<BTree>

<BeeKey>
Software Architecture Document

Version <1.1>

Revision History

Date	Version	Description	Author
16/07/2021	1.0	Introduction, goal and constraints Use-case model Logical view (Class diagram, Component diagram)	Hoàng Như Thanh Chung Kim Khánh Bùi Đăng Khoa Đỗ Vương Phúc
26/07/2021	1.1	Revise all the previous system Additional the deployment and implementation view	Đỗ Vương Phúc

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides an intensive and comprehensive architecture overview of the BeeKey mobile application

1.2 Scope

This document provides an intensive and comprehensive architecture overview of the software using various architectural views that describe separated components in detail. It depicts different aspects of the software and conveys significant architectural decisions to developers and non-developers.

1.3 Definitions, Acronyms and Abbreviations

See the Glossary document.

1.4 References

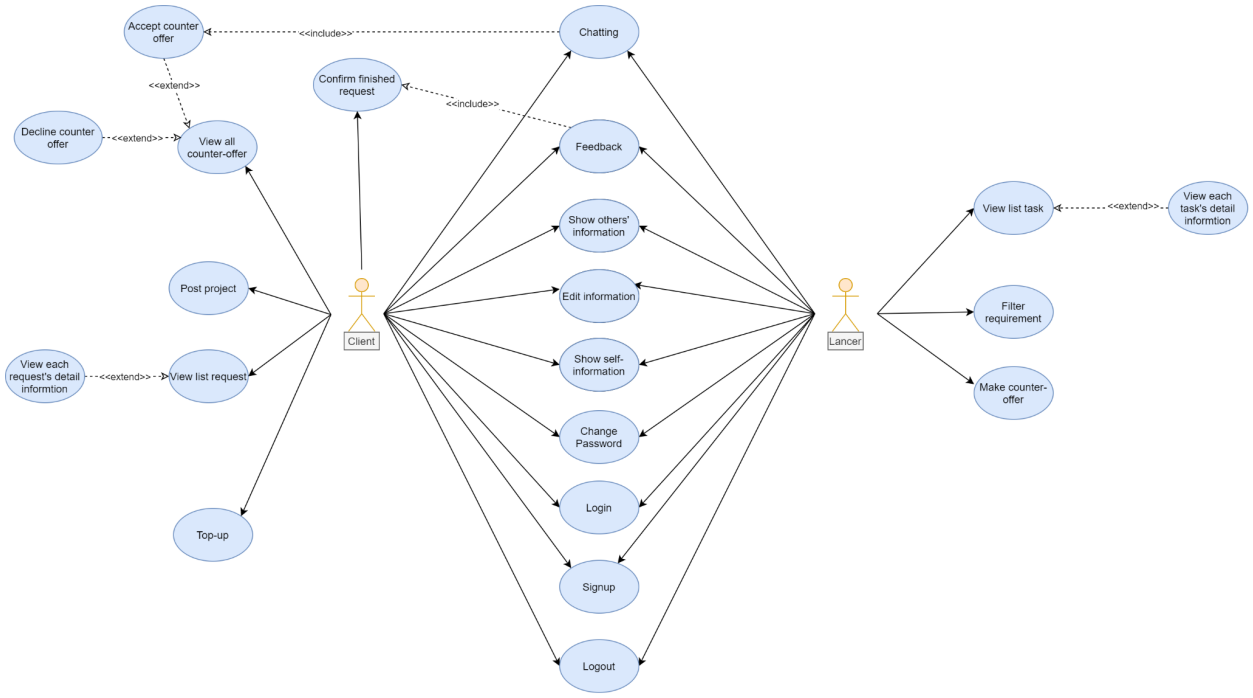
- Vision document
- Use-case specification document

2. Architectural Goals and Constraints

There are many software requirements and constraint that have significant impact on the architecture pointed out as following:

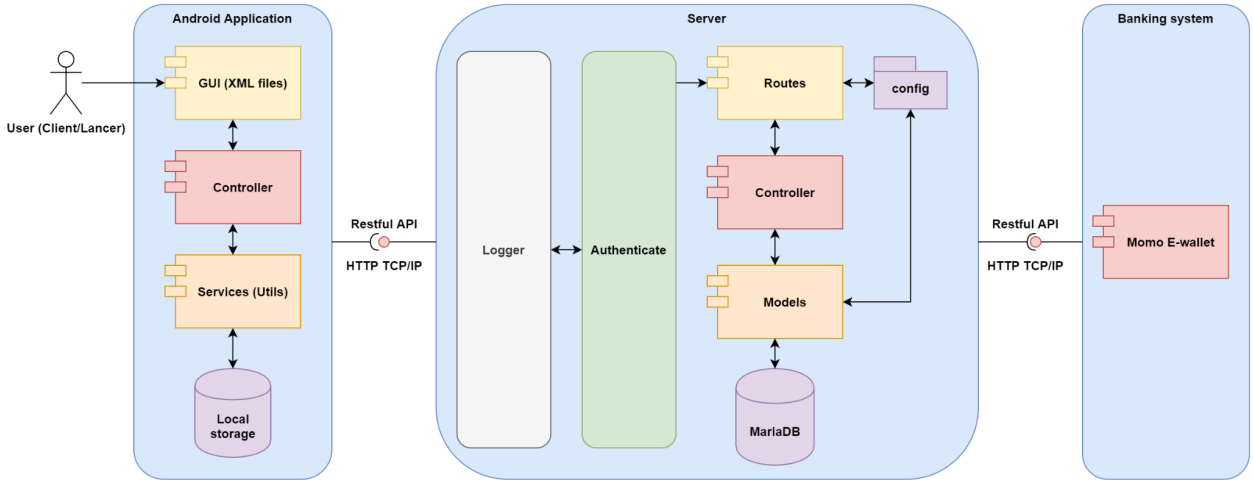
- The BeeKey mobile application is based on a client-server model. The application resides on personal mobile running at least Android 6.0. The BeeKey mobile application will be deployed on the dedicated 24/7 server on the Raspberry Pi running Ubuntu.
- The BeeKey mobile application is transactional. Therefore, the legal existing banking system must be interfaced with in order for users to make transactions.
- The system must be secured in order for users to make online transactions which results in the compulsory of logging in with username and password before being able to use any further functions. Sensitive data must be encrypted (passwords, credit card payments).
- Every sensitive action must be logged and notified to users via email, including any access on other devices and changing email or password.
- Every non-functional requirement derived in the Vision Document must be considered as the architecture is being developed.

3. Use-Case Model



For the specification, see the use-case specification document.

4. Logical View



The system includes two large components: Android application (client) and Server. Users (both client and lancer) use the Android application. Clients and the server communicate through the internet and using HTTP TCP/IP protocol with the provided Restful API (specified in the API Document). In addition, our system is also linked with an out-bound banking system (Momo E-wallet).

On the Android application, we divide into 3-tiers:

- The highest tier is the GUI which represents the view for users to interact and provide the data for the controller.
- The controller tier manipulates the data received from users through the GUI tier and using

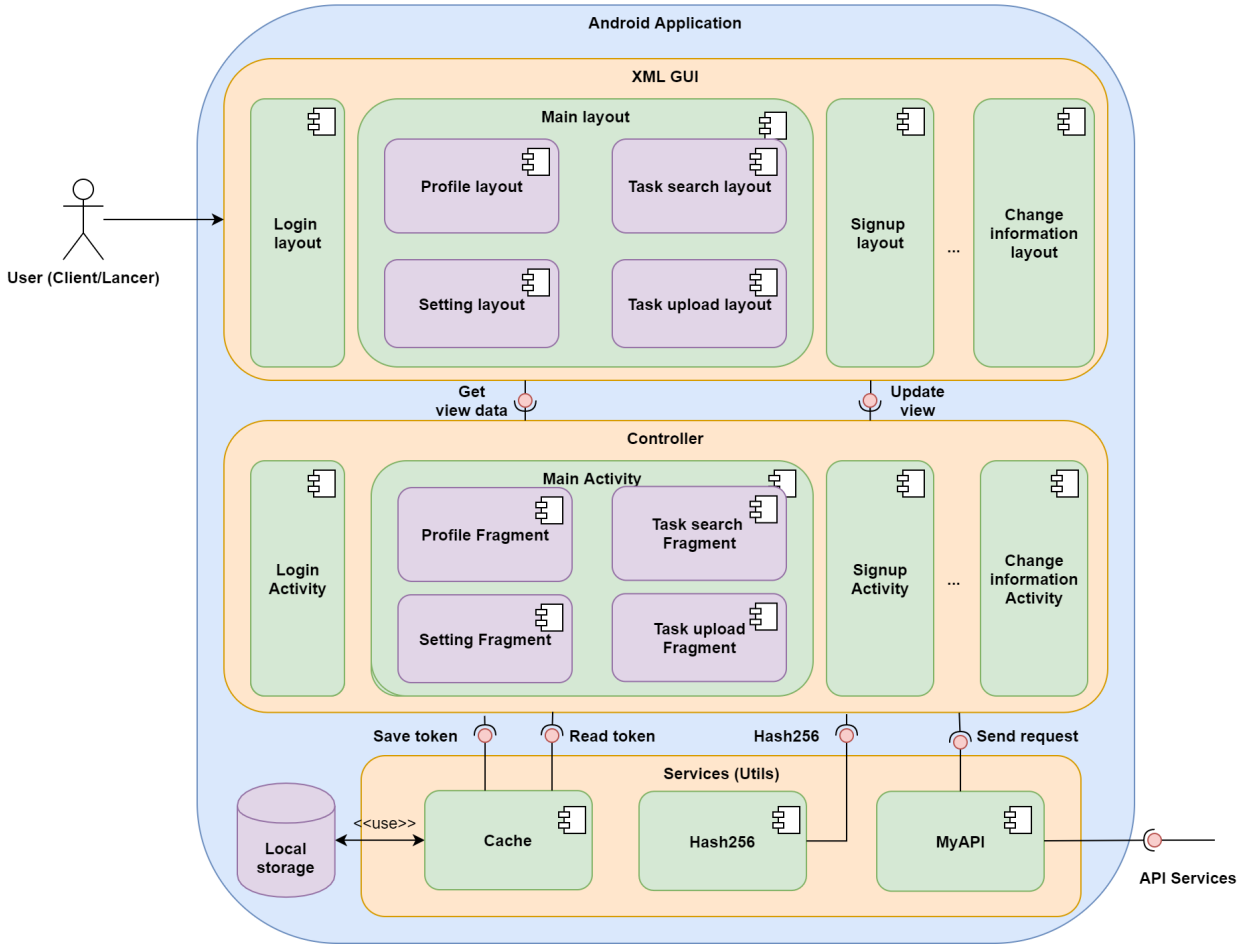
services tier to access local storage. Moreover, whenever it need to connect with server, it also call the services of API

On the server, we use the middleware architecture. It will have 5 components:

- The first one is to log down the information of the request.
- The second middleware which will check for the token of request if needed. Then it passes the request to the route middleware.
- The route middleware will route the request to the suitable business logic controller on the next middleware.
- When the request reaches the suitable controller middleware, the controller takes the required data and passes it to the models middleware. Moreover, when the model middleware has done its task, it gives back the result and the controller will send the response to the client.
- The model middleware provides services to query database

All middlewares of the server use the "config" module to get the constant configure.

4.1 Android application architecture

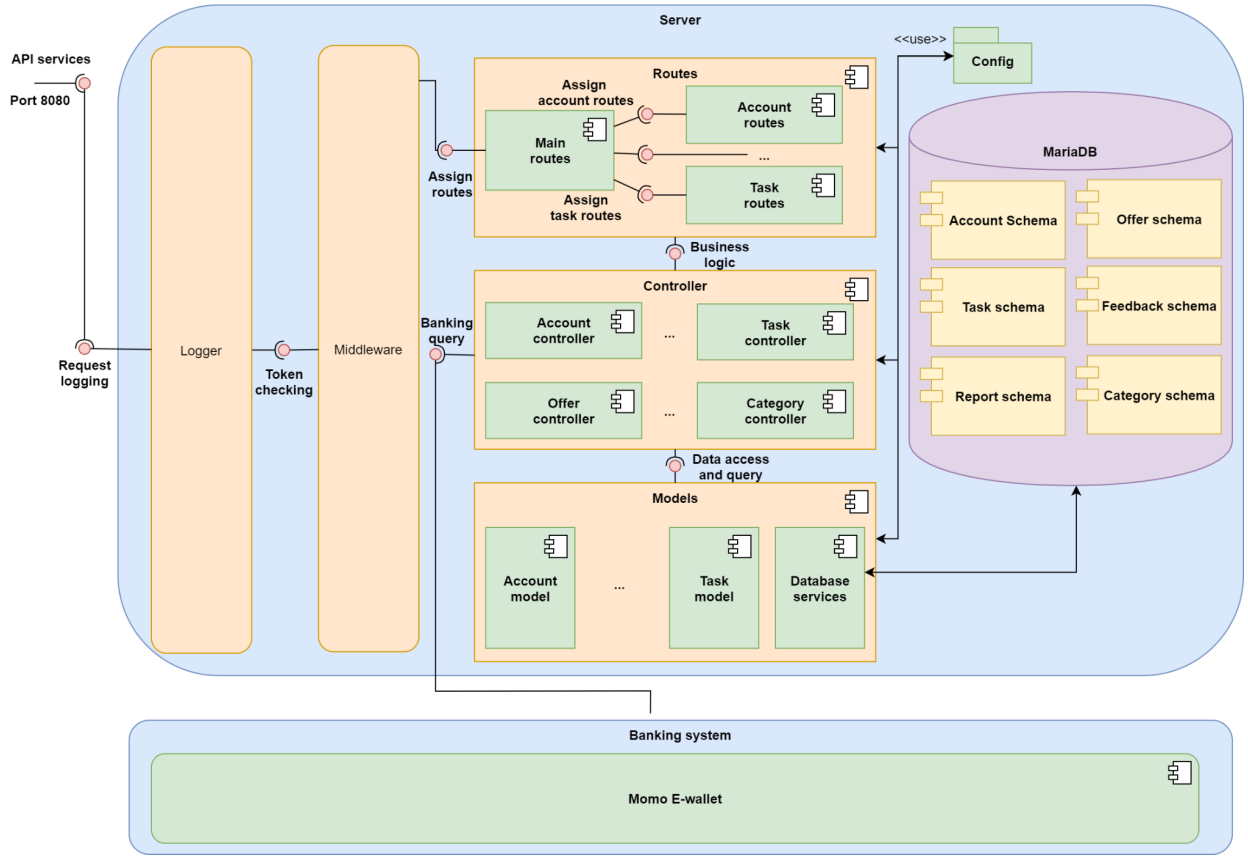


More detail about the client architecture, the software has the user interface which is represented by the GUI layer including many layouts. The first layer provides services to get the input data from the user and to update the view. The controller layer listens to the user interaction and processes the business logic. The service layer has threes main component:

- Cache: to read and write data to local storage
- Hash256: to do the encryption with Hash256 algorithm

- MyAPI: to communicate with server

4.2 Server architecture



On the server, it provides the Restful API services through HTTP TCP/IP protocol using port 8080. When a request is sent to the server, the first middleware will log down the request information for debugging.

The server uses the JSON Web Token (JWT) to authenticate the request from clients. The second middleware is to verify the token whenever it is needed.

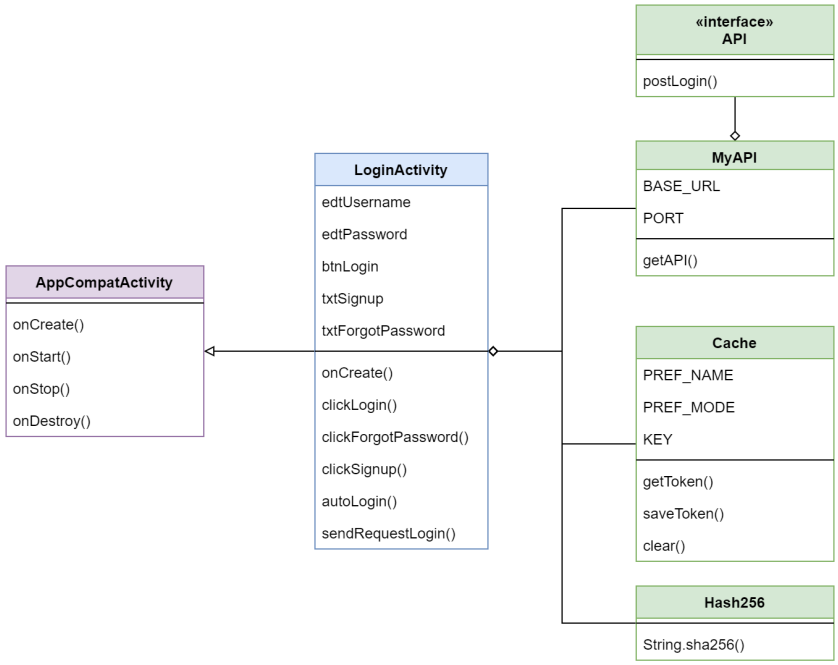
After checking the token, the next middleware will route the request to the correct controller. In the route middleware, we have one main route who is responsible for route requests. It uses the services provided by other routes components (Account routes, task routes, etc).

For each route, we have one controller used to do the business logic of our system: parse data, verify requests, pass data to the models middleware and respond to the client. It also uses the services provided by the Momo E-wallet system for users to top-up.

The model middleware has models corresponding to each schema we have on the database. Moreover, in the model middleware we also have the database utility service to access and query the database.

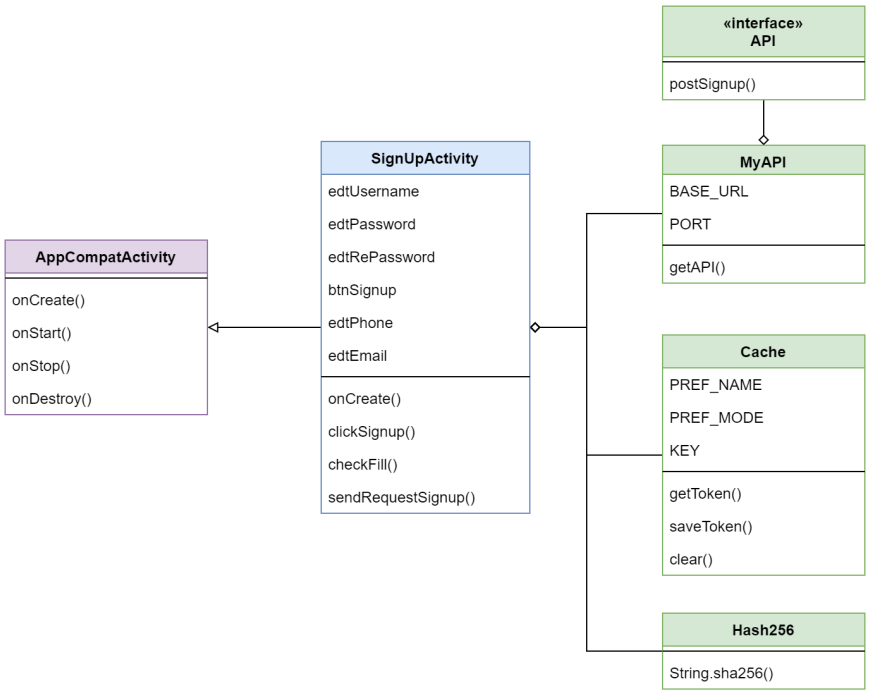
4.3 Class diagram

4.3.1 Component: Login activity



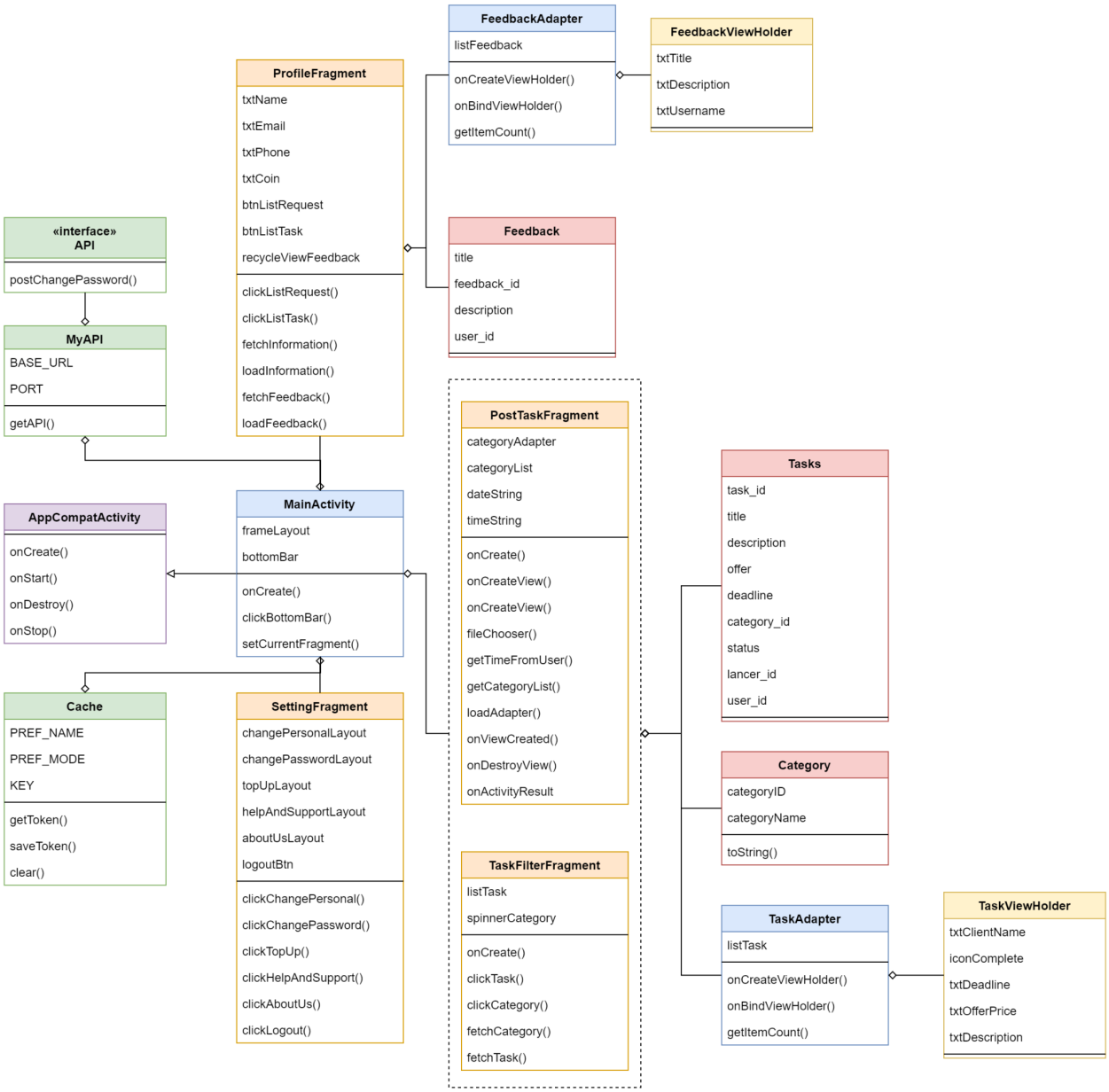
The Login activity is used for users to login by fill-in the username and password in the edit text component. Moreover, the user can change to signup activity. It uses utilities such as Cache, MyAPI and Hash256.

4.3.2 Component: Signup activity



The signup activity lets users enter information including username, password, re-enter password, phone, email to create a new account. When an account is created, the default display name is the same as username.

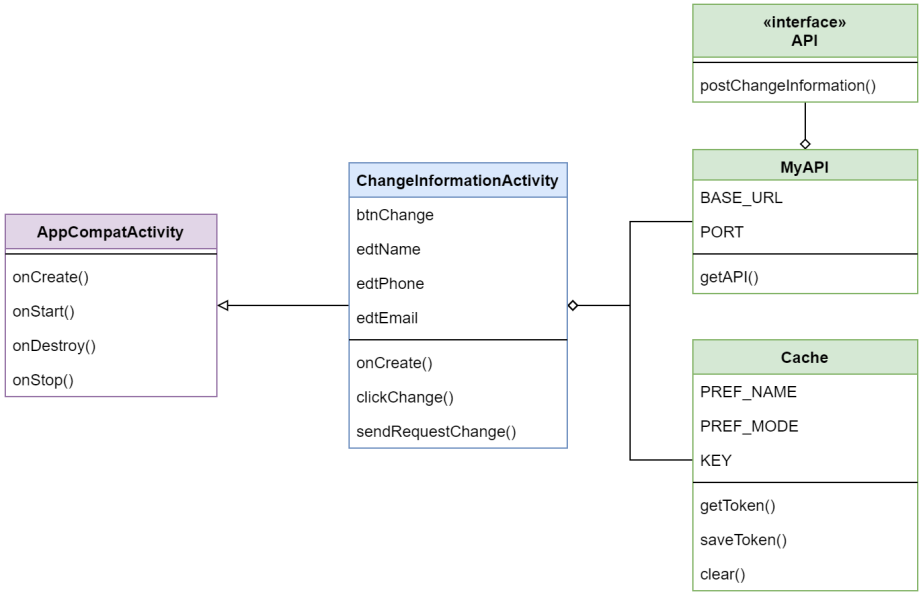
4.3.3 Component: Main activity



After login successfully, the application navigates users to the main activity. In the main activity, we use the bottom bar navigation component. On the main activity, we can navigate to four fragments view: Setting, Post new task, Profile and Search for a new task by filtering category.

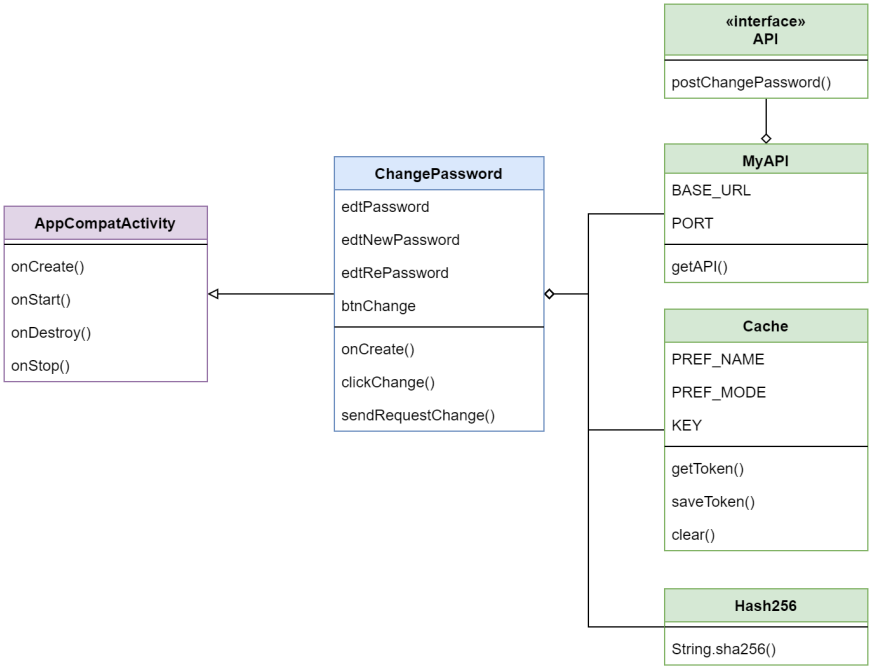
In the profile fragment, it displays the user information and feedback from other users. Because there is a lot of feedback, we use the recycler view to display which requires the adapter. The TaskFilterFragment and PostTaskFragment also use the list of Category and list of Tasks.

4.3.4 Component: Change information activity



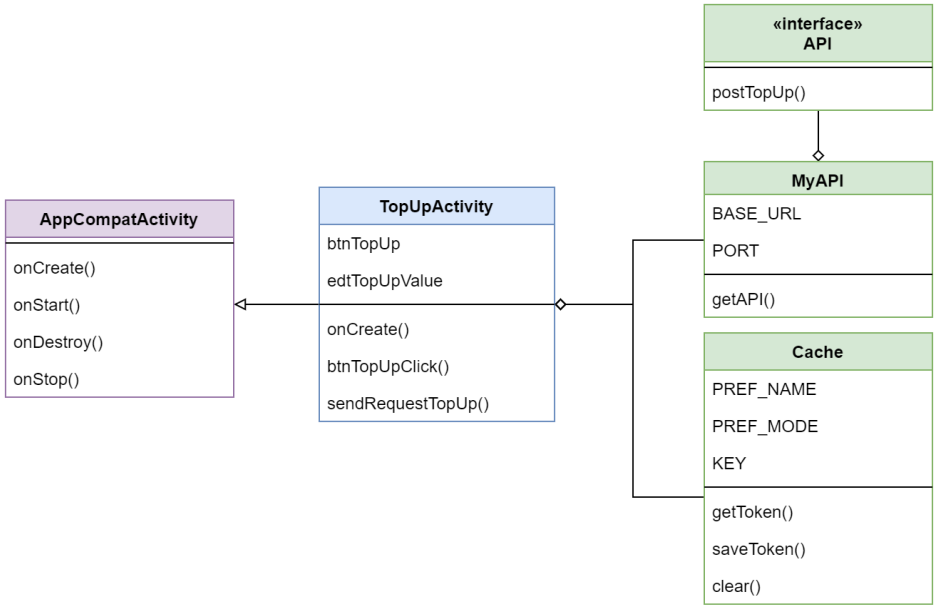
In the change information activity, it allows users to change their display name, phone and email.

4.3.5 Component: Change password activity



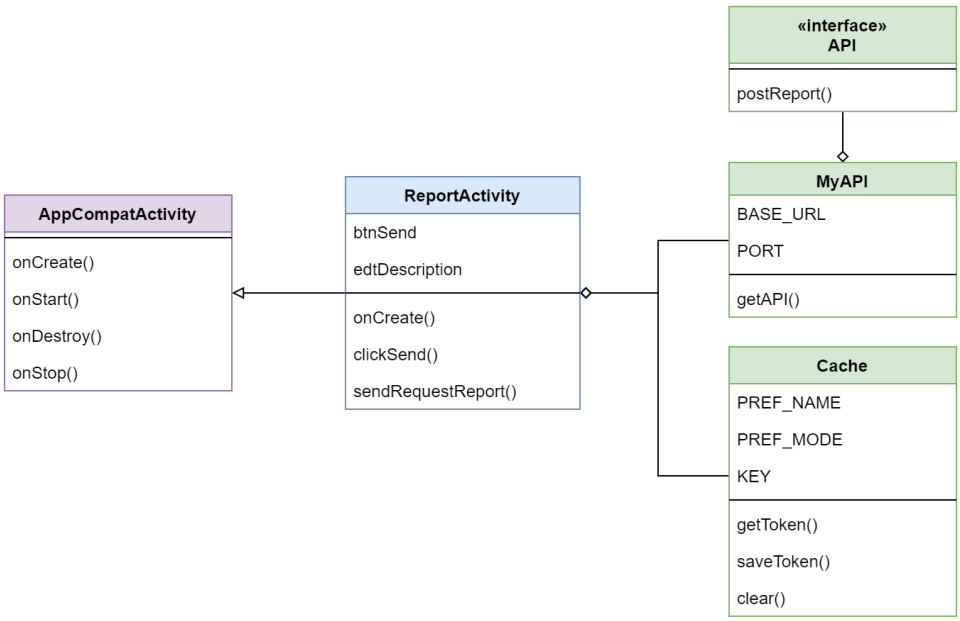
In the change password activity, users enter the old and new password to authenticate. Moreover, they must provide the new password again to ensure there is no mistake in typing.

4.3.6 Component: Top up activity



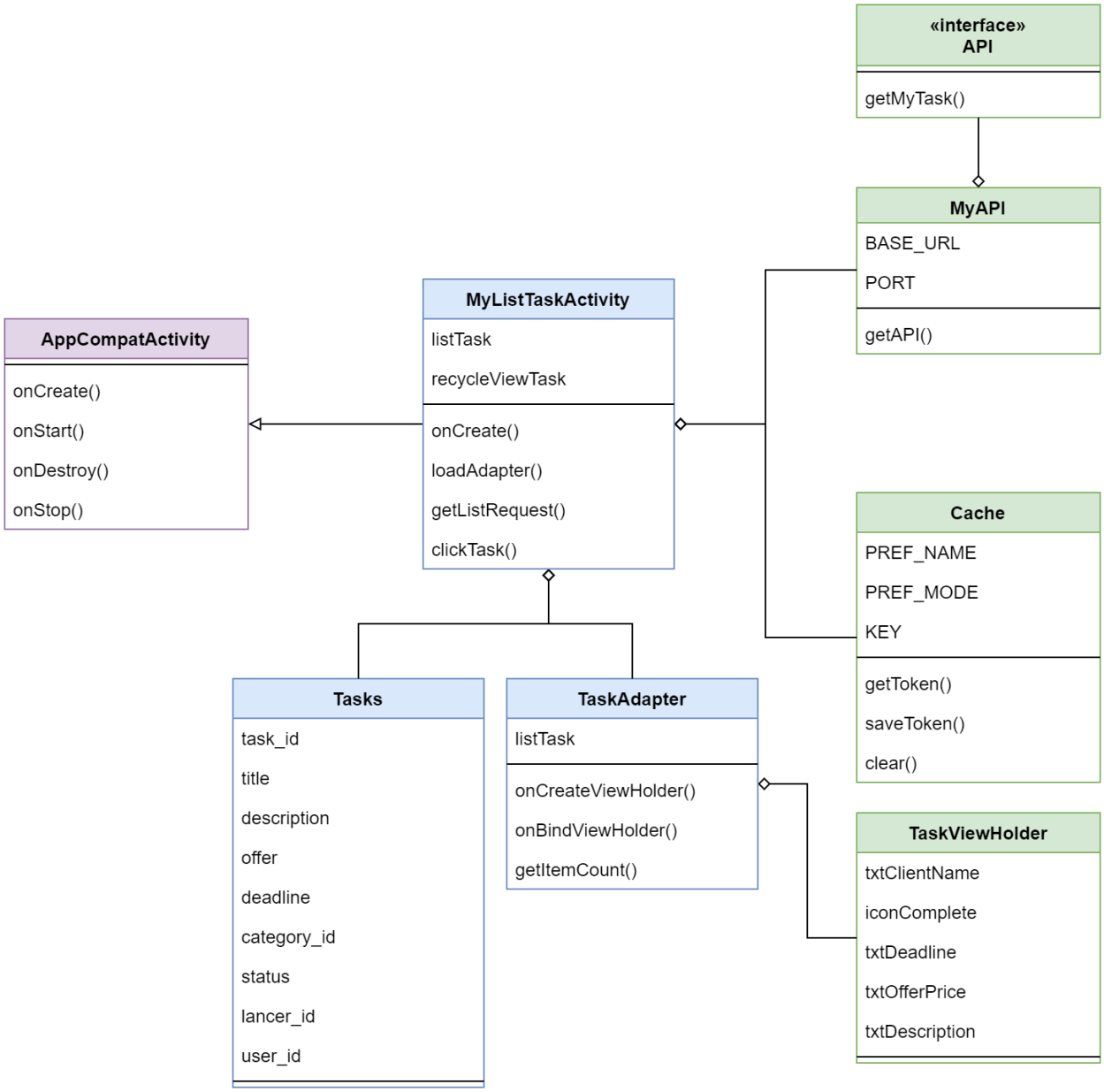
In this activity, users are able to top up the money to their account.

4.3.7 Component: Help and issue (report) activity



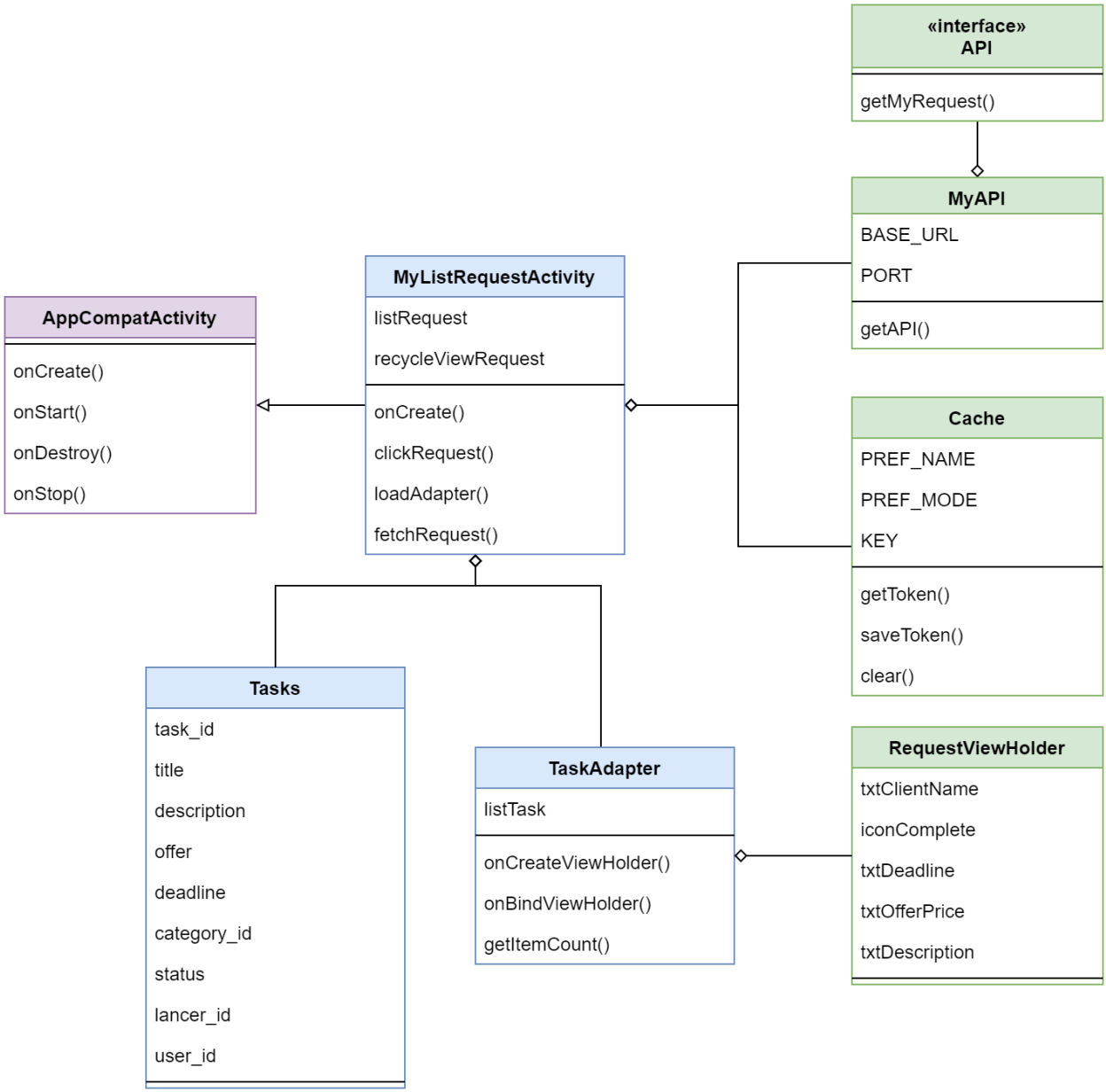
In this activity, users are able to make a report about any issue (Maybe the issue of the application or the problem between them and lancer while doing the task).

4.3.8 Component: View list task activity



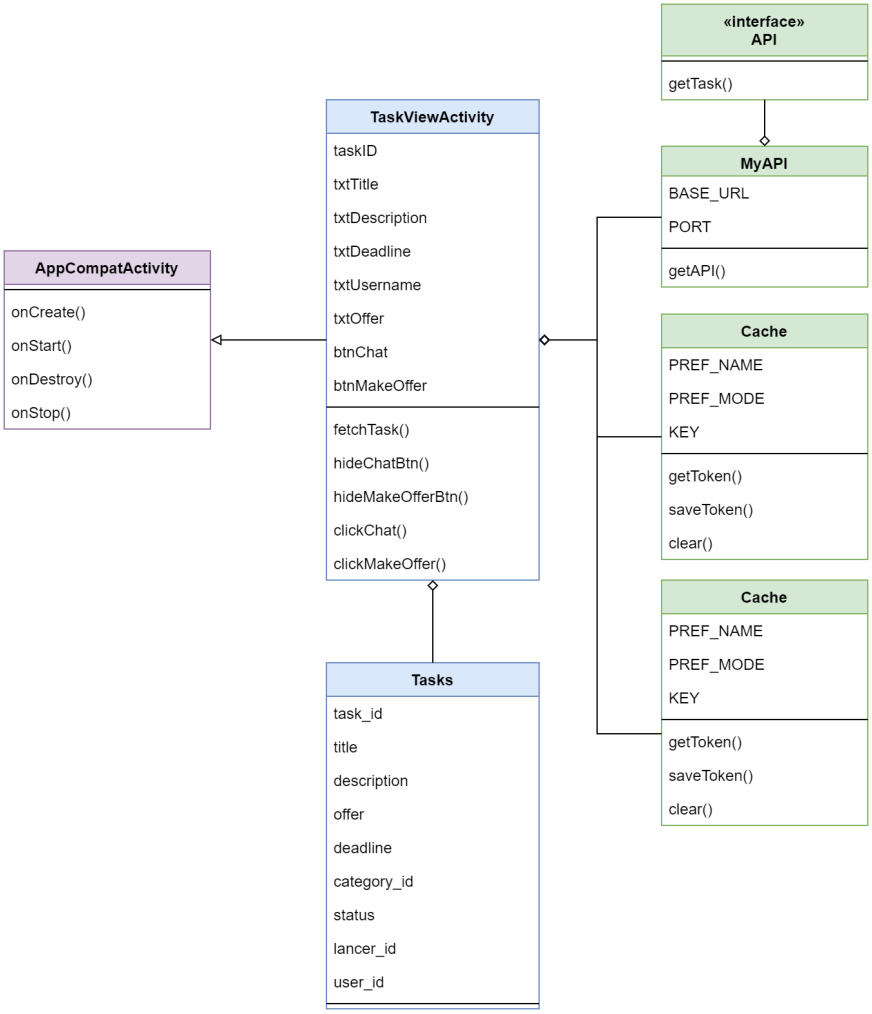
In the profile fragment, users can view their list of tasks and requests. Because of containing the list, the activity must have the adapter.

4.3.9 Component: View list request activity



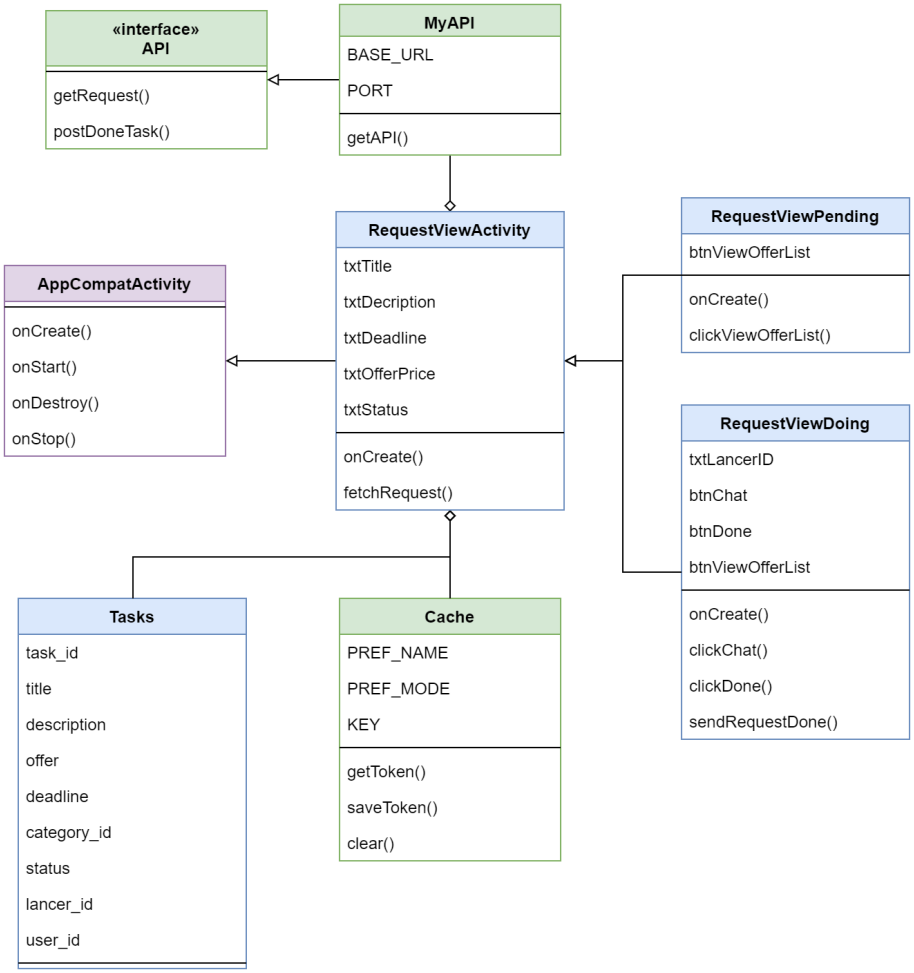
The list of requests is the same as the list of tasks. However, it has some differences, instead of finding the lancer name which is the same as the current account, it finds the uploader name.

4.3.10 Component: View task detail activity



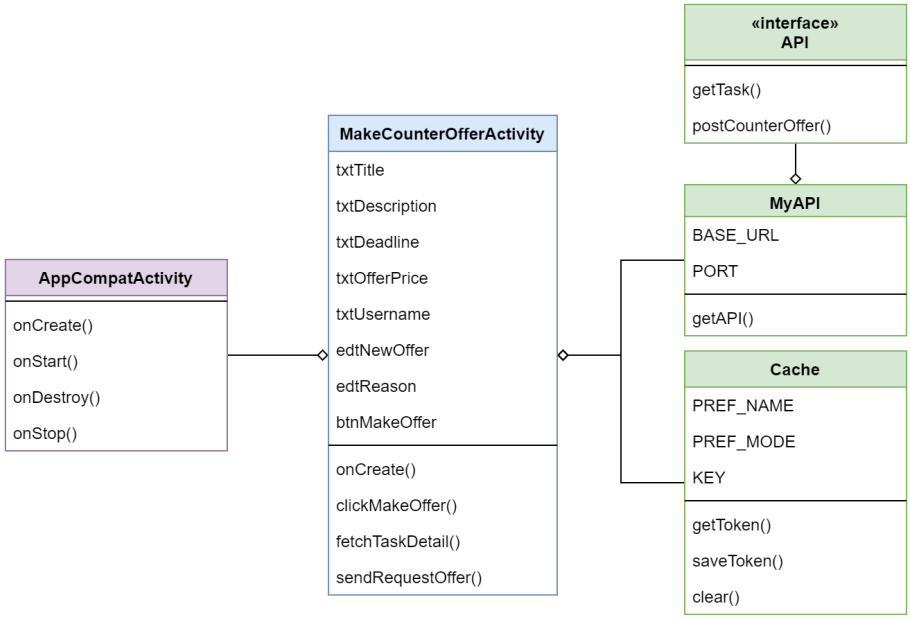
With a specified task, the user can view the details of it. Because the task has many status: Pending, Doing and Done, the activity must display correctly as its status. In the pending status, the user can only view the list of counter-offer. In the Doing status, the user can chat with the lancer who is doing the task. Lastly, when the status is done, they can only view the task in detail.

4.3.11 Component: View request detail activity



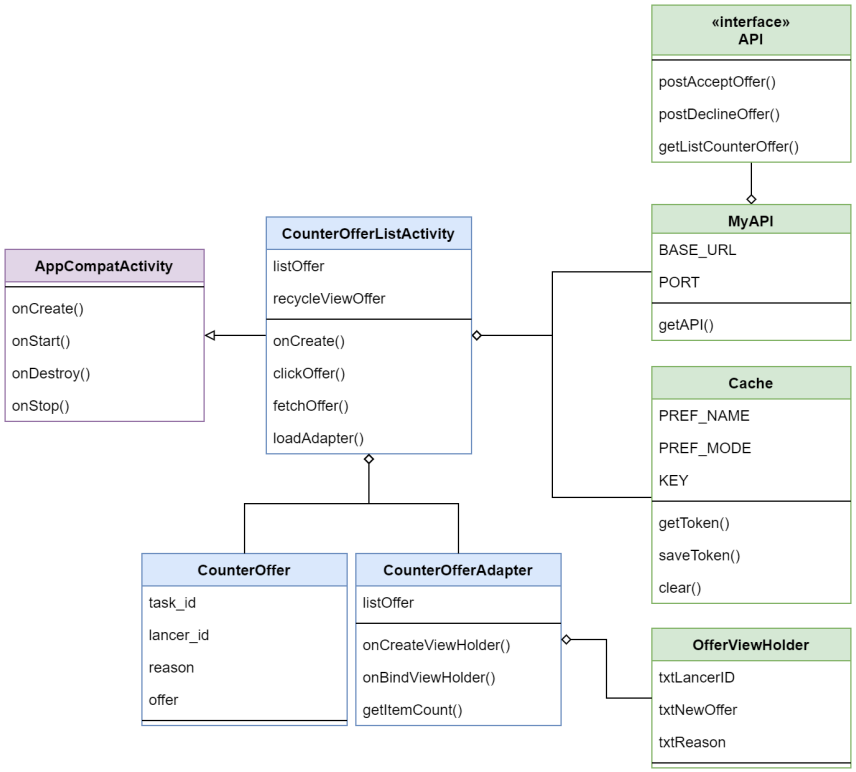
The request detail is nearly the same as the task detail. In addition, while the status of the task is “doing”, the users can choose to change the request to done status. So we split the activity into two children who inherit from the RequestViewActivity.

4.3.12 Component: Make counter-offer activity



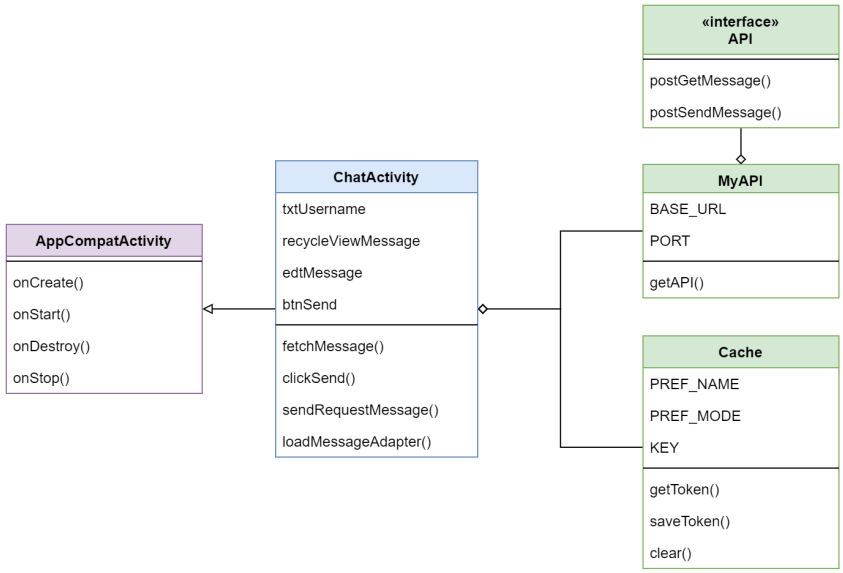
After the client creates the task, the lancer can see and make a counter-offer. In this activity, it displays the task information and lets the lancer provide the new price and the reason.

4.3.13 Component: View list counter-offer activity



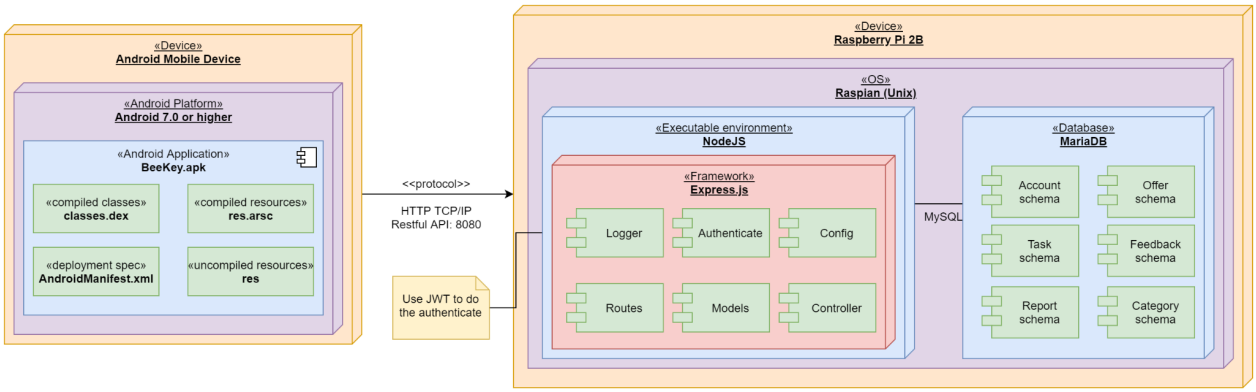
In each request, the client can see all the counter-offer which lancers created before. When the user clicks on any counter-offer they can choose to decline or accept it.

4.3.14 Component: Chatting activity



To provide more information during doing the task, the client and the lancer can chat with each other through the chatting activity. In this activity, it basically lets the client and the lancer send messages to each other.

5. Deployment



On the client device, the android application runs on the platform of Android 7.0 or higher. It used the JSON Restful API to communicate with the server through the port 8080.

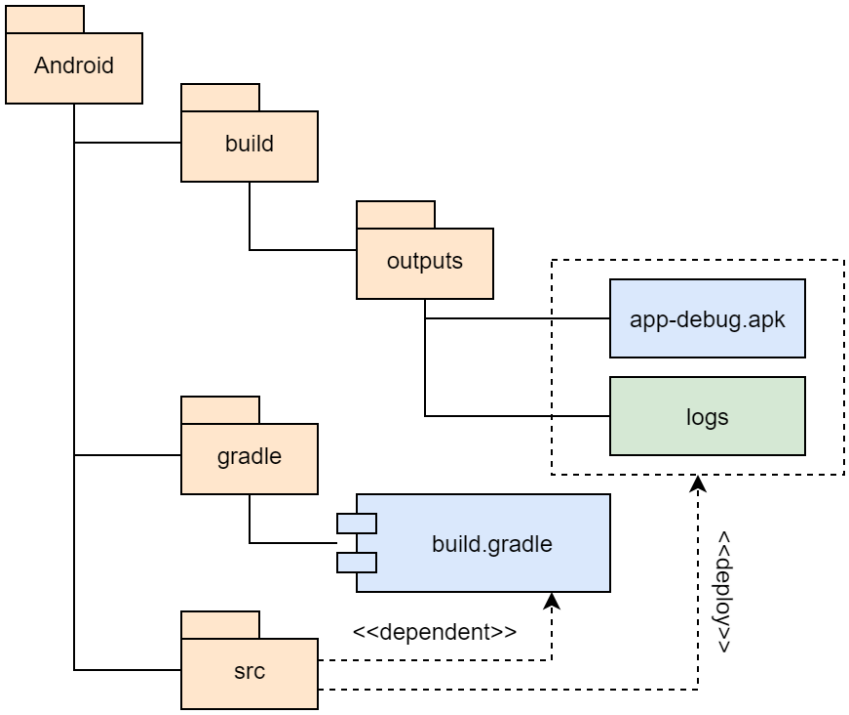
To run the server, we use the Raspberry Pi 2B embedded computer. The Raspberry uses the Raspian operating system which is a light-weight version of Unix/Debian based operating system for itself. The server incharges two roles: the application server and the database server.

As an application server, it runs on the NodeJS environment and is built with Express framework. The structure of the application server is divided into six middlewares. On the authenticate middleware, we use the JSON-Web-Token (JWT) to create the token and verify tokens.

As a database server, it uses the MariaDB database management system to provide the data. The database contains six basic schemas: Account schema, Offer schema, etc.

6. Implementation View

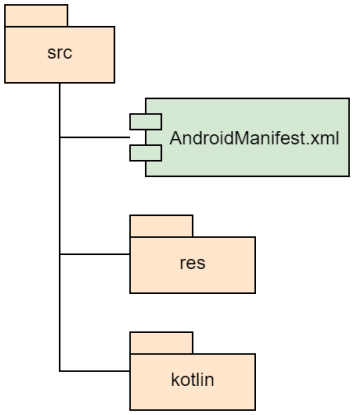
6.1 Android application implementation view



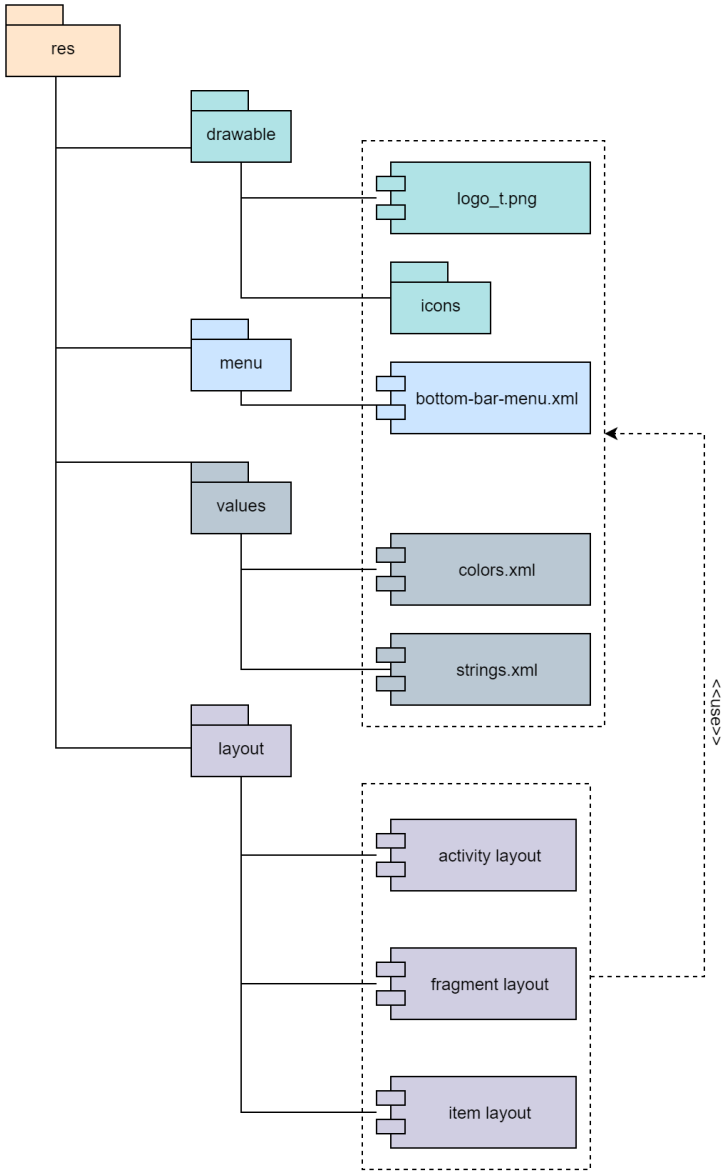
The structure of client application (Android application) including three main folders:

- build: stores the output file to install (.apk) and the logs file for debugging process
- gradle: stores the configuration of libraries and building properties
- src: stores the main source code of the project

All the source codes in the source folder will depend on the gradle to build out (deploy) the installing files.



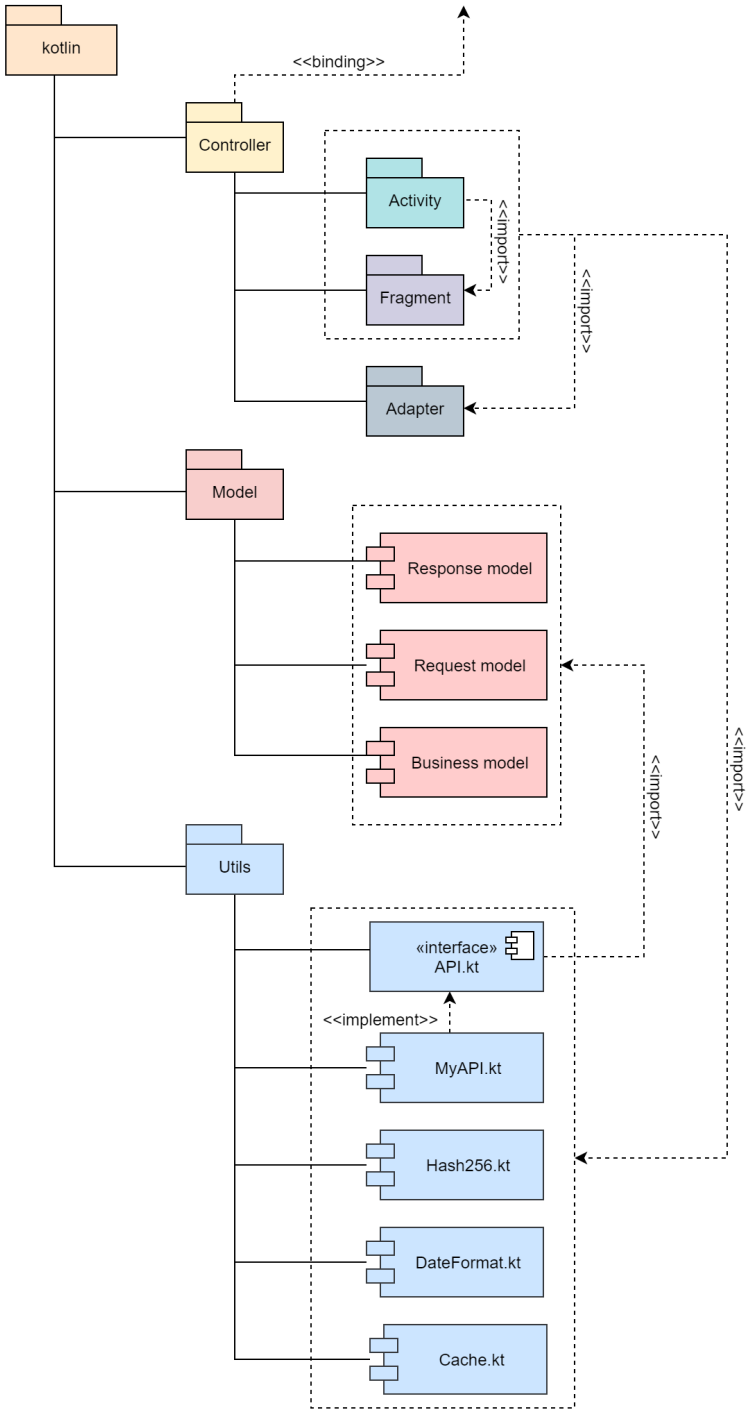
In our source code, we have a manifest file to request permission from the user and to define the activities, fragments of our application. In addition, as we already said in the architecture section, the application has 3 main tiers: View (GUI), Controller, Services (Utils). The View component will be stored in the “res” folder (resources) and the Controller, the Services will be stored in the “kotlin” folder.



In the resources folder, we have four packages:

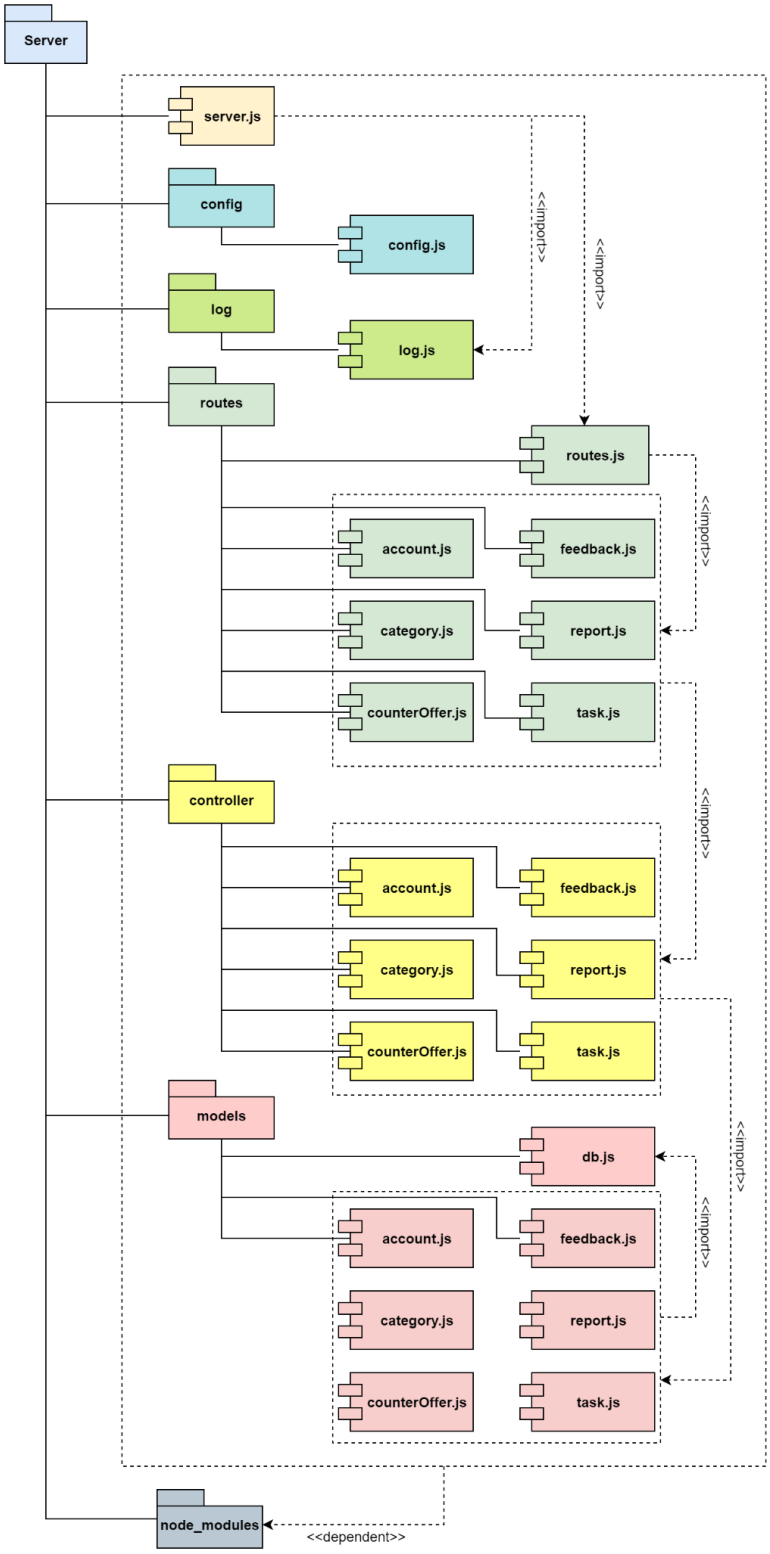
- drawable: stores logo and icon image
- menu: stores the layout of the bottom bar menu which used in the main activity
- values: define the constant string and colors (e.g. primary color)
- layout: stores all the layout of View including activity, fragment and the item layout (for adapter)

To build-up the layout, it uses the image and constant values of the drawable, menu and values folder.



In our last component, we have 3 packages to control the business logic and communicate with the server through the Restful API. The controller contains components which handle business logic of the View (GUI) tier. The activity and fragment use the adapter to load the list of data and also use the services provided by Utils folder. In the Utils package, we have the API to send requests to the server. The API interface is implemented by the MyAPI module and uses the model provided in the model package.

6.2 Server implementation view



In the server, we use the structure of middleware as mentioned in the architecture section. We have one folder called “node_modules” to store the npm libraries we depended on. All our middleware will depend on this package.

The main component is “server.js” which passes the request to the logger component then verifies the token as an authenticated component. After verifying the request, it continues to pass the request to the route component if the request is accepted.

The route components is splitted into six smaller modules to route each request to the suitable route. For each kind of request, we move it to the controller of that route.

The controller will extract the request, create the data package to query the database and the response the result to the client. To query the database, it sends the data package to the models middleware.

After that, the model middleware uses the “db.js” to create the connection to the database (MariaDB) and query the request of the controller. Finally, the result of the query will be returned to the controller and response to the client.