PSU Capstone Winter 17

Software Design Document

ASL App

Author: Josh Aldridge

Date: 06.2017

Table of Contents

Software Design Document

I. Introduction

A. The purpose of this document is to outline the high-level functionality of this ASL (American Sign Language) application produced by the Portland State University Capstone Team in Spring 2017. This document is intended for programmers who have a general understanding of class structures and the JAVA programming language. . This document details the general design considerations during the creating of this project, and is a higher level look at the software than the Technical Specification Document (which details more specific design considerations and inner-workings of classes. This document will outline the general connectivity of the application in terms of the classes used, and the general flow of data during use of the application. This application was built using Android Studio, and focuses on providing functionality to Android API 23 or higher. Please refer to our Functional Specification Document and/or Technical Specification Document for further details regarding the applications general requirements, and/or more specific implementation details. Lastly, it should be noted that features of the application that were developed, but not to completion, will also be discussed in this document; these features include the ability to edit a video, as well as the ability to send/receive videos with other users.

II. Requirements

A. The purpose of this project is to build an American-Sign-Language learning application on/for the Android platform. At a high level, this project is to develop an ASL application that will allow users to create and study (through various means) videos that represent ASL words and phrases. See the Functional Specification Document for more detailed information regarding requirements for this piece of software.

III. Terms

A. ASL - American Sign Language
B. API - Application Programming Interface
C. Activity - a single focused action that the user can do
D. Card - a recorded video paired with a text-based word representing that video
E. Deck - a collection of two or more Cards (see above)
F. URI - Uniform Resource Identifier - a sequence of characters that identify a resource

G. Toast - a “pop-up” text displayed to the user

IV. System Design

A. Design Considerations - Visual application outline/map

********See **Appendix A** below for FULL MAPPING of ASL APP functionality*******

B. Design Considerations - general overview of the application functionality

1. Base Deck

a) Provide a base deck to allow first time users some example data to work with. Contains videos of existing ASL words.

b) This deck can be deleted if the user wants.

2. FrontEnd

a) *Home Activity*

(1) this is the homepage of the ASL application. The user has the option to (1) Take Quiz, (2) Manage Decks, or (3) Manage Cards which will take the user to the following described activities.

b) Create/Edit/Delete Card

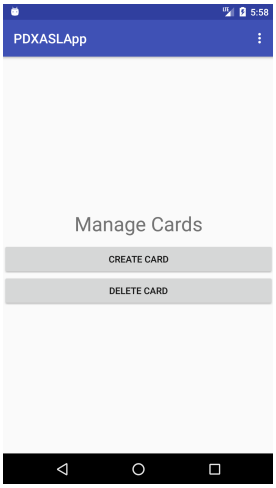


FIG 1: Manage Card Sub-Menu

(1) Create - To fully create a card, a user will need to give the card a name (**FIG 2**), which should represent the word or phrase that is recorded in the video.

Users will be allowed to record a new video using their device (**FIG 3**) or use an existing video in the Android Gallery (**FIG 4**) to create a card. Once the card is made, the card will be added to the user-selected deck when <SUBMIT> is hit (**FIG 5**).

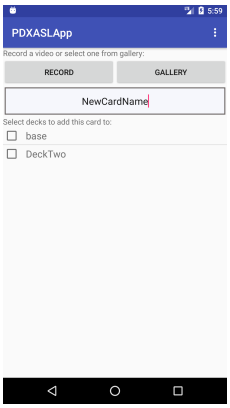


FIG 2

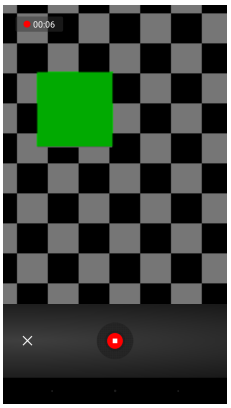


FIG 3

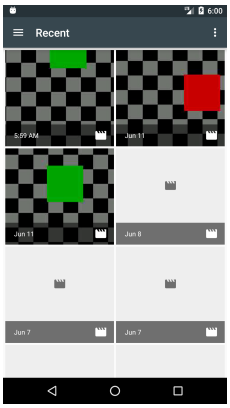


FIG 4

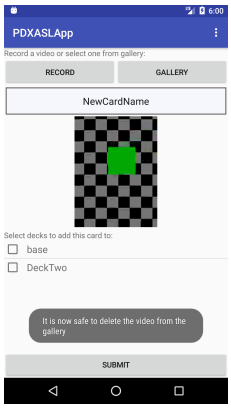
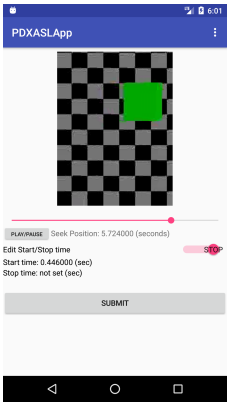


FIG 5

(2) Edit (more backend development needed) (**FIG 6**) - users should then be allowed to “time-crop” the video in order to shorten the video to a specified amount. NOTE - videos must be between 2-30 seconds in length. Once the user has selected their desired video length, a call to backend will be made to actually crop the video. The video will then be passed into any deck the user selects, assuming the deck contains the minimum card count (2 cards). ***IMPORTANT*** *- currently the layout is created for editing, but it is not utilized since there were implementation bugs when time cropping the actual video files on the backend side.*

FIG 6 : Edit Card Layout; currently built and provided, but not connected to current application



(3) Delete - to delete a card, the user simply needs to scroll through the list of all cards (videos) available in the application, and select the cards they want to delete. It should be NOTED that if a particular deck's card count drops below the minimum (2), the deck will also get deleted.

FIG 7 : Delete Card Layout

c) Create/Edit/Delete Deck

(1) Create - a user can create a new deck (collection of videos) if desired. To do so, they must name the deck (**FIG 9**), and select at least two cards from the existing list of cards to add to the deck (**FIG 9**). *NOTE* This implies that if a user wants to create a new deck with will hold new cards, the user must first create those new cards, can then create the new deck, and can, at this point, add the new cards to the new deck. *NOTE* Deck names must be unique - the application will not allow a user to create two decks with the same name.

FIG 8 : Create/Edit/ Delete Deck Layout

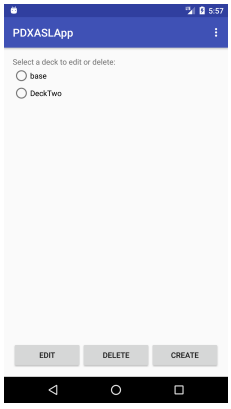
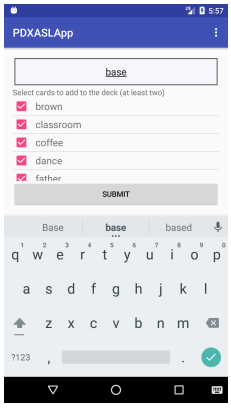


FIG 9 : Name deck; Select cards from checklist



(2) Edit - a user can choose to edit an existing deck; this can be done by selecting the deck they wish to edit (**FIG 8**), and then adding and/or removing

cards from that given deck (**FIG 9**); additionally users can rename the deck if desired (**FIG 9**). NOTE, as mentioned above, deck names must be unique or renaming will not be permitted.

(3) Delete - a user can view the list of available decks, and select which decks they want to delete. The user should be prompted with an “Are you sure” prompt prior to the deck actually being deleted. The user simply needs to hit the <DELETE> button after selecting a deck to delete (**FIG 8**).

d) Quizzes

(1) Users will be allowed to evaluate their own ASL learning by utilizing any of the three provided quizzing functionalities. The user can: take a multiple-choice quiz, take a “fill-in-the-blank” quiz, or they can simply go over their existing set of flash cards to test themselves. The multiple-choice and fill-in-the-blank quiz formats will provide the user with feedback, indicating how many questions that the user got correct. The user will be required to select the deck to determine which set of data the quizzes will be based on (see **FIG 10**).

FIG 10 : Choose Quiz Type Layout

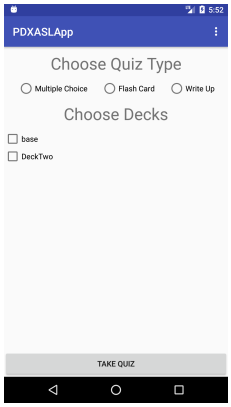
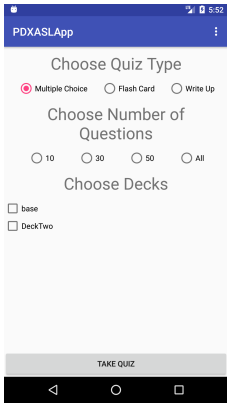


FIG 11 : Choose Quiz Type: Multiple Choice selected.



(2) Multiple-Choice Quiz - Once the user has selected the Multiple Choice option, a new list of options containing the number of questions will appear (**FIG 11**). here a user can select the number of questions that they would like to be

tested on; they can select *10, 30, 50 or All* cards to get tested on. The user will then be given the number of desired multiple choice questions (or less if the amount requested is too large for the existing card count). During the quiz, the user is shown the video, and then must select the correct answer given a list of possible words (**FIG 12**). Feedback is provided for each question, and then the full score is given to the user at the end of the quiz (**FIG 13**).

FIG 12 : Multiple Choice Layout

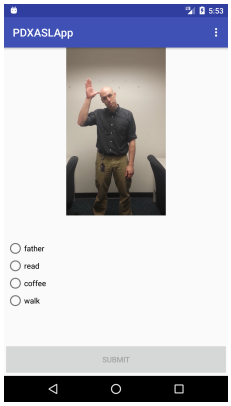
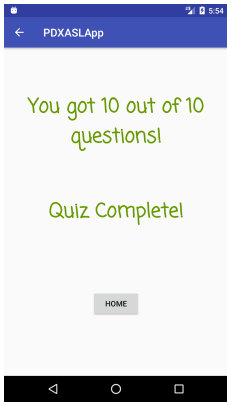
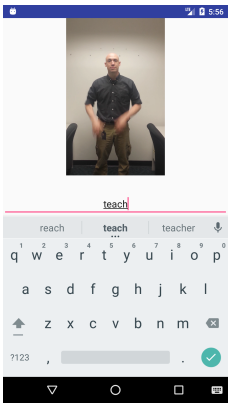


FIG 13 : Quiz Complete Layout



(3) Fill-in-the-Blank Quiz - The user will be shown the card videos, which are played in a loop. The user then needs to type in which word/phrase the video being shown represents (**FIG 14**). There will be a Toast indicating correct/ incorrect. Responses should not be case-sensitive. Feedback provided at the end.

FIG 14 : Fill-in-the-blank Quiz Layout



(4) Flash-Cards - this is not an internally assessed quiz. Users can go through their cards, view the words, and then if needed, hit the <SHOW ANSWER>

button to actually display the video (**FIG 15**, **FIG 16**). No feedback provided, as this quiz format is based on self-assessment.

FIG 15 : Flash Card - video not displayed

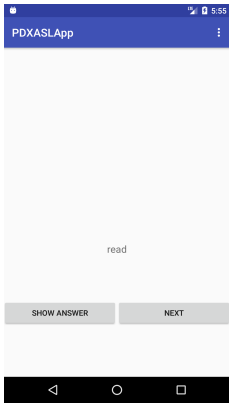
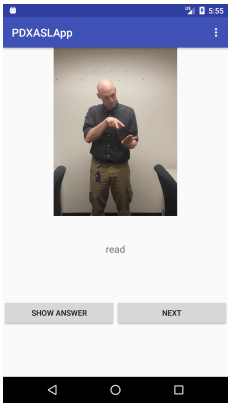


FIG 16 : Flash Card - video displayed



e) Sharing/Receiving (more backend development needed) - these two features have the UI and backend calls established, but more work is needed to develop the backend implementation of scanning QR Codes and receiving the data. Once the backend functionality is implemented, one should only need to connect to the existing “hooks” of the Sharing/Receiving layout implementation to get full functionality.

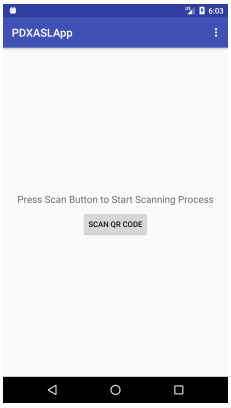


FIG 17 : Scan QR Code Layout

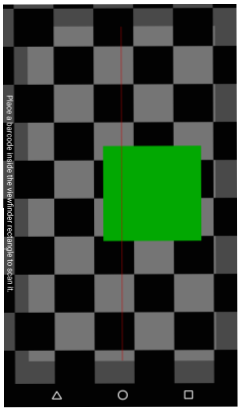


FIG 18 : Scanner Layout

FIG 19 : QR Code Layout

3. Backend

a) Create/Edit/Delete Card

(1) Create - once the video is created and paired with a word or phrase, it can then be passed to a Deck. Decks are differentiated based on (1) name and (2) the deck ID. Based on this, we can pass the video (found via it's URI) to the necessary deck (the user will select the deck to add the new card to). Once Edit Card is developed, this will pass the video's URI to Edit Card. Currently, the newly created card is passed to the database to be stored. Once the card is added to a deck, it can be used for quizzing.

(2) Edit - (Edit is fully developed on the FrontEnd, but not completely in Backend). Once this is fixed, users will be able to edit cards by time-cropping. Users will be able to "trim" the video down to 2 seconds if need be (the video must be between 2-30 seconds). At this point the video itself is unchanged, but the time "measurements" have been taken; this information is passed back to the Create Card activity. Create Card then passes this information to the backend to actually alter the bytes of the video stream. The backend side will take the full video, crop to the user's desired lengths, and give the video back to the UI to display to the user.

(3) Delete - The UI displays the list of all cards in the system. Once the user has selected the card to delete (from a list of radio buttons), that card will get removed from the system (i.e. from every deck that contains that card). Essentially two ArrayLists are created: one for the remaining decks, and another for the decks to delete. Once the selected card is removed from a particular deck, that deck is tested for its size. If the size is larger than 2, the deck is kept, otherwise it is flagged for removal at the end of this activity, and is subsequently removed from the application's database. Decks can be retrieved from the database via their deck ID or their name.

b) Create/Edit/Delete Deck

(1) Create - The user wants to create a new deck. They must enter a valid deck name (no duplicate deck names!). They must then select (from the radio group) all cards that they initially want in the deck (minimum = 2). The backend call to build a new deck requires the name of the new deck, and a

List of card objects to add to the deck. The new deck is added to a linear linked list of Deck Lists that the application uses at runtime. Each node contains a Deck name and a List of Decks. The new deck (with a deck ID) and the cards for that deck (also each with a card ID) are stored in an SQL Lite Database. The call to insert into the SQL Lite DB, given the name of the table, will return the necessary deck ID.

(2) Edit - The user wants to edit an existing deck (i.e. rename the deck and/or add or remove cards from an existing deck). The deck is retrieved from the List of decks based on its deck name. From this deck a List of the deck's cards can be obtained to edit if necessary. Once the <EDIT> button is selected, the user can rename the deck or use the radio group to select/ deselect which cards to include and/or exclude from the edited deck. The <SUBMIT> button will actually execute the changes. The deck is iterated over and cards to be removed from it are removed. Additionally any new cards selected from the List of all cards are added to the existing deck. At this point, the commit() method can be called from the modified List of cards (Deck) to actually update the database.

(3) Delete - The user wants to delete an existing deck. First, the radio button group representing the list of decks is checked for selection. Assuming a deck has been selected, a call to Backend is made with the selected deck's index to get access to this deck. A call to delete the deck is then made, and then a new radio group of decks is populated for the user to see. NOTE - if no deck is selected when the user hits the <DELETE> button, a Toast will then indicate that a deck must first be selected.

c) Quizzes

(1) Quiz Selection - A list of checkboxes paired with the deck name is first created to display to the user. This list is established by a call to backend via the getDecks() method, which returns a list of all the available decks in the application's database. Any decks that become checked will be logged. If no decks are selected, when the user attempts to take a quiz, there will be a Toast indicating to the user that they need to select a deck to proceed. If the user decides to take a multiple choice quiz, by clicking the corresponding

radio button, a new radio group of of buttons will be displayed to the user to now select the number of questions to have in the quiz (10, 30, 50, ALL). The question-count radio group will not be displayed for the other quiz types. At the end of the multiple-choice quiz and the fill-in-the-blank quiz, the user will receive feedback indicating how many questions that got correct out of the total questions provided (e.g. 9/10 Correct!).

(2) Multiple-Choice - This Activity will first retrieve a Test object from the External Test Manager class by passing it the Decks on which the test should be based, as well as options indicating the test type needed (multiple choice in this case), the question count and ordering. The External Test Manager class will return possible answers paired with the video. This Activity then will load the Card to display its video to the user. Based on the user's selection, there will be a Pair<Boolean, String> object used to represent the user's selection and the result (true/false, i.e. correct/incorrect). Based on this information, the statistics for the user's quiz score will be updated internally, and a Toast will be given to the user indicating correctness.

(3) Fill-in-the-Blank (Write-Up) - This Activity will also retrieve a Test object from the External Test Manager class by passing it the Decks on which the test should be based, as well as options indicating the test type needed, and the question count. The user is shown the video on a loop. Similar to the multiple choice test, based on the user's selection, there will be a Pair<Boolean, String> object used to represent the user's selection and the result; based on this information, as above, the statistics for the user's quiz score will be updated internally, and a Toast will be given to the user indicating correctness.

(4) Flash-Card - This Activity will also retrieve a Test object from the External Test Manager class by passing it the Decks on which the test should be based, as well as options indicating the test type needed (flash card in this case), and the question count. If the user wants to see the video, a call to set the visibility of the video to true is made. There is no internal assessment for the Flash Card Activity.

d) Sharing/Receiving

(1) As noted above, this portion of the application will be left in the development branch for further work. Currently there is no working functionality for sharing Cards and/or Decks with other users of this application.

C. Assumptions and Dependencies - user characteristics, devices, memory

1. Devices/Memory
   a) The minimum Android API is 23 for this application, per our sponsor's request. The user's device will be responsible for storing data; if the user has limited memory storage availability, this application will likely not operate optimally.
2. User Characteristics
   a) Just an interest in learning ASL. Any person capable of using a smart phone, recording videos, and playing videos should be able to utilize our application. See Functional Specification Document for further detail.

D. Class/Interfaces List (see Technical Specification Doc. for specific details)

1. BACKEND

a) Interface Stubs

(1) Card (extends Parcelable)

(2) CardManager

(3) Deck (extends Parcelable)

(4) DeckManager

(5) Question

(6) SharingManager

(a) class TxOptions

(b) class RxOptions

(7) SharingReceiveListener

(8) SharingTransmitListener

(9) Statistics

(10) StatisticsManager

(11) Test (extends Interator<Question>)

(12) TestManager

(a) class Options

(13) Video (extends Parcelable)

(14) VideoManager

(a) class ImportOptions

(b) interface VideoImportListener

b) Video Handling

(1) ImageUtil

(2) PreprocessingPipeline

(3) StubPreprocessingPipeline (extends PreprocessingPipeline)

(4) EncodeableObject (interface)

c) Managing Data

(1) BaseAssetsImporter

(2) ExternalCard (implements Card, EncodeableObject)

(3) ExternalCardManager (implements CardManager)

(4) ExternalDeck (implements Deck, EncodeableObject)

(5) ExternalDefaultDeck (implements Deck)

(6) ExternalDeckManager (implements DeckManager

(7) ExternalTest (implements Test)

(8) ExternalTestManager (implements TestManager)

(9) ExternalVideo (implements Video, EncodeableObject)

(10) ExternalVideoManager (implements VideoManager)

d) DataBase

(1) AslDbContract

(2) AslDbHelper (extends SQLiteOpenHelper)

e) Sharing

(1) Broadcast (extends BroadcastReceiver)

(2) ClientSession

(3) Protocol

(4) ServerSession (implements Runnable)

(5) SharePackage

(6) SharingClient (extends Service)

(7) SharingServer (extends Service)

(8) SharingManager (implements SharingManager interface)

2. FRONTEND

a) FrontEndTestStubs

(1) testCard (implements Card)

(2) testDeck (implements Deck)

(3) TestingStubs

(4) testMultiChoiceQuestion (implements Question)

(5) testMultiChoiceTest (implements Test)

(6) testTestManager (implements TestManager)

b) UI Classes

(1) BaseActivity (extends AppCompatActivity)

(2) CompleteQuizActivity (extends AppCompatActivity) (implements View.OnClickListener)

(3) CreateCardActivity

(4) CreateDeckActivity

(5) CreateEditDeleteDeckActivity

(6) CustomArrayListAdapter

(7) DeleteCardActivity

(8) DispQRCodeActivity

(9) EditDeckActivity

(10) EditVideoActivity

(11) FlashCardActivity

(12) HomeActivity

(13) ListRow

(14) ManageCardsSubMenuActivity

(15) ManageDecksSubMenuActivity

(16) MultipleChoiceActivity

(17) NoExtrnlStrgPrmsActivity

(18) ReceiveDeckActivity

(19) ShareDeckActivity

(20) TakeQuizSubMenuActivity

(21) VideoAspectRatioLayout

(22) VideoLayout

(23) WriteUpActivity

c) Test-UI

(1) ChooseQuizTest

(2) CreateCardTest

(3) CreateDeckTest

(4) HomeActivityTest

(5) NavigationFlowTest

(6) CardsInstrumentedTest

(7) DecksInstrumentedTest

(8) QuestionInstrumentedTest

(9) StatisticsInstrumentedTest

(10) TestInstrumentalTest

(11) VideoInstrumentalTest

V. Appendix A

A. Application Layout Map