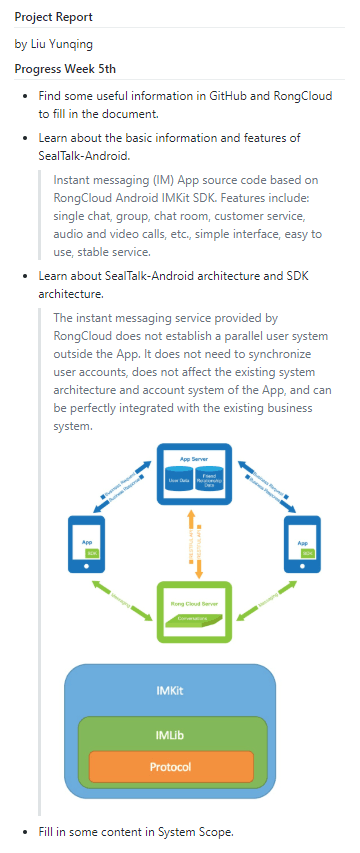
**Progress Report**

**——sealtalk-android**

**刘云清**：



**徐楚玉**：

After choosing SealTalk as our team's project, my main achievement at present is to write the introduction part of the project report. In order to accomplish my assignment, I searched a lot of relevant information on the Internet and read SealTalk 's official documents in detail about the functions of this product at first. It makes me have a general understanding of SealTalk. Also it is necessary to know the technical side. I tried to read the source code of the project and got a general idea of the structure of the code. But my understanding of the source code is still very limited at present. I will try to understand the source code as thoroughly as possible later.

**段宇昕**：

Understand SealTalk project application scenarios and basic development information

Understand the information of SealTalk project company -- RongCloud, including its partners, competitors, etc.

Reading the handout material “Delft Student on Software Architecture DESOSA 2017” and study the reports of Arduino and Gradle

Finish the initial analysis of stakeholders and power vs. interest.

**李祖鑫**：

- Find Users, Acquirers, Suppliers, Developers, Competitors, Maintainers as stakeholders in sealtalk-android.

- Fill in some content in stakeholders

- Find some useful information in github and RongCloud mainpage to fill in the document.

**方焱志**：

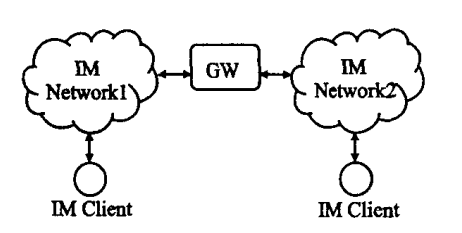
- Have some insight into Instant Messaging

- Analyze packages in the source code of this project

- Learn about the Context View

- Find out the external entities it interacts with

The project we chose is [sealtalk-android](https://github.com/sealtalk/sealtalk-android). This is an Instant Messaging(即时通信) app, such as QQ and Skype, all of which are IM software. Therefore, I first learned about instant messaging. Instant messaging is mostly based on the following four communication protocols: the first is an instant messaging and space based protocol, IMPP; the second is a space and instant messaging based protocol, PRIM; and the third is based on attendance extensions and instant messaging sessions, SIP; the fourth is based on the extensible message and presence protocol, XMPP. Among the above four mainstream communication protocols, the XMPP protocol has relatively outstanding scalability and flexibility.

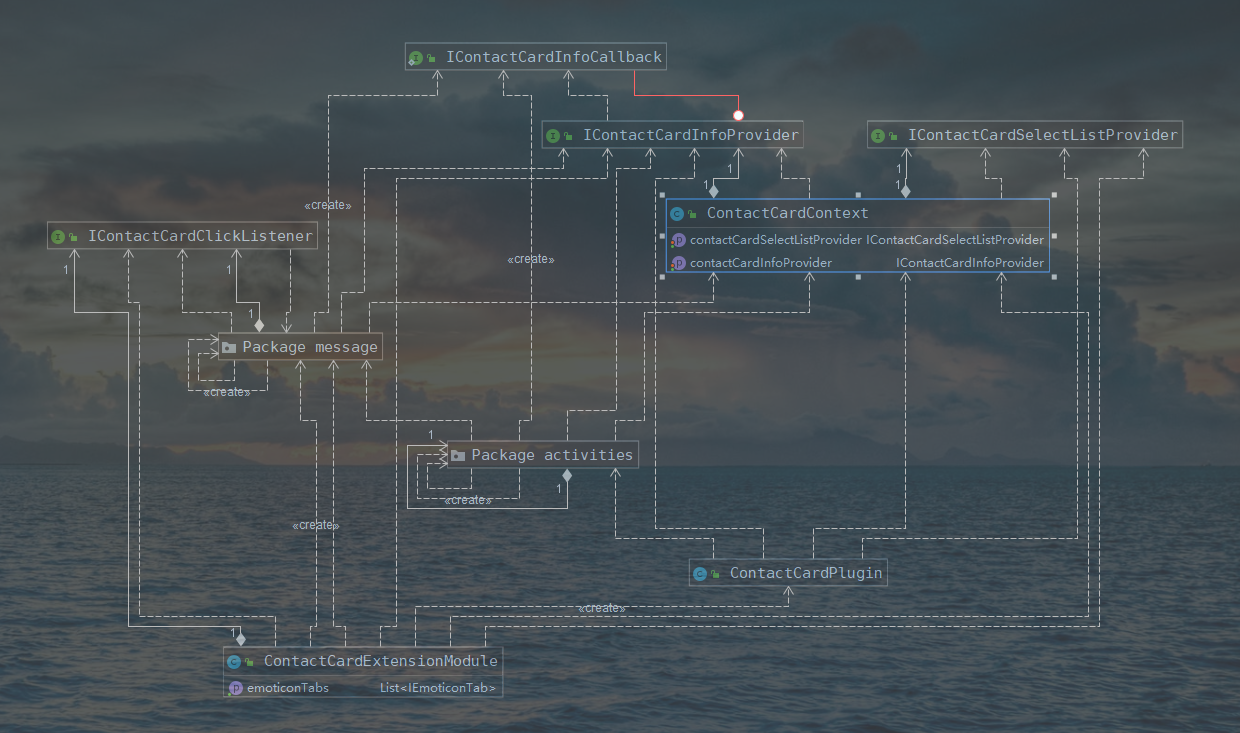


At the same time, instant messaging platforms usually have the following non-functional requirements: 1. **Scalability**: Developers should pay attention to system scalability, reserve enough upgrade interfaces, and embody in the system design architecture to meet the future of the system. The need for function upgrade or architecture optimization. 2. **Error log**: developers should pay attention to the automatic generation and real-time alarm of the application, in order to facilitate system debugging and maintenance upgrade. 3. **Data Transmission security**: In the process of system development, certain encryption technologies must be adopted to ensure the absolute security of the user's sensitive private data after being put into use.

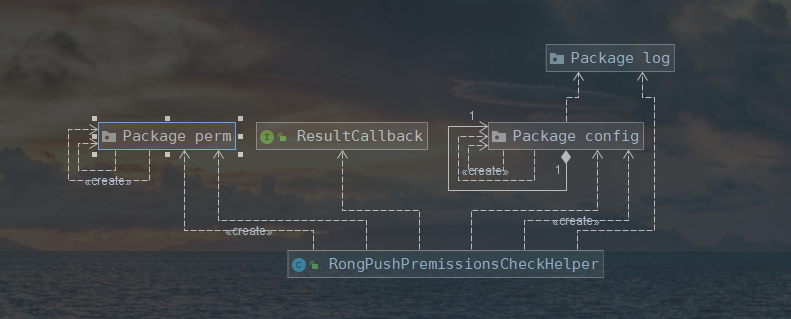
Since the sealtalk project is based on C/S implementation, the points mentioned above are mainly reflected in the [sealtalk-server](https://github.com/sealtalk/sealtalk-server/blob/master/src/util/n3d.js) (based on Node.js). For example, [util/n3d.js](https://github.com/sealtalk/sealtalk-server/blob/master/src/util/n3d.js) is the encapsulation of the encryption function, which corresponds to the third point of the above non-functional requirements. Here, the description of the server side will not be described in details.

After a general understanding of some of the principles of instant messaging, it is now moving to its Android-side architecture. The project consists of four main modules:

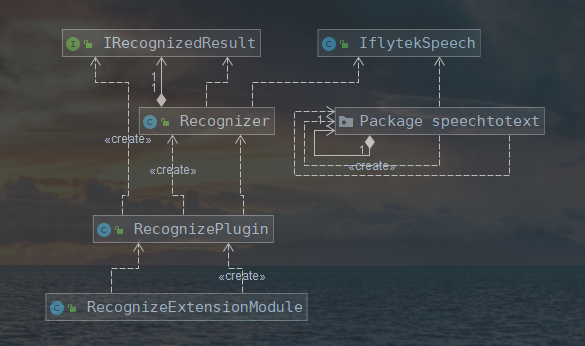
1.contactcard：Contact card



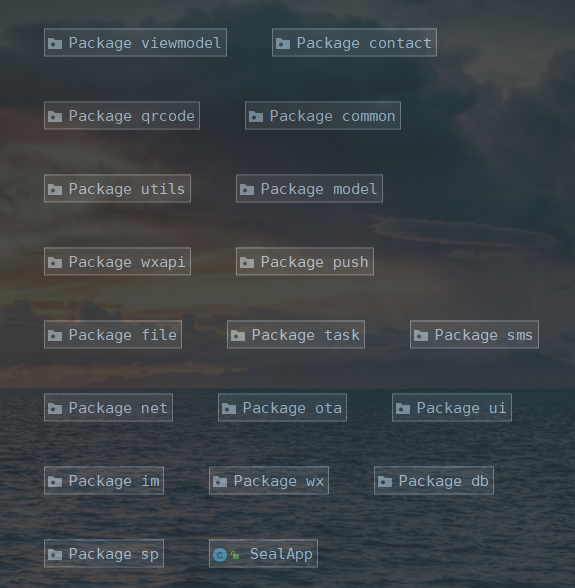
2.pushpermission：Check push permissions



3.recognizer：Speech recognition from Third Party（科大讯飞）



4.sealtalk：app

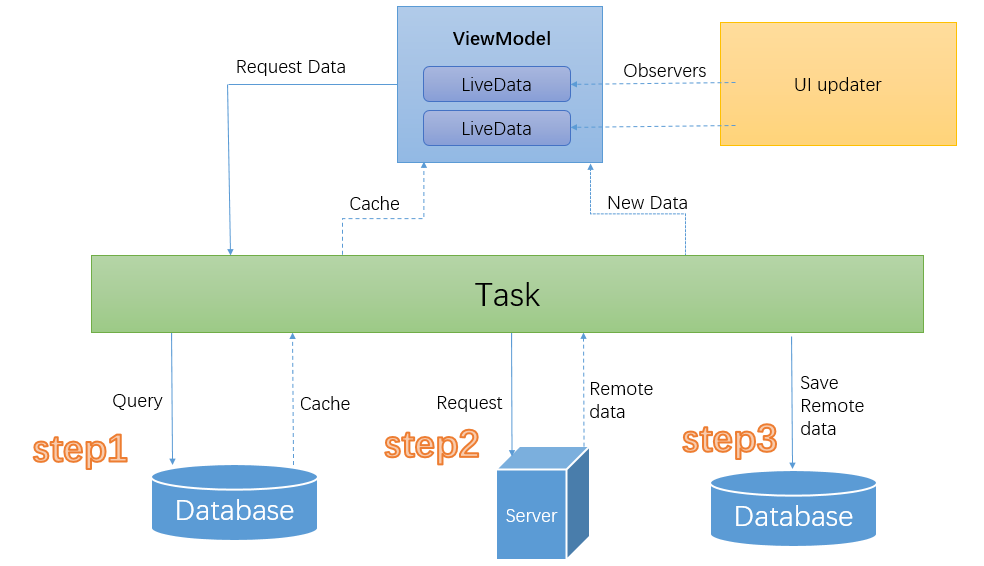


Obviously, we should focus on the main part of the app.

The following table shows the function corresponding to each package:

|  |  |
| --- | --- |
| Package Name | Function |
| common | Encapsulate url, error code and other constants |
| contact | Phone contact management |
| db | Database management, using the **Room** framework |
| im | Instant messaging implementation |
| model | Data layer, providing various data services for the upper layer |
| net | Encapsulate network request operations, using the **Retrofit** framework |
| ota | Online updating tools |
| viewmodel | Bridging View and Model, data returned via LiveData |
| sms | Short message service |
| task | Different Tasks are packaged according to different interfaces or data attributes |
| ui | The View layer |

The relationship between viewmodel/task/ui is as follows:



Among them, ViewModel's LiveData follows the **observer pattern**. LiveData will notify the Observer when the lifecycle state changes. You can merge the code that updates the UI into these Observer objects. Every time the data changes, Observer will update the UI.

Note that here in the Task layer, the database will be queried first, then the cached data in the current database will be returned, and then the server will be requested to obtain the latest data, and then the new data will be saved to the database. Finally, a database query will be performed to obtain the latest data in the database. The code corresponds to the following (task/ FriendTask.java):

