System Design Document

SKAJ Corp - KvittoSkanner

Version: 1.9

Date: 2017-04-05

Author: Sanjin Slavnic, Joakim Mattsson, Kevin Brunström & Anne Keller

This version overrides all previous versions

Table of contents

1. Introduction

The application runs on the latest Android version as of early 2017. Reason being is to get access to the newest available libraries for the project. The application therefore runs on Android devices capable of running Android Nougat 7.1 with API 25.
The application will use a MVC model base to structure code and divide class functionality and responsibility. Main focus is to obtain loosely coupled design.

1.1. Design goals

Application design must be testable. It should be easy to isolate classes in the model and construct separate tests. The design should be loosely coupled for future extension, such as adding new features and functionality but also making it possible to redesign or update the GUI. To make this process user-friendly for future modifications, the application model will be encapsulated, meaning briefly it will be easier to understand and change.

1.2. Definitions, acronyms and abbreviation

A.

API
- Application Programming Interfaces is a set of tools and protocols used for building software application, specifying how software components should interact.

Apache License
- The Apache License is a free software license that allows the user of the software the freedom to use the software for any purpose.

Activity
- See 2.1.1.

Adapter
- A bridge between the GUI components and the data source that fill data into the component.

B.

BottomBar
- A custom view component that mimics Material Design[1] Bottom Navigation pattern.

Bitmap
- A specific type of Drawable object, mainly .png, .jpg or .gif, that can be drawn on the screen.

[1] https://material.io/

C.

Context

- Abstract class allowing access to application-specific resources and classes and application-level operations such as launching activities, receiving intents.

F.

FAB

- Floating Action Button.

Fragment

- See 2.1.2.

G.

Gradle

- Open source build automation system designed for multi-project builds.

GUI

- Graphical user interface.

I.

Intent

- An intent is a messaging object used to request an action from another app component.

J.

JSON

- JavaScript Object Notation is an language-independent open-standard format that uses text to transmit data objects consisting of serializable values.

L.

Library

- A library is a collection of resources, including pre-written code, classes, values, specifications and documentation used by programs to develop software.

Layout Manager

- A layout manager positions item view inside a RecyclerView (see 2.1.7.).

M.

MVC

- Model-View-Controller is a way to organize the structure of an application. The Model containing the data and logic that determines how the application will operate. The View responsible for everything visible (the GUI) and Controller which handles user input and controls the model and view.

O.

OCR

- Optical character recognition gives a computer the ability to read text that appears in an image.

S.

Serializable

- The process of translating data structure or object state into a format that can be stored.

STAN

- Structure analysis tool for Java, visualizing code design, providing understanding of code, measuring quality and design flaws.

U.

URI

- Uniform Resource Identifier is a compact sequence of characters that identifies an abstract resource.

TextRecognizer

- Object from Google library that collects text information from images.

X.

XML

- Extensible Mark-up Language is a format used for sharing data on the internet.

2. System architecture

2.1. Overview

The application is written in Java using Android Studio IDE. The Gradle build system for Android Studio made it easy to include external library modules as dependencies.

2.1.1. Activity

It is important to acknowledge that Android Studio initiates its program in an Activity starting with a call on onCreate() callback method, similarly to the Java's main() function. Activities are fundamental on the Android platform and are central to how a user navigates within an application. This makes it challenging working with a MVC model as Activities are defined somewhere between a traditional view and controller, sometimes functioning as both. To solve this we simply created a separate package for all Activities in our application.

2.1.2. Fragment

Fragment is another important Android component. It holds part of the behavior and UI of an activity. They resemble functionality of activities and extend an activity in some ways. This provides modularity by dividing activity code across fragments and reusability by placing shared behaviour across multiple activities. Fragments are located in the view package.

2.2. Libraries

2.1.3. TextRegognizer

This detector object possesses OCR capabilities. The Mobile Vision API, licensed under the Apache 2.0 License, provides the TextRecognizer for mobile Android devices. Once initialized it can be used to process all types of images and determines what text appears in them. It's located in the service package.

2.1.4. Bottom Bar

Bottom navigation bars are primarily for use on mobile devices. They make it easy to explore and switch between top-level views. It works as a controller and is located in the MainController class. It enables the user to navigate between different top-level views the app provides (Archive, Company and Supplier. It's licensed under the Apache 2.0 License.

2.1.5. Wizard Pager

A wizard is originally a GUI design pattern that leads the user through the interface step by step to do tasks in the prescribed order. In order to implement this we included a library of a Wizard GUI for Android licensed under the Apache 2.0 License. It's responsible for main case functions and is functionally decomposed into several classes, located in different packages.
This library has been extremely difficult to work with, preventing MVC model base code structure. After several discussions with our adviser Adam, we chose to place and structure the main part in a separate wizard package, located in the view package.

2.1.6. Floating Action Button

A circular button, shortly names FAB. It descends from ImageView and enables user to add new data in top views. It's located in the controller package and licensed under the Apache License 2.0.

2.1.7. Text Drawable

This library provides images with text similar to the Gmail application. It extends Drawable class and can be used with ImageView classes. It's primary use in KvittoSkanner adding design to the listviews, grouping for example different categories by color and first letter.

2.1.7 RecyclerView

The RecyclerView widget is a container for displaying large data sets whose elements change at runtime. To use RecyclerView, one have to specify an adapter and layout manager. We chose to

implement Base RecyclerView Adapter Helper, extending the RecyclerView.Adapter class, making it fairly easy to work with lists. It's licensed under the Apache license 2.0. and located mainly in the view package, working as an adapter for lists.

2.1.8. App Intro

This is a library, easy to implement, introducing first-time users to get familiar with KvittoSkanner. It's a form of Wizard and is licensed under Apache license 2.0.

2.1.9. PhotoView

This library helps to produce an implementation of zooming Android ImageView. It's located in the view package, licensed under the Apache license 2.0.

2.1.10. SpinKit

This library extends ProgressBar, a visual indicator of progress. It's usage is located in the "AddNewReceipt" Activity, indicating user that the application is copying or collecting data from an image. It's licensed under the MIT License.

2.2. Software decomposition

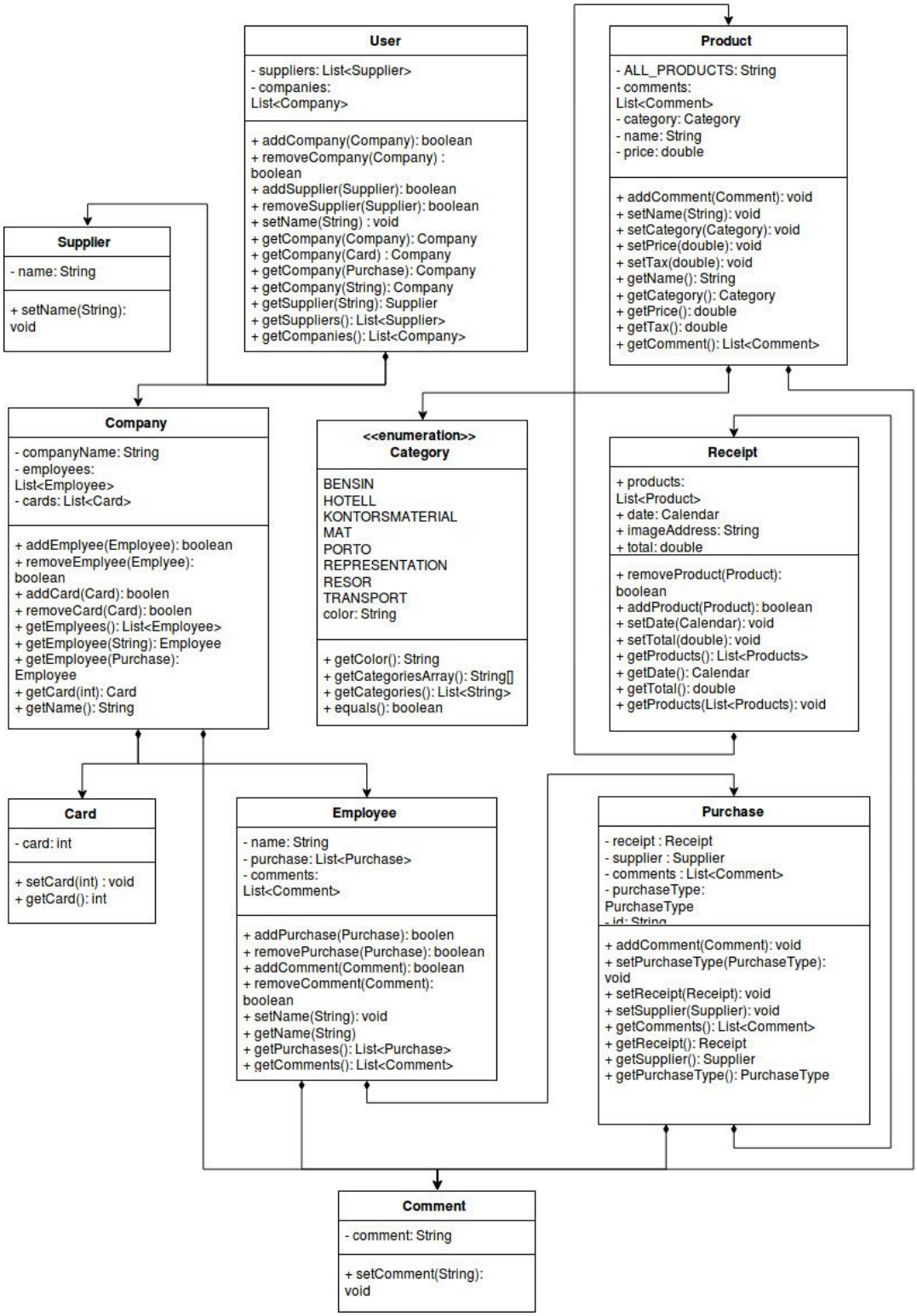

Figure 2.0. Showing design model for KvittoSkanner.

Application model consists mainly of lists, holding data with the user at the top of the model hierarchy.

2.2.1. Dependency analysis

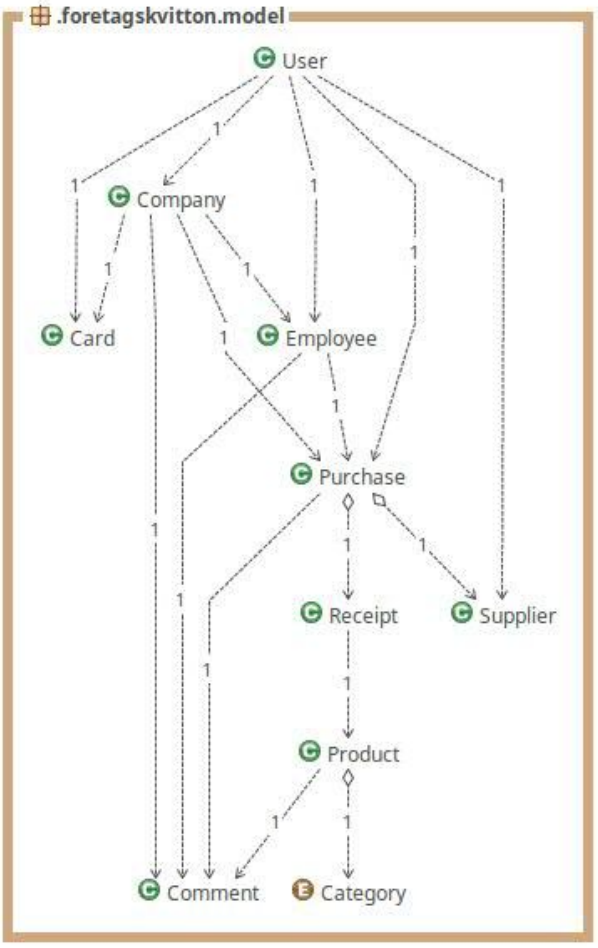

Figure 2.1. Showing model package dependencies.

Parts of the model, Purchase for example, is a part-of-a-whole. It holds a Receipt, holding data from a saved receipt, including an image reference.

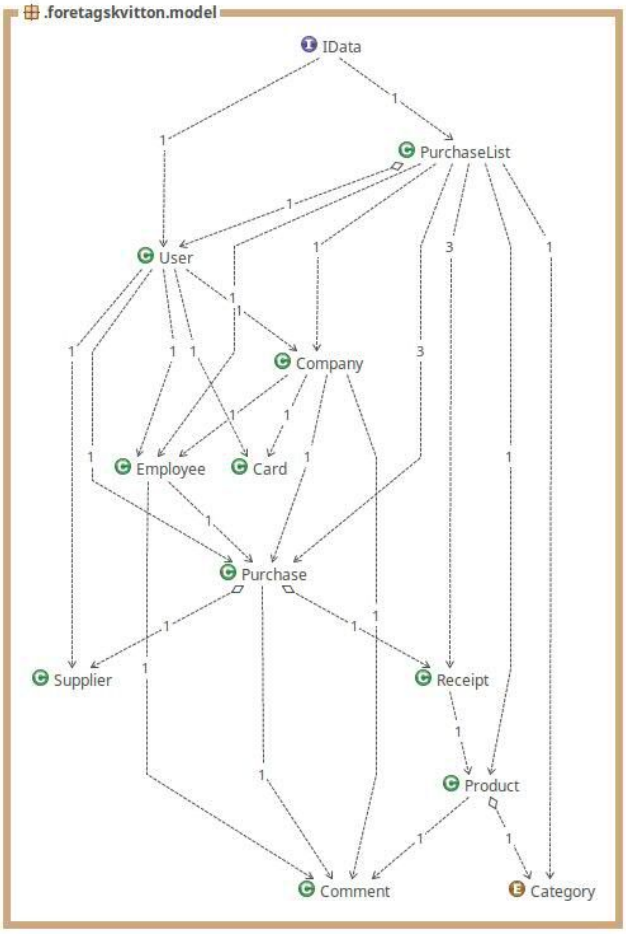

Figure 2.2. Showing a complete picture of dependencies in KvittoSkanner model.

Figure 2.2. includes an interface, enabling user to locally save data. There is also a PurchaseList class, extending ArrayList, enabling access to all saved Purchases, allowing user to filter and sort information.

2.3 Concurrency issues

Android application has usually one main thread that starts when Application class is created. This thread is responsible for drawing the GUI. Any code added to an Activity runs with leftover resources so the app stays responsive. This caused issues when TextRecognizer collected data from an image, taking sometimes a few seconds to execute. The AddNewReceipt Activity simply shut down and application moved on to next Wizard Activity in order to not delay the GUI.
Same issue emerged when an image was copied.
In order to fully complete executions a parallel thread was created, running a progressbar while data was being collected or copied.

3. Subsystem decomposition

There is little documentation in the code as per agreement with our advices Adam. Information can instead be collected with analysis tools. Focus was instead naming classes, methods and variables, making is easier to determine their responsibility and functionality.

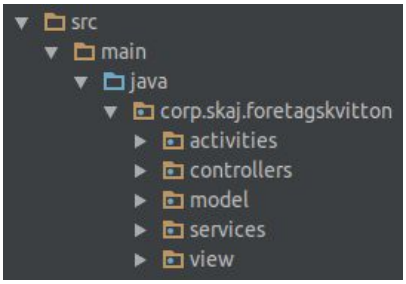

Figure 2.0. Showing showing main packages. Picture doesn't include internal service package textcollector, responsible for algorithms collecting gathered data from images. It also excludes wizard package located in view.

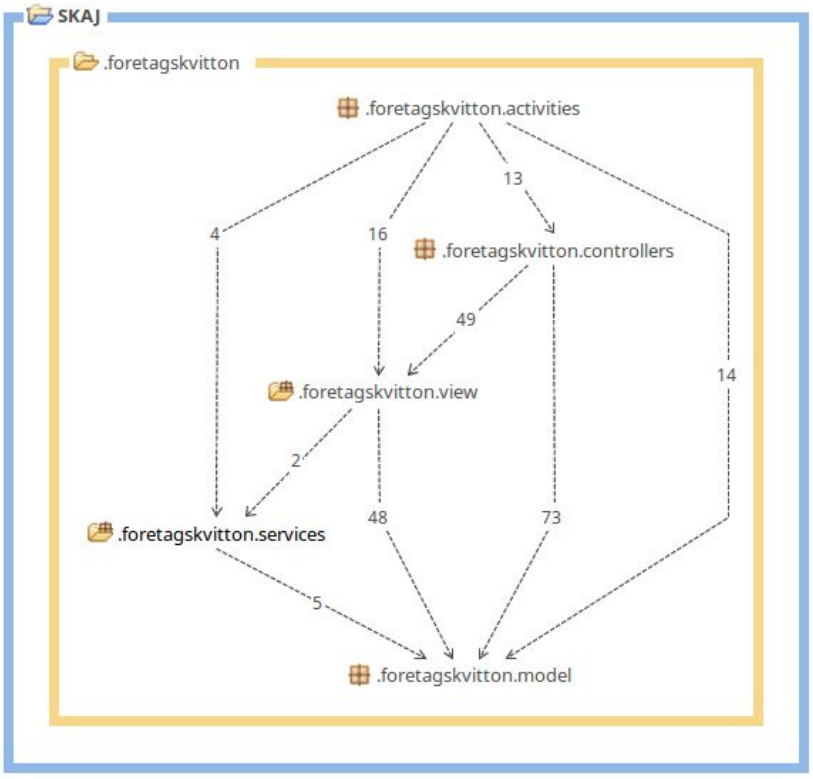

Figure 2.1. Showing application package dependencies and model encapsulation.

3.1 Activities

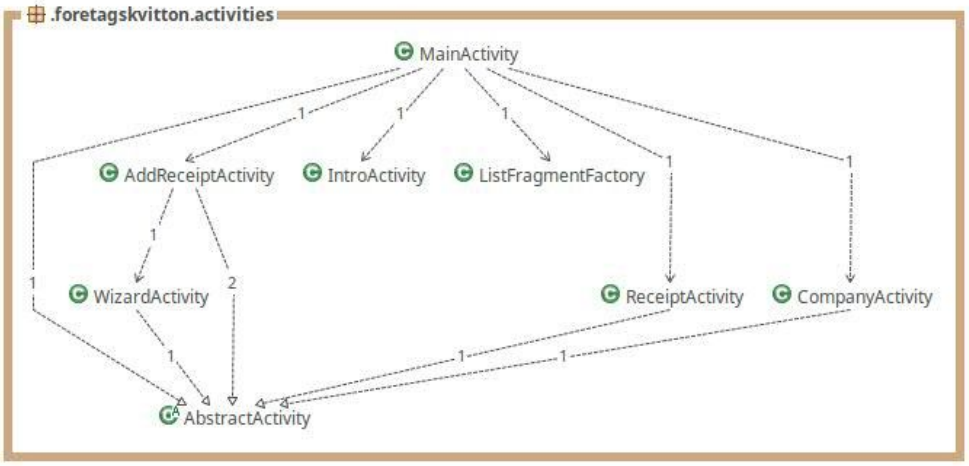

Figure 2.2. Showing activity package dependencies.

This package is hierarchically structured, with MainActivity at the top, working figuratively as a glue for controllers and views (See 2.1.1. for further explanation). AddReceips holds IFileHandler for image management and initiates Wizard to confirm and collect required data, enabling Purchases to be saved. AbstractActivity holds common package functionality.
There is also a Factory, building top-views, adapters and controllers.

3.2 Controllers

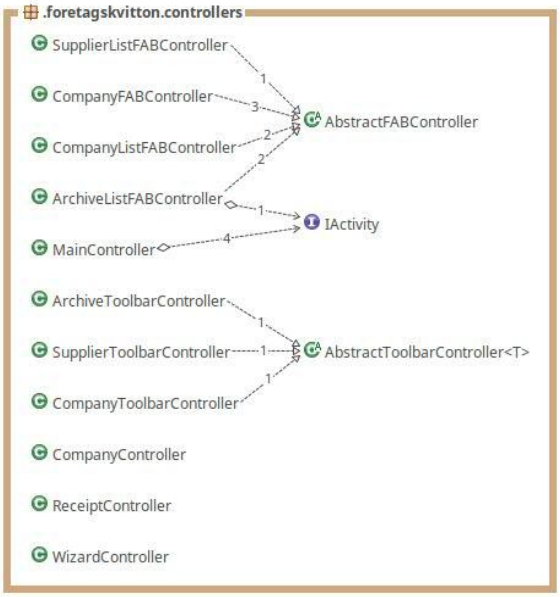

Figure 2.3. Showing controller package dependencies.

Controller package is composed of classes responsible for controlling different views and user inputs. FAB controllers handle user input for adding items in lists. Their functionality is abstracted in the AbstractFABController class, making it fairly easy to add in other views. Same goes for Toolbar controllers, visually changing the action bar at the top of screen, adding functions that allows user to manipulate lists and remove data.
IActivity interface allows communication between controllers and activities.

3.3 Model

See 2.2.1.

3.4 Services

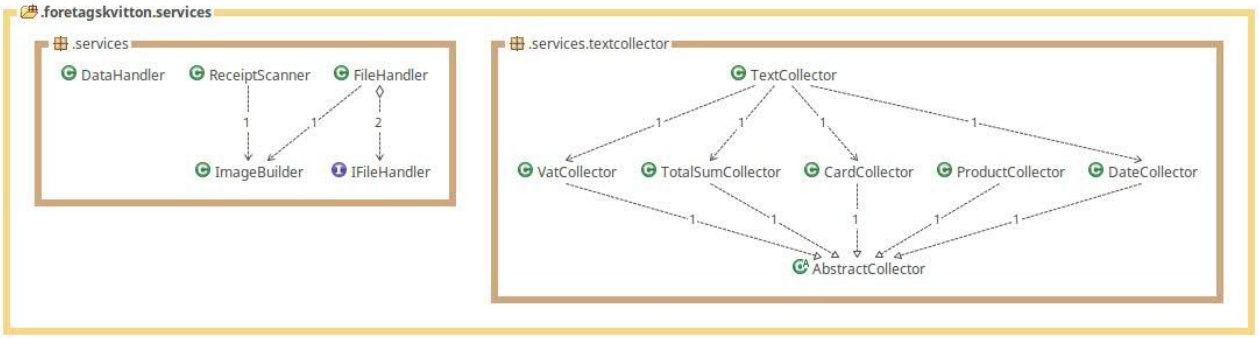

Figure 2.4. Showing service package dependencies.

Service package is mostly globally accessible, being responsible for handling read and write data functions. DataHandler (See 4.1.4. for further explanation.) saves model data locally and FileHandler handles file management for receipt images, allowing communication by implementing IFileHandler interface. This is used by AddNewReceiptActivity, enabling application to read, write and copy images.

ReceiptScanner static class uses TextRecognizer (See 2.1.3. for further explanation.) to collect data from images, enabling user to cumulate required data for Purchases. TextCollector is static, holding package private help classes, allowing functionality to be globally accessible. It consist of algorithms for filtering collected data by ReceiptScanner. It's purpose is to scan list of strings and find desired data.

Imagebuilder is static and builds a Bitmap from an Uri.

3.5 View

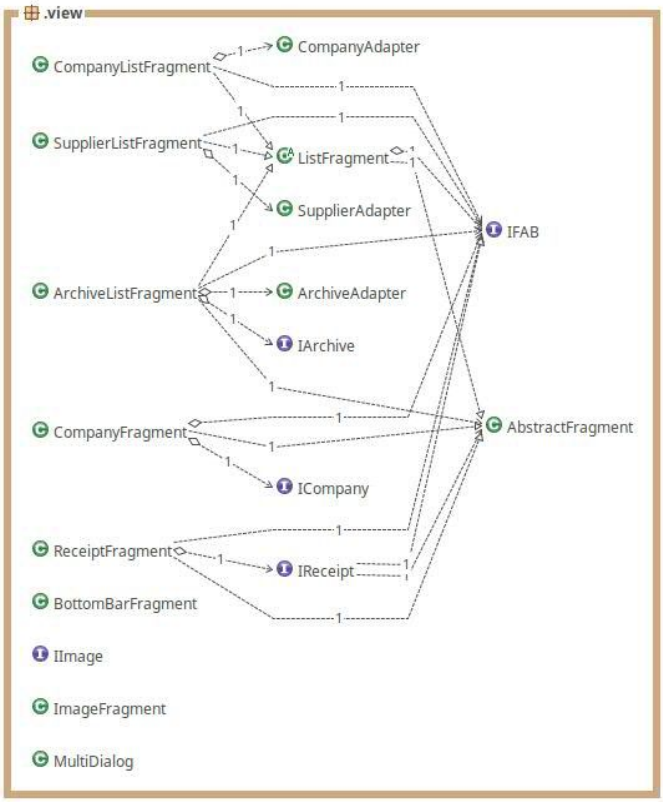

Figure 2.5. Showing view package dependencies.

This is so far the most entangled package, consisting of adapters, several interfaces for setting listeners, communicating indirectly with the controller package, a general pop up builder MultiDialog and fragments. It's responsible for the different views of the application.

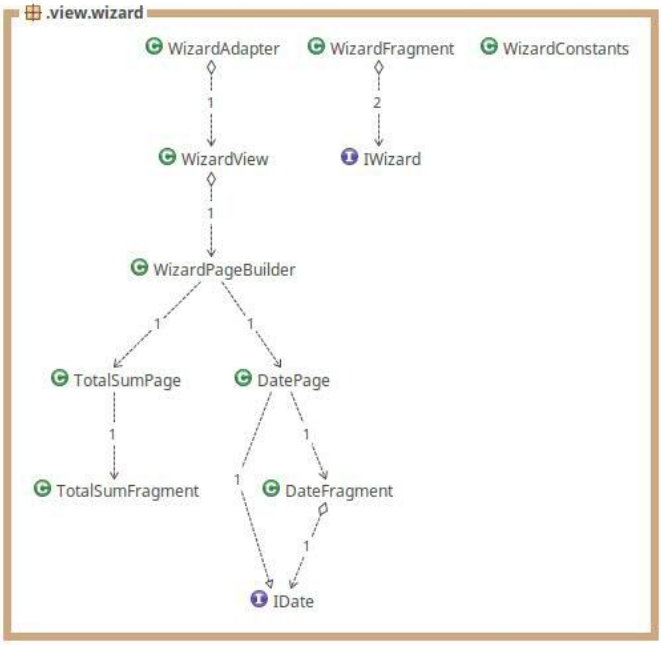

Figure 2.6. Showing wizard package dependencies.

This package consists mainly of an implemented library, guiding user through several steps for registering a new receipt. It was a challenge to fit it to the top level package structure as the source code was not build to follow a MVC model. Code was instead package structured. See 2.1.5. for further explanation.

4. Persistent data management

4.1 Overview

Application data is saved locally using a single SharedPreference instance that you can call from any Context. App user object, holding data, is instantiated when application starts. If there is no data saved a default will be made, registering a single company. Images of receipts are saved locally in application folder, generated by the app itself. The model holds an Uri in string form in order to be converted into JSON, referencing to the image location.

4.1.1. SharedPreferences

SharedPreferences is an interface, pointing to the file that contains the values of preference. It allows users to save and retrieve data in the form of key-value pair. It can be used to save primitive data.

4.1.2. Gson

In order to save all model data, specifically the User class object, the application implement Gson to convert Java Object into a JSON. Gson is a Java serialization library, able to convert Java Objects into JSON and back, licensed under Apache 2.0 license.

4.1.4. DataHandler

DataHandler is a generic class extending Application, a base class for maintaining global application state. It provides read and write data function for all types, keeping application data globally accessible. In order to get DataHandler, Context interface has to be obtainable. Calling getApplicationContext returns global application context, in difference from other contexts where an activity context may be destroyed or made unavailable when the activity ends.

4.1.5. FileHandler

This class is responsible for building an image folder and files, enabling images to be saved. It reads and removes, but most importantly copies images attached to purchases, being dependent on the image data. It uses static ImageBuilder to build Bitmaps.

4.1.6. XML

The application's layouts and the screen elements they contain are designed using XML. Android provides three types of XML, but recommends XMLPullParser because it's easy to use. Each layout must contain one root element, which is a View or ViewGroup object. You can additionally add layout objects or widgets as child elements.

5. Access control and security

The application is intended for single a user. As of today data is saved locally, enabling future extensions, making it possible to use databases, possibly with login function to secure and access receipt from different devices.

From a package perspective the application model is encapsulated, preventing modification. Locally saved copies of images are still accessible from outside the application, enabling users outside the application access, making the application vulnerable if images were to be deleted.

6. References

TextRecognizer

- https://developers.google.com/vision/introduction

WizardPager

- https://github.com/TechFreak/WizardPager

BottomBar

- https://github.com/roughike/BottomBar

FAB

- https://github.com/futuresimple/android-floating-action-button

RecyclerView

- https://github.com/CymChad/BaseRecyclerViewAdapterHelper

AppIntro

- https://github.com/apl-devs/AppIntro

PhotoView

- https://github.com/chrisbanes/PhotoView

SpinKit

- https://github.com/tobiasahlin/SpinKit