SLOTH EXPRESS

PACKAGE DELIVERY SYSTEM DESIGN MODEL

1452764 何冬怡 1452798 李 想 1452697 彭嘉琦 1593369 Luc Berro

CONTENT

1 Overview

As for the whole system, we used **LAMP** (**L**inux, **A**pache, **M**ySQL, **P**HP) as our web App platform. The web App deploys on them. As for APPs on Android/iOS cellphone, we have JAVA/Swift as our programming language. And we used **FTP Server** stores all the pictures on the website and **Session Server** maintains the HTTP session. The entire deployment is shown in the next part.

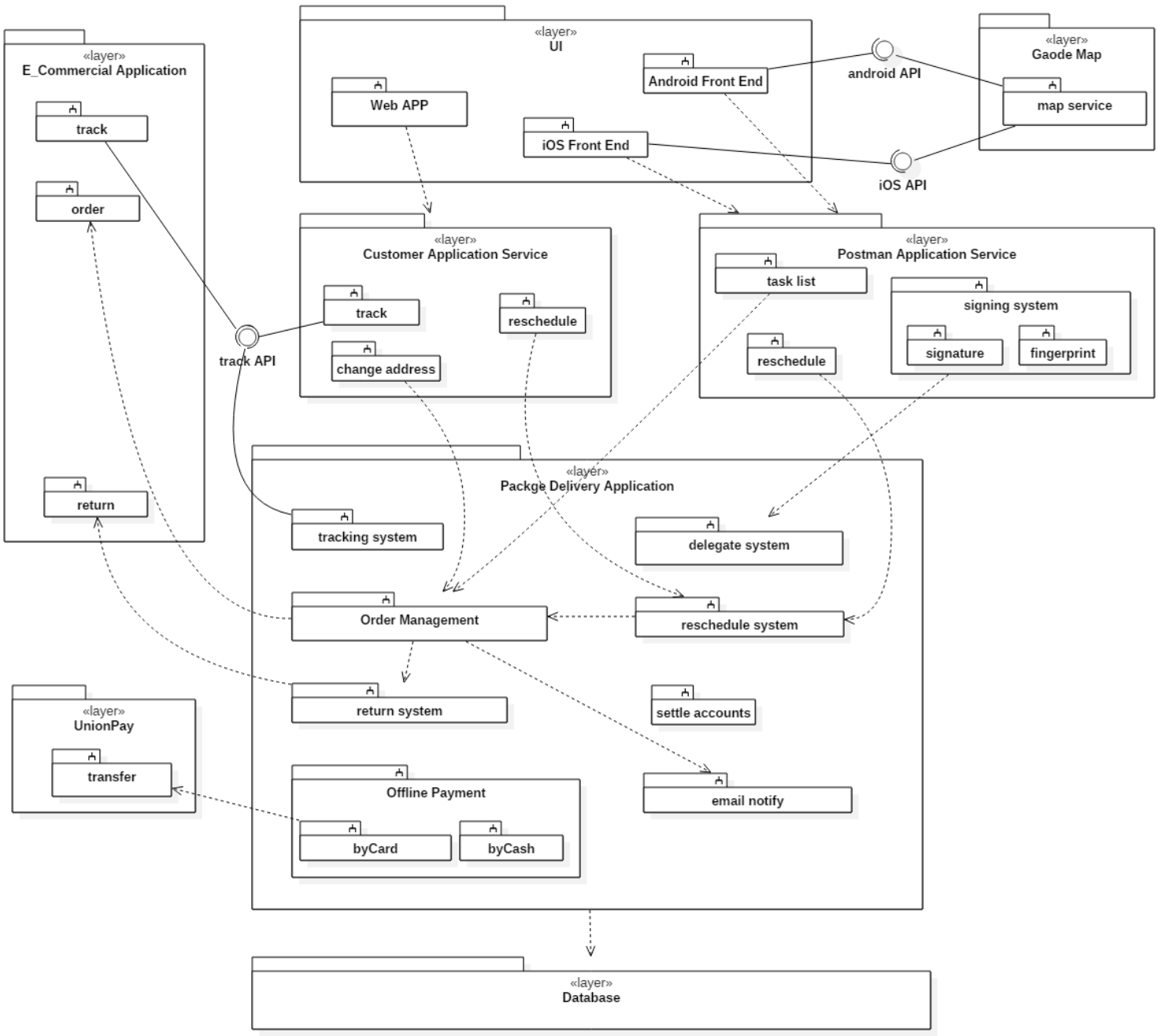
The offline payment uses a **mPOS terminal device** and its corresponding application. The mPOS service is a complete solution provided by **UnionPay**. With mPOS terminal customer can swipe their IC card and input password for payment, and via **Bluetooth**, the corresponding application deployed in postman's smartphone will get the payment message and connect to UnionPay via network.

Our **user interface** includes 3 parts, web App, iOS front end and Android front end. Because both postman and customer need to track the package, all the Apps import map service in **Gaode map**. The iOS and Android App use postman App service and web App uses customer app service.

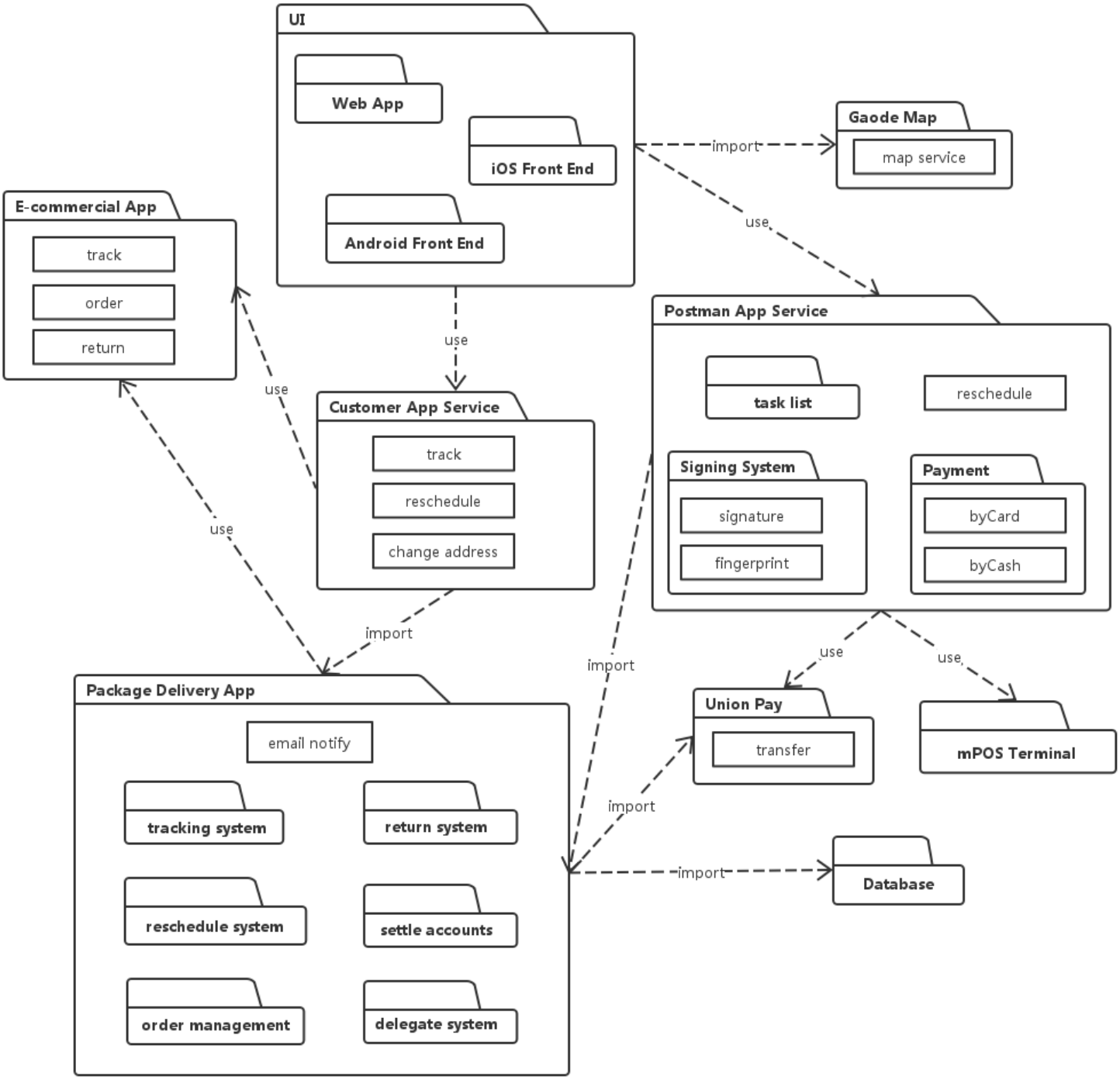
The **postman App service** includes task list, reschedule, signing system and payment. They import some sub packages in other layers. For example, the task list imports order management in package delivery application, the payment by card uses mPOS terminal.

The **customer App service** is similar. In package delivery application, in addition to the above situation, the import relationship is also between the sub packages. For example, the reschedule system imports the element in order management to finish its work. Except for this, the package delivery App also imports many data in Database, and uses the sub packages in e-commercial App, just like the tracking system uses the track from track API.

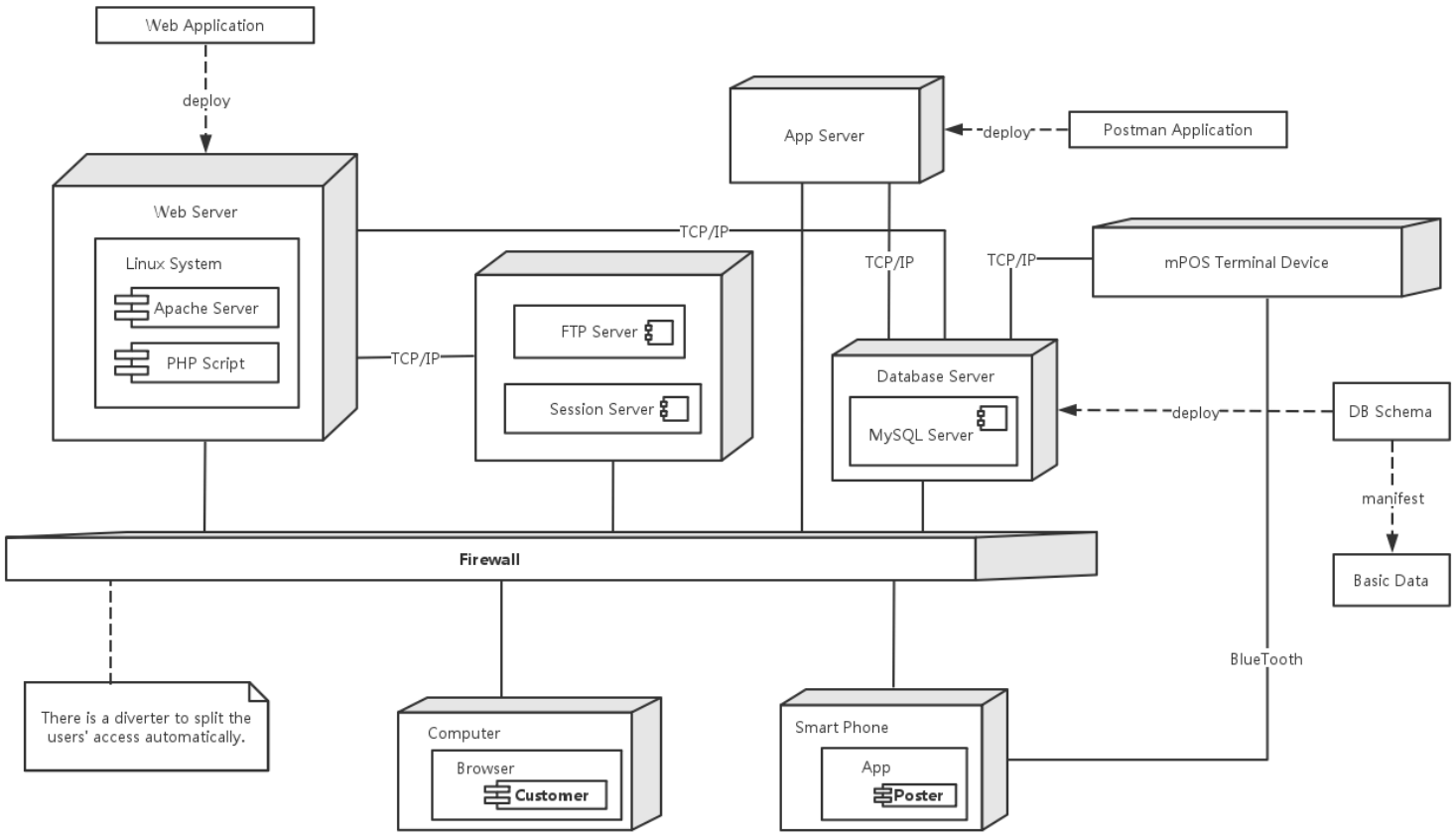
2 Architecture structure



2.1 Logical architecture (package diagram)



2.2 **Physical architecture (Deployment diagram)**



We use **LAMP** as our web App platform. L for **Linux** system, A for **Apache**, M for **MySQL** and P for **PHP/Python**. The web App deploys on them.

FTP Server stores all the pictures on the website and **Session Server** maintains the HTTP session. The two is connected to Web Server through the TCP/IP protocol. Database Server uses MySQL to save data on the website. DB Schema deploys on it and manifests Basic Data. App Server executes the console program regularly, and the postman App deploys on it.

The **mPOS Terminal Device** is connected to the DB Server through the TCP/IP protocol and the smart phone by **Bluetooth**. The postman uses smart phone to access server and customer uses browser on the computer. And all the operations on the server are filtered through the **firewall**. When plenty of users operate our server, the diverter here can split the access automatically.

3 Interface specification

3.1 Subsystems and interfaces

Tracking	+PackageInfo getPackageInfo(string orderID)
	+PostmanInfo getPostmanInfo(string postmanID)
OrderManagement	+modifyOrderInfo(string orderID, Address address, string phoneNumber)
	+settleAccounts(string orderID)
	-getPrice(string orderID)
DistributionManagement	+getOrderList(OrderList orderlist, bool isReturned)
	+addToDistributionList(string orderID)
	+deleteFromDistributionList(string orderID)
	+orderDistribution(string orderID)
	-choosePostman(string orderID)
DeliveryManagement	+arrangeDelivery(string orderID, string postmanID)
	+addToDeliveryList(string orderID,string postmanID,OrderInfo orderInfo)
	+deleteFromDeliveryList(string orderID)
	+finishOrder (string orderID, Paymathod paymathod)
	-finishOrderPayOnline(string orderID)
	-finishOrderPayOffline(string orderID)
MessageManagement	+notifyCustomer(string orderID,PostmanInfo postmanInfo)
	+notifyPostman(string postmanID, OrderInfo orderInfo)

Internal APIs

GaodeMapAPI	MapDisplay	findViewById(R.id.map)
		aMap.setTrafficEnabled(true)
		aMap.setMapType(AMap.MAP_TYPE_SATELLITE)
	MapLocation	+onLocationChanged(AmapLocation amapLocation)
		+LocationManagerProxy.requestLocationData(LocationProviderProxy.AmapNetwork,long interval,long distance,AMapLocationListener)
	RoutePlanning	searchRouteResult(int routeType, int mode)
UnionPay API	mPosPayment	+requestForPayment(BillInfo)
		+getCardInfo()
		+messageAssembly(CardInfo,BillInfo)
		+messageEncryption(string)
		+messageDecryption(string)
		+getTransactionMessage()
		+displayResult()

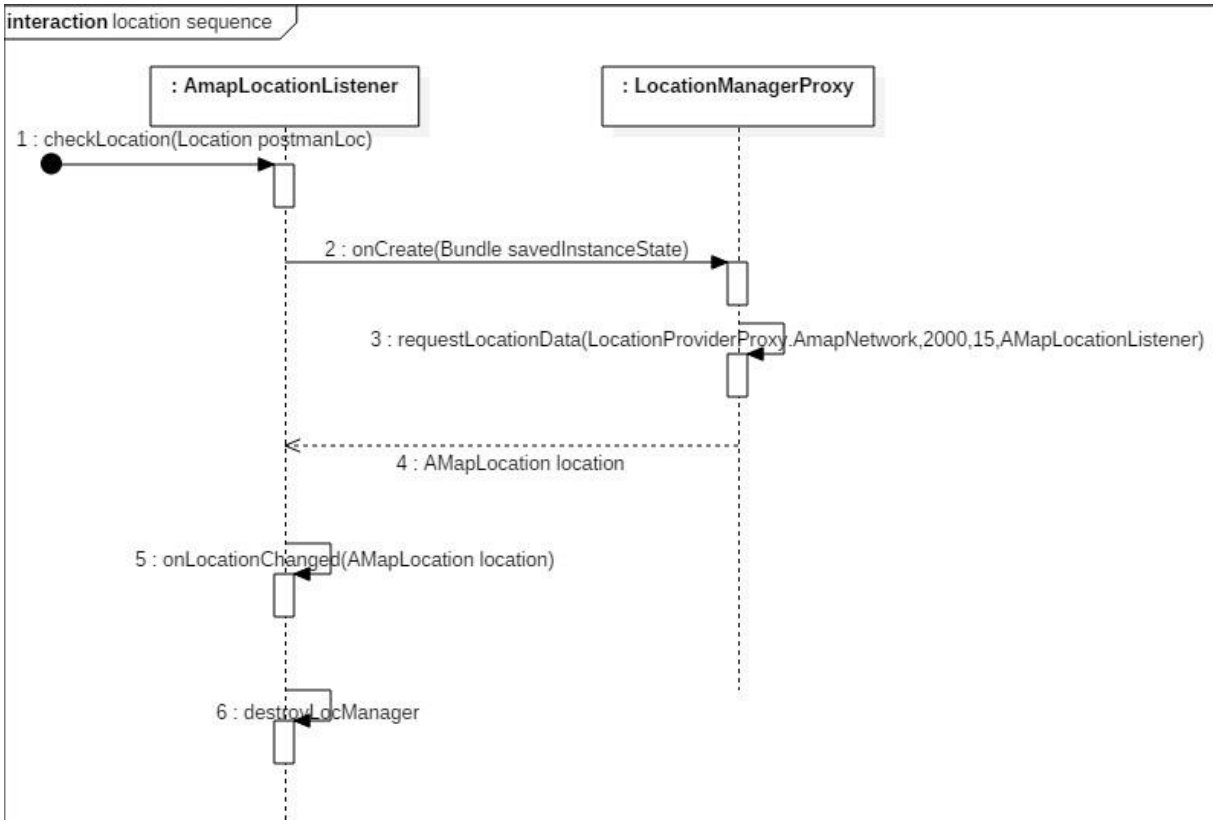
External APIs

3.2 **Detailed examples of two API**

3.2.1 Location

		usage	example	comment
name	mLocationManagerProxy.requestLocationData	monitor location		
parameter	locationProviderProxy.ProxyType	type of location	AmapNetwork	mixed location(GPS first)
	long interval	Interval of location request		unit: ms -1 means location once
		distance of location		only used in gps location
	AmapLocationListener		this	

		usage
name	onLocationChanged	location callback
Parameter	AmapLocation amapLocation	result of location

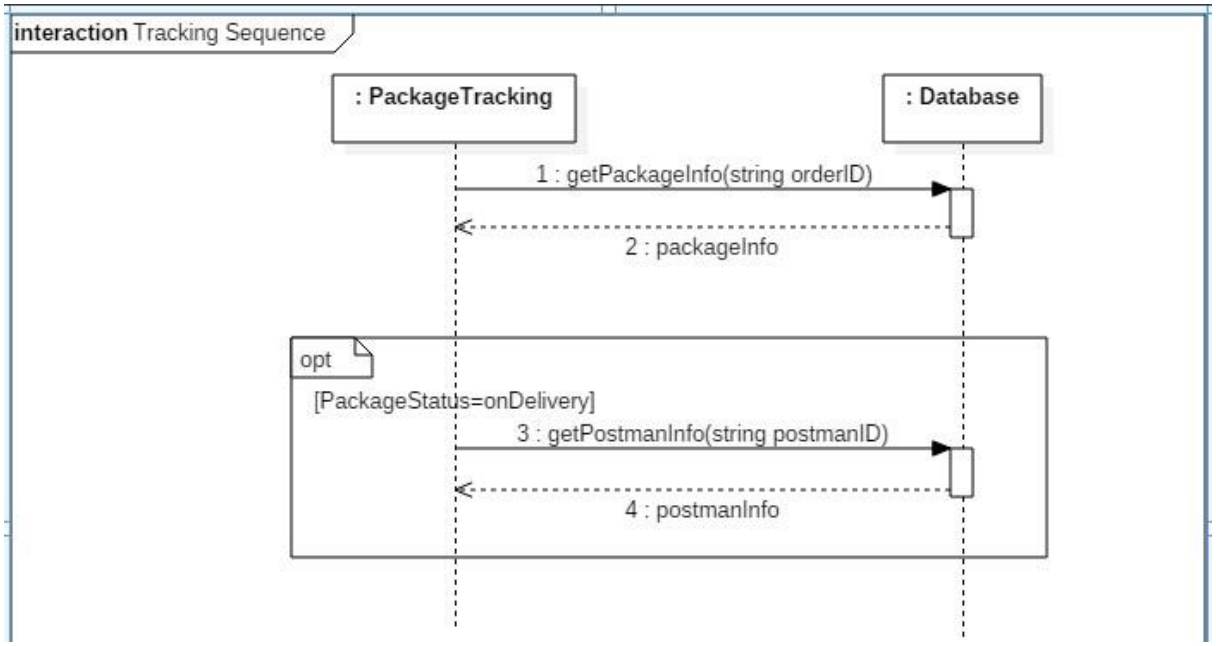


3.2.2 Track

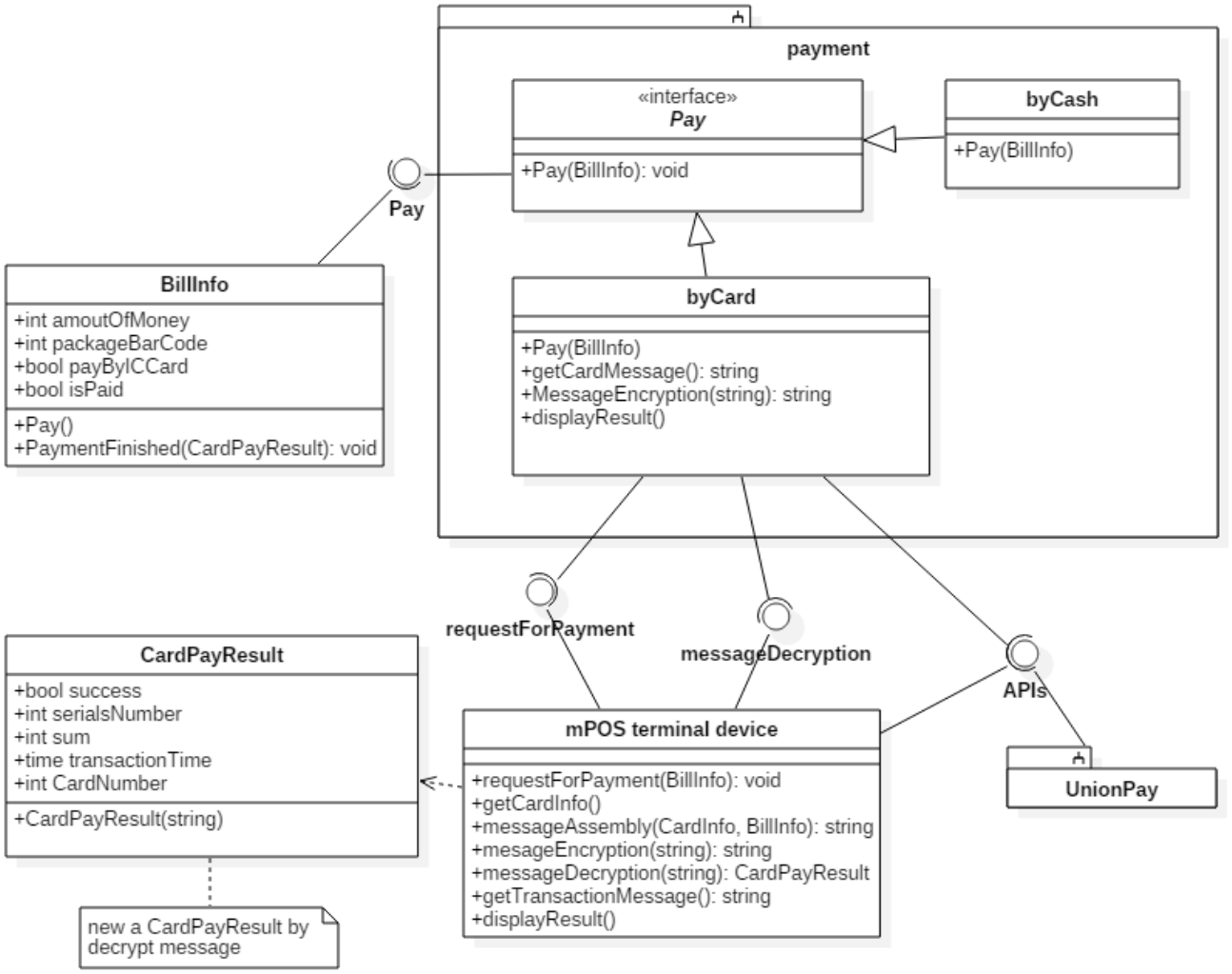
		usage	example
Name	getPackageInfo	get detail information of package	
Parameter	string orderID	ID of package	12345789
return value	PackageInfo		

		usage	example
Name	getPostmanInfo		
Parameter	string postmanID	ID of postman	021
return value	PostmanInfo		

PackageInfo	PostmanInfo	PackageStatus
string currPosition string nextPostion PackageStatus packageStatus Date date	string postmanName string postmanPhone	handling onTheWay onDelivery



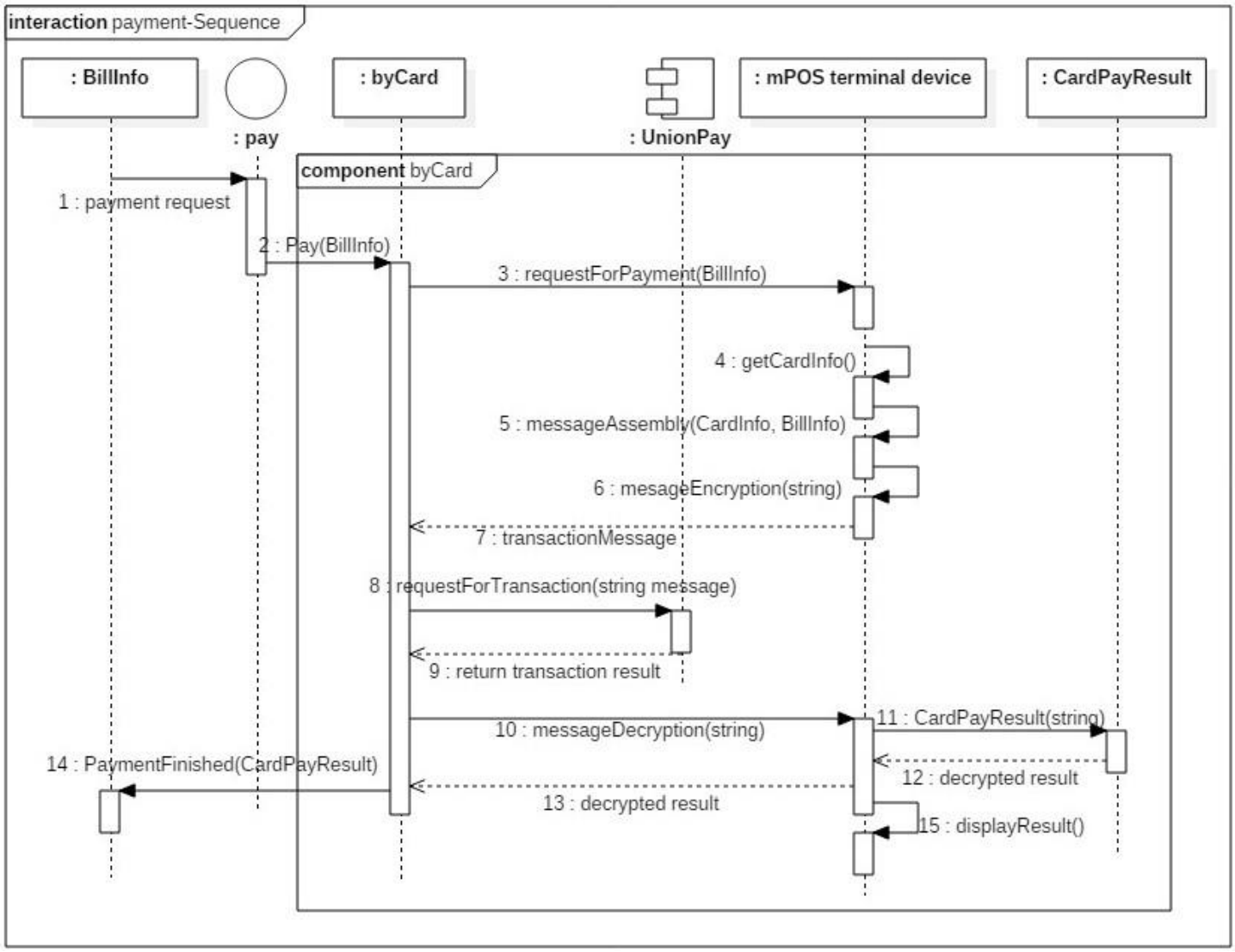
3.3 Payment subsystem interfaces (example)



The Payment system provides a **Pay** API and uses APIs from **mPOS terminal device** and **UnionPay**.

The Pay interface is realized by an abstract class Pay, and implemented by two payment method: byCash and byCard. As for byCard method, it depends on interfaces provided by mPOS device and UnionPay.

The COD (Cash on delivery) payment used the mPOS solution provided by **UnionPay**. Since the document from UnionPay is not opened to the public, interfaces in the diagram cannot be specified very clearly.



4 Mechanism

4.1 User session mechanism

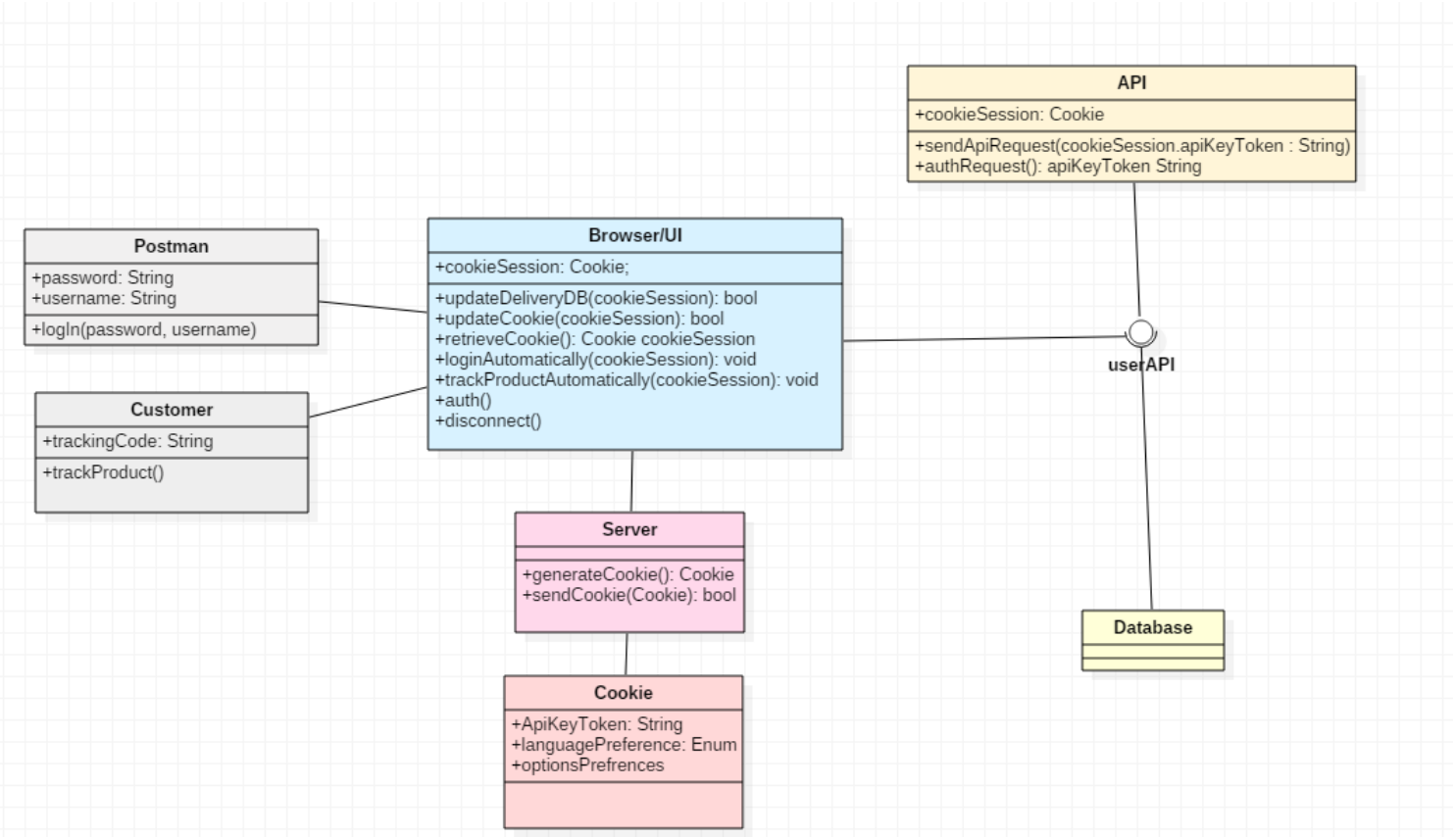
The « **User Session Mechanism** » is focused on keeping the important information of the user intact during his whole session.

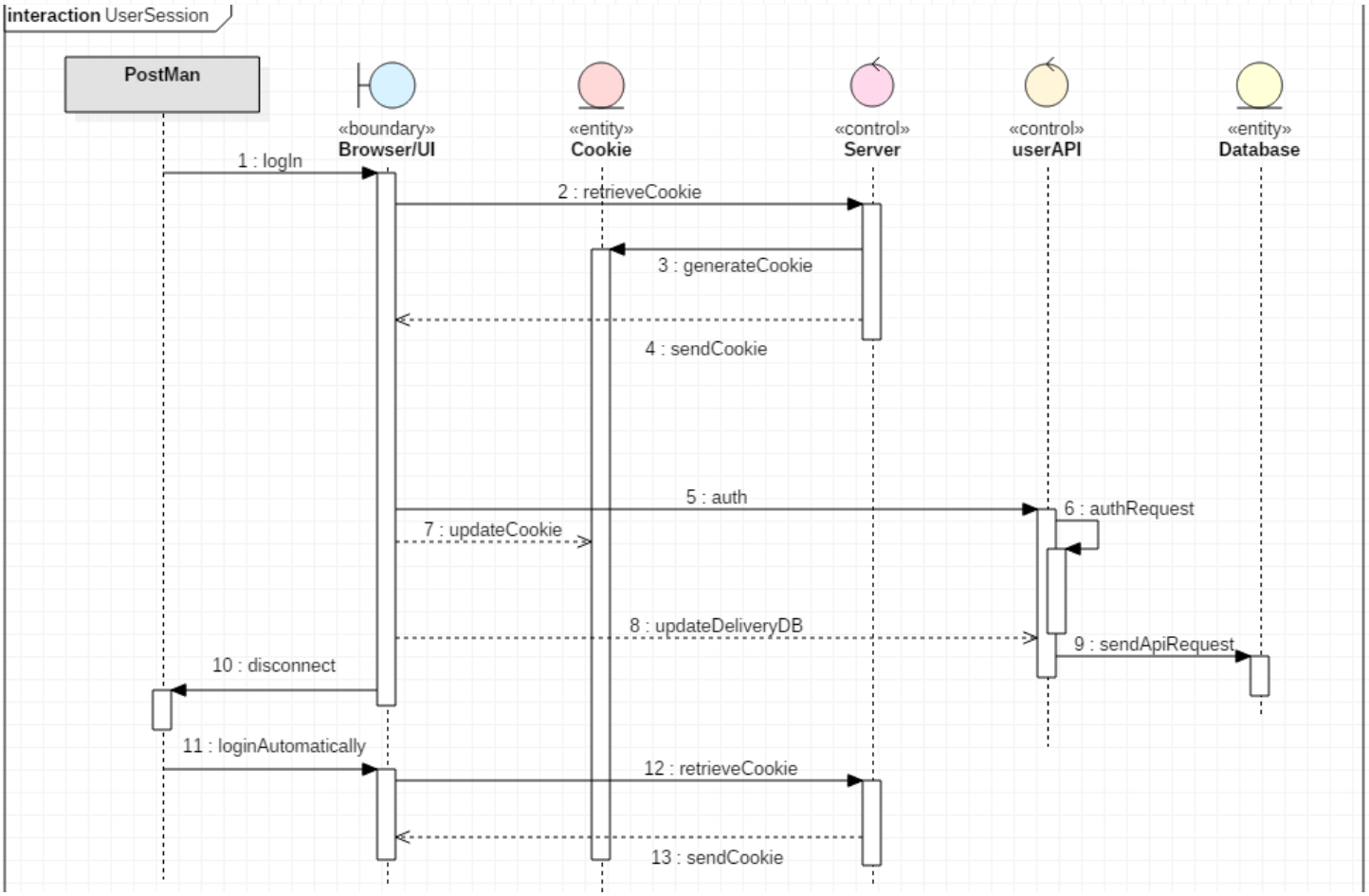
Once the user has connected once into the UI, his information are stored locally on the user's device. This information is the option preference of the UI (« remember my password » functionality, for example) and the language chosen for the UI.

Because our solution is using heavily its API, we wanted to increase the security and the friendly usage of our API. When the backend is doing any request, the API-key must be provided.

The API-key is a one-session-use key that allow the backend to freely do any request. If the API-key is not provided, the request can't be accomplished.

The API-key is automatically catch by the backend once the user has connected to the UI; the key is then stored in the Cookie to avoid asking the user to authenticate every time a backend request must be made.


API
+cookieSession: Cookie
+sendApiRequest(cookieSession.apiKeyToken : String)
+authRequest(): apiKeyToken String
Postman
+password: String
+username: String
+logIn(password, username)
Browser/UI
+cookieSession: Cookie;
+updateDeliveryDB(cookieSession): bool
+updateCookie(cookieSession): bool
+retrieveCookie(): Cookie cookieSession
+loginAutomatically(cookieSession): void
+trackProductAutomatically(cookieSession): void
+auth()
+disconnect()
userAPI
Customer
+trackingCode: String
+trackProduct()
Server
+generateCookie(): Cookie
+sendCookie(Cookie): bool
Database
Cookie
+ApiKeyToken: String
+languagePreference: Enum
+optionsPrefrences

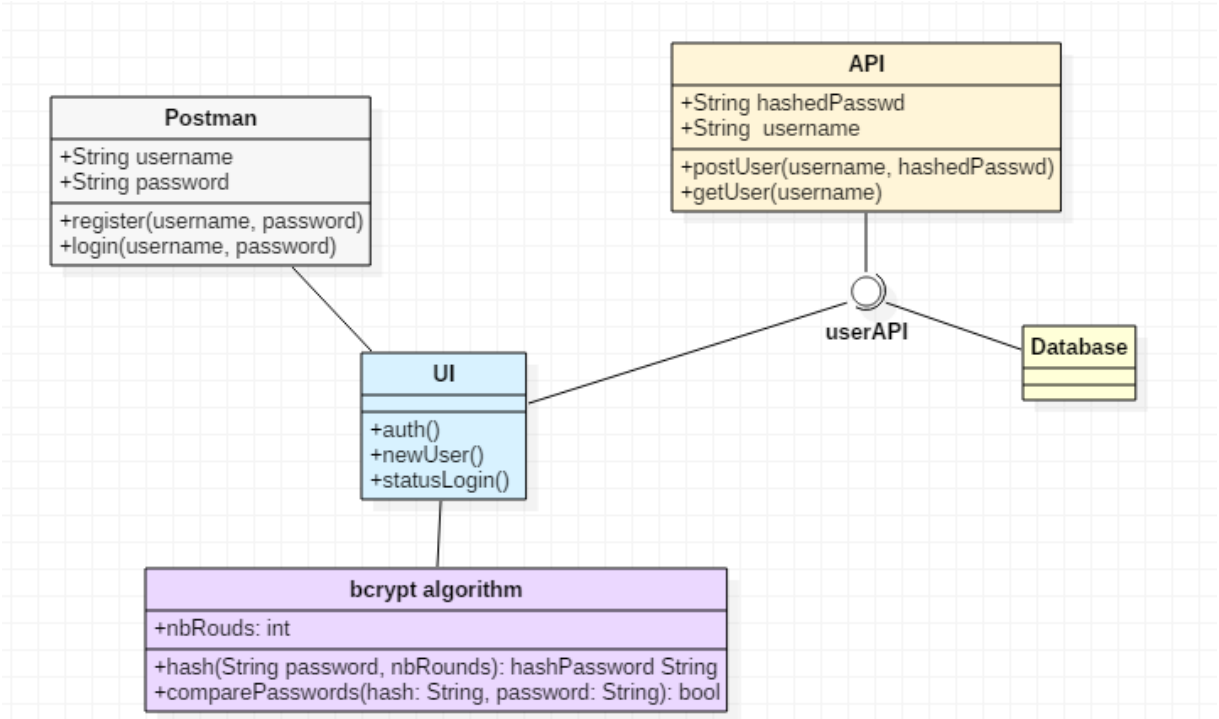

interaction UserSession
PostMan
«boundary»
Browser/UI
«entity»
Cookie
«control»
Server
«control»
userAPI
«entity»
Database
1 : logIn
2 : retrieveCookie
3 : generateCookie
4 : sendCookie
5 : auth
6 : authRequest
7 : updateCookie
8 : updateDeliveryDB
9 : sendApiRequest
10 : disconnect
11 : loginAutomatically
12 : retrieveCookie
13 : sendCookie

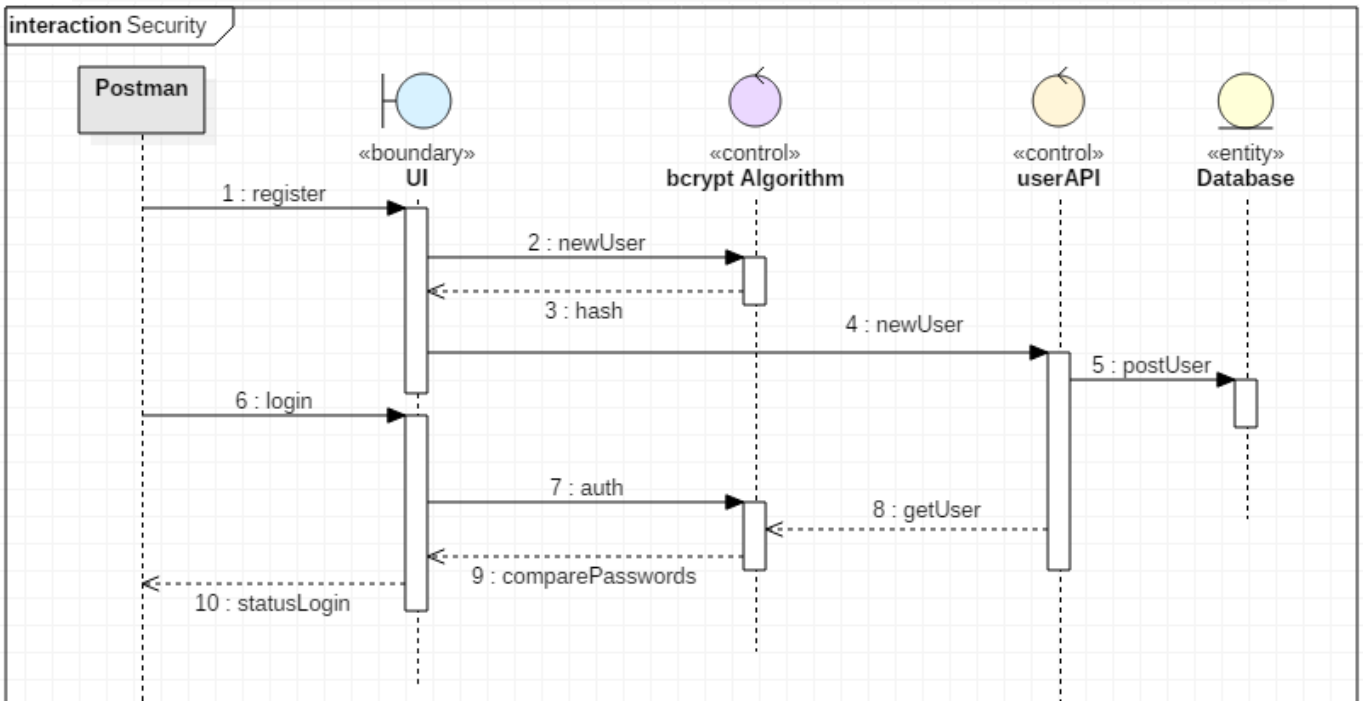
4.2 **Security mechanism**

The Security Mechanism is about not providing any sensible information to any attempted intrusion. We choose to encrypt the password by using « Bcrypt ». Bcrypt is a password hashing function, it is resistant to brute-force search and it can be implemented in Java, Node and other languages.

Once the user has created an account and input its password, the database automatically receive the password hashed by the server. This way, if any intrusion is made on the database, the password is encrypted and can't be used.

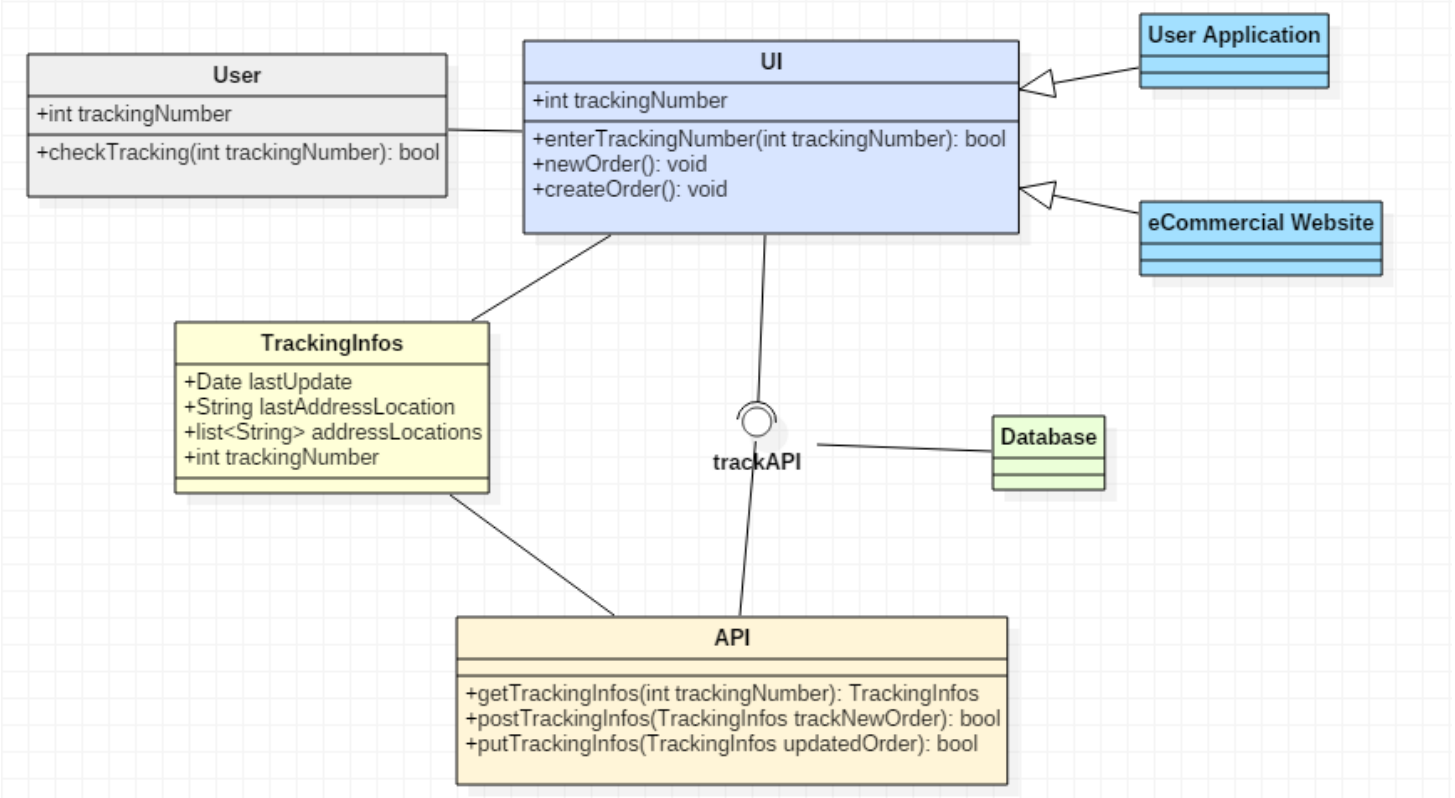
When the user is logging in, the password inputted is encrypted and compared with the one inside the database. If it matches, it means the password is correct, and the authentication can proceed.

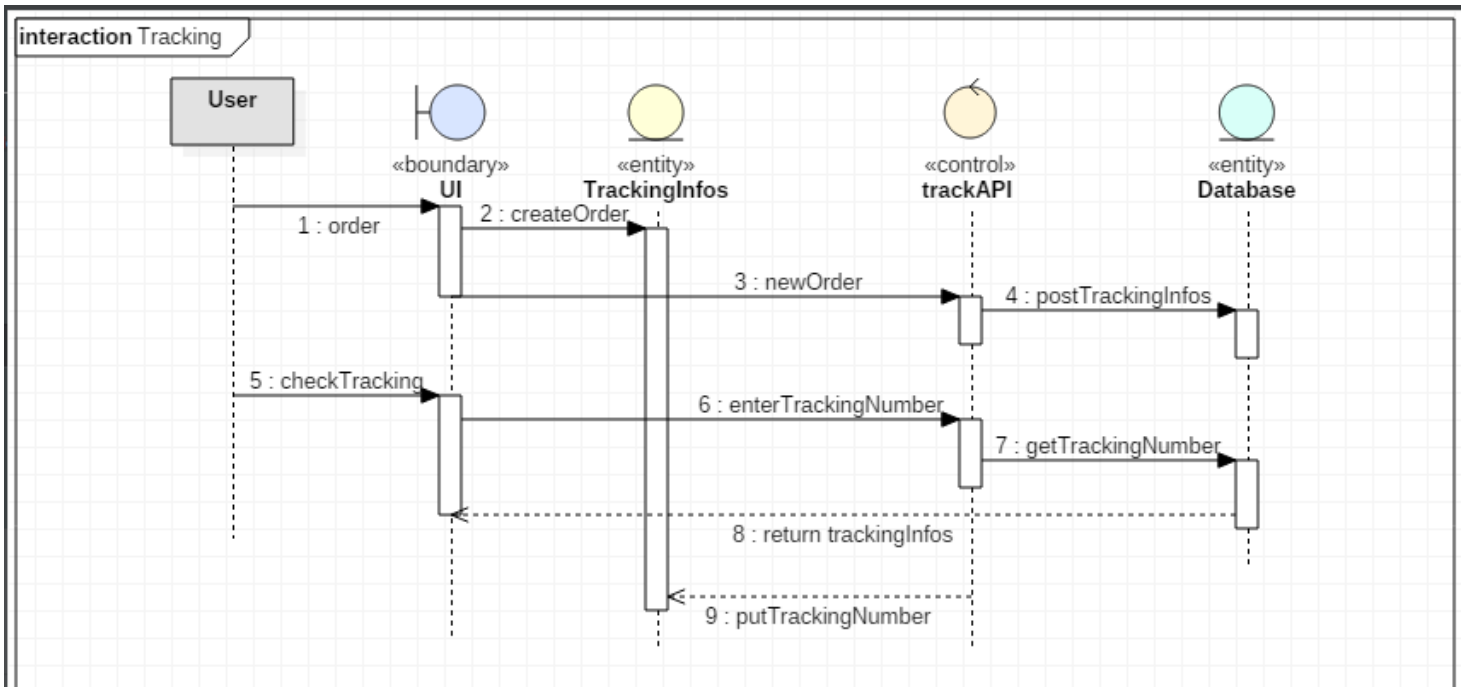




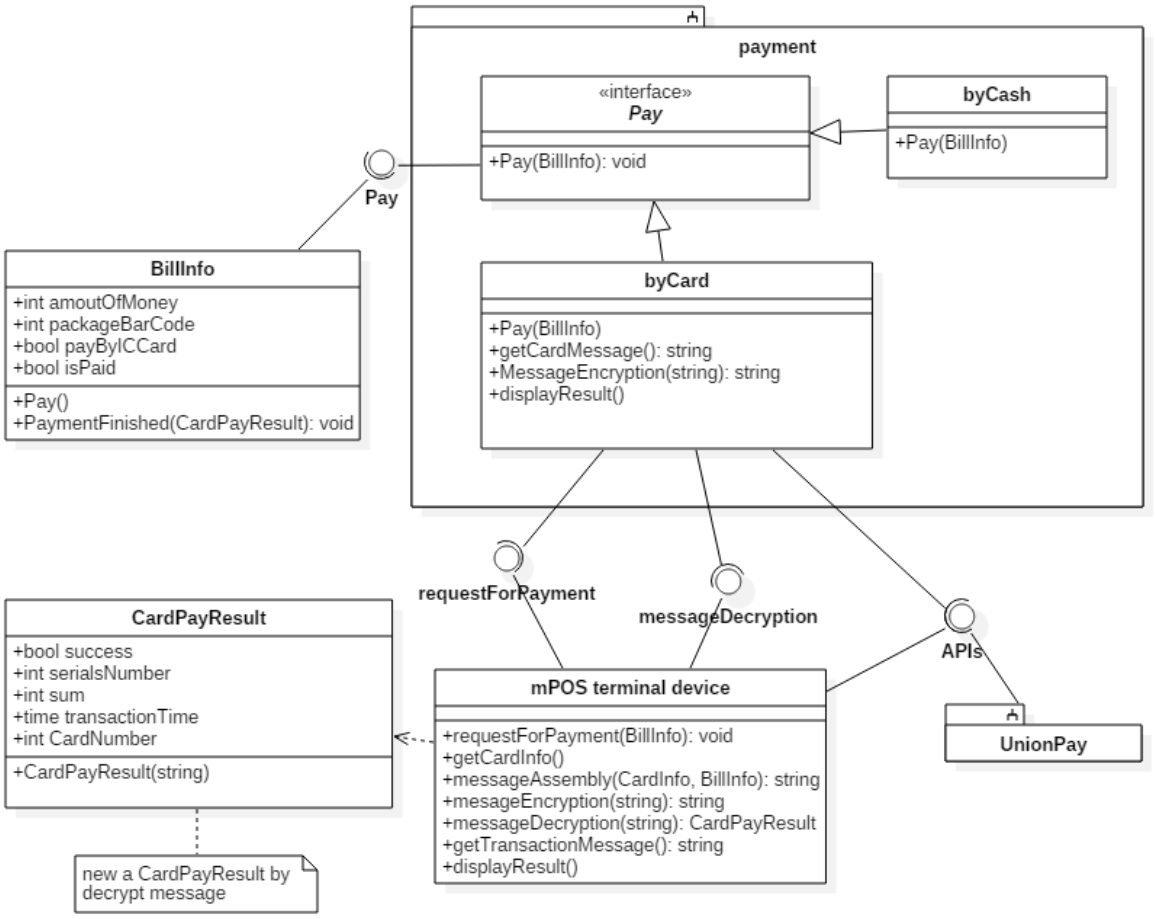
5 Use case realization

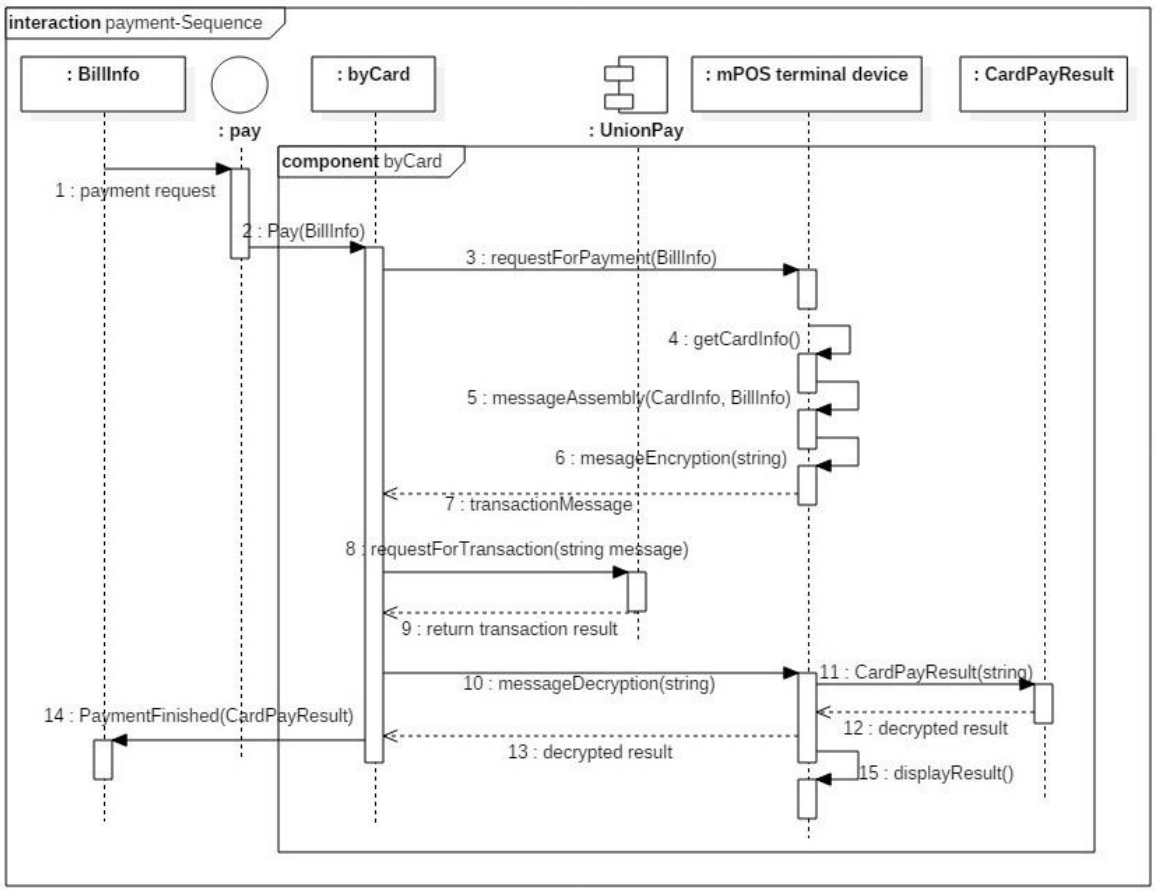
5.1 Tracking





5.2 **Payment**


payment
«interface»
Pay
+Pay(BillInfo): void
byCash
+Pay(BillInfo)
Pay
BillInfo
+int amoutOfMoney
+int packageBarCode
+bool payByICCard
+bool isPaid
+Pay()
+PaymentFinished(CardPayResult): void
byCard
+Pay(BillInfo)
+getCardMessage(): string
+MessageEncryption(string): string
+displayResult()
requestForPayment
messageDecryption
APIs
CardPayResult
+bool success
+int serialsNumber
+int sum
+time transactionTime
+int CardNumber
+CardPayResult(string)
mPOS terminal device
+requestForPayment(BillInfo): void
+getCardInfo()
+messageAssembly(CardInfo, BillInfo): string
+mesageEncryption(string): string
+messageDecryption(string): CardPayResult
+getTransactionMessage(): string
+displayResult()
UnionPay
new a CardPayResult by decrypt message


interaction payment-Sequence
: BillInfo
: pay
: byCard
: UnionPay
: mPOS terminal device
: CardPayResult
component byCard
1 : payment request
2 : Pay(BillInfo)
3 : requestForPayment(BillInfo)
4 : getCardInfo()
5 : messageAssembly(CardInfo, BillInfo)
6 : mesageEncryption(string)
7 : transactionMessage
8 : requestForTransaction(string message)
9 : return transaction result
10 : messageDecryption(string)
11 : CardPayResult(string)
12 : decrypted result
13 : decrypted result
14 : PaymentFinished(CardPayResult)
15 : displayResult()

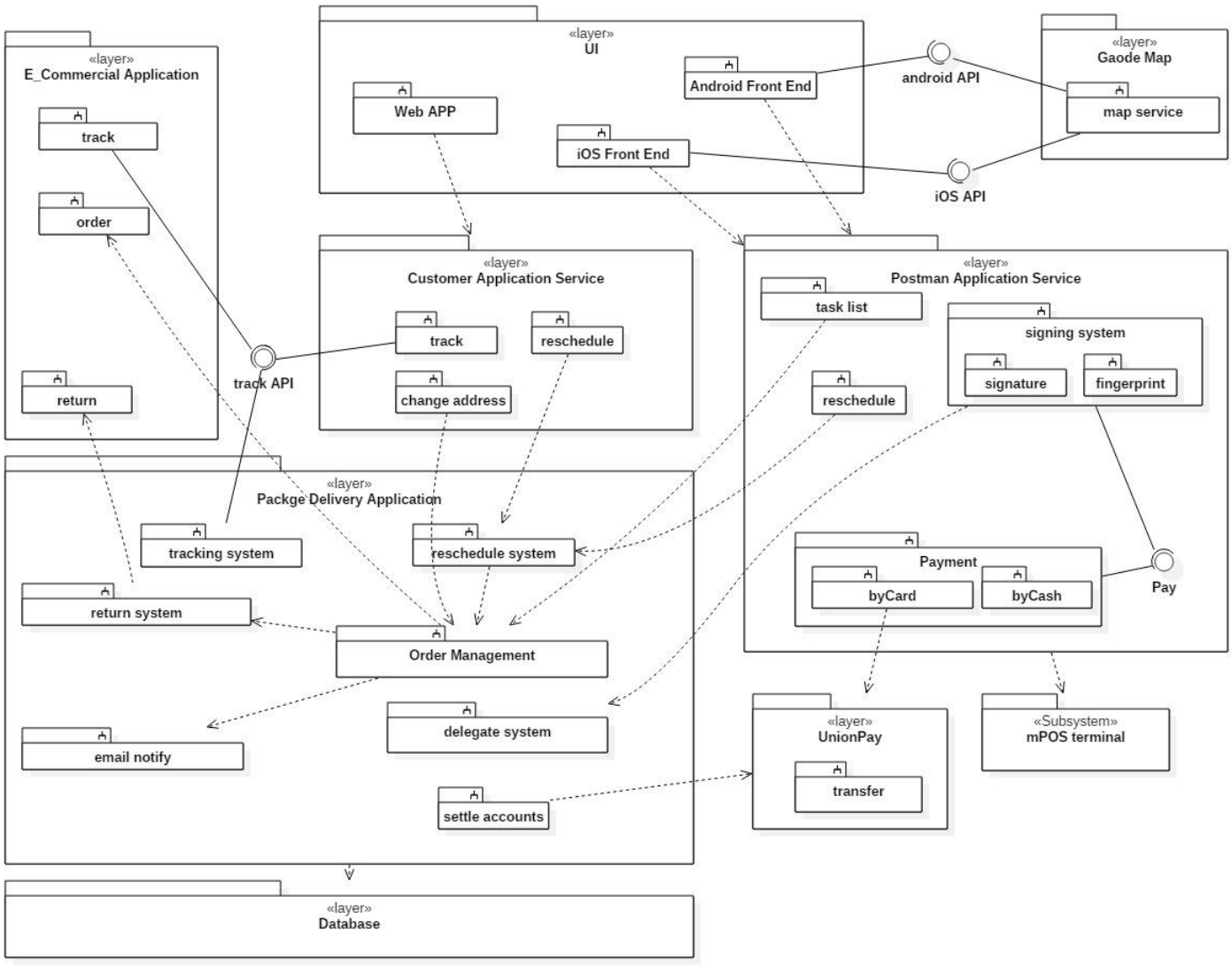
6 Progress of prototyping

Here we show the prototyping of Gaode Map location in Android.

```
//declare mLocationOption object
public AMapLocationClientOption mLocationOption = null;
mlocationClient = new AMapLocationClient(this);
//initialize
mLocationOption = new AMapLocationClientOption();
//set location listener
mlocationClient.setLocationListener(this);
mLocationOption.setLocationMode(AMapLocationMode.Hight_Accuracy);
mLocationOption.setInterval(2000);
mlocationClient.setLocationOption(mLocationOption);
//start location
mlocationClient.startLocation();
@Override
        public void onLocationChanged(AMapLocation amapLocation) {
                if (amapLocation != null) {
    if (amapLocation.getErrorCode() == 0) {
    amapLocation.getLocationType();
    amapLocation.getLatitude();
    amapLocation.getLongitude();
    amapLocation.getAccuracy();
    SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    Date date = new Date(amapLocation.getTime());
    df.format(date);
} else {
        Log.e("AmapError","location Error, ErrCode:"
        + amapLocation.getErrorCode() + ", errInfo:"
        + amapLocation.getErrorInfo());
    }
}
```

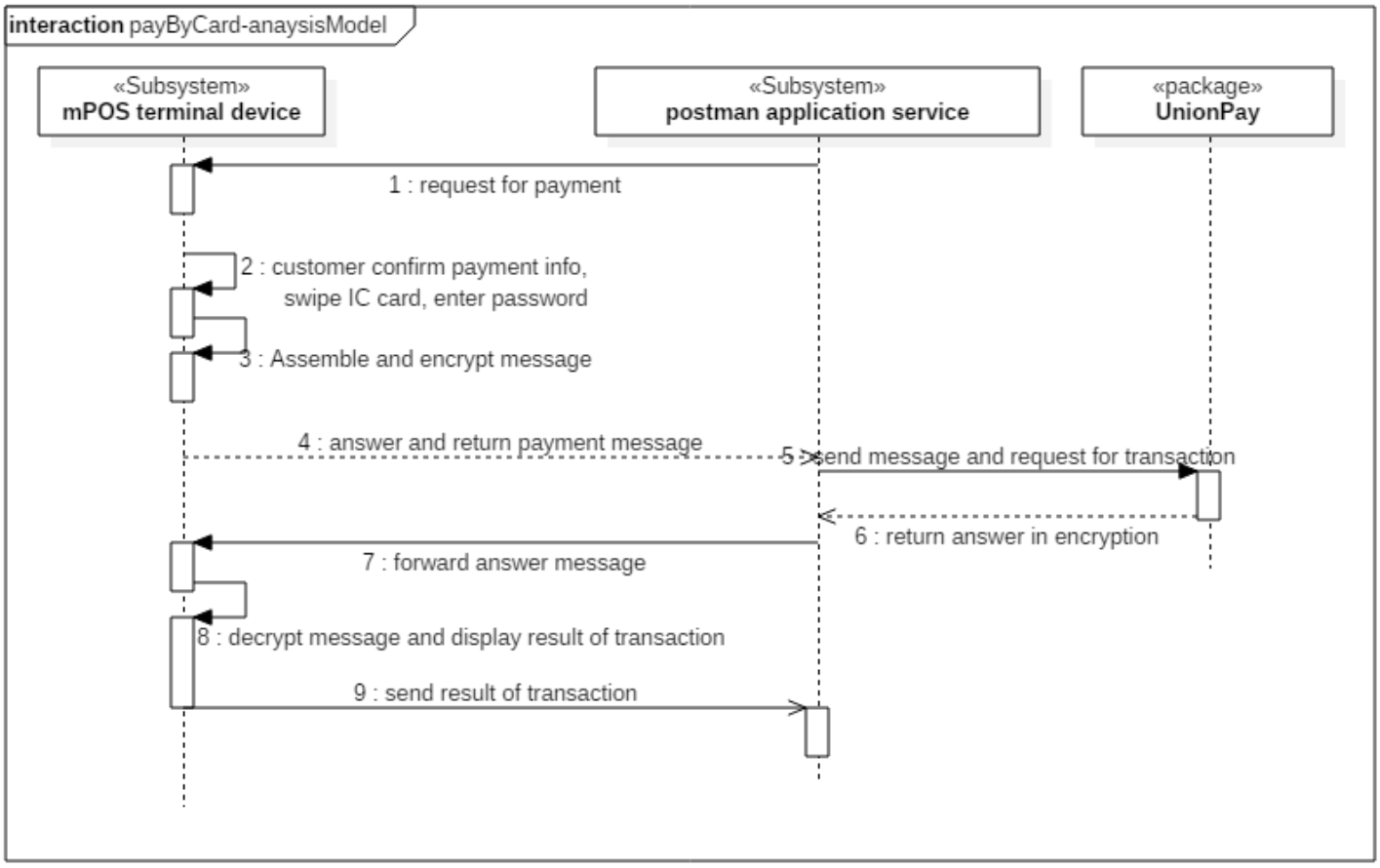
```
}
```

7 Updates of previous works



In the high-level architecture diagram in analysis model, the [**Payment**] subsystem is moved from **Package Delivery Application** to **Postman Application Service**, so that it will be deploy in smart phones and communicate with **UnionPay** directly.

And we enrich the sequence diagram of card payment in analysis model.



8 Contributions of team members

1452764　何冬怡(Leader)

Document summary and arrangement;
Payment use case realization;
Progress of prototyping
Updates of previous works

1452798　李　想

Architecture structure
(package diagram & deployment diagram);

1452697　彭嘉琦

List of subsystems and interfaces;
Detailed description of two interfaces
(Location and tracking)

1593369　Luc Berro

Design mechanism specifications
(User session & security mechanism);
Tracking use case realization;

9 Enclosure

With URLs below to view our **Package diagram** and **Deployment diagram**:

Package diagram　http://www.processon.com/view/link/5734b3cae4b0d4e09767820b
Deployment diagram　http://www.processon.com/view/link/5734b45fe4b0a43bbdd89e6b

Other models　/architecture.mdj