System design document for Flash Cards

Version: 2.0

Date: 29/5-2016

Author: Group 10

This version overrides all previous versions.

1 Introduction

1.1 Design goals

The design is constructed so that we can change the services e.g if we want to swap the database.

1.1.1 GUI

The GUI is fairly easy to grasp for users on all experience levels since it's very straight forward as well as following accepted design conventions to eliminate excise for the user.

1.2 Definitions, acronyms and abbreviations

GUI - Graphical User Interface

2 System design

2.1 Overview

The application will use a MVC model adapted to the android studio specifics. The frontend will be built using XML and the backend will be done with Java in android studio.

2.1.1 Aggregates

Each view in the GUI will have its own XML file that is directly linked to a Java controller class, one controller class per XML file. The controller classes are connected to each other via a main model class FlashCards that keeps track of everything, with the help of a Singleton class that allows a single shared

instance between all classes.

2.1.2 Model Functionality

The decks and the cards are being stored in a SQLite database which is being handled by a database wrapper class and a database controller interface via Android Studios built in SQLite database API. There the decks and cards along with the different attributes are stored in SQL tables. Each deck contains an array list that holds its respective cards.

When you start the application you will see a list of all your decks. Clicking on either of them sets the current deck to the selected one and sends it to the main model FlashCards class. The information about the deck is fetched by the controller class from the SQLite database. The user can then select what mode they want to play the deck with, which also is stored as a private variable in the FlashCards class. Upon starting the deck the controller class fetches the information about the card(s) and presents them on the screen.

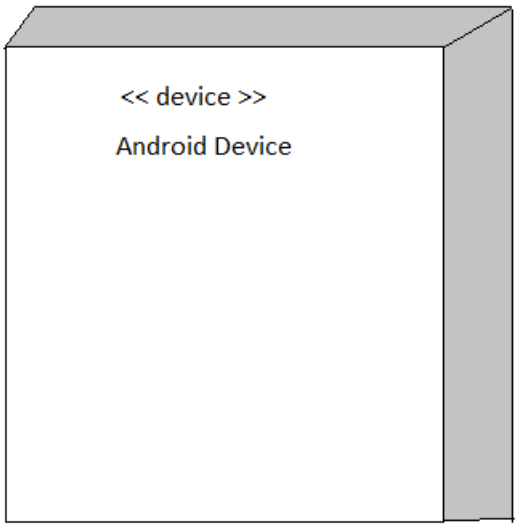
When you create a new deck or a new card the respective activity (either CreateDeckActivity or CreateCardActivity) calls the database where it saves the card/deck.

When you open an app specific import URL the uri scheme is recognized by the app which opens it and launches the ImportDeckActivity. That activity then strips out the JSONObject from the URL using the JsonService class, imports the deck and then launches the ImportCardActivity which imports all cards using the same URL and then sends presents the DeckActivity view to the user with the newly imported deck set to current deck.

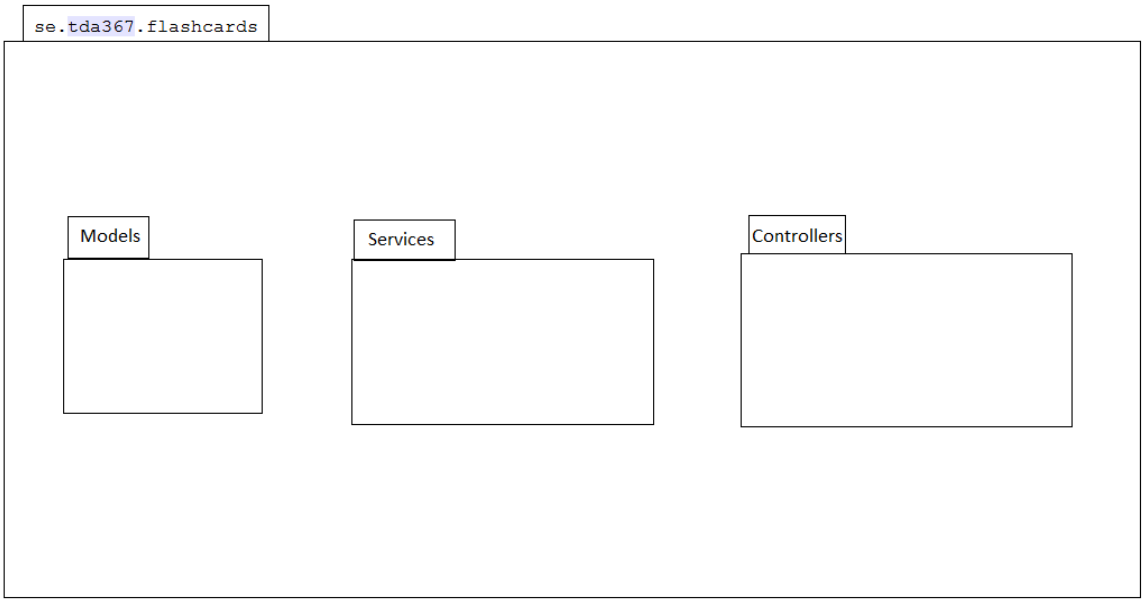
2.2 Software decomposition

2.2.1 General

The only thing needed to run our application is an android device with API level 15 or above as shown below in the deployment diagram.



The application is decomposed into the following top level packages:



- Models package contains the models for the application, including the head model class.
- Services contains different service classes and interfaces such as our Database Service.
- Controllers contains all controller classes.

2.2.2 Decomposition into subsystems

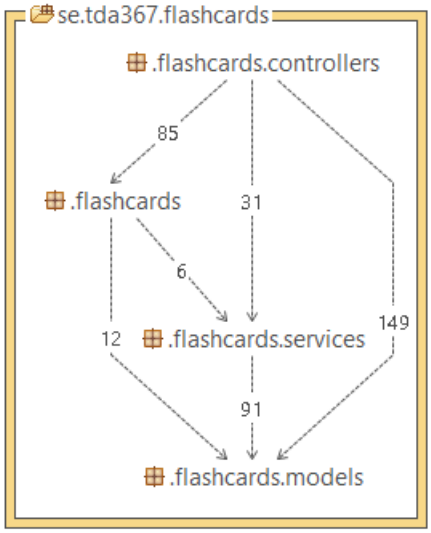
We have two subsystems, DatabaseService and JsonService. DatabaseService is a wrapper for the android built in SQLite database. JsonService is a class that converts decks and cards to JSONObjects and URLs by importing androids built in JSONObject, URLEncoder and URLDecoder classes.

2.2.3 Layering

Layering is as indicated in the figure below.

2.2.4 Dependency analysis

Dependencies are as shown below.



2.3 Concurrency issues

NA. None that we are aware of.

2.4 Persistent data management

All persistent data will be stored in a SQLite database.

2.5 Access control and security

NA.

2.6 Boundary conditions

NA.

3 References

1. Flash Card concept: https://en.wikipedia.org/wiki/Flashcard
2. MVC, see: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

APPENDIX

Design Model.