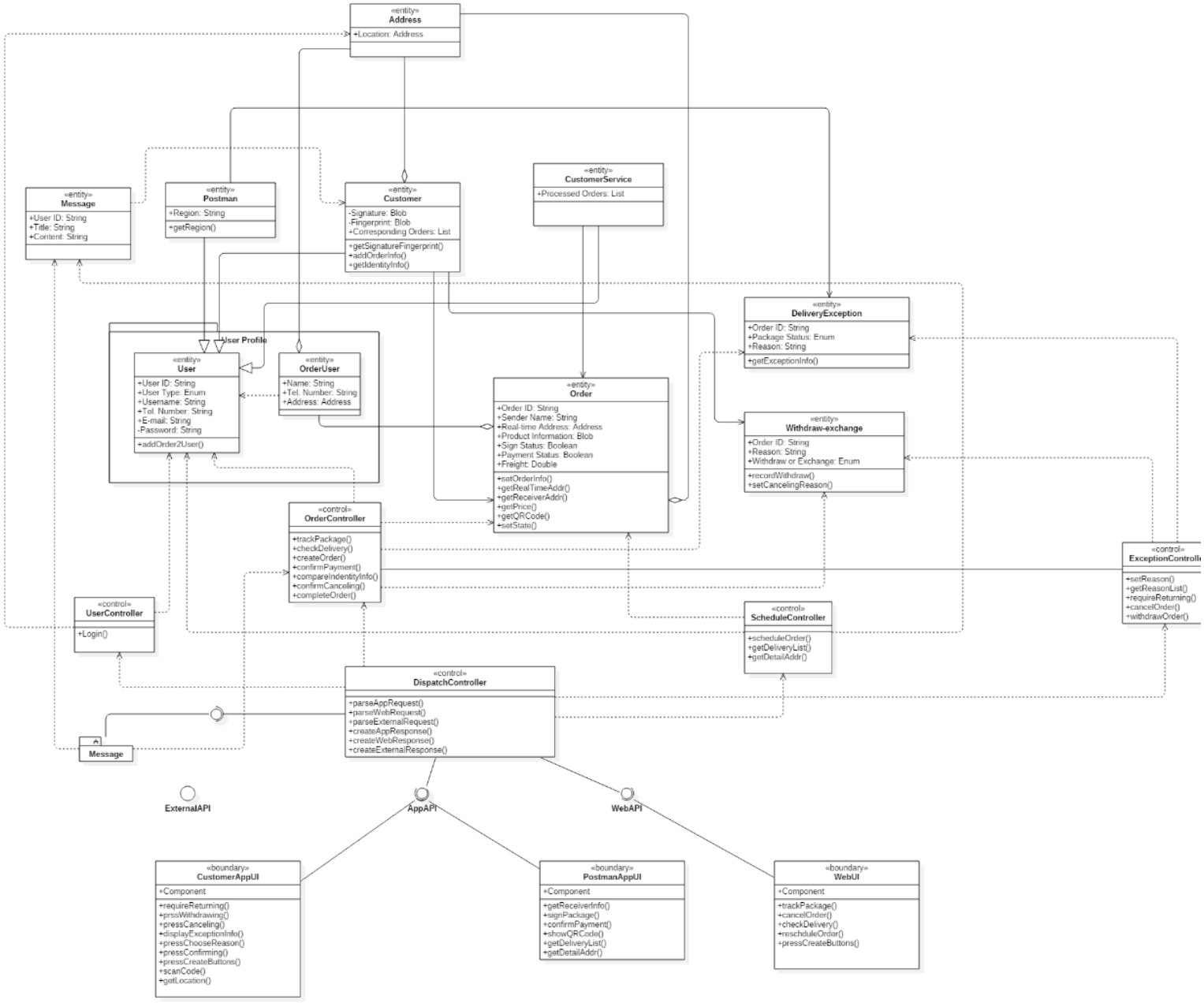
NHibernate is a realization of Hibernate based on .NET framework. It is also used to map data in relational database to object-oriented form. Considering our system should support different frameworks and subsystems in application layer, I think it is necessary to implement NHibernate.

3.1.5 Installation Topology



Our system separates the server into the web server and the App server. For each server, the Hibernate and Spring frameworks are used to construct the server. The Web Application Server is also designed for the system. We separate it from the Web server to let it provide service for web browsers. A server is also designed for file management and session handling. We name them as FTP server and Session Server to provide service for the Web Server. In order to make FireWall connect with the PC or the Client fast, we use CDN(content delivery network).

3.2 Global Class Diagram



The class diagram contains three parts.

The **first part** is the entity classes.
The *UserProfile* package includes two classes: User and OrderUser. The *User* Class records the basic information of a user. User ID is the primary key of Use and User Type is an enum consisting of [CUSTOMER, POSTMAN, SERVER, SUPERUSER]. The user can login the system by email or id or telephone number and password. The *OrderUser* class depends on the class User. In this newly created class, it stores the detailed information of the order, including user's name, phone numbers and address. This class is designed as a self-define data type and can be used to compare the information of users stored in our system.
The class Customer and Postman and *CustomerServices* inherit from the User.
The class *Customer* has a corresponding order list and the attributes to sign the package. The *Postman* has the attribute 'region' to identify the areas where he is in charge of. We define the public API of User to provide services for the controller.
The class *Order* record the basic information of an order. The Order ID is the primary key of Order.
Since the sender or the receiver may not register an account in the system, the Order class records the name, Tel number and address of sender and receiver. The class Order is designed for containing information about our orders and all the basic services will be offered by the controller, such as [createOrder(), checkDelivery(), etc.]. The operation of the Order are limited within data domain like [setState(), getRealTimeAddr(), etc.]. When the user tracks the package, System will return the real-time address of the order.
The *Message* is edited by an independent subsystem Message and sent to user when the package is sent or the package is damaged or lost. And Message can be other recommendation information. The class Message is also designed for providing services composing of fundamental functions for the controller.
The *Withdraw-Exchange* is edited by customer when the customer is unsatisfied with the product after receiving it and later the request will be sent to System.
The *DeliveryException* is edited by postman when the product is lost or damaged during the process of delivery and the info will be sent to System.
The *Address* records the address which the postman has collected.

The **second part** is the controller classes.
All the requests from UI are sent to the dispatch controller. The dispatch controller provides three APIs to web UI, app UI and E-business. The controller will parse the request and delivery the request to corresponding controller. The corresponding controller collects the necessary information to record in the database and handles the request and returns the result to UI. For example, when the user tracks the package by using the APP UI, the APP UI will call the APP API and sends the request to dispatch controller. The *DispatchController*(DC) will then parse the request and sends it to order controller. The *OrderController*(OC) selects the corresponding order from database and calls the getRealTimeAddr(), and returns the result to the DC. DC returns the result

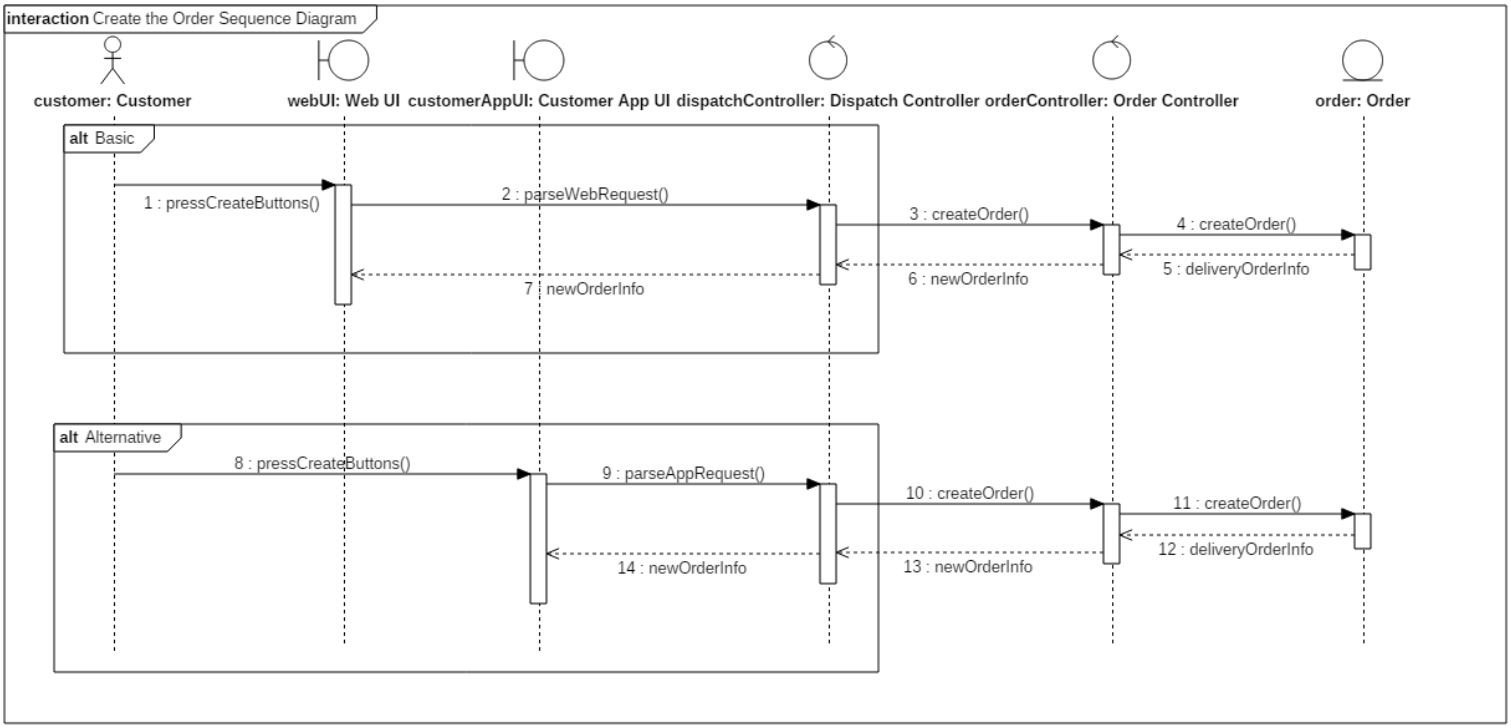
to APP UI. The *ExceptionController*(EC) class is specially abstracted for controlling the exception handling and all the schedule issues will be handled with *ScheduleController*(SC).

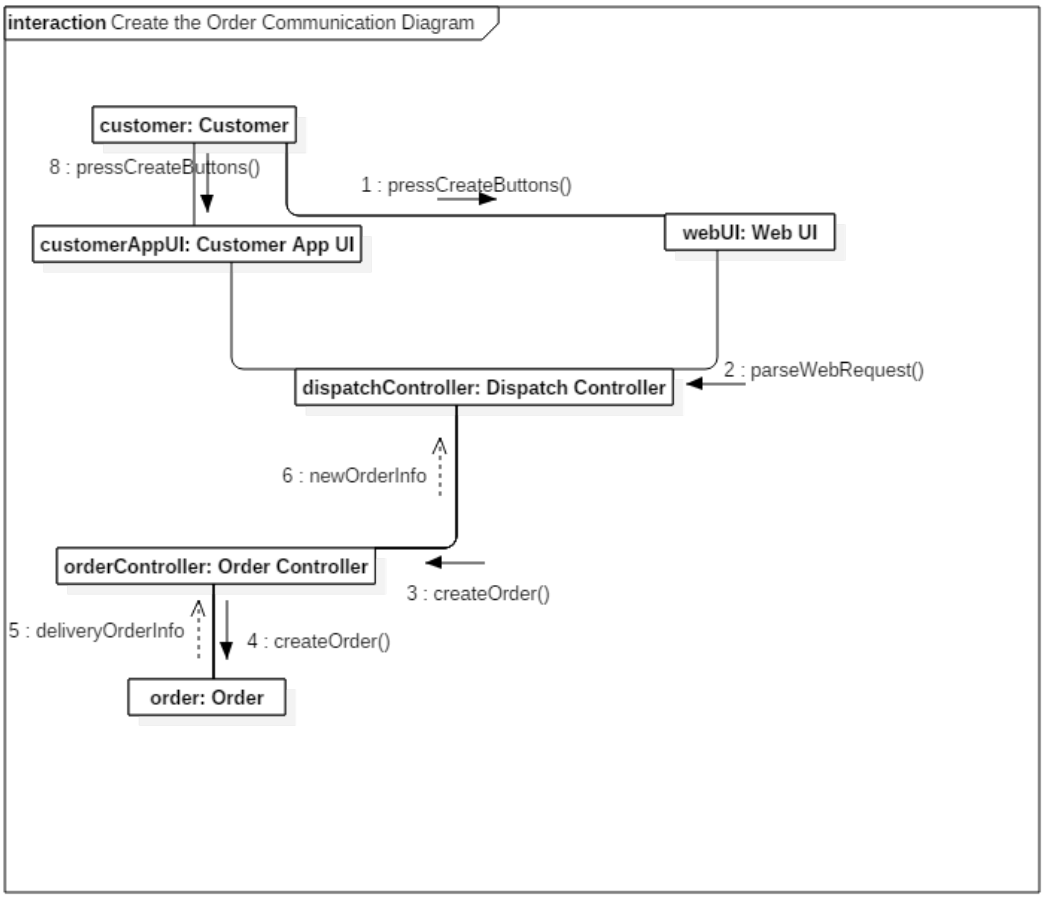
The **third part** is the boundary classes containing the UI class.
The *CustomerAppUI* provides a lot of functions for customer, which is the same with the *WebUI*.
The *PostmanUI* provides a lot of functions for postman to delivery products.

3.3 Use Case Realization

Create an Order

The customer can create the order in our express system. The customer will create the order by pressing the corresponding buttons on our website or in the App. After getting the request, those two user interfaces will send the requests to the dispatch controller. The dispatch controller will then send messages that tell the order controller to create the orders. The order controller, as seen from its name, will send the creating messages to the class order and control its creation of orders. And new orders will be created in the class Order. After creation in the Order class, the Order class will send new orders' information to the order controller. The order controller will send the information to the dispatch controller. And this dispatch controller will send back new orders' information and the system's response will be shown to the use interfaces.

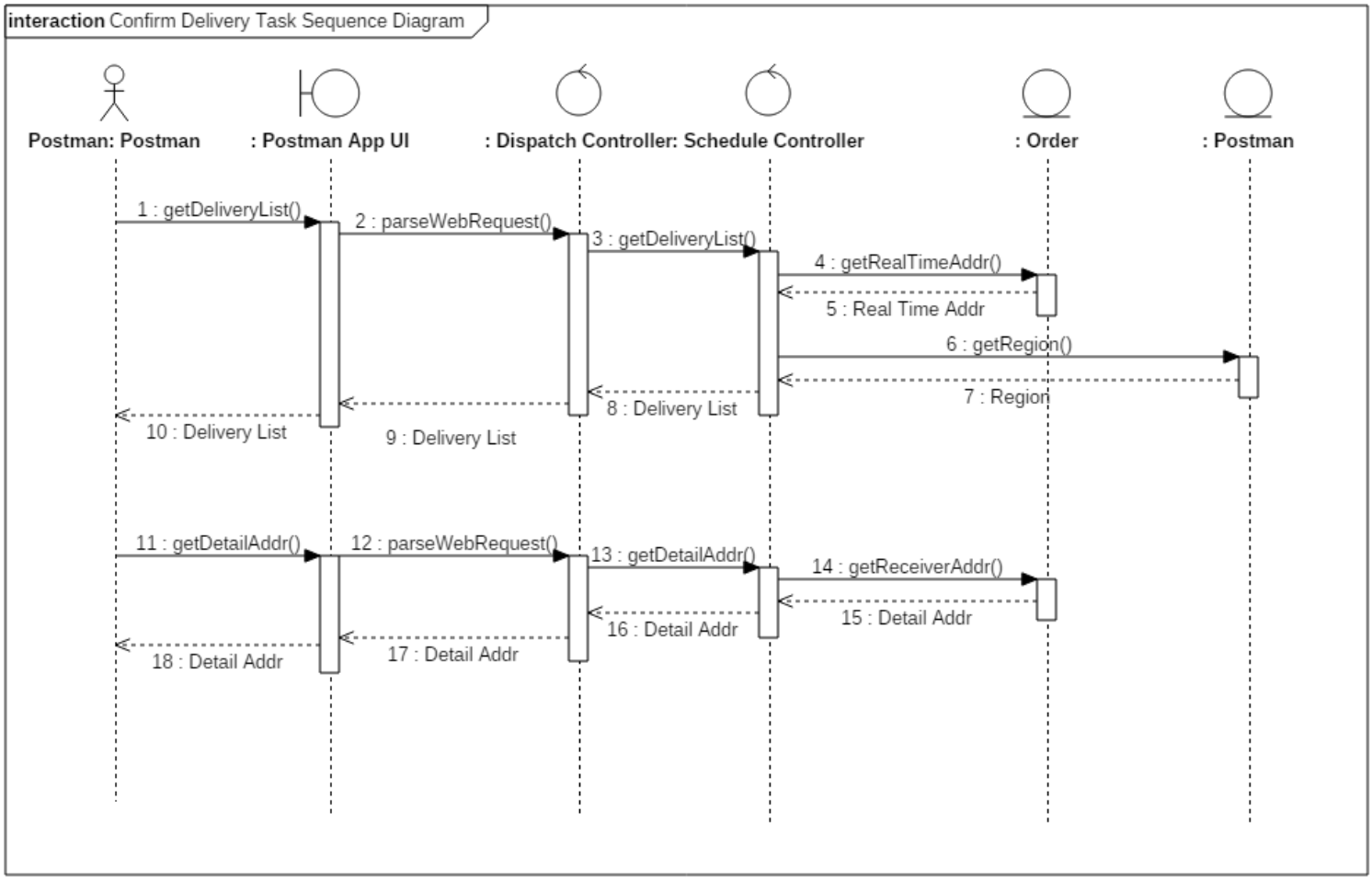


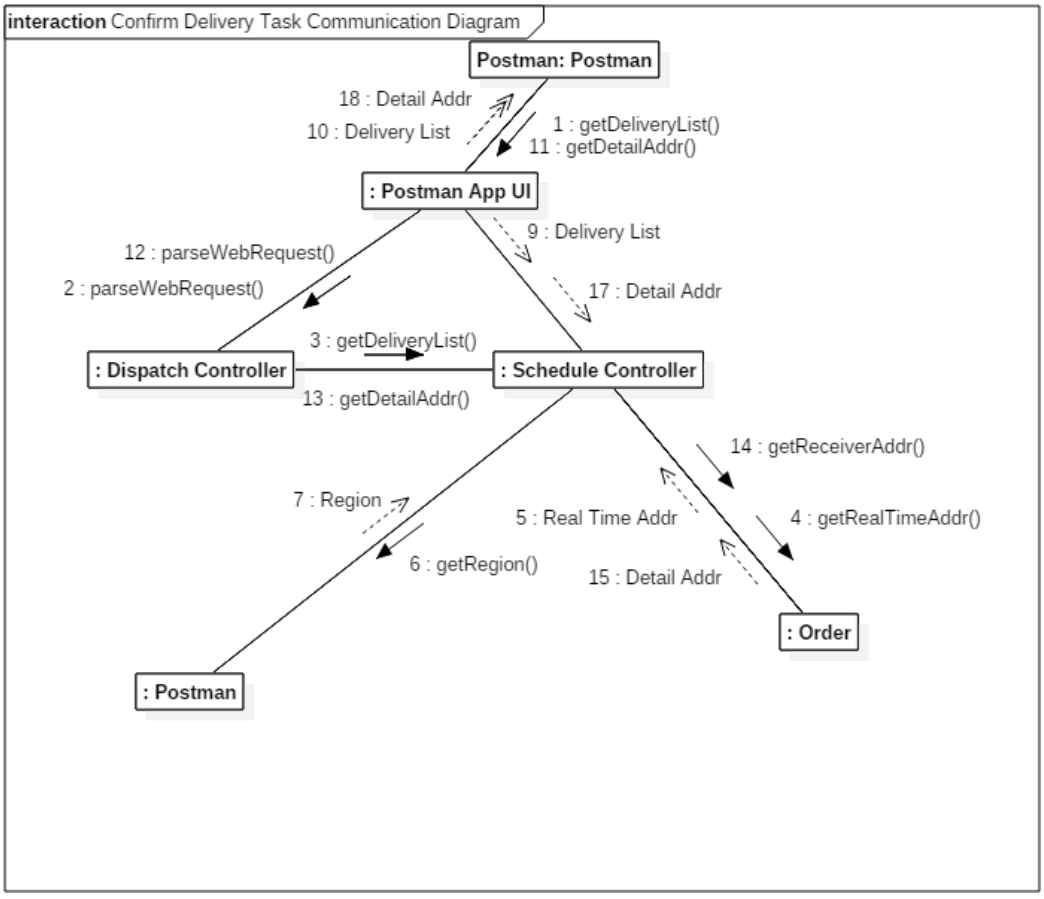


Confirm Delivery Task

First, postman can make a request of the delivery list from the Postman UI, and the UI will call the API implemented by dispatch controller. The dispatch controller checks the request and sends the request to Schedule controller. The Schedule controller gets the postman's regions and returns all the orders whose real-time address is included in the region. The order list will return to the dispatch controller and return to the postman UI.

Second, postman can make a request of the detail address for one order. The UI calls the Dispatch controller's API. The dispatch controller checks the request and sends it to schedule controller. The schedule controller gets the detailed address of the order and returns the address info to dispatch controller. The dispatch controller returns the info to postman UI.

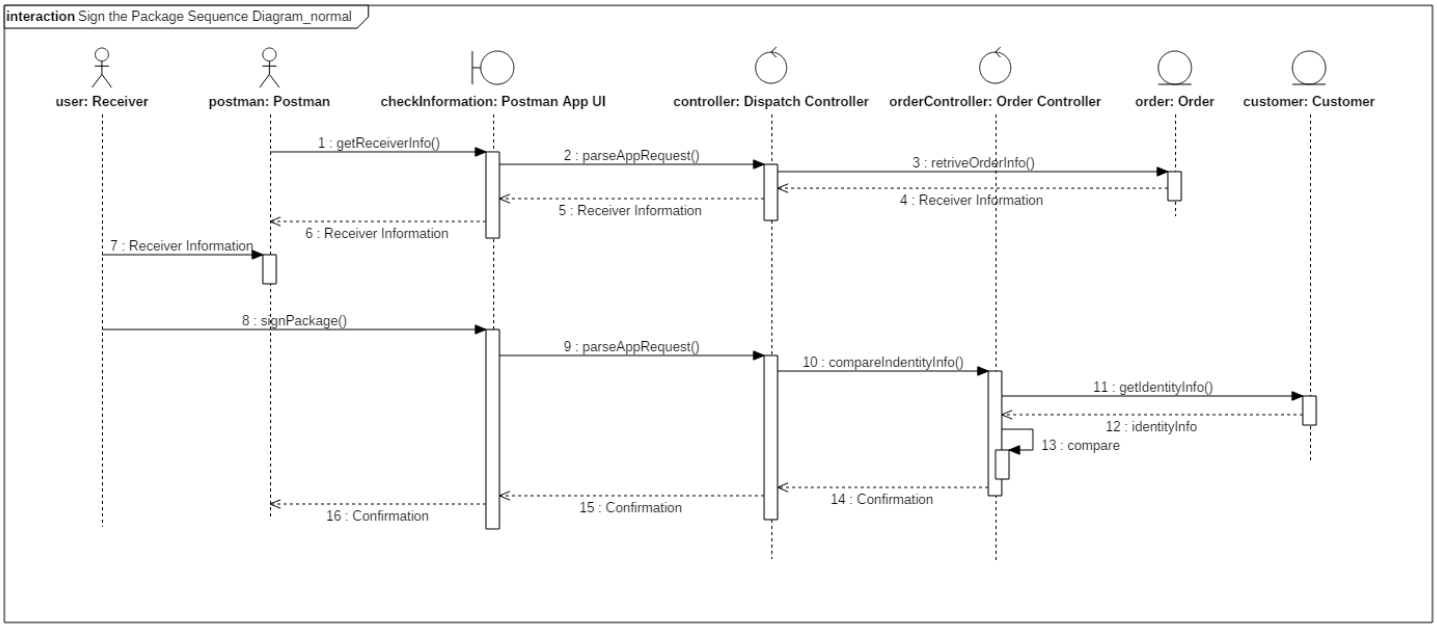

interaction Confirm Delivery Task Sequence Diagram
Postman: Postman
: Postman App UI
: Dispatch Controller: Schedule Controller
: Order
: Postman
1 : getDeliveryList()
2 : parseWebRequest()
3 : getDeliveryList()
4 : getRealTimeAddr()
5 : Real Time Addr
6 : getRegion()
7 : Region
8 : Delivery List
9 : Delivery List
10 : Delivery List
11 : getDetailAddr()
12 : parseWebRequest()
13 : getDetailAddr()
14 : getReceiverAddr()
15 : Detail Addr
16 : Detail Addr
17 : Detail Addr
18 : Detail Addr

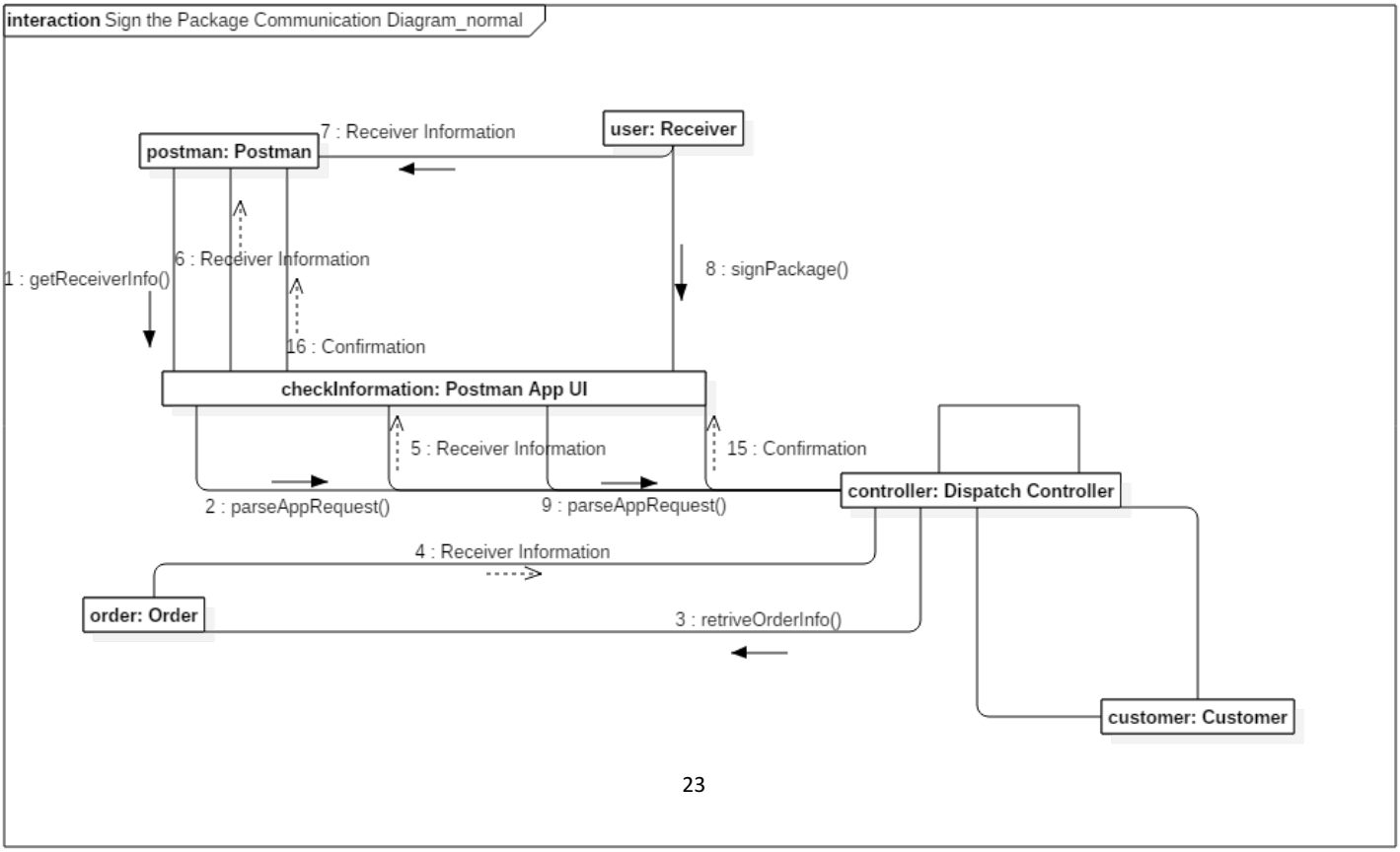

interaction Confirm Delivery Task Communication Diagram
Postman: Postman
18 : Detail Addr
10 : Delivery List
1 : getDeliveryList()
11 : getDetailAddr()
: Postman App UI
9 : Delivery List
12 : parseWebRequest()
2 : parseWebRequest()
17 : Detail Addr
3 : getDeliveryList()
: Dispatch Controller
: Schedule Controller
13 : getDetailAddr()
14 : getReceiverAddr()
7 : Region
5 : Real Time Addr
4 : getRealTimeAddr()
6 : getRegion()
15 : Detail Addr
: Order
: Postman

Sign the Package

We abstract the main sequence of signing the package. In the first part, the postman requires the information of the receiver from the system by Postman App UI. The Dispatch Controller parses the request and transfers it to Order entity, then the Order returns information including telephone numbers and so on. Notice that if the receiver is an agent, the confirmation will also be returned.

After postmen check the information and confirm the particular receiver, the receiver is supposed to sign the package through Postman App UI by using the signature or the fingerprint. The Order Controller will compare it with that previous stored in Customer domain.

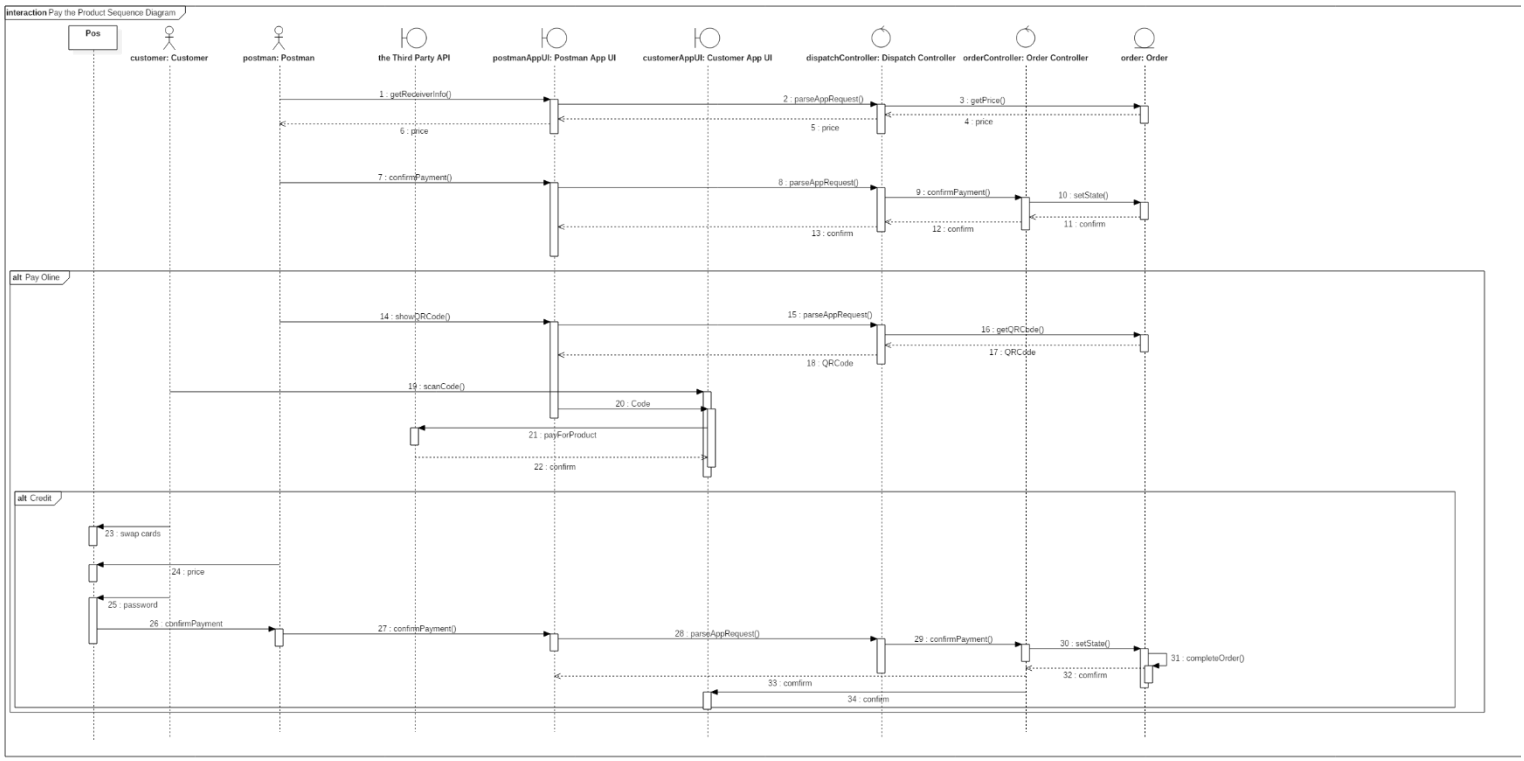


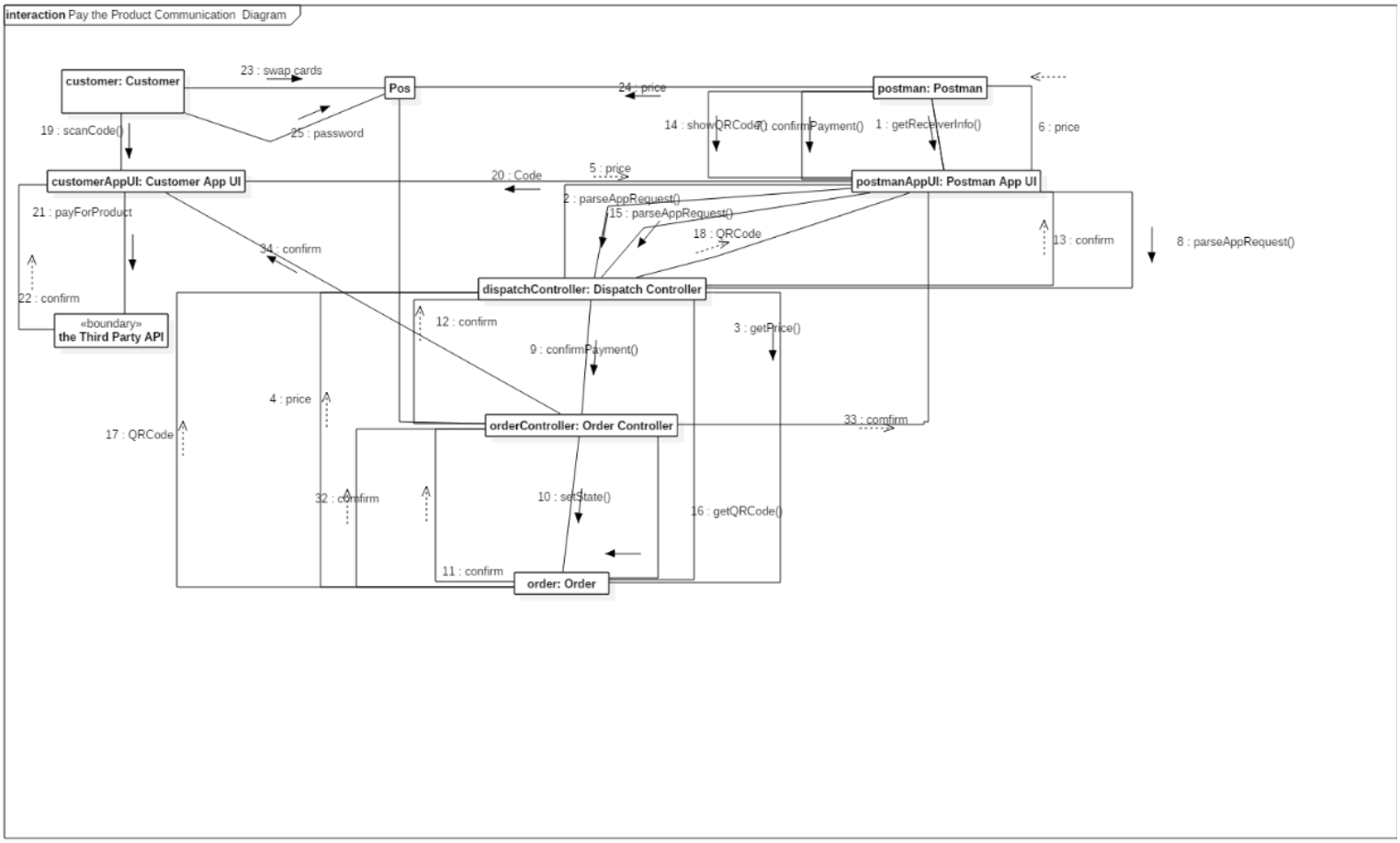


Pay for the Product

After the product is sent to receivers and if it has not been paid, receivers should pay for it. First, the postman presses the button on the App for postman to check how much money receivers should pay. App then will send the request message to dispatch controller in the way of url link. Dispatch controller will parse the URL link and use the function of order entity to get the price and after that, dispatch controller will send it back to App. After receiving the cash, postman will press the confirm button on the App to confirm the payment, the operation message will be sent to dispatch controller, too. Dispatch controller will invoke the method of order controller to change the status of order entity.

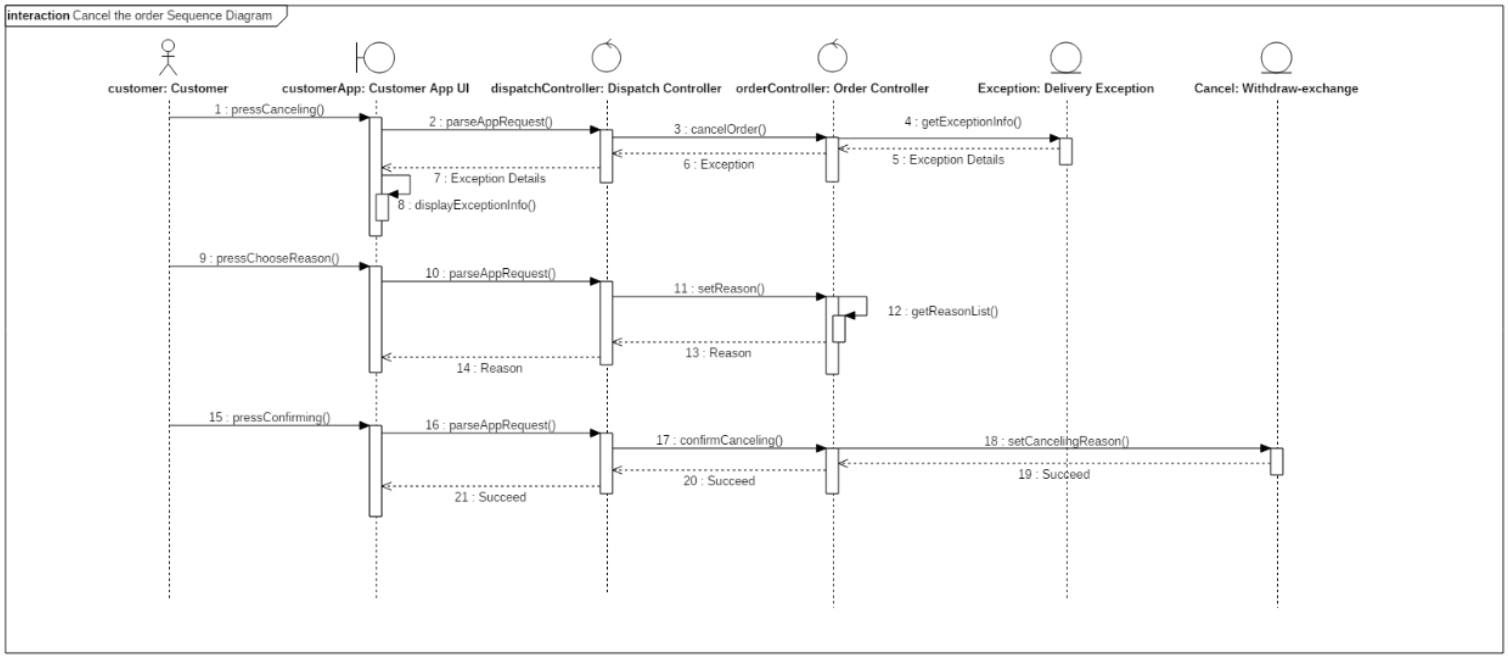
There is an alternative flow in the case that receivers pay for the order online or by credit cards. If receivers decide to pay online, postman will press the button on the App to get the QRCode. The dispatch controller will get QRCode and send it to App UI for postmen, which is the same way with getting the price of the products. Then receivers will scan QRCode showed on postmen's App with customers' App. Customer's App will invoke the Third Party API to finish the payment. If receivers want to pay by credit cards, postman will supply the POS device, after swapping the card, inputting the price and the passwords, POS device will show the confirm message on the screen, and postman will confirm the payment just as the method of paying by cash.

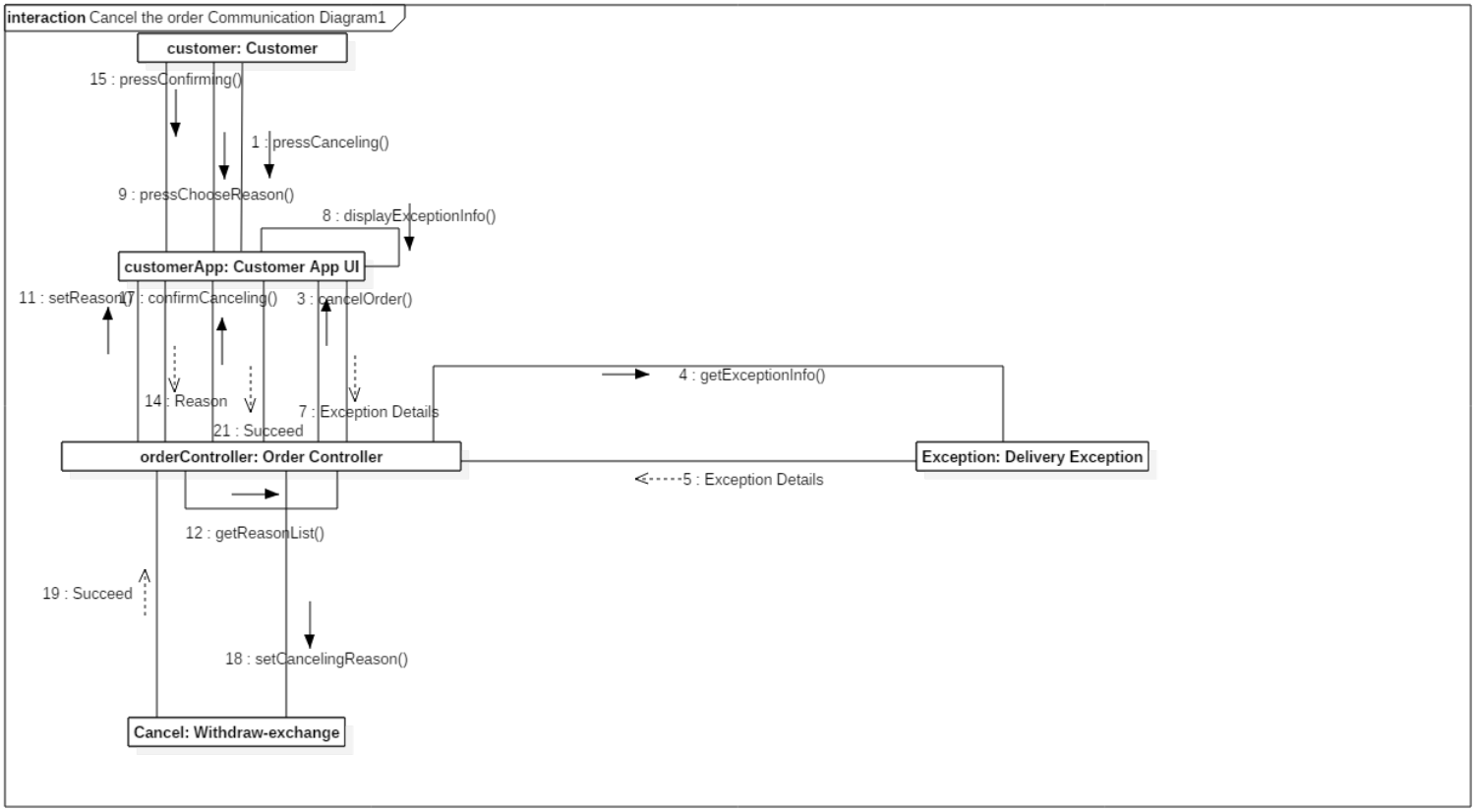




Cancel the Order

After being informed that the package is damaged or lost, the customer can choose to cancel the order. He/she needs to press canceling button to enter Verify-Order app UI to check the exception details and he/she must press choosing-reason button to continue. Finally, the customer needs to choose reasons and confirms the canceling. These three steps all need UI to use parseAppRequest function to tell dispatch controller to call different functions of Order Controller. And Dispatch Controller will send back exception details, reasons and success message individually to customers.

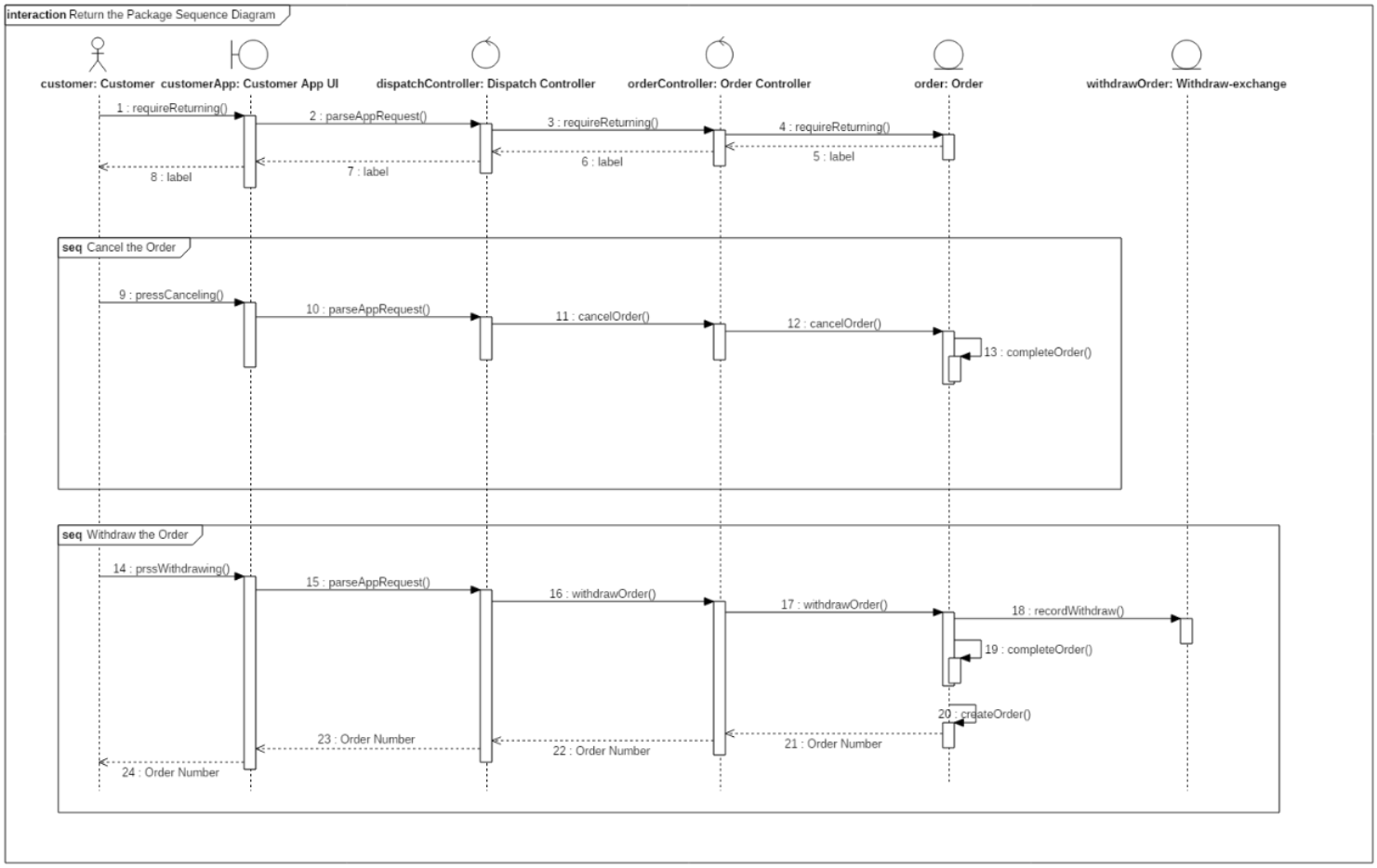

interaction Cancel the order Sequence Diagram
customer: Customer
customerApp: Customer App UI
dispatchController: Dispatch Controller
orderController: Order Controller
Exception: Delivery Exception
Cancel: Withdraw-exchange
1 : pressCanceling()
2 : parseAppRequest()
3 : cancelOrder()
4 : getExceptionInfo()
5 : Exception Details
6 : Exception
7 : Exception Details
8 : displayExceptionInfo()
9 : pressChooseReason()
10 : parseAppRequest()
11 : setReason()
12 : getReasonList()
13 : Reason
14 : Reason
15 : pressConfirming()
16 : parseAppRequest()
17 : confirmCanceling()
18 : setCancelingReason()
19 : Succeed
20 : Succeed
21 : Succeed

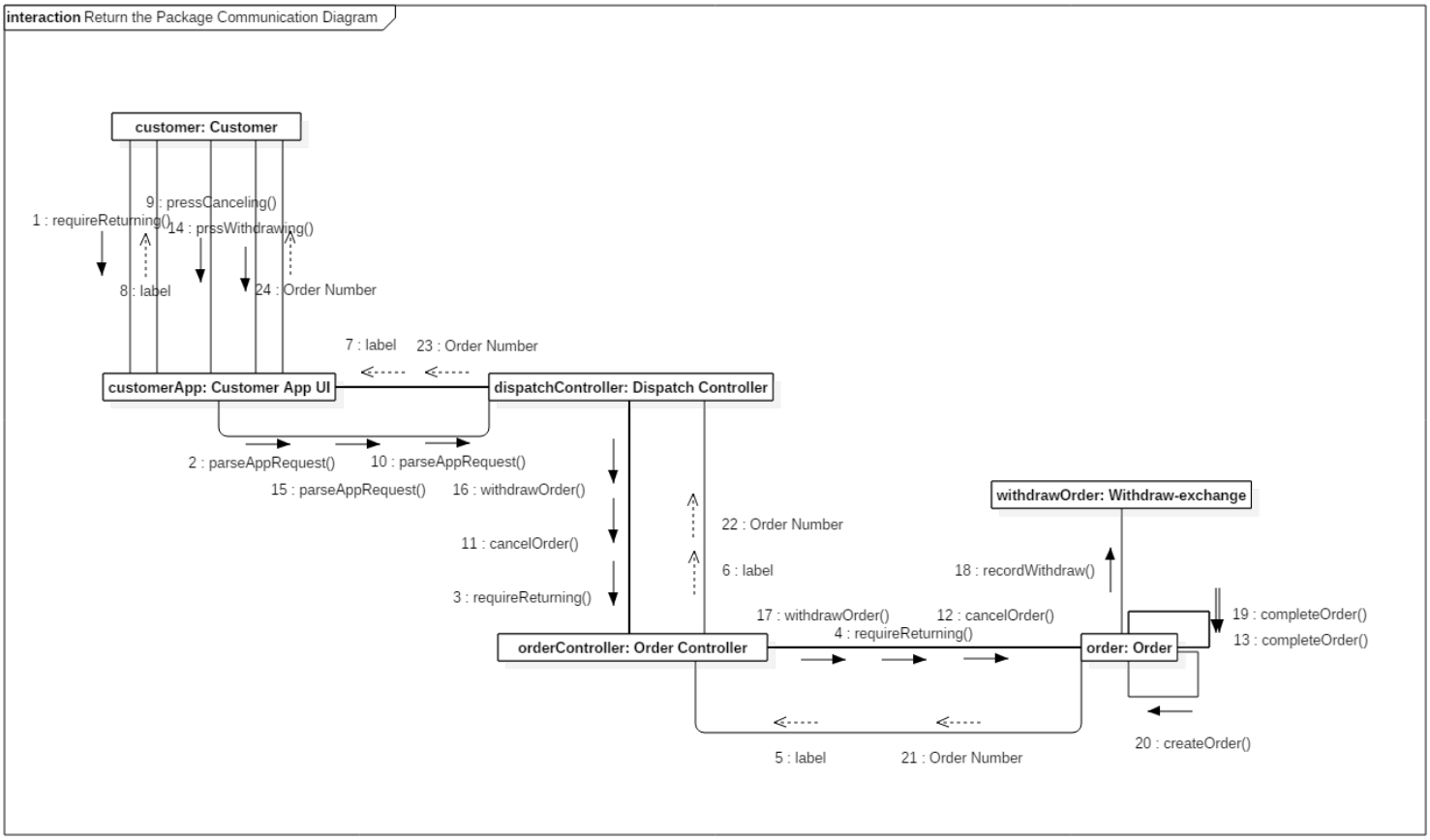

interaction Cancel the order Communication Diagram1
customer: Customer
15 : pressConfirming()
1 : pressCanceling()
9 : pressChooseReason()
8 : displayExceptionInfo()
customerApp: Customer App UI
11 : setReason()
17 : confirmCanceling()
3 : cancelOrder()
14 : Reason
21 : Succeed
7 : Exception Details
4 : getExceptionInfo()
orderController: Order Controller
Exception: Delivery Exception
5 : Exception Details
12 : getReasonList()
19 : Succeed
18 : setCancelingReason()
Cancel: Withdraw-exchange

Return the Product

If the customer is unsatisfied with the product, he/she can return the product. Customers need to apply for returning through Customer App UI. Then Dispatch recognizes that request and assigns it to Order Controller so Order Domain will return a label containing the address to send it back and the postage.

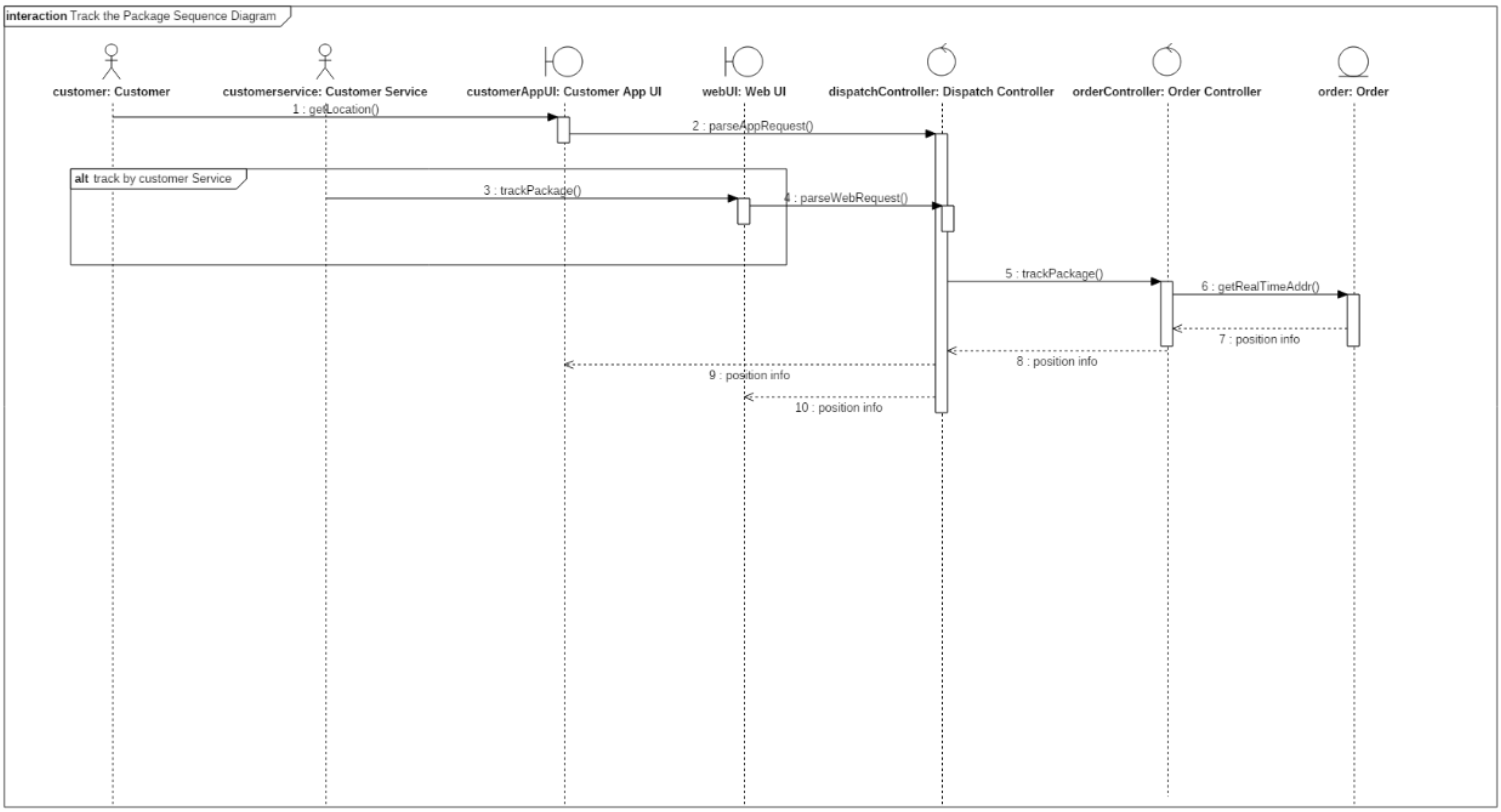
If the customer chooses to cancel the order, after a similar process, the order will be completed. Else if he/she chooses to withdraw the order, the original order will be recorded in the Withdraw-exchange Domain and then be completed. Afterwards, a new order with the same information as the original one is created.

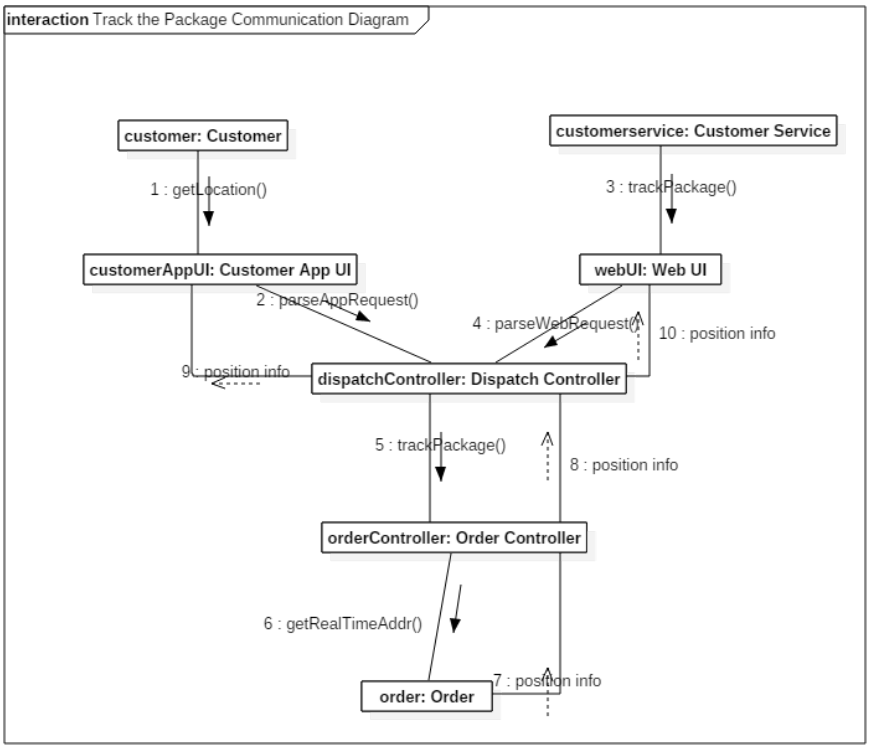




Track the Package

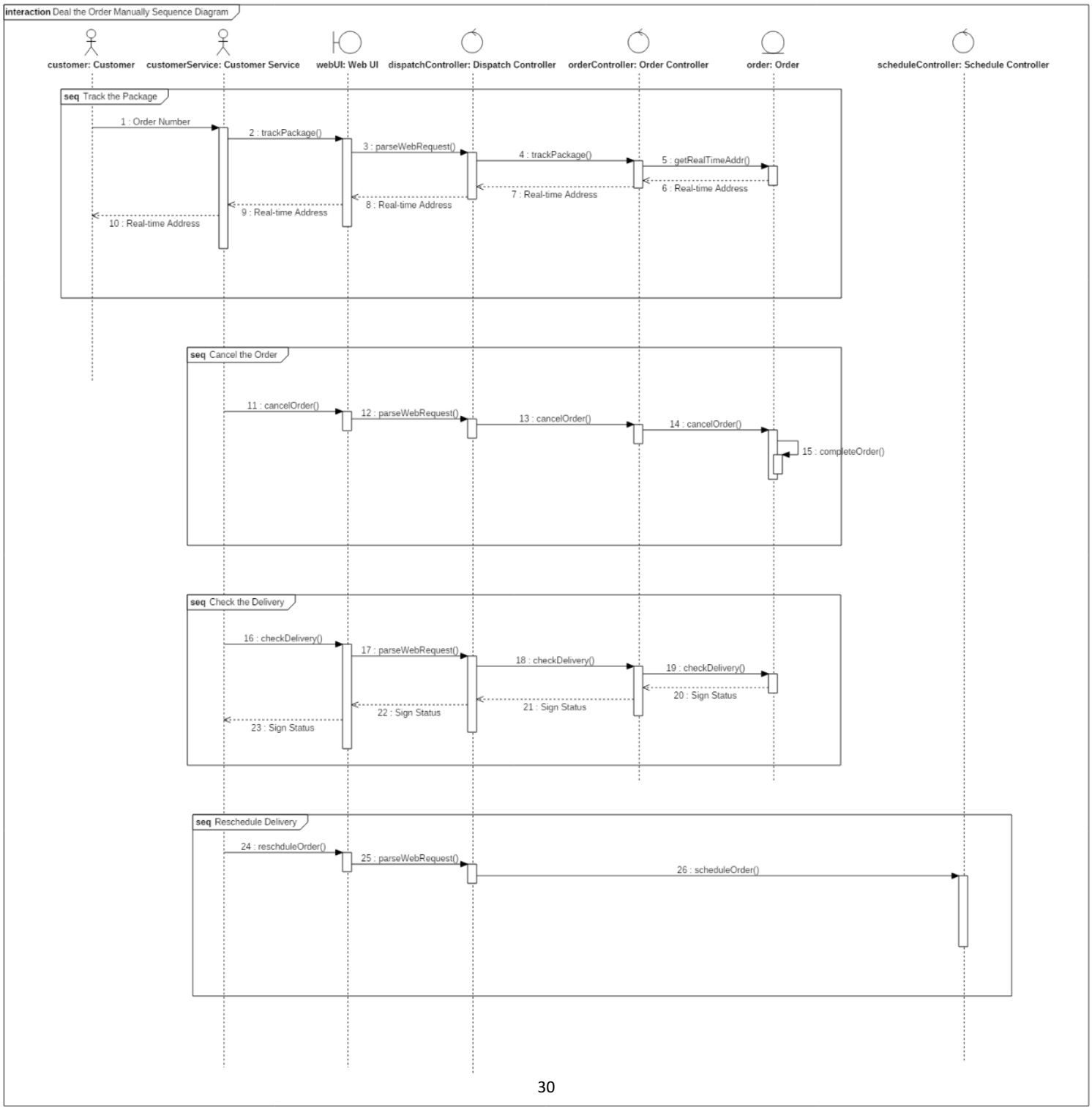
Customers and Customer Service have the right to track the product. Customer will invoke the function on App for customers, and customer service will track the package on web. Both messages will be sent to dispatch controller in URL link. After parsing the link, dispatch controller invokes the getRealTimeAddr function of order controller. Order controller will then get the location by invoking the function of order entity. After getting the real time address, the system will return this information to the client who invokes the function.

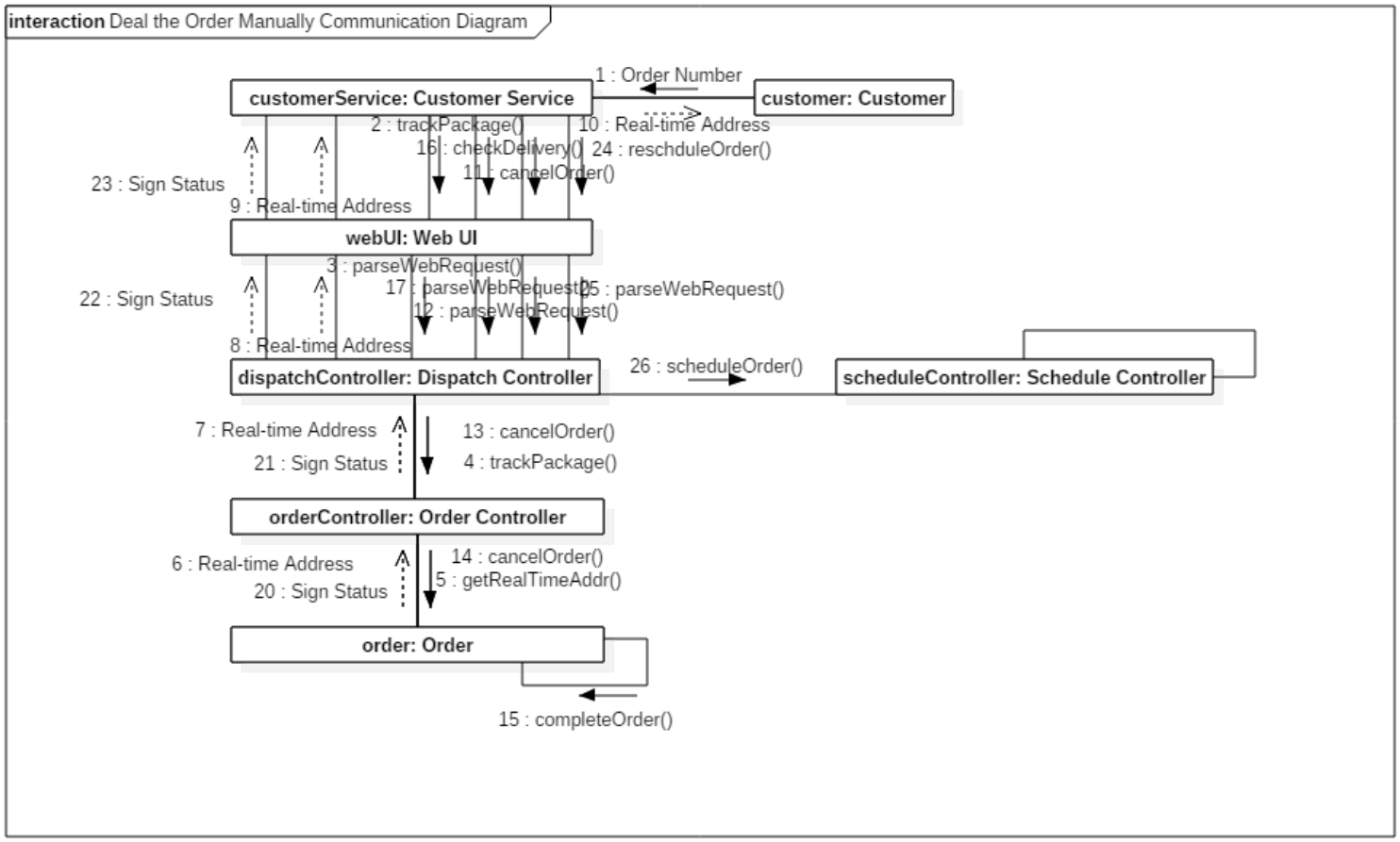

interaction Track the Package Sequence Diagram
customer: Customer
customerservice: Customer Service
customerAppUI: Customer App UI
webUI: Web UI
dispatchController: Dispatch Controller
orderController: Order Controller
order: Order
1 : getLocation()
2 : parseAppRequest()
alt track by customer Service
3 : trackPackage()
4 : parseWebRequest()
5 : trackPackage()
6 : getRealTimeAddr()
7 : position info
8 : position info
9 : position info
10 : position info


interaction Track the Package Communication Diagram
customer: Customer
customerservice: Customer Service
1 : getLocation()
3 : trackPackage()
customerAppUI: Customer App UI
webUI: Web UI
2 : parseAppRequest()
4 : parseWebRequest()
10 : position info
9 : position info
dispatchController: Dispatch Controller
5 : trackPackage()
8 : position info
orderController: Order Controller
6 : getRealTimeAddr()
7 : position info
order: Order

Deal the Order Manually

There might be tasks that system cannot deal with automatically. Furthermore, system itself may make mistakes. In that case, customer service is supposed to take the responsibility of dealing the request from customers to track the package, canceling the order with permission in some special situations, making up for system's omitting of the signed mark and rescheduling the delivery if the receiver is not available. The first and the third situation have returned value and in the second condition the order will be ended automatically. For the last situation, note that the schedule will be made by Schedule Controller.




interaction Deal the Order Manually Communication Diagram
customerService: Customer Service
customer: Customer
webUI: Web UI
dispatchController: Dispatch Controller
scheduleController: Schedule Controller
orderController: Order Controller
order: Order
1 : Order Number
2 : trackPackage()
10 : Real-time Address
16 : checkDelivery()
24 : reschduleOrder()
11 : cancelOrder()
23 : Sign Status
9 : Real-time Address
3 : parseWebRequest()
17 : parseWebRequest()
25 : parseWebRequest()
12 : parseWebRequest()
22 : Sign Status
8 : Real-time Address
26 : scheduleOrder()
7 : Real-time Address
13 : cancelOrder()
21 : Sign Status
4 : trackPackage()
6 : Real-time Address
14 : cancelOrder()
20 : Sign Status
5 : getRealTimeAddr()
15 : completeOrder()

4 Design

4.1 Subsystems

4.1.1 List of Interfaces

Login Subsystem

This subsystem is designed for offering interfaces to controller for all kinds of users, including login and logout.
The corresponding interfaces are as follows.
•String Login(String identify, String Password)
This interface is designed for analyzing the identify (eg. Tel number/Job number/e-mail/...) information and check whether the password is right. If it is right, the return value will be a sessionid. Otherwise the return value will be null.
•Boolean Logout(String sessionid)
First of all, users are able to logout manually or automatically. By get the parameter "sessionid", this interface will change the status of the user. The return value indicates whether users have logged out successfully or not.

OrderCreater Subsystem

This subsystem offers interfaces to controller for customers and e-business companies to create orders.
The corresponding interfaces are as follows.
•Boolean createOrder(ExternalOrder orderinfo)
This interface is used to create orders from customers. It will receive the external orders' information in the format of what is specified in the class "ExternalOrder". The change and updates of corresponding class User Profile and Order will be related with the implementation of this interface. The return value indicates whether the creating process is successful.
External Order Information format:Pair: { int key, string value}
•ArrayList<OrderId> createOrder(E-BType type, ArrayList<ExternalOrder> orderlist)
This interface is used to analysis order-request data according to types of e-business companies and create orders for them. The return value is an ArrayList containing pairs of order numbers from e-business companies and their corresponding order_ids in our system.
In the implementation of this interface, it will invoke createOrder(ExternalOrder orderinfo) to finish creating a single order.
The E-Business Type and its order list will be stored in a basic class, named as E-BOrder.

Payment Subsystem

This subsystem offers interfaces to controller for customers and postmen to finish payment(including package and freight).
The corresponding interfaces are as follows.

•Boolean payEB(Array<Order_Type> order_list, double amount)

E-business companies pay the order through the backend and then invoke this function. Our system will check the amount. If the system makes sure that the amount is right and our account has received the money, the function will return true and amend the state of these orders. Otherwise it will return false.

•Boolean payIndiCash(String order_id, double amount)

This interface receive the order_id and the amount of the order as parameters and return true or false. This is used in the scenario that the receivers pay by cash and postmen check the payment.

•Boolean payIndiOnline(String order_id)

This interface is used in the scenario that receivers pay online(eg. Alipay) and Alipay will check the payment. The third-party payment system will send the result of payment to our system. And the corresponding information will be changed in our system.

•Boolean payIndiCard(String order_id)

This interface is used in the scenario that receivers pay by credit card and bank systems will check the payment. After that, bank system will send the result of payment to use, which determines the return value.

•Invoice provideInvoice(String order_id,Invoice_Type type)

If customers ask for a invoice, system will return a struct called Invoice according to the type, required by users. The Invoice is a user-defined data type.

EWEHandler Subsystem

This subsystem is used to handle the exceptions during the delivery process and withdrawal ore exchange request from the customer.
This subsystem offers interfaces to controller for postmen and customer service to deal with the exception (eg. damage/lossing/... of package) and the returning.
The corresponding interfaces are as follows.

•Boolean sendException(String ordernumber, Package_Status status, String reason)

This interface is used to handle the exception and send the message to the system. This interface receives the 3 parameters and will change the status of the package in our system. The Package_Status marks what happened to the package by a num.

•Boolean sendWithdrawInfo(String ordernumber, String reason)

By sending the ordernumber and the withdrawal reason to this function, the corresponding information will be changed in the system.The return value marks whether the message has been sent successfully.

•Boolean sendExchangeInfo(String ordernumber, String reason)

This interface is used to handle the exchange requests and send the message to the system. The return value marks whether the message has been sent successfully.

•Boolean cancelTheOrder(String ordernumber)

After receiving the order number, the order status will be changed in our system. The

return value marks whether the canceling process successes.

Message Subsystem

This subsystem offers interfaces to controller to sends different types of messages.
The message type are defined as receive,exception,application...
The corresponding interfaces are as follows.

•void receiveMessage(Message_Type type,String user_id, String content)

This interface will receive the message and analyze the content of the message by the message type and the user id.

•Boolean sendMessage(String user_id,String content)

In this interface, it will organize and encode the content of the message and send it by the user id. The return value marks whether the sending successes.

RoutePlanner Subsystem

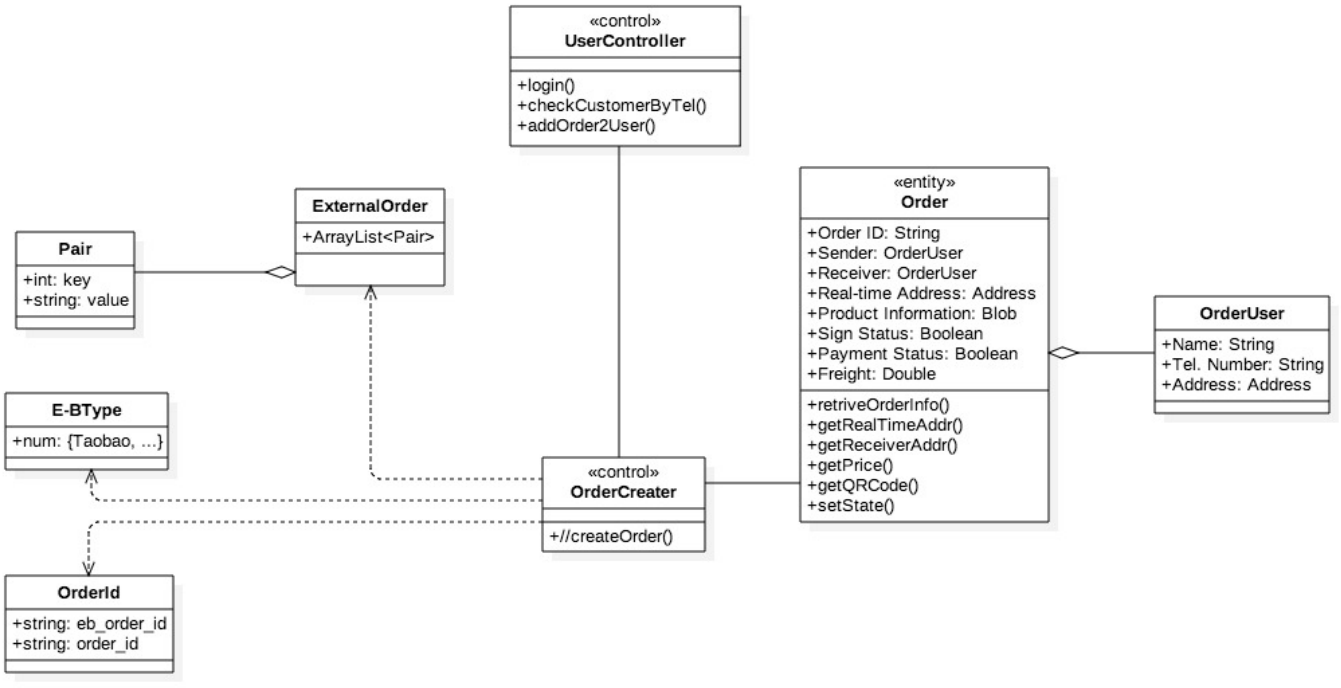
This subsystem offers interfaces to controller to plan the delivery routes.
The corresponding interfaces are as follows.

•Route planRoute(String orderId)

The controller will receive the order id and the sender and the receiver's addresses will be found in the corresponding class Order. By calling this interface, the controller will return a Route type value, containing the information of the recommended route.

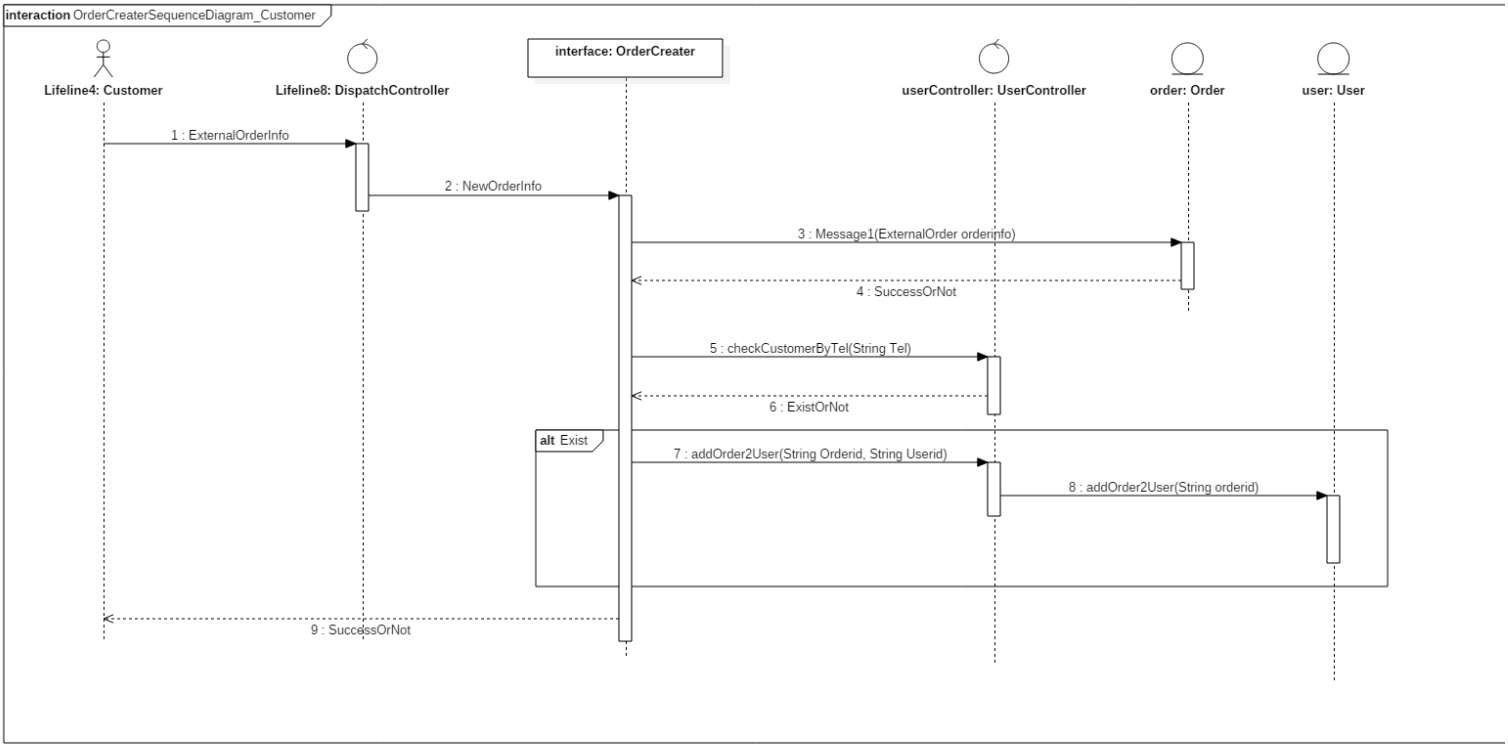
4.1.2 One Detailed Subsystem: orderCreater

This subsystem offers interfaces to controller for customers and e-business companies to create orders.

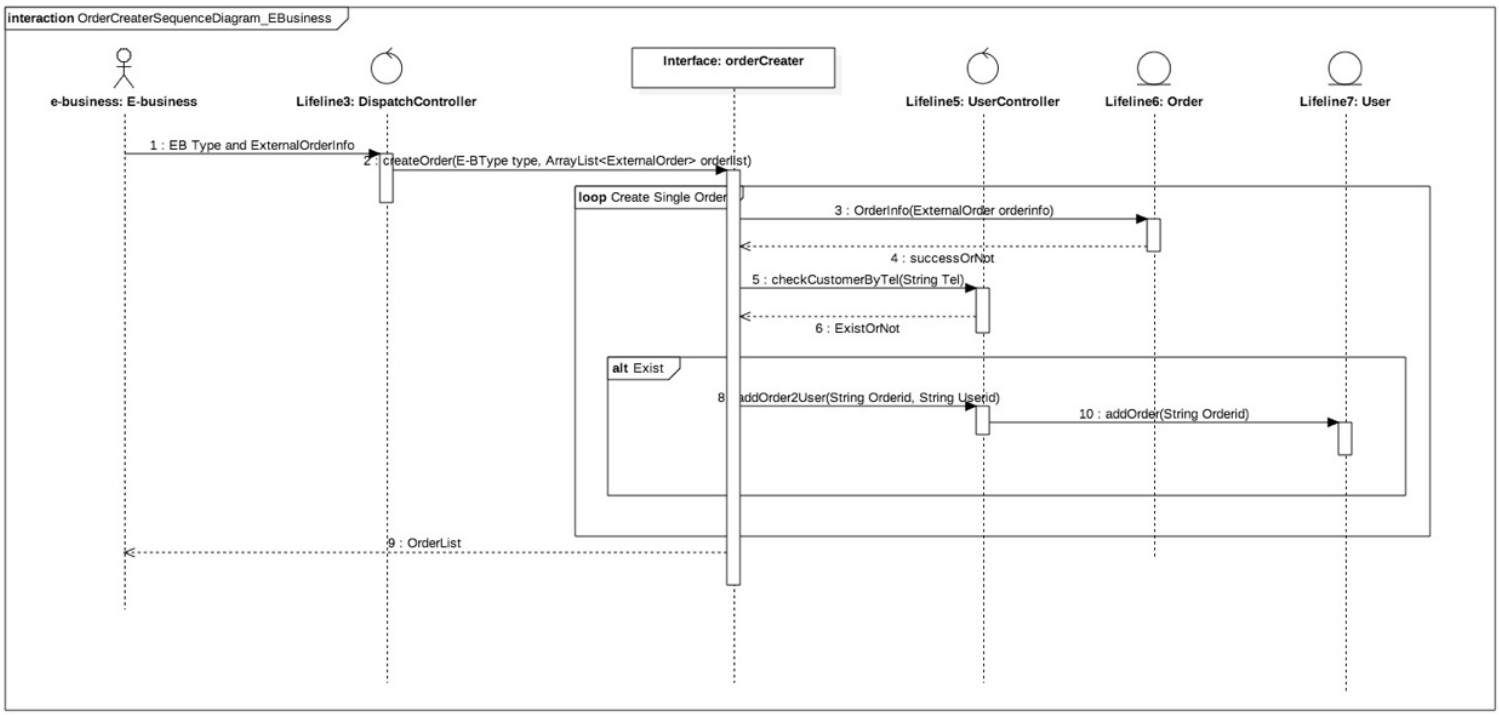


The **class diagram** of this subsystem is showed above. Our system will get the external

order information in the format of Pair from customers and E-business type if it creates orders for E-business. By invoking the interface in “Create the Order” class, the system will return the order ids for both customers and E-business. During the process, it will invoke the interfaces provided by the class User Controller and Order. The related interfaces are showed in the class diagram.



This **sequence diagram** shows the process of creating the order for the customer. After getting the request from the customer, the system will check if the customer already exists in our system. If the customer does exist, the system will add the external order information provided by the customer into our system. And return the result to the customer.



This **sequence diagram** shows the process of creating order list for the E-business. In this subsystem, it will create the order list by creating one single order each time. Until the loop of creating a single order finishes, it will return the whole order list back to the E-business. For creating a single order, the process is the same as creating the order for the customer, as showed above. By checking the existence of the customer, the system will add the new order's information of the customer.

The **interfaces** we have designed are showed as follows.

•Boolean createOrder(ExternalOrder orderinfo)

This interface is used to create orders from customers. It will receive the external orders' information in the format of what is specified in the class "ExternalOrder". The change and updates of corresponding class User Profile and Order will be related with the implementation of this interface. The return value indicates whether the creating process is successful.

External Order Information format:Pair: { int key, string value}

•ArrayList<OrderId> createOrder(E-BType type, ArrayList<ExternalOrder> orderlist)

This interface is used to analysis order-request data according to types of e-business companies and create orders for them. The return value is an ArrayList containing pairs of order numbers from e-business companies and their corresponding order_ids in our system.

In the implementation of this interface, it will invoke createOrder(ExternalOrder orderinfo) to finish creating a single order.

The E-Business Type and its order list will be stored in a basic class, named as E-BOrder.

Protocol:

Mainly we would use TCP/IP (Internet Protocol Suite) and the common four layers: link layer, transport layer, internet layer and application layer.

• Link Layer: We adopt Ethernet and particularly take MAC (Medial Access Control) model in our system.

• Transport Layer: We employ IP to ensure the transportation of data and ARP&RARP as the transformation between IP address and the physical address. In addition, considering our system is used around the regions of Jiangzhehu, we use IGMP to process the trans-regional transportation.

• Internet Layer: Because in OrderCreater, the content is orders, we decided use UDP and through the error handling to ICMP. If the transportation failed, ICMP in Transport Layer will send error message.

• Application Layer: In this layer because there is no need to assure real-time communication.Besides, it is unpractical and will cost a lot, we use HTTP/HTTPS.

Json Format of Order:

```
{
   "OrderID" : "IDNumber" ,

   "OrderState" : "State"

   "Price" : Price(double),

   "Paid"    : True / False,

   "SenderName" : "NameA",

   "SenderPhoneNum" : "PhoneNumber"

   "SenderAddress" : {
                "Country": "CountryName",
                "State"    : "StateName",
                "City"     : "CityName",
                "District" : "DistrictName",
                "Road"     : "RoadName",
                "Building" : "BuildingNumber"
                "Unit"     : "UnitNumber",
                "Room"     : "RoomNumber"
    }

   "ReceiverName" : "NameB",

   "ReceiverPhoneNum" : "PhoneNumber"

   "ReceiverAddress" : {
                "Country": "CountryName",
                "State"    : "StateName",
                "City"     : "CityName",
                "District" : "DistrictName",
                "Road"     : "RoadName",
                "Building" : "BuildingNumber"
                "Unit"     : "UnitNumber",
                "Room"     : "RoomNumber"
    }

}
```

The **created Order** Format:

Order Format

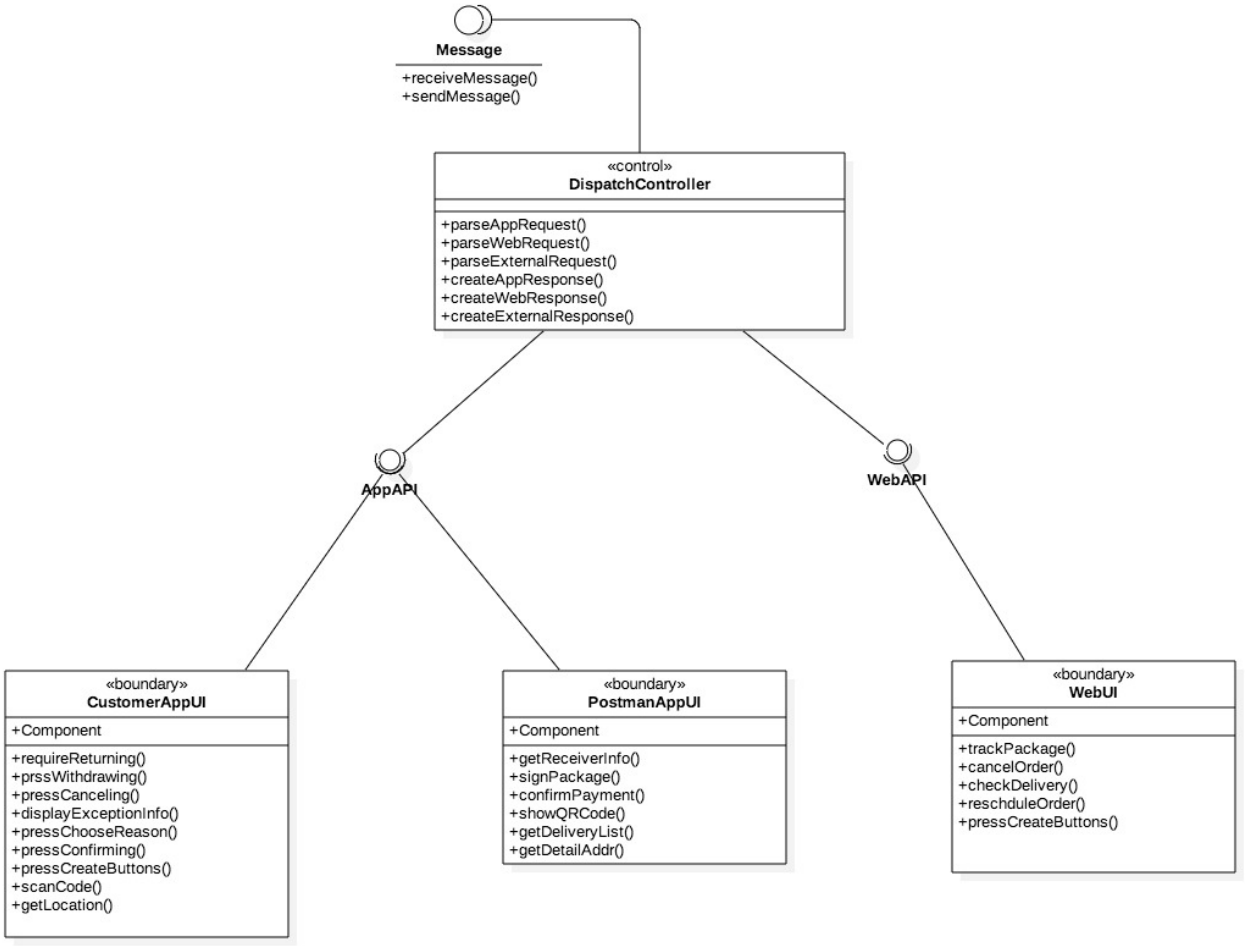
Order ID		Freight	
Sender Name		Sender Tel.	
Sender Address			
Receiver Name		Receiver Tel.	
Receiver Address			
Real-time Address			
Product Information			
Payment Status		Sign Status	

4.1.3 Interfaces Between Our System and External Systems

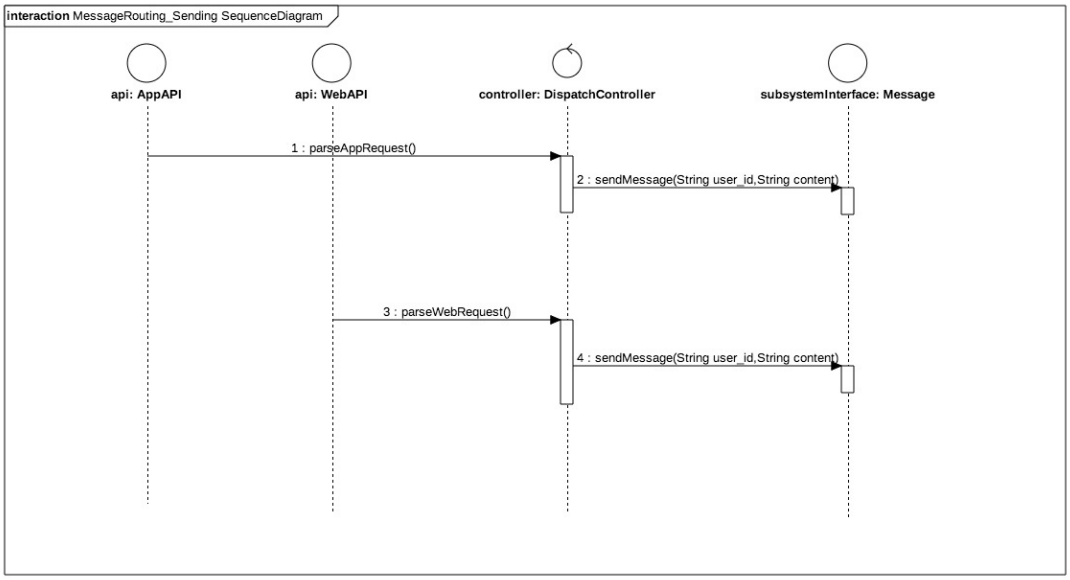
As showed above, we have designed create the order interface for communication between our system and external systems. The external systems include the E-business websites and the customer.
The Encryption class provides interfaces for the third party messaging system. These interfaces are showed in the security mechanism.

4.2 Mechanisms

4.2.1 Communication Mechanism

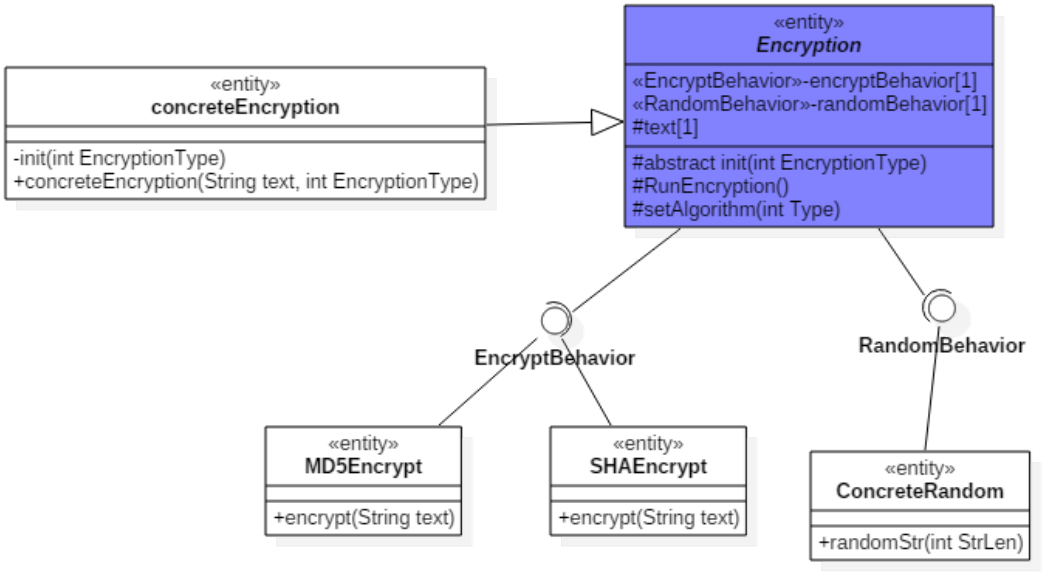


Our system provides App API and Web API for the Customer App UI, Postman App UI and Web UI. The Dispatch Controller will control these two APIs and use the interfaces provided by the Message class. The communication will be performed in the Message class by using its interfaces.



This sequence diagram is used to describe the message receiving process of the subsystem's interface. This interface will receive the messages and then the Dispatch Controller class will create responses and send back the messages to APIs.

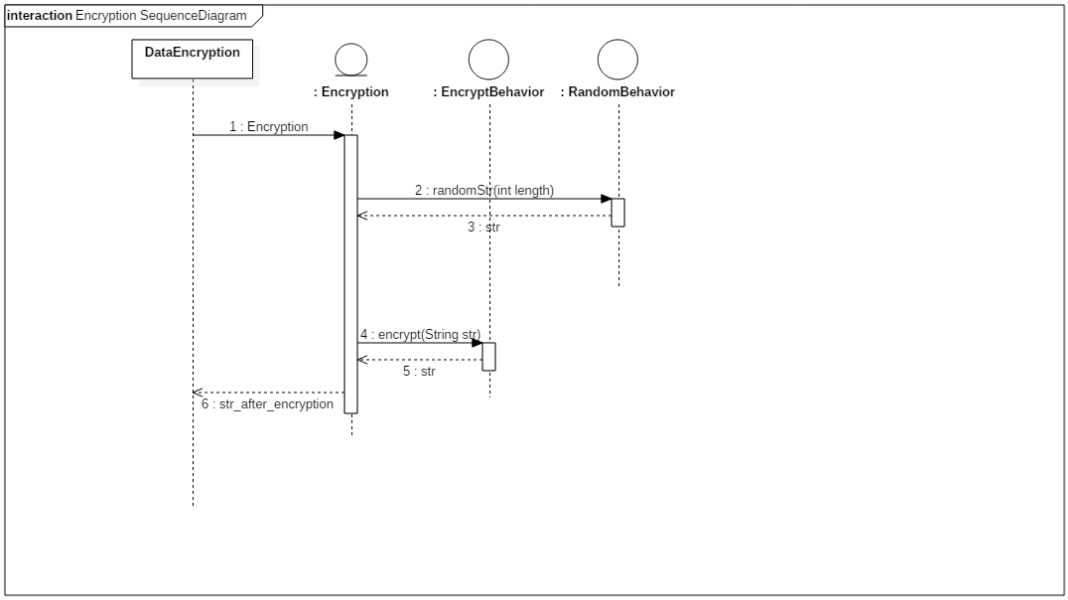
4.2.2 Security Mechanism



Our system provides 2 interfaces in this mechanism and defines two corresponding functions, including encryption and random string generation. There are two entity classes MD5Encrypt and SHAEncrypt which are designed for implementing the interface EncryptBehavior. Inheriting from the abstract class Encryption, the entity class ConcreteRandom is used to implement the RandomBehavior interface and will invoke these two interfaces.

Encryption is an abstract class, without the encryption() operation and randomStr() operation. And two attributes which are two interfaces are private. What is more is that we add a string attribute to store the content, an abstract operation init(int EncryptionType, string text) for subclass to init the text attribute and set algorithm at the beginning, a RunEncryption() opeartion to process the encryption and a setAlgorithm(int Type) operation to set the algorithm.

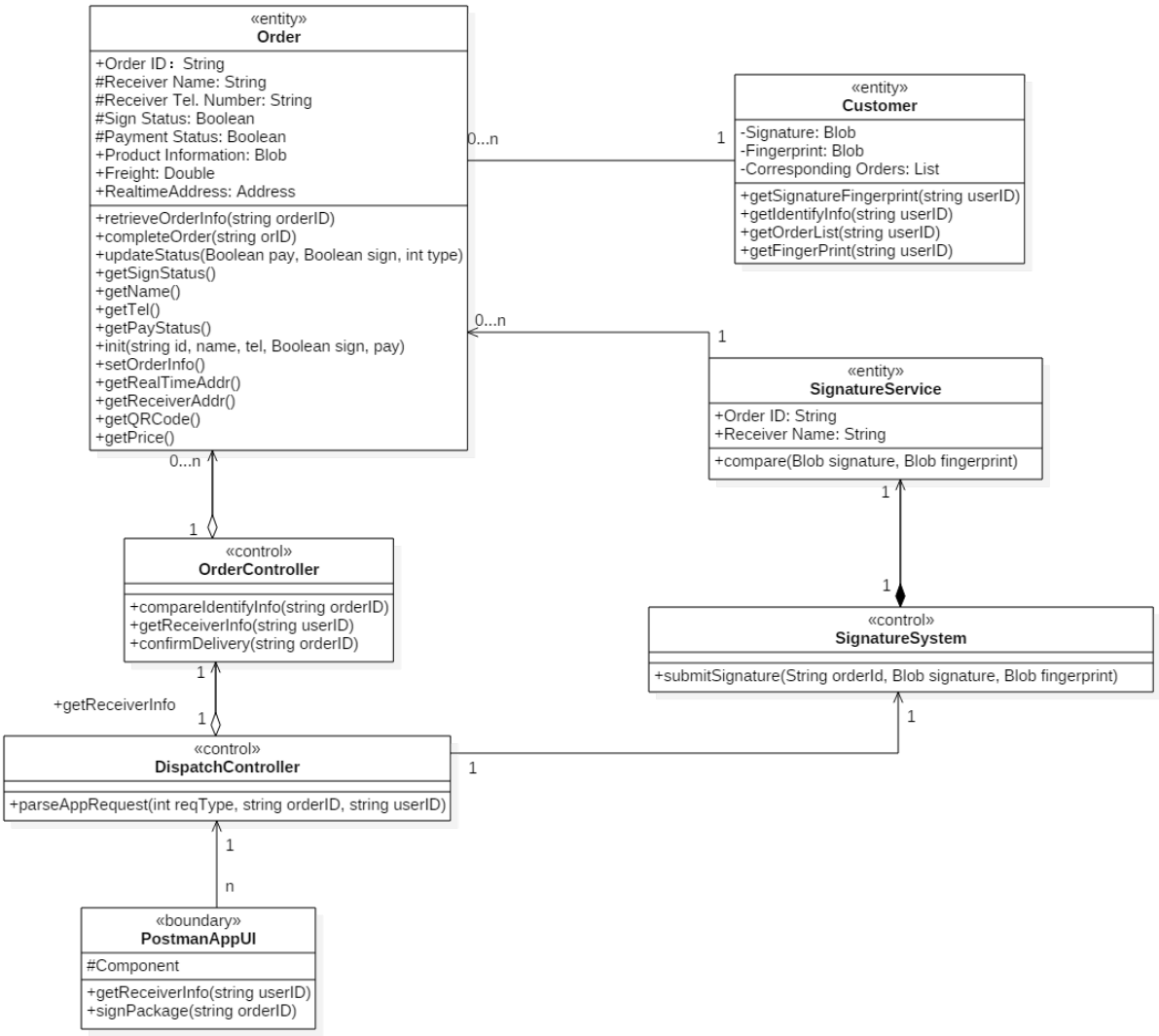
If someone want to use the encryption, they can build their own concrete class inherited the Encryption class. They need to rewrite the init(int EncryptionType, string text) operation to set the options of encryption, in which operation they can use setAlgorithm(int Type) to set the algorithm. After finishing these, they just invoke the RunEncryption() operation and get the result. So no one can know how the Encryption class process the task.



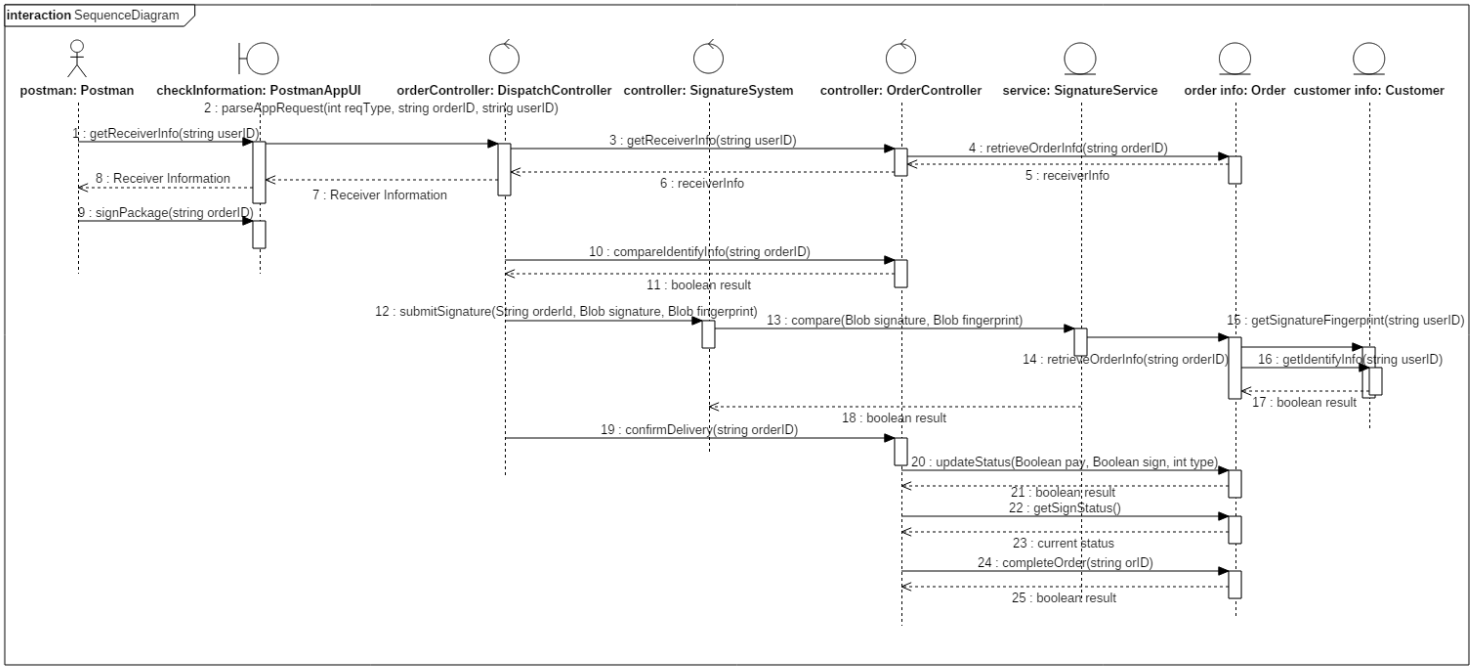
The sequence diagram of this security mechanism shows the encryption process. The system will generate a string randomly according to the length defined in the Encryption class. The encrypt() interface will put together the original password and the random string together to complete the encryption. And return the encrypted string. This encrypted string is used to identify the user's identity by comparing the input of password with what stores in our database.

4.3 Use case Realizations

4.3.1 Sign the Package

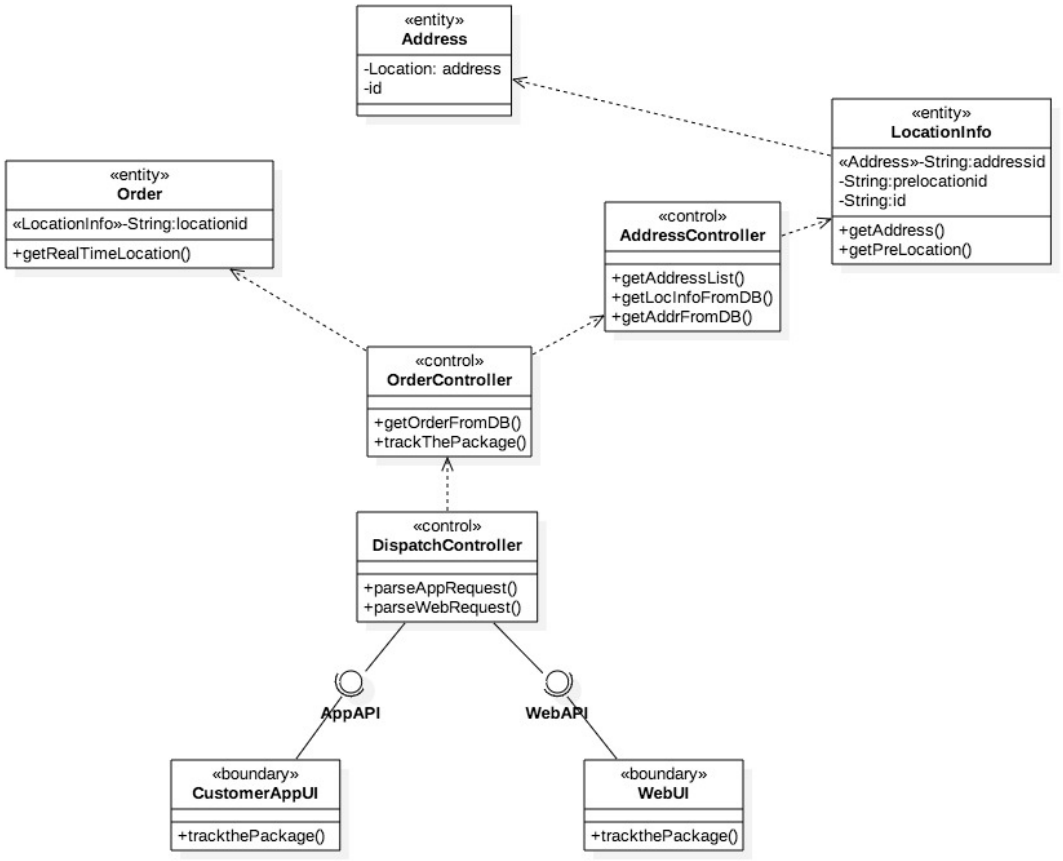


In this use case, the postman uses his mobile device to let the receiver sigh the package. Since the receiver is the generalization of the customer and the third-party agent, we add additional controller: Signature system, to control the signing way: fingerprint. After receiving and parsing the request in the Dispatch Controller from Postman App UI, it will use the Order Controller to compare the receiver information and use the Signature System to compare the signature or fingerprint with what store in the system. We have defined a series of “get” functions and corresponding parameters to implement this use case. All the attributes except the orderID are defined as protected, so that the security is guaranteed. We define the signature, fingerprint and order list attribute of the class Customer as private ones to improve security. And the aggregation as well as the relationships such as aggregation, dependency are showed in the class diagram.

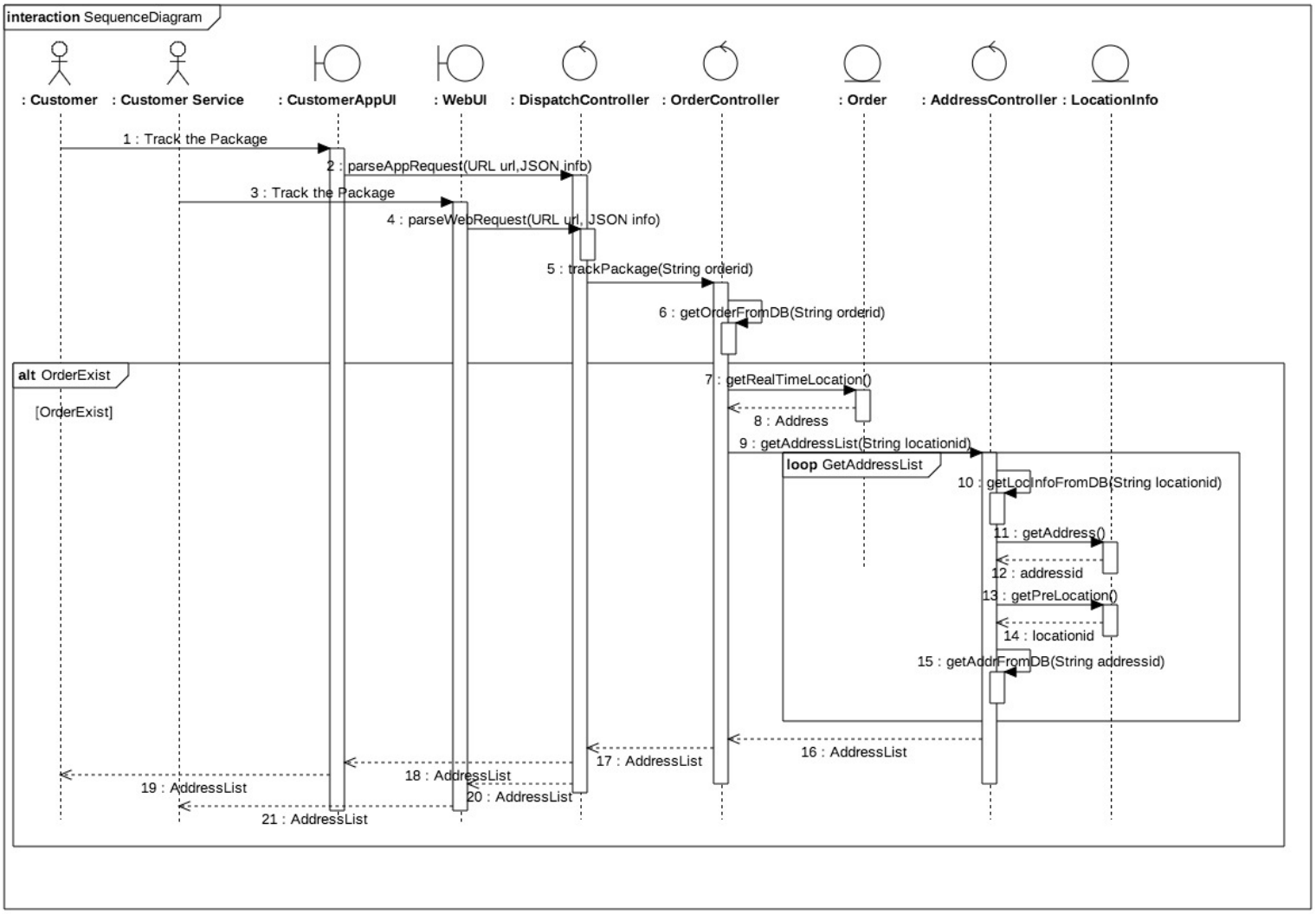


The postman will get the receiver's information by using the Postman App UI. This boundary class will call its interface getReceiverInfo(). The corresponding request will pass through Dispatch Controller, Order Controller and Order class. Each class will call its own interface showed in the class diagram and retrieve the order and receiver information finally. During the signing process, the postman App UI will call its interface signPackage(). If the package is signed by the agent, the system will identify the information and compare them by using the order controller and calling compareIdentifyInfo(). If the package is to be signed by the customer himself, the customer has alternative choices, including signature and fingerprint. We will submit the signature or fingerprint by calling the interface submitSignature(), provided by the signature system. And pass the corresponding information to the signature service which will compare the signature or fingerprint with what are stored in our system. If the corresponding information are matched with each other, the order controller will call its interface confirmDelivery(). Then the order class will use updateStautus() to complete the package delivery and return success signal to the getStatus() interface.

4.3.2 Track the Package



When the Customer and the Customer Service track the package by using the Customer App UI or Web UI, the UI will call the API implemented by dispatch controller and send the Request by the HTTP. The request contains the order id and the content-type is JSON. The dispatch controller parses the request and sends the order id to Order Controller. Order Controller gets the order information from the database. And if the order exists, Order Controller will get the location info of the order and sends it to the Address Controller. The Address Controller will get the addresses from the location information by a loop and put them in an order list. This process will come to an end when the prelocationid is null. After this process, the Address Controller return the Address list to the customer.

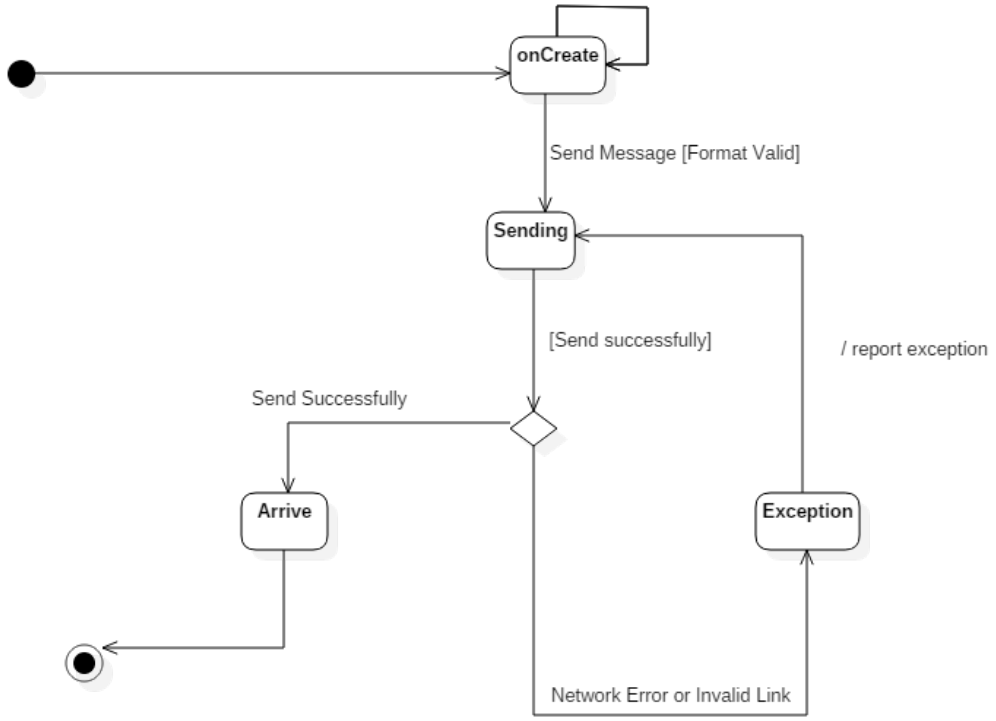


The related interfaces are showed in the sequence diagram above.

After invoking the interfaces parseWebRequest(URL url, JSON info) and parseAppRequest(URL url, JSON info) of Dispatch Controller, the system will get the tracking request from the customer or the customer service. It will use the getOrderFromDB(String ordered) provided by the Order Controller to get the package, required to track by the customer or the customer service. If the package exists in the database, it will invoke the getAddress() and getPreLocation() interfaces to get the current and former addresses of the package by its order id. Then the system will return the Address list to the customer or the customer service.

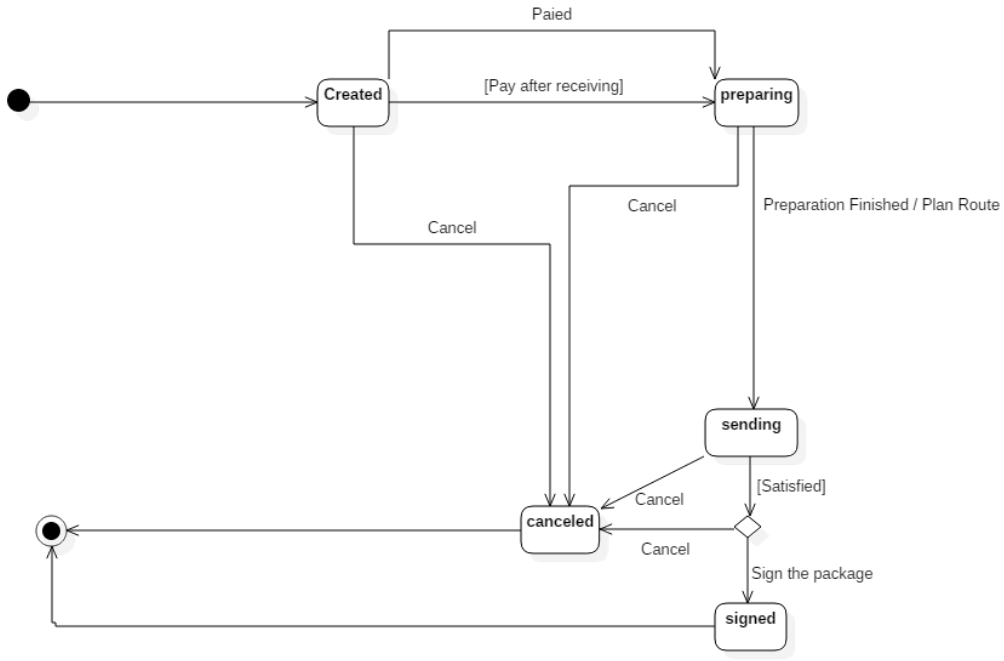
4.4 State Mechanism

Send messages:



This diagram is a state machine of sending message. We set four states, onCreate, Sending, Arrive and Exception. After a message being created, its state will be onCreate, which means it is being edited. Once the message's edition has finished, the system will check the format to make sure that it's valid. If so, the message's state will be Sending. If it arrives successfully, its state will turn to Arrive, otherwise its state will be Exception and system will send it again until it arrives.

Send Packages:



The state machine of the order class:
It has five states named as created, preparing, sending, signed and canceled. An order initiate with the created state and record the information of goods. When the “pay the order” event occurs or “pay after receiving” condition is true, it transfers into preparing state. And starting from the preparing, it will turn into sending state by the occurrence of preparation finished event. Meanwhile, it will call the PlanRoute() function to design the sending routes. And if customer choose to sign the package, sending will become signed state. However, the customer has the right to “Cancel the Order” at any state of order except the signed state. Finally, canceled and signed state will come to an end and the order entity will be destroyed accordingly.

5 Discussion

5.1 Critical Pattern

Our system takes advantage of some basic design patterns to implement the MVC infrastructure.

Factory Pattern:

Our system defines two kinds of Order classes, which are orders for E-business and those of system's. The order of the E-business is designed as a base class. Subclasses such us orders of Taobao, orders of Jingdong can be derived from the base class according to the concrete information provided by a certain E-business.

By using this E-business Order factory, our system is able to expand the function and business and even cooperate with other E-business. During the expanding process, the original program doesn't need to be modified and we just add new programs into the original one.

Facade Pattern:

Our system encapsulates similar functional modules into subsystems. The subsystem will integrate its internal interfaces into one interface which is quite convenient for the usage of the subsystem. In the meantime, these internal interfaces are still exposed to other modules. And modules can invoke and use these interfaces. In that way， the flexibility of our system is improved.

Strategy Pattern:

In our system, we encapsulate some similar algorithm clusters by adopting the strategy pattern. The algorithms existing can be used by the internal modules or users whenever they need. Take the encryption mechanism as an example. We have provided several algorithms of encryption and generation of random strings. If there are any encryption algorithms in the future, we will just expand the program instead of modifying it a lot.

Observer Pattern:

Our system adopts the observer pattern based on the MVC model to reduce the coupling degree between each layer. When receiving the request from the front-end, the system will pass the request from the front-end to the controller to handle. Whenever the data information were changed, the system will inform the views that are observing the data of the changing information. The updates will be done then. What's more, the propelling function is implemented based on the observer pattern.

Composite Pattern:

Our system can provide users with information in different format according to users' requests. A certain format is the combination of different views. And our system implements the combinations of different views by adopting the composite pattern.

5.2 Open Issues

Expandability & Maintainability

In the design of our system, we encapsulate the modules which are possible to be changed and reserve interfaces for extension. Take the handling parts of E-business orders for example. Our system provides interfaces and base class of E-business order. For the changeable paying way, the system encapsulates the payment subsystem. Whenever we want to change the paying way, we can just extend the functionalities of this subsystem. For the management of messages and the propelling function, the message subsystem is encapsulated. We can extend other functions based on the original subsystem.

Robustness

The robustness of our system mainly reflects in four parts.

First is the information distribution of the dispatch controller. All requests will be parsed and handled in this controller. As for the error messages, it has error processing mechanism.

Second one is the management of CSRF(Cross-site request forgery). The system will verify the request by using the session_id stored in the database and the cookie stored in the client-side. The system uses its encryption mechanism to generate the session_id and verify requests.

Thirdly, it's about the load balancing technique which is used for flow controlling. When the server breaks down or is attacked by lots of requests, the system will still be able to operate normally.

Finally, our database utilizes cluster mode for data storage and request handling. If the server breaks down, the system will still operate normally.

6 User Interfaces

6.1 Website

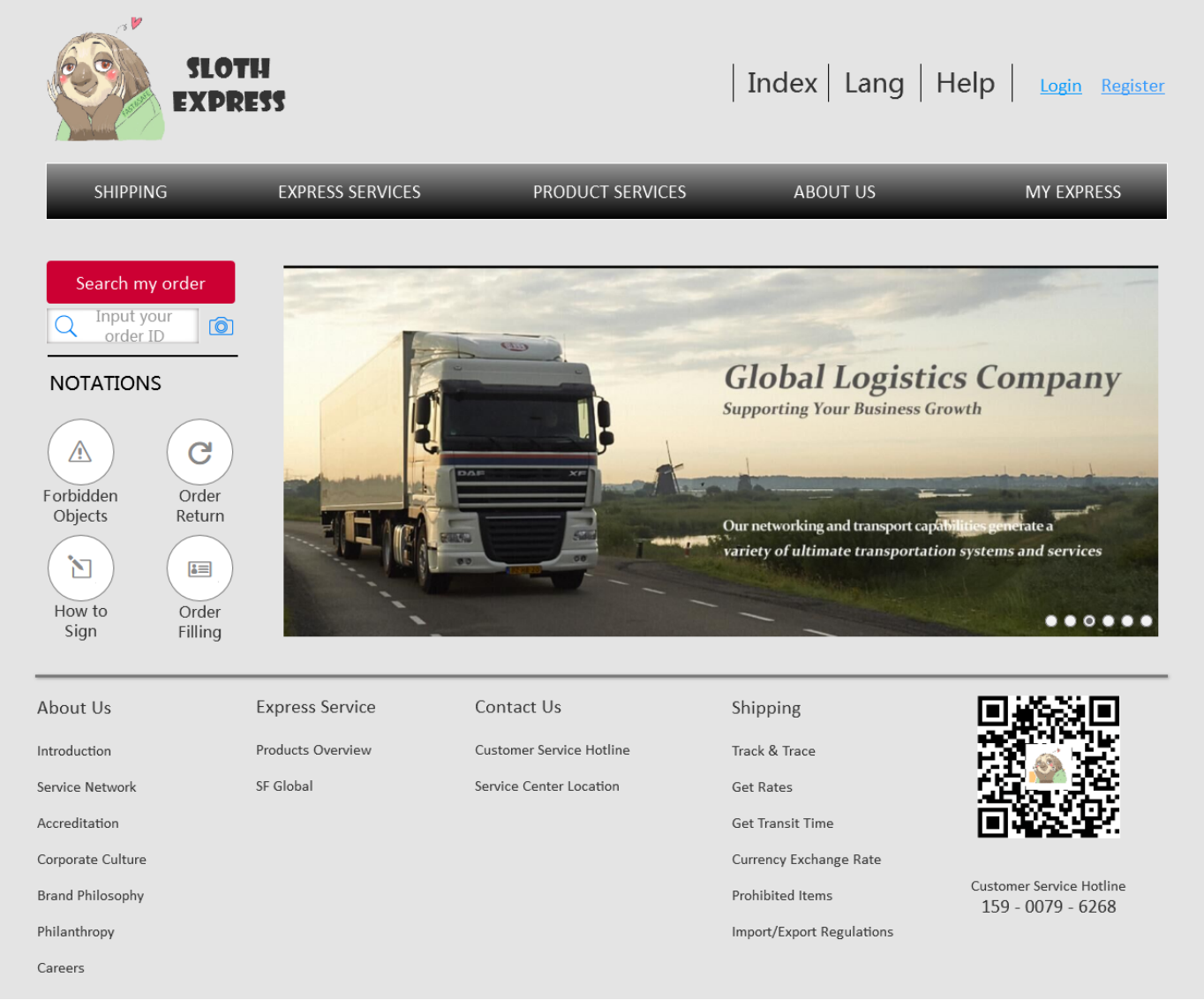


Figure6.1 The web page of the express system

There is basic information of the system showed on the page. The services can be ordered on this website after the user logs in. The custom can search for their packages' particulars after logging in. In addition, the significant notations of the sloth express are showed on the web page, such us the forbidden objects etc. Any common browsers of the website can contact the sloth express company and know about the company freely. The service hotline and the QR code are showed in the bottom right corner.

6.2 Mobile Devices (APP)

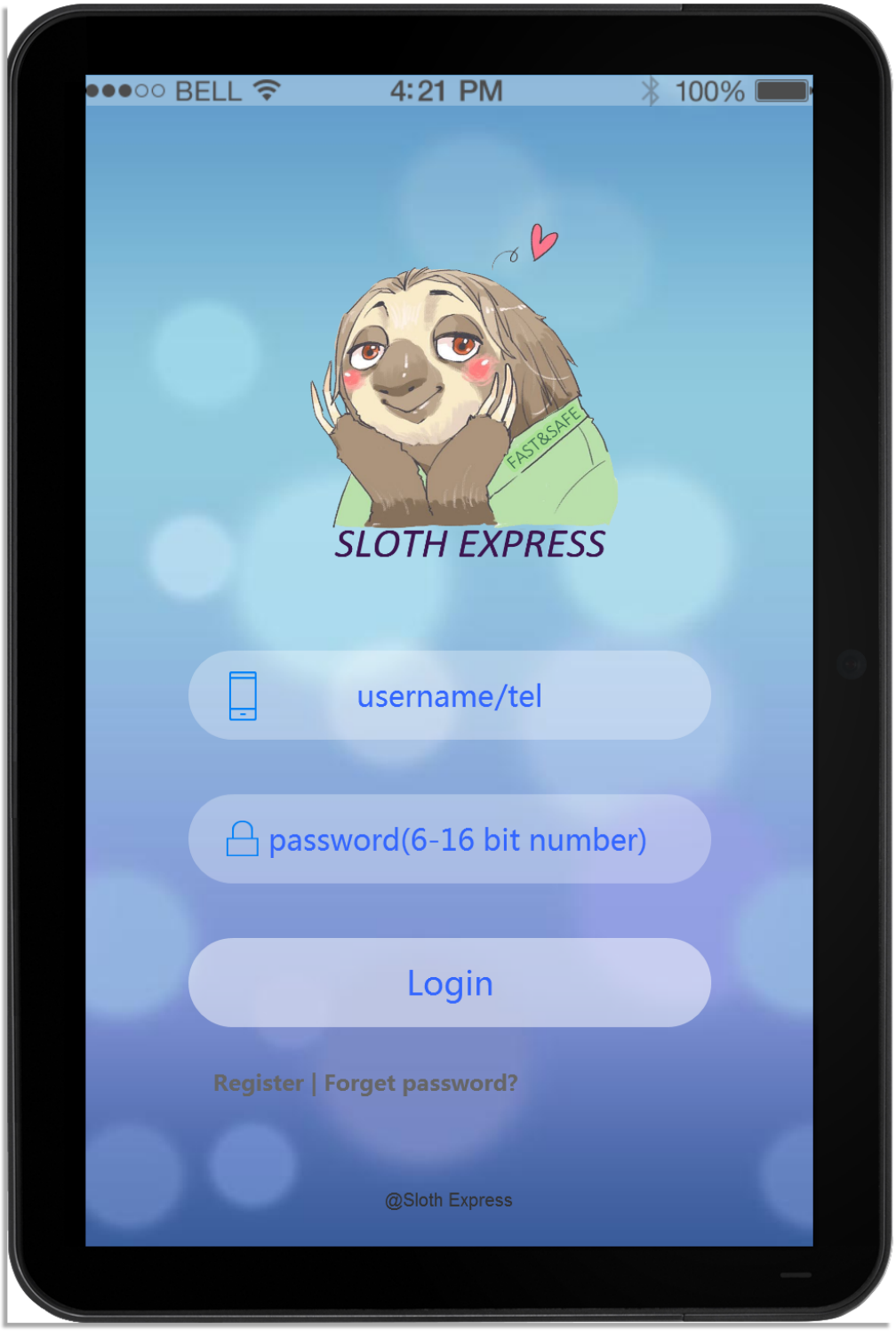


Figure6.2.1 Home Screen for both the customer and the postman

The use inputs his/her username and the password into the system. The system will recognize the identities of the users automatically. In addition, this page provides the service of registration and retrieving the password.

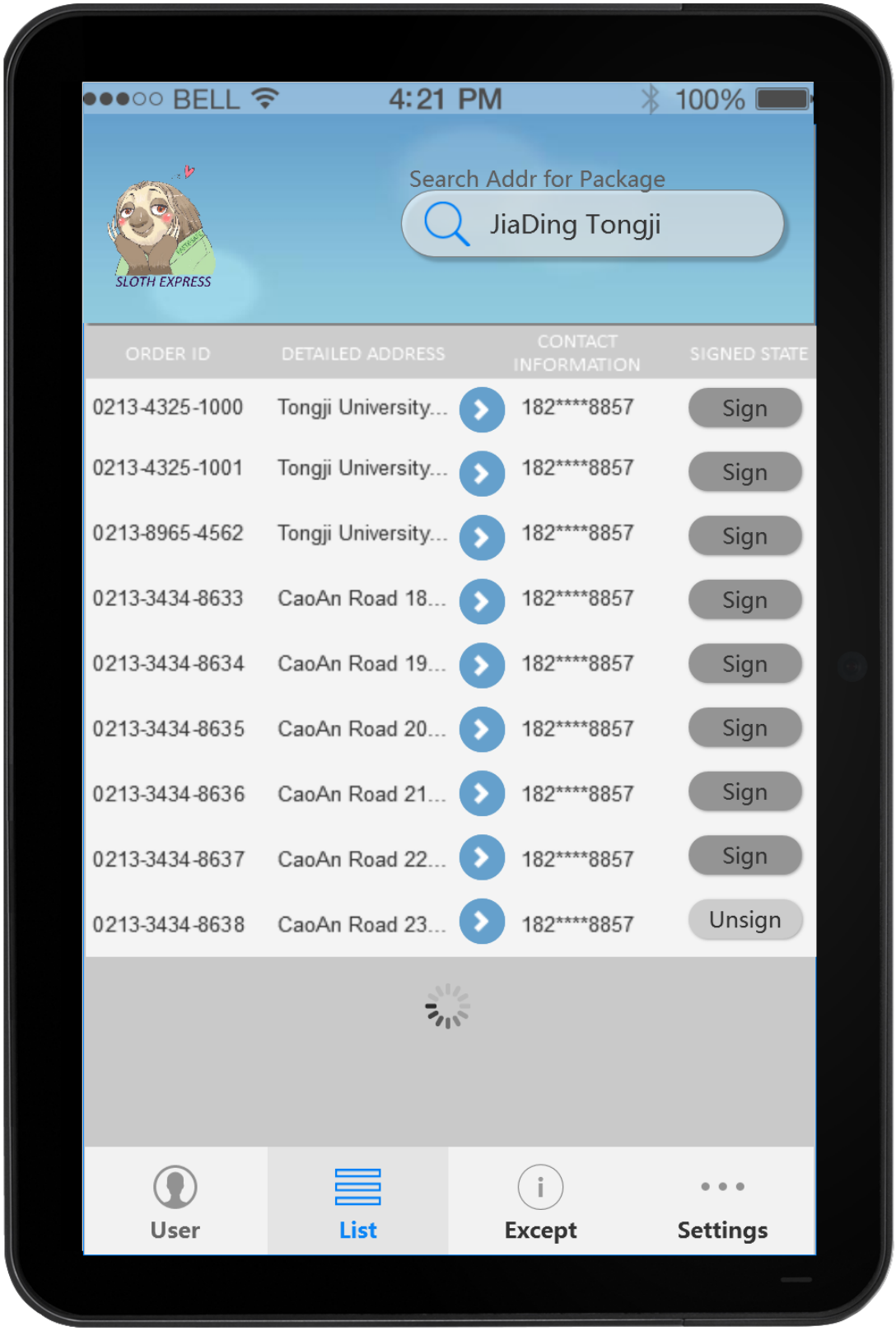


Figure6.2.2 The product list screen for the postman

After logging in, the postman will search for the appointed delivery address and then find out all the delivery missions. The postman is able to click the buttons for detailed addresses and look over the delivery status of each order.

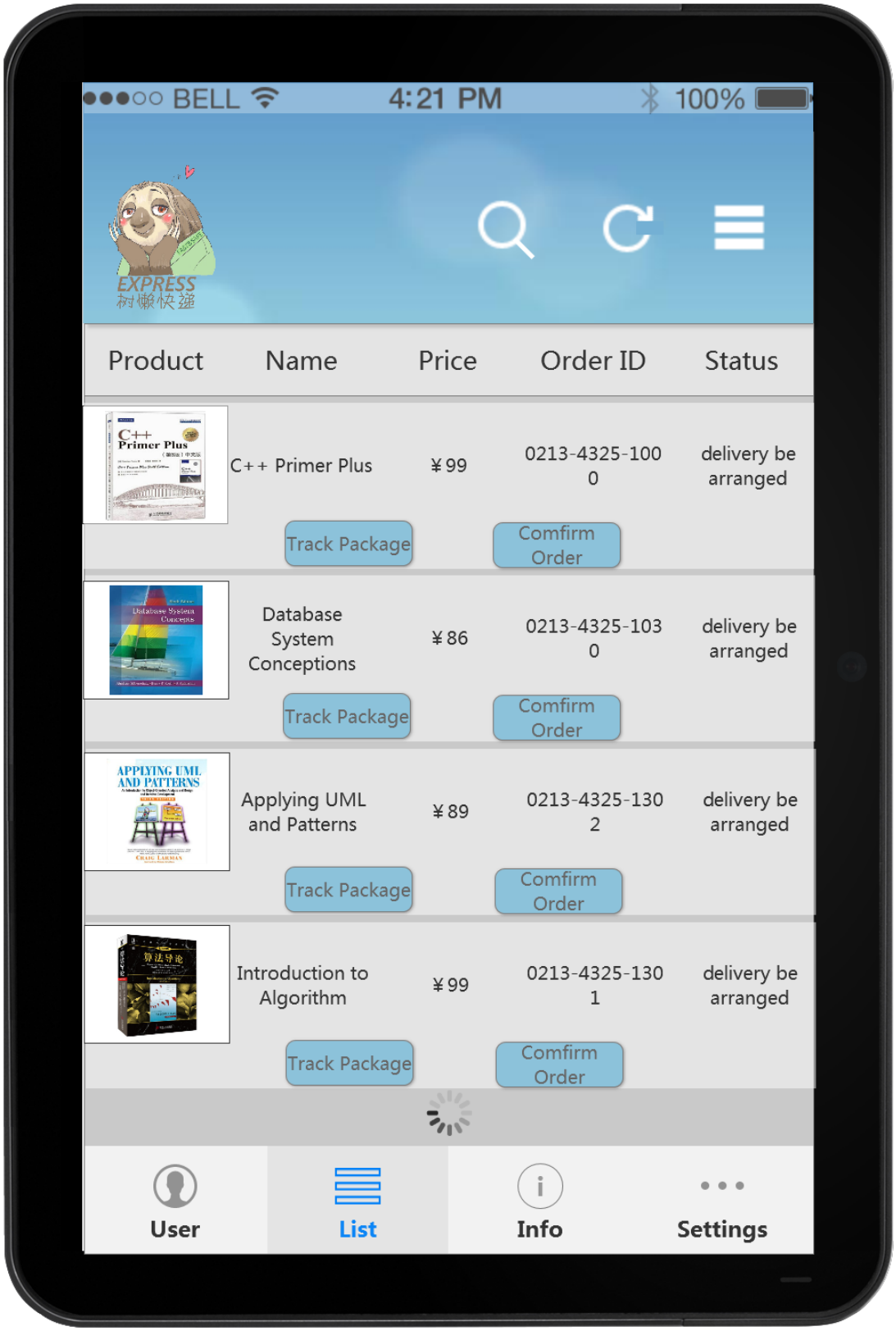


Figure6.2.3 The delivery particulars screen for the customer

After logging in, the customer can find his/her order list and the delivery details. The order IDs are provided for specifying each order's delivery information. Tracking the package and confirming the delivery will be recorded in the system.

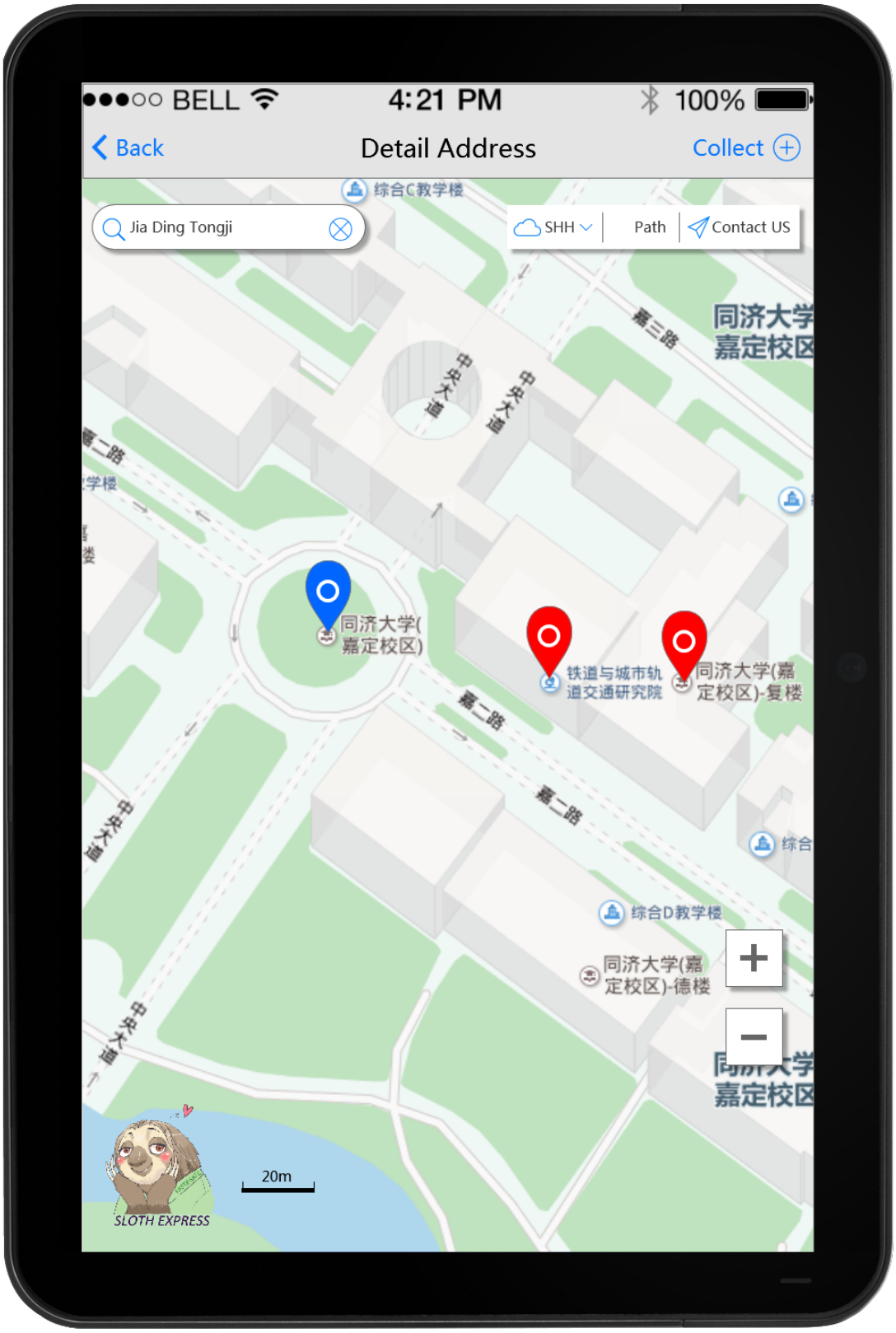


Figure6.2.4 The address inquiry screen for the postman

After clicking the buttons for detailed addresses, the postman can get the GPS from a real-time updated map, which enables him to find out where he is and his destinations. The postman can click the top-right button to collect new addresses into the database.

Figure6.2.5 The signing screen designed for the postman

After scanning the QR code (button on the top-right), the receiver can sign the package. There are two methods: signatures or a fingerprint. If the product is unpaid, one more column needs to be checked. Click the submit button to complete the delivery.

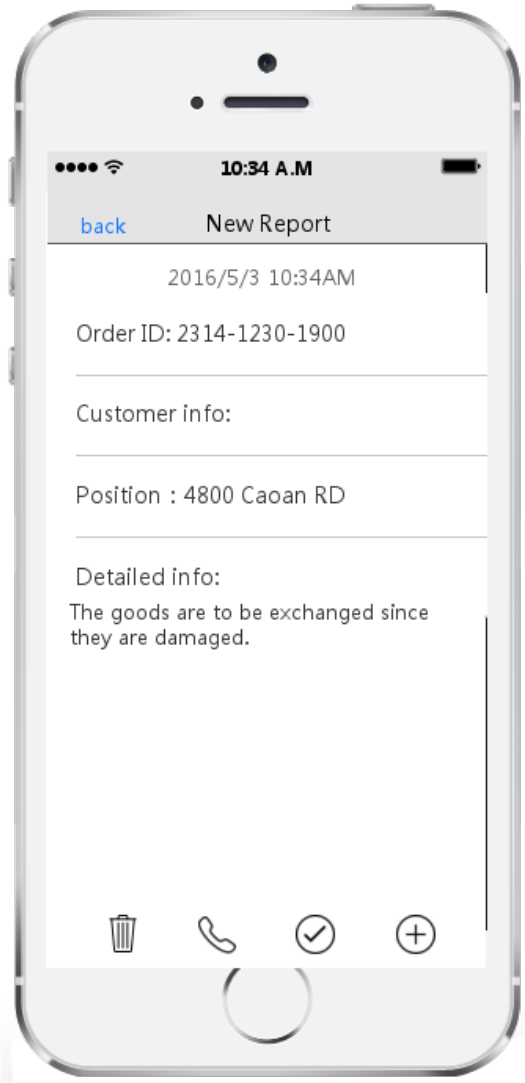


Figure6.2.6 New report page designed for the postmen

If the goods are damaged or lost during the delivery process. The postmen need to edit reports to record the information and the exceptions. They should include the details in the reports including the order ID, customer info and the goods' current positions. This report will be sent to the system and it will be dealt with by the customer service.

Figure6.2.7 Message page designed for the customers

They are used to connect the system with the customers. On the left, it is the message that will be sent to the customers. The system will tell the customers the exceptions of their products and the choices for them to deal with the matter. On the right, the user interface shows the way for customers to contact with the system. The customer can edit the message and chooses what he wants to do with the packages, including exchange and withdrawal. The customer needs to input the order ID and his phone number for our customer service to contact with him.

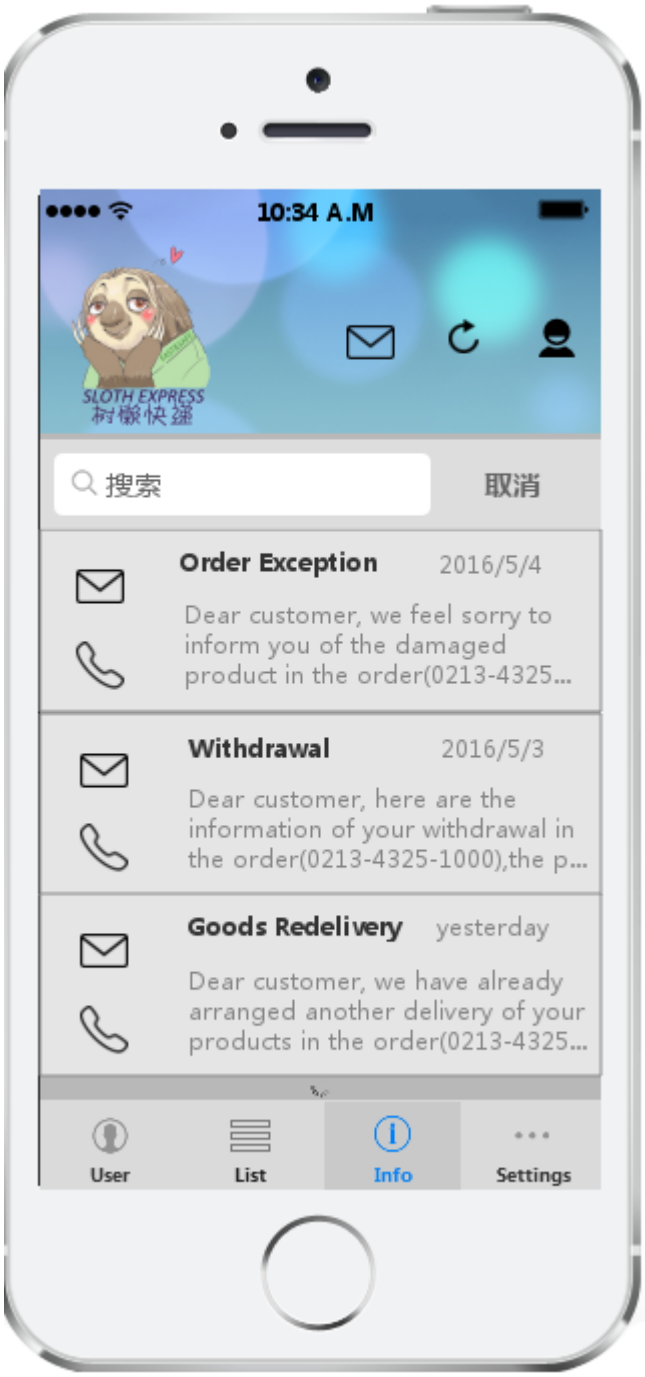
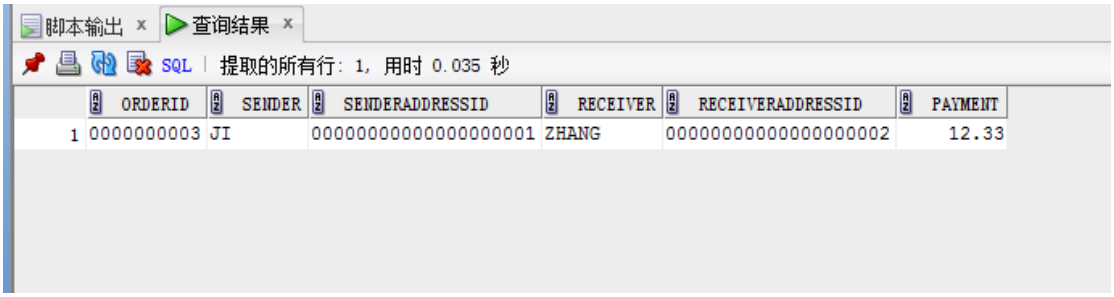


Figure6.2.8 Message list page designed for the customers

This user interface is showed to the customers. This page shows the messages list for the customer. The customer will get the information of his packages including the process of exchange and withdrawal, the exceptions during the delivery process. The customer can contact with the customer service and send messages or telephone with us.

7 Prototype



We build the connection with Oracle and the attributes of the connection, including user name, password and the dialect.

We create the mapping between the Order table in Oracle and the class Order of Java code. And specify the configuration file and starts one session with the database.

After the instantiation of the Order class, the data has been stored in the database after the operations.

Encryption part:

```
DBPassword: 塭"  墊□?□K?7=,□
True
```

In order to realize extendibility and reusability, the encryption mechanism of our system is designed be using the strategy pattern. We have defined 2 interfaces, which are encryption behaviors and generation of random string. An abstract class Encryption is defined and its method is designed for implementing the 2 interfaces. We may define a class inherited from the Encryption class and define the two interfaces in detail. In our prototype, we design two concrete class, named as MD5Encrypt and SHAEncrypt. And 2 other concrete classes are defined for the generation of random string and encryption of the password. The detailed test is in the test file.

The source code is showed in the compressed package.

8 Contributions

1452693 Wang Guansong Document, Presentation	20%
1452712 Wang Jiahui Document, Presentation	20%
1452716 Zhang Yinjia Modeling, Presentation	20%
1452762 Zhu Fangrui Document, Presentation	20%
1452768 Ji Xunzhen Modeling, Presentation	20%

Appendix

I. Glossary of Terms

- **after-sales service**: Also called customer service, after-sales service is the provision of service to customers before, during and after a purchase.
- **article:** The material in the package which is sent by a normal customer.
- **bi-directional read**: The information can be read in both direction.
- **cash-on-delivery express**: The sale of goods by express where payment is made on delivery rather than in advance.
- **claim**: When packages are damaged or lost, customers have right to ask for compensation.
- **courier**: A courier is a person who delivers messages, packages, and mail. Here it refers to postmen.
- **customer service staff:** The staff in the logistics company serving customers.
- **damaged express item**: The package that is damaged during express.
- **decision support system (DSS)**: A decision support system is a computer-based information system that supports business or organizational decision-making activities.
- **delivery:** A single task to send the package to a customer.
- **delivery terminal:** The destination of the delivery where the receiver receive and sign the package.
- **dispatch list**: The digital list of information of the packages to be delivered in postmen's port.
- **distribution center**: A station in a large district to transfer packages to the regional distribution center.
- **door-to-cfs**: From the shipper factory or warehouse to the destination or the Container freight station of the discharging port.
- **door-to-door**: From the shipper factory or warehouse to the consignee's factory or warehouse.
- **Electronic Data Interchange (EDI)**: Electronic Data Interchange is an electronic communication method that provides standards for exchanging data via any electronic means.
- **Electronic Order System (EOS)**: Electronic Order System is to meet demand instantly, with perfect quality and punctuality.
- **express item**: Packages to be delivered.
- **express item tracking system**: A sub system in our system to track the packages with GIS automatically.
- **express network**: A service network within the scope to help delivery.
- **express waybill**: An express receipt given by the carrier to the shipper acknowledging receipt of the packages being shipped and specifying the terms of

delivery.

- **first time delivery**: The first time for a particular postman to send the package to a position.
- **Global Positioning System (GPS)**: The Global Positioning System is a space-based navigation system that provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites.
- **Geographic Information System (GIS):** A geographic information system is a system designed to capture, store, manipulate, analyze, manage, and present all types of spatial or geographical data.
- **handheld terminal**: Handheld terminal refers to the portable data processing terminal with some particular features. Here it refers to mobile phones with our APP.
- **inquiry**: The customer logins the system or connects with customer service staff to get information about the order, operation instruction etc.
- **Integrated Services Digital Network (ISDN)**: Integrated Services for Digital Network is a set of communication standards for simultaneous digital transmission of voice, video, data, and other network services over the traditional circuits of the public switched telephone network.
- **interchange receipt**: A voucher to certify that the customers or e-business commits articles or products to the logistics company for delivery.
- **Invoice (INV)**: An invoice is a commercial document issued by a seller to a buyer, relating to a sale transaction and indicating the products, quantities, and agreed prices for products or services the seller had provided the buyer.
- **Just-in-time logistics (JIT logistics)**: Just-in-time logistics is a modern logistics method based on the JIT management philosophy.
- **lost express item**: The package that is lost during express.
- **order number**: The number generalized when the order is created.
- **order processing**: A series automatic operation in system to deal the order, such as creating an order, completing an order and so on.
- **package:** The material to be delivered after customers or the e-business company create orders.
- **product:** The material in the package which is ordered by customers from the e-business company.
- **QR code**: QR code (abbreviated from Quick Response Code) is the trademark for a type of matrix barcode (or two-dimensional barcode) first designed for the automotive industry in Japan.
- **Receiver**: Generalized from Customer and Agent, the person receiving and signing the package directly.
- **Redelivery**: When no one can sign the package, the postman will carry it back to the delivery terminal and the order will be rescheduled in the system.
- **redirect express item**: When customer changes the destination or the destination is out of scope, the package will be reassigned.
- **regional distribution center**: The substation in a certain region of the logistics

company to assign packages to postmen.

- **Return**: If customers are unsatisfied with the product, he or she can send it back with a label from system.
- **sender**: The customer or the e-business company who sends the package.
- **serial number of express**: i.e. the tracking number of packages in the system.
- **sign in**: The receiver sign the package and get it.
- **sorting**: The packages in the regional distribution center are sorted to transfer to corresponding postmen or the packages in the distribution center are sorted to transport to regional distribution centers.
- **tracking number**: Especially for tracking the real-time GPS location of the package.
- **withdrawal**: If the customer is unsatisfied with the product and has sent it back, he or she can choose withdraw the order and the payment will be reimbursed.

II. References

i) Book

[reference] Freeman, E., Freeman, E., Bates, B., & Sierra, K. (2005). *Head First Design Patterns*. *Head First design patterns =*. Southeast University Press.

Simple design pattern

1. Design Principle Identify the aspects of your application that vary and separate them from what stays the same.
2. Design Principle Program to an interface, not an implementation.
3. Design Principle Favor composition over inheritance.
4. The Strategy Pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

The decorator pattern

1. Design Principle Classes should be open for extension, but closed for modification.
2. Be careful when choosing the areas of code that need to be extended; applying the Open-Closed Principle EVERYWHERE is wasteful, unnecessary, and can lead to complex, hard to understand code.
3. The Decorator Pattern attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclasses for extending functionality.
4. But **Java** I/O also points out one of the downsides of the Decorator Pattern: designs using this pattern often result in a large number of small classes that can be overwhelming to a developer trying to use the Decorator-based API.

The factory pattern

1. The Factory Method Pattern defines an interface for creating an object but lets subclasses to decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
2. Design Principle Depend upon abstractions. Do not depend upon concrete classes.
3. The Abstract Factory Pattern provides an interface for creating families related or dependent objects without specifying their concrete classes.
4. The intent of Factory Method is to allow a class to defer instantiation to its subclasses.
5. The intent of Abstract Factory is to create families of related objects without having to depend on their concrete classes.

The Adapter and Facade Patterns

1. Here's how the client uses the Adapter
    1.1 The client makes a request to the adapter by calling a method on it using the target interface.
    1.2 The adapter translates that request into one or more calls on the adaptee using the adaptee interface.
    1.3 The client receives the result of the call and never knows there is an adapter doing the translation.
2. The Adapter Pattern converts the interface of a class into another interface the clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.
3. A facade not only simplifies an interface, but also decouples a client from a subsystem of components.
4. Facades and adapters may wrap multiple classes, but a facade's intent is to simplify, while an adapter's is to convert the interface to something different. (P260)
5. The Facade Pattern provides a unified interface to a set of interfaces inn a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
6. Principle of Least Knowledge - talk only to your immediate friends.
7. The principle tells us that we should only invoke methods that belong to: (P266)
    The object itself;
    Objects passed in as a parameter to the method;
    Any object the method creates or instantiates;
    Any components of the object.

The proxy pattern

1. By invoking methods on the proxy, a remote call is made across the wire and a String, an integer and a State object are returned. Because we are using a proxy, the GumballMonitor doesn't know, or care, that calls are remote (other than having to worry about remote exceptions).
2. The Proxy Pattern provides a surrogate or placeholder for another object to control access to it.
3. Use the Proxy Pattern to create a representative object that controls access to another object, which may be remote, expensive to create or in need of securing.

ii) Articles

- The paper presents and assesses a layout scheme for UML class diagrams that takes the architectural importance of a class in terms of its stereotype (e.g., boundary, control, entity) into account. They believe that organizing a class diagram bases within the context of these techniques but that emphasizing architectural importance is a better method of layout. And to prove their idea, they implemented a pilot study online and asked expert, intermediate and basic

participants to answer the study online. Through analyzing the result, they draw a conclusion that it is more important to have an architecturally meaningful UML class diagram.
[reference] Andriyevska O., Dragan N., Simoes B. & Imaletic J. (2005). Evaluating UML Class Diagram Layout based on Architectural Importance. *Visualizing Software for Understanding and Analysis.*

- This paper introduces a general monitoring system based on B/S software structure. This system manages to build data communication server, web server and database server by applying ASP.NET and database technology. With the addition of data-acquisition units placed in turnouts, users with permission can monitor the real-time running condition through the Internet based on TCP/IP and UDP protocol. Typically, it focuses on security, since almost all of the monitoring objects are security devices. Besides, to collect and store enormous data, the system has powerful compatibility and generality. It employs Microsoft SQL Server 2005 as the engine of its database. The system also supports analyzing the running trend of the target as the data source. It has been initially proved in an experimental railway that the design idea is correct and monitoring method is feasible.
  [reference] Y. Wang, J. Li, X. Yin, Y. Liu, & J. Dong. (2013). Designing of general railway monitoring system based on B/S structure. *Control Conference (CCC), 2013 32nd Chinese* (pp.8189-8194). IEEE.

- "Network business security" provides theoretical basis for security defense of enterprise automatic production system and enterprise management information system. It is defined in 3 parts: data security, network system security and network business security.
  [reference] K. Wu, T. Zhang, W. Li, G Ma. 2009. Security Model Based on Network Business Security. Computer Technology and Development.

- A network security system base is designed from the network between the transport layer and application layer. The network system involves digital signature, encryption, decryption, authentication and data transmission function. It has passed the strict test, which means that it realizes a real security of the Internet.
  [reference] Z. Huang. 2014. The Design and Implementation of Security Network System Based on Web. Advanced Research and Technology in Industry Applications (WARTIA), 2014 IEEE Workshop

- The distributed database is a database in which storage devices are not all attached to a common processing unit such as the CPU, and which is controlled by a distributed database management system. In another way, its data is stored in many devices respectively, but they are managed by DDBMS (Distributed Database System).

[reference] O'Brien, J. & Marakas, G.M. (2008). Management Information Systems (pp. 185-189). New York, NY: McGraw-Hill Irwin

- To serve content to users with high availability and high performance, CDN (Content Delivery Network) is useful. CDN provides storage space to users from respective servers, by choosing which server is best available to terminals. It makes data transfer faster and more steady, so as to offer a better user experience. It can also protect the system from DoS attack.
  [reference] Wikipedia. (2016). Content delivery network. Retrieved on April 2, 2016. From:
  https://en.wikipedia.org/wiki/Content_delivery_network

- SDN (Software Defined Network) is a solution to computer networking that allows the manager of the network service to administrate the architecture of the net system without changing the physical equipment, just on soft-level. It is more dynamic and cheaper for building and changing the arrangement of the network.
  [reference] Wikipedia. (2016). Software-defined Network. Retrieved on April 2, 2016. From:
  https://en.wikipedia.org/wiki/Software-defined_networking

- To cache dynamic data, the cache has to have certain features: expirations, evictions, caching relational data, synchronizing a cache with other environments, database synchronization, read-through, write through & write behind, cache query, event propagation, cache performance and scalability, high availability and performance.
  [reference] Khan, Iqbal. "Distributed Caching On The Path To Scalability". MSDN (July 2009). Retrieved 2012-03-30.

iii) List of References

[1] J. Pan. (2013). *Software Methodology, Volume 1, Business Modeling and Requirements.* Tsinghua University Press
[2] Paul, S; Z Fei (2001-02-01). *"Distributed caching with centralized control". Computer Communications* 24 (2): 256–268
[3] Wikipedia. (2016). *Electronic business*. Retrieved on April 2, 2016. From:
https://en.wikipedia.org/wiki/Electronic_business#Security.