System design document for Pinjobs project (SDD)

Contents

Version: 2.0

30th May 2015

Authors: Filip Slottner Seholm, Filip Larsson, Isaac Gonzalez, Carl Albertsson

This version overrides all previous versions.

1. Introduction

1.1 Design goals

The design must be loosely coupled and modular to make it possible to switch GUI as well as switch database and extend the program with new features. The design must be testable which means it should be possible to isolate models for test.

1.2 Definitions, acronyms and abbreviations

Parse - cloud database using NoSQL

NoSQL - database that can handle big data and multiple queries.

MVC - model-view-controller, design pattern used for…

Java - object oriented programming language.

Android Studios - integrated developer environment for develop android applications.

GUI - graphical user interface.

Xml - extensible markup language, documentation readable for both human and computer.

Activity - A android specific class which works as a controller in the system.

Package-by-feature - package structure where one package contains classes specific for one feature in the application.

Ad - A Class in the system which represents an advertisement that consists of different attributes.

Intent - An object used to start an activity, which contains different information about the activity the intent is sent from and can be used to send additional data to the new intent.

Serializable - An interface which, if implemented, make objects of custom created classes containing only types already defined in java available for sending through intents via the putExtra method.

Parcelable - An interface which, if implemented, make objects of custom created classes containing serializable objects available for sending through intents via the putExtra method.

Service - There are two classes that have all the communication with the database. These classes are called Services.

2. System design

2.1 Overview

The application uses a modified android MVC structure. It also uses a “package-by-feature” structure and services classes for communication with a database.

2.1.1 MVC-android structure

The MVC design used by the application is to have models as plain java classes independent of any android specific classes which implements an interface.

The controllers are represented as subclasses of the android specific class Activity. The system consists of one controller for each view with a similar name as the view. eg. CreateAdView has the controller CreateAdActivity. The activitys communicate with both the model, view and other controllers and coordinates what to do depending on input and states of the view and model.

The view is represented by both .xml files and Java classes where data from the .xml file to the Java view file is given by sending the Activity to the view. The view itself then extracts the references of the desired components. The .xml file holds the code for the elements of the view and the java class is used to manipulate these elements.

2.1.2 Services for database

The purpose of a Service class is to communicate with the database without compromising the sustainability or exchangeability of the program. The communication is made through Service classes. Their purpose is to save and retrieve data from the database. These classes also have their own interfaces which cover the base functionality of the classes. To change “storage type” from our database Parse to some other database or local storage, all one needs to do is change or replace the Service class and implement the service interface. This follows the facade design pattern (Skrien, 2008) which simplifies the usage of the subsystem Parse. This also makes the program independent from parse API, since it is only the interface of the Services that is used in the outside code.

2.1.3 Communication between controllers

The communication between the controllers in the application are a bit different compared to normal Java application. Since the program is android based and uses Activities as controllers the communication between the activities is used by intents.

Intents works almost like a constructor in the way that when an Intent is called to an activity, the activity starts and calls the onCreate method. If you want to pass some kind of data through an intent this is done by a putExtra method which takes a keyword that the data is related to. To send own objects unknown by the android API the class that is sent though the extra method has to implement either the interface Serializable or Parcelable.

The application sends data to different controllers in this way, this means that some of our classes implements Serializable or Parcelable. Advertisement is a model class which needs to implement

Parcelable but since it is a model the class should not consist of any android API specifics which is necessary when implementing Parcelable. This results in a wrapper class which holds an advertisement that can be sent through an intent. By utilizing the Proxy design pattern the wrapper class works as a placeholder for the advertisement class (SourceMaking, 2015).

There is also a StartforForResult method for intents which is used quite frequently in the application.The method starts an activity and when the newly started activity is done and the finish() method in that activity is called the onActivityResult method in the first activity is called. This is very useful because the second activity can finish and send data back to the creator activity without any knowledge of the creator activity. This is done in the application to stand clear of circular dependencies between the controllers and depending on what result is given, the creator activity will do different tasks.

2.1.4 Unit testing

To make the application testable, some model classes has to implement an interface. For example the model class advertisement implements the interface IAdvertisement. The Interface contains exactly the same methods as the model class. In doing so the possibility to create mockclasses for testing is available. Mockclass testing is used in the application when tests of the applications advertisement class is done. If the advertisement class were to be tested without a mockclass it would depends on a profile object. But if a mockclass of the profile class is used during the test the advertisement test is independent and does not rely on any other class. Using interfaces in this way is supported by the design pattern facade (Skrien, 2008).

2.2 Software decomposition

2.2.1 General

The application consists of following modules, see figure 1.

advertisement - Contains all classes relevant for creating and modifying ads. Some classes for showing ads created by the user. Contains model, view and controller classes which are divided by packages. advertisement also has a utils package and a service package which communicates with the database

main - Contains mainView and mainActivity, mainActivity is the first activity that is opened when the application is started.

profile - Contains all classes relevant for creating, modifying and showing profiles. Contains model, view and controller packages. Also contains a service package used for communication with the database.

handler - Contains classes for displaying all advertisements in the system. Contains model, view and controller classes. Handler also has a background thread which fetches all the advertisements from the database and saves them in the system.

user - Contains classes for logging in to a user’s profile. View and model for MVC.

utils - Contains one class for getting the user's position and one class for checking if information is put in correctly when getting input from the user. The package contains of help classes which are used from mutiple other modules.

2.2.2 Decomposition into subsystems

The only subsystem used is the database Parse. The System will not work properly without it. One of the main goals in designing this application is to make it as *modular* as possible. Therefore the implementation of Service classes were made. As mentioned earlier this is implemented by following the *facade pattern.*

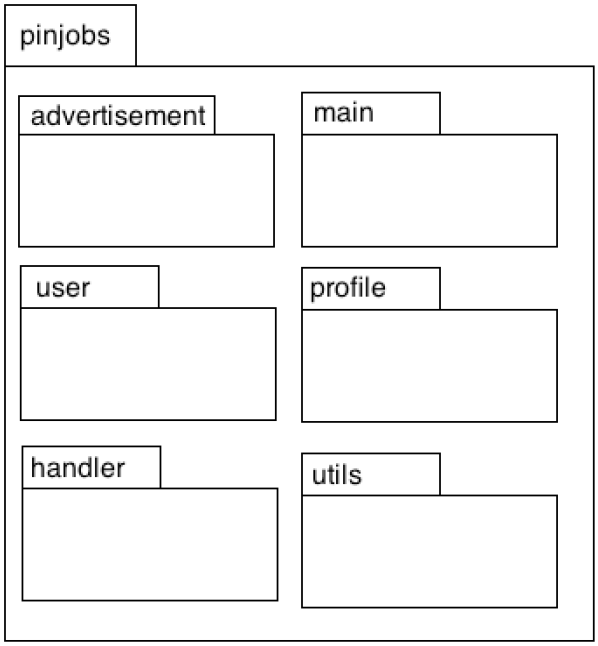
2.2.3 Layering

The system uses a package by feature structure, which means that there is a single package for every small module that is independent. The majority of modules includes a model, a view and a controller, some models only consists of a view and controller. In some packages there is also extra utility classes as well as service classes, these are all divided into packages within the main package.

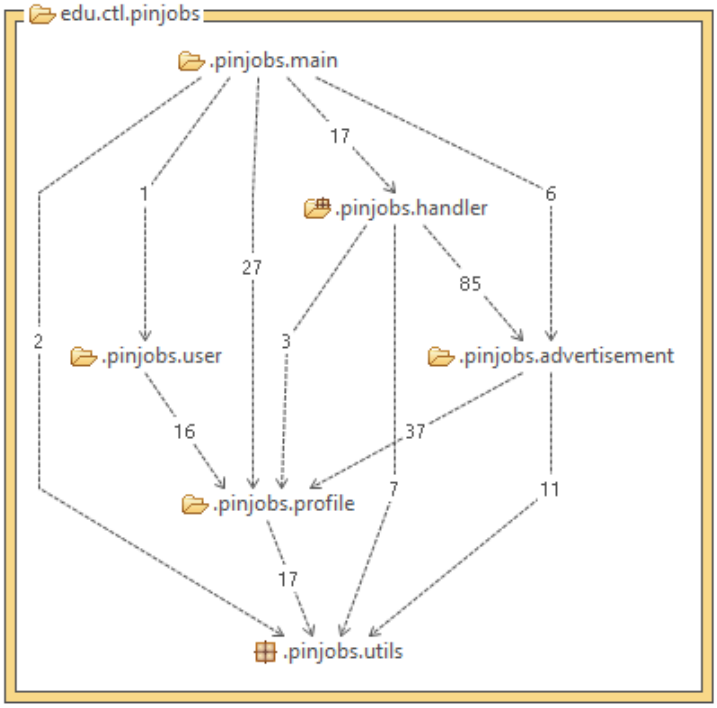
The lowest package in the hierarchy is the profile package (figure 2) This layer is completely independent of the other packages in the system. If profile needs to notify another package this is done using the *observer pattern* (Skrien, 2008). By having an interface in the Profile package which is implemented outside of the package, the lowest layer can communicate with the rest of the system without any knowledge of it. This is done in the whole hierarchy and this is how every lower layer notifies the higher layers.

2.2.4 Dependency analysis

Dependencies are as shown in Figure 2. There are no circular dependencies.



Figur 1. High level package.



Figur 2. STAN graph.

2.3 Concurrency issues

The program uses a background thread to receive a list of advertisements but because Parse is a NoSQL database no concurrency issues are possible. NoSQL database can handle multiple queries (MongoDB, 2015). The background thread is used to mitigate concurrency issues, by fetching ads and communicating with parse in the background thread the main thread doesn't have to wait for parse and the program can continue as usual. This also follows the None-functional requirements.

When the background thread is done fetching the ads the system saves it in *AdvertisementListHolder* which uses the Singelton design pattern. This is done due to there should only be one list and one instance of the list. When the background thread is run again the list will be replaced and the system will instead use the updated list because all classes have a reference to the same object and therefore the new list.

2.4 Persistent data management

The application do not contain much data itself but it handles data contributed from the user such as profiles and advertisements. The Parse database is used to store and retrieve this data.

2.5 Access control and security

Could be some issues since the application depend on Parse. If the connection with Parse would fail during some task, an exception is thrown and an alert dialog is shown telling the user that there was a connection error.

When a user creates a Profile, the password is saved to Parse as a string as plain text. The Parse class we used to save all data in our classes was called *ParseObject*. This class can be modified and made into a "copy" of the Profile. There is another Parse class called *ParseUser* which already has some set fields, like *password*, which encrypts the string saved to it. But the application still uses it's own profile system because if the system wants to change database the application would still have a profile/login system and the exchange would be easy.

2.6 Boundary conditions

NA. The application is launched and exited as a regular android application.

3. References

MongoDB, INC. (2015). *NoSQL Databases Explained*. From URL http://www.mongodb.com/nosql-explained

SourceMaking. (2015). *Proxy Design Pattern.* From URL https://sourcemaking.com/design_patterns/proxy

Skrien D (2008). Object-Oriented Design Using Java.
USA. McGraw-Hill.

Appendix

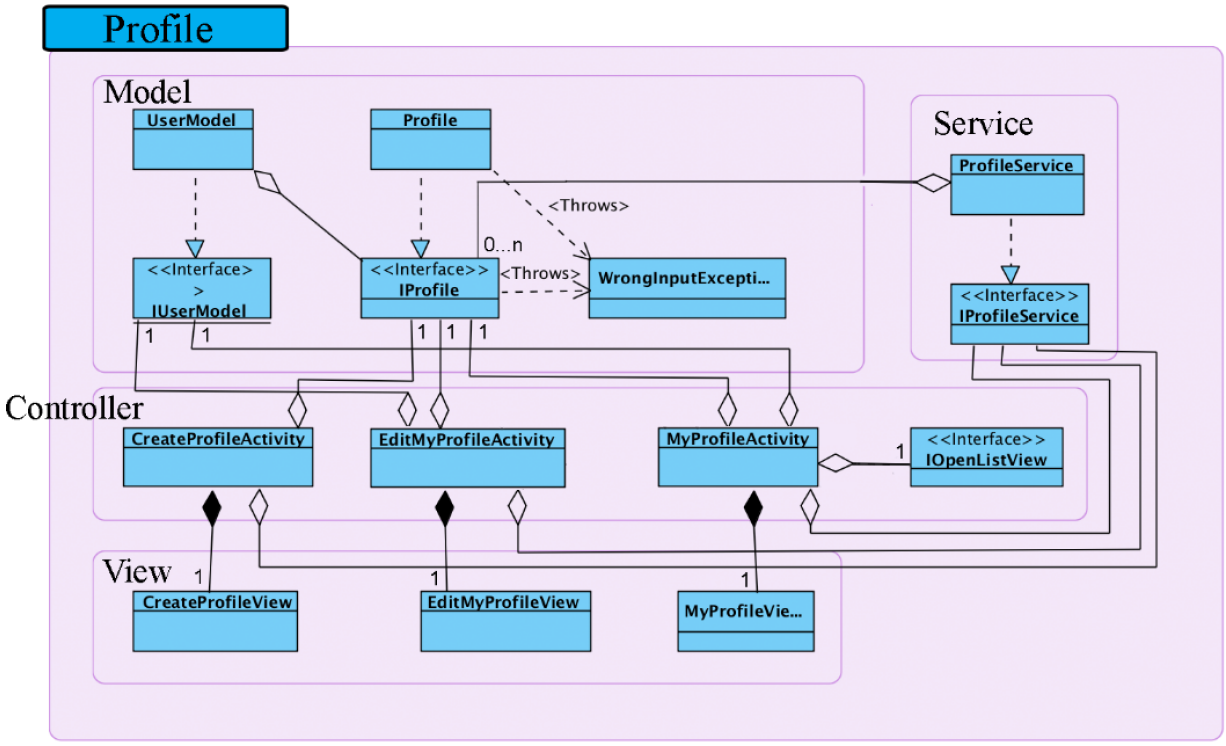


Figure 3. UML of profile package.
(Similar structure in the other packages that contains model-view-controller)