System Design Document

SKAJ Corp - KvittoSkanner

Version: 1.4
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 runs therefore 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.

B.

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

C.

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

F.

Fragment

[1] https://material.io/

- 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.

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.

T.

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.

2.1.3. 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. implementation and with Adam's advice we have integrated it as good as possible.

2.2. Software decomposition

Below is the Design Model for KvittoSkanner

2.2.1. Dependency analysis

Below is a chart of all dependencies.

//TODO Använd analysverktyg (STAN) för att grafiskt visa beroenden och kommunikation mellan klasser i modellen.

2.3 Concurrency issues

The threads that is being used in the application is used as we need to load information quick. Android do not accept long loading times and if it takes to long to load data the Activities which is main components in our program will "kill" themselves.

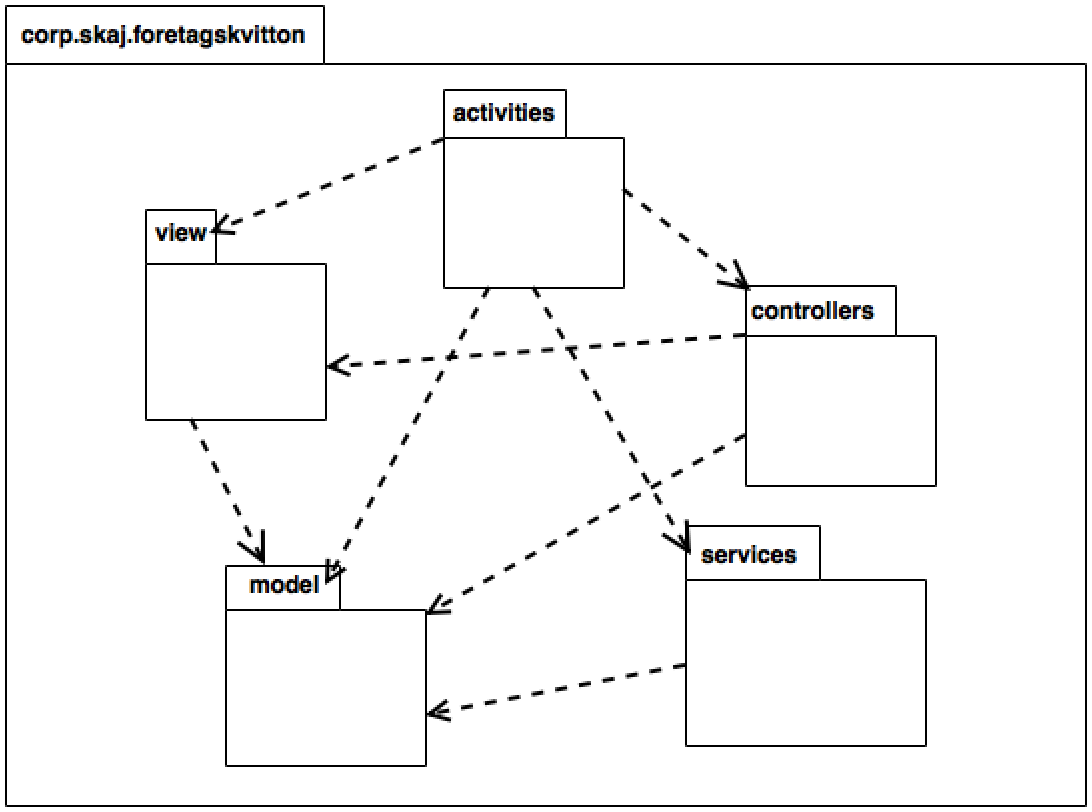
The most overall, top level, description of the system. Which (how many) machines are involved? What software on each (which versions). Which responsibility for each software? Dependency analysis. If more machines: How do they communicate? Describe the high level overall flow of some use case. How to start/stop system.

An UML deployment diagram , possibly drawings and other explanations. Possibly UML sequence diagrams for flow.

(Persistence and Access control further down)
Any general principles in application? Flow, creations, ...

3. Subsystem decomposition

Below is the package diagram for our application.
There is no description in the top of the classes as per agreement with Adam as this information can be collected with different types of tools.



3.1 Activities

Activities package holds all the applications activities.

3.2 Controllers

Controller package holds all controllers.

3.3 Model

The model package is for all domain logic.

3.4 Services

The service package is for data handling.

3.5 View

The view package is for all visual representation.

3.1 “...First software to describe” ...

Recap: What is this doing (more detailed)
Divide it into top level subsystems. An UML package diagram for the top level. Describe responsibilities for each package (subsystem). Describe interface. Describe the flow of some use case inside this software. Try to identify abstraction layers. Dependency analysis Concurrency issues.

If a standalone application

- Here you describe how MVC is implemented
- Here you describe your design model (which should be in one package and build on the domain model)
- A class diagram for the design model.

else

- MVC and domain model described at System Architecture Diagrams
- Dependencies (STAN or similar)
- UML sequence diagrams for flow.

Quality

- List of tests (or description where to find the test)
- Quality tool reports, like PMD (known issues listed here)

NOTE: Each Java, XML, etc. file should have a header comment: Author, responsibility, used by.., uses ...

4. Persistent data management

4.1 Overview

Application data is saved locally using a single SharedPreference instance that you can call from any Context, pointing to the file that contains the values of preference.

4.1.1. SharedPreferences

SharedPreferences is an interface which 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.

4.1.3. URI

The User class holds indirectly a link to the image.
//TODO Skriv mer här

4.1.4. DataHolder

DataHolder 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 DataHolder, Context interface has to be obtainable. Caling 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. 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.

- https://developer.android.com/guide/topics/ui/declaring-layout.html

//TODO Lägga till ytterligare info?

How does the application store data (handle resources, icons, images, audio, ...). When? How? URLs, pathe's, ... data formats... naming..

5. Access control and security

The application is intended for single a user. Data is Possibility for extension, using a database and enabling login function to access receipt from different devices.
Encapsulation? Prevents modification of data that should not be manipulated from outside the program.

Different roles using the application (admin, user, ...)? How is this handled?

6. References

//Under construction…

Gradle

- https://docs.gradle.org/current/userguide/dependency_management.html

TextRecognizer

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

WizardPager

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

BottomBar

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