System Design Document

SKAJ Corp - Företagskvitton

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 May 2017. Reason being to get access to the newest available libraries for our project. The application runs therefore on Android devices capable of running Android Nougat 7.1 with API 25 or higher.
The application will use a MVC model, implementing interfaces to handle communication between mainly the views and controllers. Main focus is managing to keep 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.

Apache License

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

B.

BottomBar

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

G.

Gradle

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

GUI

- Graphical user interface.

J.

JSON

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

[1] https://material.io/

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.

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 in Android Studio made it easy to include external library modules to the application as dependencies.

2.1.1. Activity

It is important to acknowledge that Android Studio initiates, similarly to Java's main() function, its program in an Activity starting with a call on onCreate() callback method. 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, 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 many 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 is a part of a larger Google library shortly named OCR, which gives a computer the ability to read text that appears in an image. 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. In this application the navigation bar is an Activity. It works mainly as a controller and is therefore located in the controller package. It enables the user to navigate between different top-level views the app provides (Archive, Graphs, Add New Post, Company, Profile. 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. This is the application's main use case and due to its several steps, involving most functionality, it’s located split up into several classes and located in different packages based on the MVC model.

2.2. Software decomposition

//TODO Vi behöver en bild på våra paket och ett diagram som visar hur våra paket kommunicerar. Vi bör beskriva i punktform vilka som kommunicerar och hur.

2.2.1. Dependency analysis

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

2.3 Concurrency issues

//TODO Skriv om trådar vi använder, varför och hur dom fungerar.

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

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 ...

3.2 "...next software to describe" ... As above....

4. Persistent data management

4.1 Overview

Application data is saved locally using a single SharedPreference instance that all clients share, 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 data from the model, specifically the User class, 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. URL

The User class holds indirectly a link to the image.
//TODO Skriv mer här om vi använder google drive länk för att lagra bilder.

4.1.4. DataHolder

Data is instantiated in DataHolder when the application starts. DataHolder is a Java class extending Application, keeping application data globally accessible. In order to get DataHolder, Context interface has to be obtainable, allowing access to application-specific resources and classes. Calling getApplicationContex returns global application context, in difference from other contexts where an activity context may be destroyed or made unavailable when the activity ends.

This class exists in order to encapsulate application model and avoid possible loading time during usage.

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