<Product\_Scanner>

Software Architecture Document

Version <1.0>

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <17/11/17> | <1.0> | Initial work | All |
| <21/11/17> | <1.1> | Reviewed following to TA’s comments | All |
| <3/12/17> | <1.2> | Add Deployment | Huynh Vinh Loc |
| <3/12/17> | <1.2> | Add Implementation view | Huseyin Kizil |

Table of Contents

1. Introduction 4

2. Architectural Goals and Constraints 4

3. Use-Case Model 5

4. Logical View 6

4.1 Component: Presentation tier 7

4.2 Component: Business service tier 8

4.3 Component: Data tier 12

5. Deployment 13

6. Implementation View 14

Software Architecture Document

# Introduction

Purpose

This document provides a comprehensive architectural overview of “Product scanner”. It provides an architectural overview of our system. The aim of “Product scanner” is to inquire the customer about the scanned product, especially its average price.

References

* H. Kato and K.T. Tan, "2D barcodes for Mobile Phones," Proceedings of 2nd. International Conference on Mobile Technology, Applications and. Systems, pp. 8, Nov. 2005.
* Ab Rahman, M. H. (2017). Development of quick scanner mobile application for grocery price checker using barcode scanner (Doctoral dissertation, Universiti Teknologi MARA)
* Gao, Jerry Zeyu, Lekshmi Prakash, and Rajini Jagatesan. "Understanding 2d-barcode technology and applications in m-commerce-design and implementation of a 2d barcode processing solution." In Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International, vol. 2, pp. 49-56. IEEE, 200

Overview

This document contains 6 sections which are described as below:

* Section 1 is an introduction to the software architecture of Product Scanner
* Section 2 gives goals and constraints for the architecture of the system
* Section 3 contains a use-case model
* Section 4 describes a logical view for Product Scanner
* Section 5 shows deployment
* Section 6 describes an implementation view for Product Scanner

# Architectural Goals and Constraints

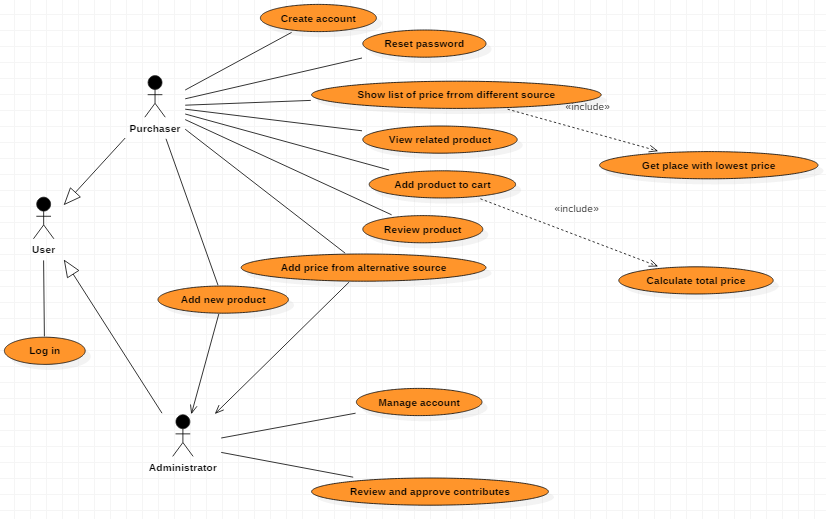
There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

1. Database system must ensure complete protection of data from unauthorized access. All remote accesses are subject to user identification and password control.
2. The application must be available on different languages
3. The application can be use in either offline mode or online mode.
4. All products in the database must be editable by admins and all information of products must be shown for users.
5. User could not directly access and edit database. All changes must be sent to admin as suggestions.
6. All performance and loading requirements, as stipulated in the Vision Document must be taken into consideration as the architecture is being developed.

The constraint for design and implementation are:

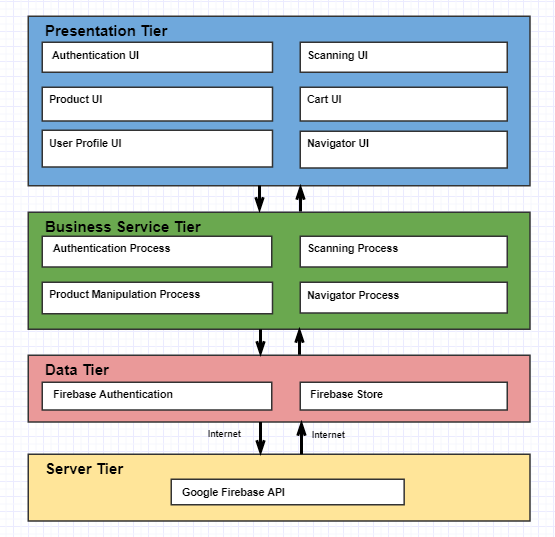
* Flexibility
* Simplicity

# Use-Case Model

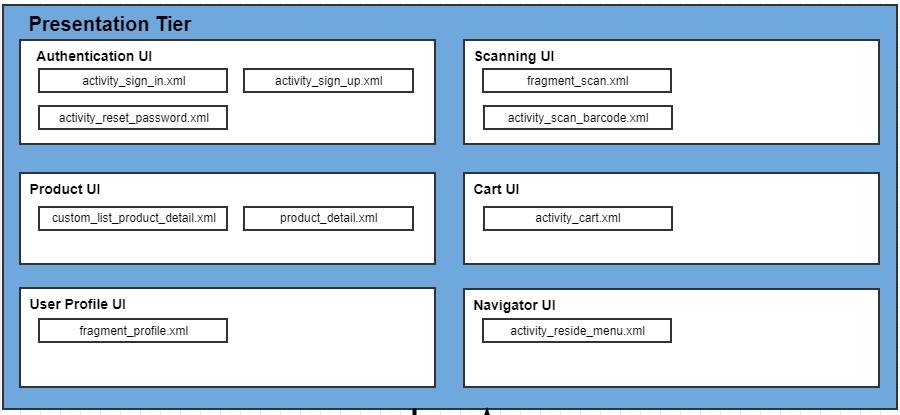
**

# Logical View

# 



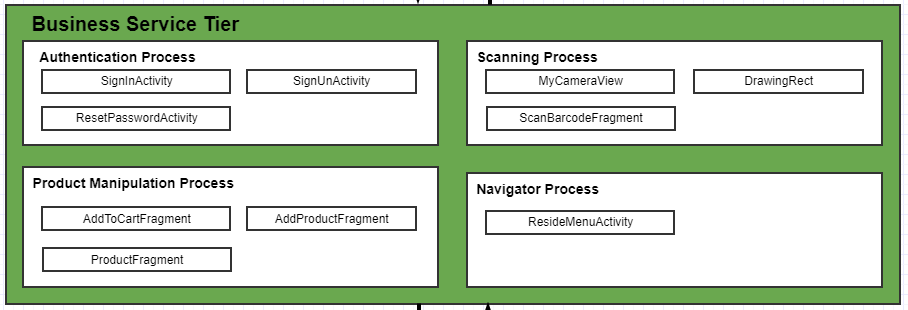
## Component: Presentation Tier



This tier shows the main UI of the application. There are 6 main parts (described in the picture above).

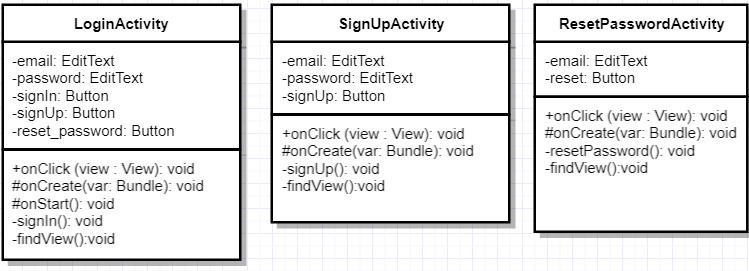
* **Authentication UI**: login, register, reset password pages
* **Scanning UI:** scanning screen
* **Product UI**: including each product detail, and list of product (our custom List View), can be reused for some further function.
* **Cart UI**: User cart layout
* **User Profile UI**: account detail and setting page.
* **Navigator UI**: the navigator menu of the application, here we use the re-side menu.

## Component: Business Service Tier



## This Tier controls our whole project functionality by performing detailed processing. Up to this moment, there are 4 main processes in the application descripted by the picture above.

## 4.2.1 Authentication:



When the user launches the application, authentication is needed.

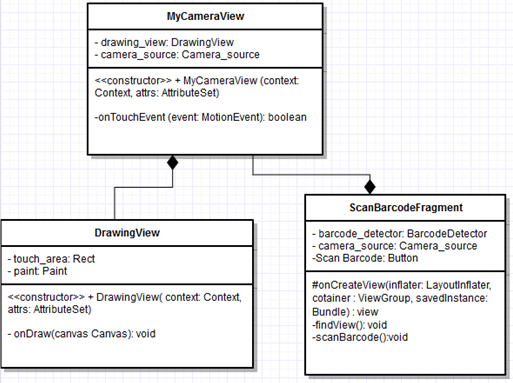
If he has an account, he fills the 2 EditText “mail” and “password” then he clicks on the button “signIn”. That runs the function *signIn()* which checks if the entered email and password are available by sending a request to the FireBase.

Then if he forgets his password, he clicks on the button “reset\_password” to launch the corresponding activity. Here, he is has to fill his email address and to click on “reset”. By this way, if there is a corresponding email address in the database, a procedure to reset the password is sent to the user. He just has to follow the instruction to reset it.

If he does not have an account, he clicks on the button “signUp” which launches the corresponding activity. He informs his email address and his password. Information is sent to database for verification.

## 4.2.2 Scanning Process

## 

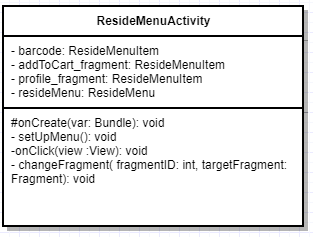


The role of **ScanBarcodeFragment** is to launch the activity to detect a barcode. When he presses the button “Scan Barcode” the class launches *scanBarcode(*). This method creates a **MyCameraView** in order to scan the object. When **MyCameraView** catches a barcode, it sends to **ScanBarcodeFragment** which handle this barcode.

**MyCameraView** detects barcode through the module “com.google.android.gms.vision”. When the users tap the screen, the application focuses on the clicked surface to detect the barcode via the modules. After scanning, information is returned to the main class to be processed.

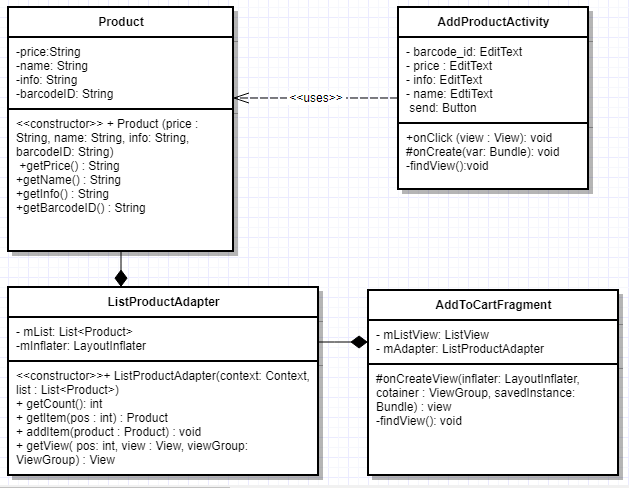
DrawingView class is constructed here to draw a rectangle around the touched surface by user.

**4.2.2 NavigatorProcess**

****

**NavigatorProcess** is used to switch among layouts. There are 3 layouts that user can access by this reside menu: user profile, add to cart, and scan barcode. There are several functions in this activity such as **setUpMenu** (initialize and add fragment to the menu ) , **onClick** ( let user click on menu icons to change ) and **changeFragment** ( support changing fragment ).

**4.2.2 Product Manipulation Process**



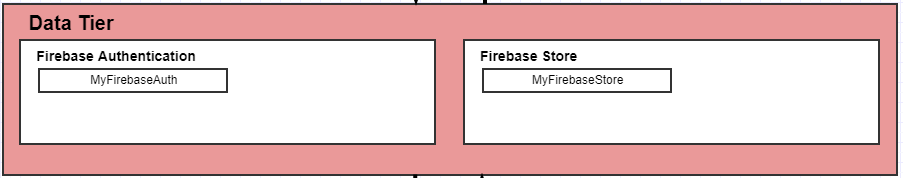
This process control some manipulated-functions on the product, including **Adding product to cart**, **Add new product** (the **Sharing** and **Rating** functions seems too small compared with these functions, so we don’t list them here)

**Product Class** holds the information of the product: name, price, information, barcode. **ListProductAdapter** is a custom ListView that holds a list of products consists of many items and inflate directly into layout. **getCount** to get total number of products, getItem to select specific ones, **addItem** to add new ones into list.

**AddproductActivity** let user input new product by 4 properties in edit text. Click the send button to finish and send request to admin to accept changes.

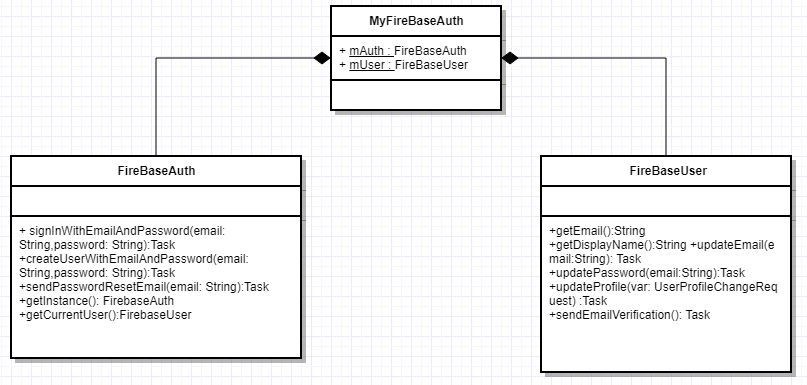
**AddToCartFragmen**t is also hold a list of product but selected by user themselves for further use such as calculate total price.

## 4.3 Component: Data Tier



## This tier can be considered as database-warehouse, where user-data, product-data is stored and retrieved. Data in this tire is keep independent of Business server tier

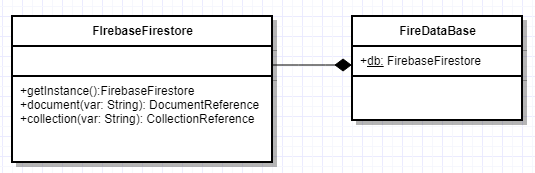
## 4.3.1 Firebase Authentication



**MyFireBaseAuth** class handles 2 main method that we use in the application: **FireBaseAuth** (using for handle signing in, signing up, reseting password,...) and **FireBaseUser** (using to update account information, update password, ….)

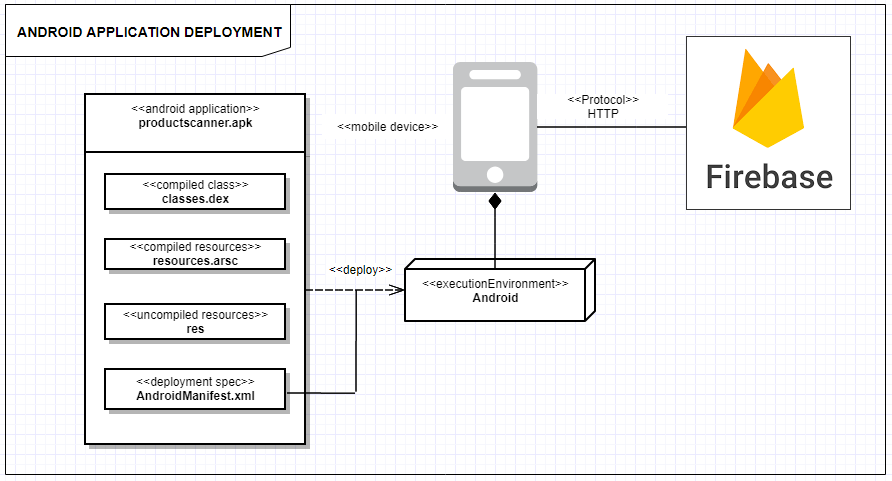
This class plays an important role, ensure the “**Authentication Process**” works correctly.

## 4.3.2 Firebase Store



This is a **NonSQL** document database. We use it to store our product, user data, and can be easily sync data across devices in both online and offline mode.

# Deployment



File productscanner.apk contains the code, required data and resources files of the project. The .apk file represents one Android application to be deployed to the Android-enabled mobile devices.

The classes.dex file ( Dalvik Executable) contains the classes compiled in the DEX file format understood by the Dalvik/ART virtual machine.

The resources.arsc file contains the XML content from all configurations of the res/values/ folder.

Res contains resources that aren't compiled into resources.arsc.

AndroidManifest.xml: Contains the core Android manifest file. This file lists the name, version, access rights, and referenced library files of the app. The file uses Android's binary XML format.

The application runs in Android environment and it connects to Firebase for user authentication and database.

# Implementation View

The folder “java” contains the source code files

The folder “drawable” contains the images needed for UI and related .xml files to position them.

The folder “layout” contains .xml files in order to design the UI.

The folder “strings” contains 4 files to translate the application into 4 languages: English, French, Turkish and Vietnamese.