

Flash Card

Final Document

**5 May 2015,**

**Team3:**

Derek Grove

Douglas Harshberger

Dylan Small

Evan Connors

Michael Nazzario

Table of Contents

I. Introduction 4

I.I         → Purpose

I.II        → Scope

I.III → Intended Audience

I.IV      → Rationale

I.V → Major Constraints

II. Project Organization 5

II.I        → Process Model

II.II       → Organizational Structure and Responsibilities

II.III → Actual Effort Schedule

III. Team-Specific Aspects 8

III.I        → Management Objectives and Priorities

III.II       → Team Availability

III.III → Team’s Range of Skills and Experience

IV. System Overview 9

IV.I → System Architecture

IV.II → System Explanation

IV.III → Use Case Model

IV.IV → Use Case Scenarios

V. Requirements 12

V.I → Functional Requirements

V.II → Requirement Traceability Matrix

V.III → Nonfunctional Requirements

VI. Design Considerations 13

VI.I → Assumptions and Dependencies

VI.II → General Constraints

VI.III → Goals and Guidelines

VI.IV → Development Methods

VII. Architectural Strategies 14

VII.I → Engineering Paradigm

VII.II → System Hierarchy (Packaging)

VII.III → System IO

VIII. Class Design 15

VIII.I → High-Level Class Diagram

VIII.II → Detailed Class Diagram

VIII.III → Class Explanation

IX. Processes 23

IX.I → High-Level Sequence Diagram

IX.II → High-Level Sequence Diagram Explanation

IX.III → Manage Sequence Diagram

IX.IV → Manage Sequence Diagram Explanation

IX.V → Practice Sequence Diagram

IX.VI → Practice Sequence Diagram Explanation

IX.VII → Quiz Sequence Diagram

IX.VIII → Quiz Sequence Diagram Explanation

X. Database Design 27

X.I → DB Relational Schema (High-Level)

X.II → DB Schema Definitions

X.III → DB E-R Diagram

X.IV → DB Glass-Box Explanation

XI. Data Flows 31

XI.I → Context Data Flow Diagram

XI.II → Context Data Flow Explanation

XI.III → Level 1 Data Flow Diagram

XI.IV → Level 1 Data Flow Explanation

XI.V → Level 2 Data Flow Diagram

XI.VI → Level 2 Data Flow Explanation

XII. GUI 35

XII.I → Login Case Diagram

XII.II → Login Explanation

XII.III → Manage Case Diagram

XII.IV → Manage Explanation

XII.V → Practice Case Diagram

XII.VI → Practice Explanation

XII.VII → Quiz Case Diagram

XII.VIII → Quiz Explanation

XIII. API 43

XIII.I → Project to Eclipse Environment

XIII.II → Project to Android Studio Environment

XIII.III → External Android Device Testing

XIII.IV → Load Local SQLite DB to SQLiteBrowser

XIII.V → Find Apache2 WebApp Files

XIII.VI → Locate MySQL DB

XIV. Glossary 51

→ Terms and Definitions

I. Introduction

.I Purpose:

The purpose of the Japanese Flash card application was to provide a better way for our client’s students to study Japanese. By updating and repairing the app while also adding functionality we have ensured a more pleasant experience than what was previously offered. We intended to create an easy way for the client to change the content of the app as well as add more functionality for the end user.

The purpose of this document is to be a reference to any person who wants to implement or any person interested in the architecture of the Flash Card application. This document describes the application’s architecture and each sub-architecture along with their associated interfaces, database schemas, and the motivations behind the chosen design.  Both high-level and detailed-level designs are included in this document.

In order to properly understand this document one should have a technical background and be able to interpret Data Flow Diagrams (DFD) as well as Unified Modeling Language (UML) diagrams.

.II Scope:

Japanese Flash Card app will be utilized by Dr. Yongtaek Kim and given to his students to be used as a tool to study the Japanese language in a flash card system. The app has the ability to display an image and give options for the user to select the answer that best describes the image, as well as another tool which provides the picture and shows the answer on the back. There will be support for audio to help with the answering process. There has been functionality added to allow the client to manipulate the database in a more user-friendly manner. We also changed the language from Kanji to Hiragana at the request of the client. At this time there is no functionality for user logins however this may be implemented in the future to make the application arbitrary for every class.

The scope of this document will include the architecture and design of the Flash Card application. This application is to provide a student with a flash card based quiz and assist with the memorization of terms that an instructor or an administrator inputs.

.III Intended Audience:

The intended audience of this document includes the technical team currently developing the application, managers,  Dr. Yongtaek (current client), and future developers of the application.

.III Rationale:

The purpose of this application is to test the end user on their knowledge of the Japanese language. We provided a clean and easy to follow GUI that would enable the user to have a pleasant learning experience. We also must updated the app to the client’s specifications by adding additional chapters and fixing the text element so that it displays the correct Japanese symbols for each image.

.IV Major Constraints:

In order for the server-side to work, one must implement a LAMP Server which will host the MySQL DB and the PHP server scripts. One must also have access to an Android device in order to run the application.

II. Project Organization

.I Process Model:

The High-Level timeline was as follows:

* Requirements Iteration
  + Feb 5th - Problem Statement
  + Feb 12th - Project Plan & Software Requirements Specification
* Analysis Iteration
  + Feb 26th - Software Design Specification v1, High-Level Design
  + Feb 19th - Planning Checkpoint Review
* Design Iteration
  + Mar 26th - Software Design Specification v2, Detailed Design
  + Mar 5th -   Prototype
* Implementation Iteration
  + Mar 23rd - Beta
  + Apr 4th -  Verification and Validation Results
  + May 5th - Product Release
* Post-Delivery Iteration
  + May 5th - Legacy documentation and Final Report

.II Organizational Structure and Responsibilities:

In order for someone to pick-up on this project where we left off, one should at least assign the following roles:

Derek is the Business System Analyst Team Leader, his responsibilities include:

* POC for any problems/concerns with the project
* Organization and publishing of team deliverables
* Planning/scheduling and reviewing/editing of documents; Operations

Doug is the Technical Platform Specialist, his responsibilities include:

* System logic and data flows
* Engineering of components/objects of the system
* Glass-Box Testing

Dylan is the Lead Architect, his responsibilities include:

* Development of architecture documentation
* GUI and event module integration and management
* Black-Box Testing

Evan is the Technical Platform Team Leader, his responsibilities include:

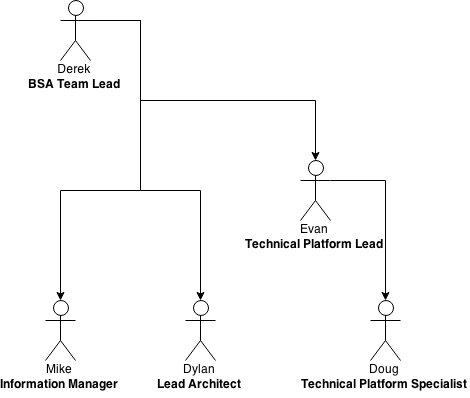
* System logic and data flows
* Engineering of system structure and hierarchy
* System management, API documentation

Mike is the Information Manager, his responsibilities include:

* Data Entry and Documentation
* Manual/Minimal-programmatic changes
* Hardware/OS integration and validation

Team Member 6 is the Graphical designer, his/her responsibilities include:

* Creating image buttons for the application and icon
* Creating a meaningful title screen and recurring background
* Creating images for the language so team can hold rights to the app for Google Play

[](https://drive.draw.io/#G0B8wVQUrDBEuiS3NlYUpwVHVWUUk)

.III Actual effort schedule:

|  |  |
| --- | --- |
| **Date & Task** | **Hours** |
| Project Plan | 5 |
| Software Requirements Specifications | 2 |
| Software Design Specification V1 (High Level Design) | 7 |
| Planning Checkpoint Review | 2 |
| Software Design Specification V2 (Detailed Design) | 10 |
| Prototype | 10 |
| Verification and Validation Plan | 10 |
| Beta | 50 |
| Verification and Validation Results | 10 |
| Product Release | 5 |
| Legacy documentation and Final Report | 10 |
| **Total Hours** | 121 |

III. Team-specific aspects

.I Management Objectives and Priorities:

Our top initiative was to have a fully documented and working product at the end of the spring 2015 semester. We assessed our progress at the end of each week with scorecards in which we highlighted achievements and outstanding items that carried onto the following week. We followed the Agile process and good programming practices, and abided by all SE codes of conduct.

.II Team Availability

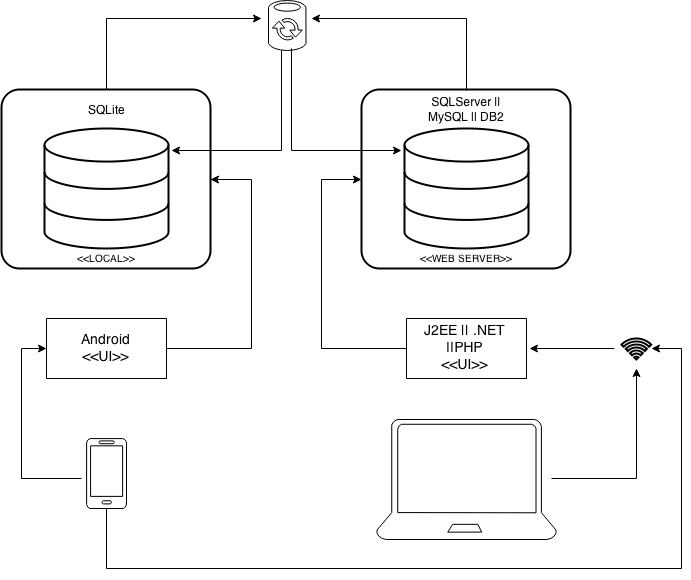
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** |
| **Derek** | After 6:00PM | After 7:00PM | After 6:00PM | After 7:00PM | After 3:00PM |
| **Douglas** | After 2:30 PM | After 6:30PM | After 12:30PM | After 6:30PM | After 12:30PM |
| **Dylan** | After 3:30PM | After 4:30PM | After 3:30PM | After 4:30PM | After 3:30PM |
| **Evan** | After 2:30PM | After 3:30PM | After 12:00PM | After 3:30PM | After 3:00PM |
| **Michael** | After 4:30PM | After 9:30AM | After 3:30PM | After 9:30AM | After 3:30PM |
| **Group Meeting** | 4:30 PM | 6:30 PM | 3:30 PM | 6:30 PM | 3:30 PM |

.III Team’s Range of Skills and Experience

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Skill (1-5)** | **Derek** | **Doug** | **Dylan** | **Evan** | **Mike** |
| Java J2SE | 4 | 4 | 2 | 4 | 2 |
| Human-Computer Interaction (User interface) | 3 | 2 | 4 | 4 | 3 |
| General Database | 3 | 4 | 4 | 4 | 3 |
| SQL | 3 | 3 | 3 | 4 | 3 |
| Team work | 3 | 4 | 4 | 3 | 4 |
| Team management | 4 | 3 | 4 | 3 | 3 |
| Time management | 5 | 3 | 4 | 3 | 4 |
| Professional writing | 4 | 3 | 5 | 2 | 5 |
| Editing/Proofreading | 4 | 3 | 4 | 4 | 5 |
| Debugging/Troubleshooting | 3 | 2 | 2 | 4 | 5 |

IV. System Overview

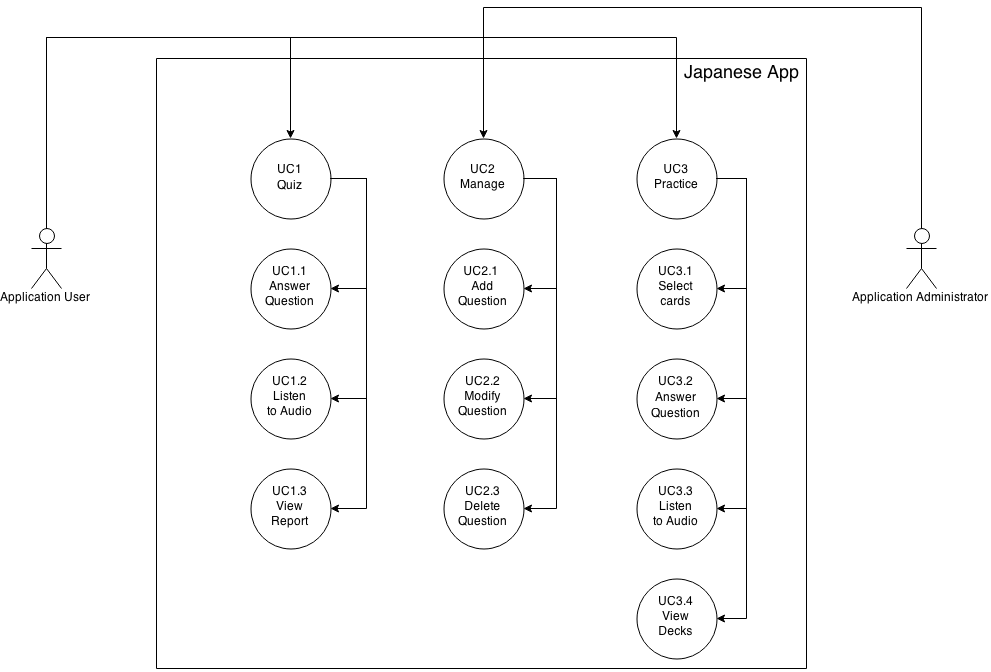
.I System Architecture:

***[](https://drive.draw.io/#G0B3Y_Ai05HOYwQ3pJbGZuLWxfWVk)***

.II System Architecture Explanation:

There will be two UIs. One interface which will be the client-side viewing UI will be that with an Android device and the app installed. This system will use a local DB, SQLite, to view and manipulate data. The other UI will be a server-side language UI to a server DB. This will be the primary way for admins to go in and manipulate their databases in which the clients can sync up with the server. The server-side UI and DB are undecided currently, however it is probable that the UI will be a PHP web-page to a MySQL DB.

.III Use Case Model:

***[](https://drive.draw.io/#G0B8wVQUrDBEuiRnBmQk1YeUM4alE)***

.IV Use Case Scenarios:

|  |
| --- |
| **UC01: Quiz** |
| **Brief Description**  This use case allows the app user the ability to answer questions in a weighted system and see their final score of a section. |
| **Step-by-Step Description**   1. App User opens up the app and selects “Quiz”. 2. App User will then be given a question after they choose a lesson.    1. App User then selects an answer for the question.    2. App User may listen to audio for the question.    3. App User may choose to “Check Answer”.    4. App User chooses to go “Next” or “Previous”. 3. App User answers all of the questions in selected section. 4. App User receives a final report. |
| **UC02: Manage** |
| **Brief Description**  This use case allows the application administrator the ability to import, modify, and delete questions and answers into the app using SQLiteBrowser. |
| **Step-by-Step Description**   1. App Administrator opens up the app and selects the “Management” tab. 2. App Admin will then be able to modify the database that the app accesses.    1. App Admin adds a question after filling in all required attributes in order to instantiate the object.    2. App Admin selects a question in which to edit a property contained within.    3. App Admin selects a question in which to delete and presses the “Delete” tab. |

|  |
| --- |
| **UC03: Practice** |
| **Brief Description**  This use case allows the app user the ability to practice questions within the app. |
| **Step-by-Step Description**   1. App User opens up the app and selects “Practice”. 2. App User will then be given a question after they choose a lesson.    1. App User will select cards from a list in which to practice.    2. App User then selects an answer for the question, answer will show after given.    3. App User may listen to audio for the question.    4. App User chooses to go “Next” or “Previous”. 3. App User answers all of the questions in the cards selected for the deck. 4. App User may now see the cards that were put in “Wrong” and “Correct” categories. |

V. Requirements

.I Functional Requirements:

FR1: The system shall quiz students on their Japanese vocabulary words, give them the correct answer, and also give them a total score after completing the entire lesson.

FR2: The system shall keep a database of corresponding images, audio files, and Japanese equivalent words (in Hiragana), and false (trick) answers. Information will be organized according to class lessons (7 - 12).

FR3:The system shall allow the instructor, Dr. Yongtaek Kim, to add/delete/change content on database easily, so he can update the application to fit with class material.

FR4: The system shall allow students to use a practice mode.

.II Requirement Traceability Matrix:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **UC1** | **UC2** | **UC3** |
| **FR1** | X |  |  |
| **FR2** |  | X |  |
| **FR3** |  | X |  |
| **FR4** |  |  | X |

.III Nonfunctional Requirements:

NR0: The system shall have minimal downtime, and once completed, the system should be available 95 % of the time, with 5% given for maintenance, database updates, and unforeseen failures.

NR1: The system shall support Android phones and tablets using Android Versions 4.0.3(Ice Cream Sandwich) to 5.0 (Lollipop) when released.

NR2: The system shall be light-weight, and be able to run on cheaper Android devices with low processing ability.

VI. Design Considerations

.I Assumptions and Dependencies:

Our app will be able to install on any Android phone via an Android APK file.  The app will use SQLite on the local end to control all of the data going into the questions and answers.  The user will be able to import the multiple components required to create a question object into the app and the app will auto-generate questions and answers from the file.

Assuming that that the server and server-side DB is running, we will need the user to have an internet connection in order to synchronize with the server DB for updates, or re-download the APK if the server is not running and the local storage is the only system available. The admin/teacher should have a desktop or laptop in which they can ping the admin web portal to manipulate the server DB in a efficient manner. However data is able to be manipulated from the application, in which can be synced from local to server.

.II General Constraints:

Due to the nature of the project there are a few constraints associated with the project. One is that the end-user must be using a smartphone using the Android Operating System. Another constraint is the general way of updating questions must be manually entered into the database using SQLiteBrowser or on the server side in MySQL Workbench.  Also, the app is a relatively large application in order to accomodate all devices. Devices on Android 4.0+ will run significantly better than lower APIs.

Our biggest constraint is that the application is only for the Android operating system. Time was also a constraint, we only had until the end of the semester to complete the application for the client so that he may present the application at a conference. This was difficult as the first time we had programmed in an Android environment.

.III Goals and Guidelines:

Our main goal is to generalize this application and implement more features so that the student and admin will have a better experience overall with the application. We want to make the Flash Card application into a tool for future professors to use in their classes, in an arbitrary fashion.

.IV Development Methods:

This project will utilize a paradigm with 3 builds: Alpha, Beta, and Gamma. Each time becoming more stable and robust with more features implemented. We will make the final product very user intuitive. Also allowing the Admin to easily access data and options to edit the data.

VII. Architectural Strategies

.I Engineering Paradigm:

The architectural design of this application uses the Object Oriented Paradigm (OOP). Android application are coded in Java along with XML referencing. Using an Object Oriented language like Java allows us to easily design and implement modules. We will also be using SQLite for the local DB. The database can currently be edited/updated using the SQLiteBrowser tool, which provides a graphical and tabulated user interface for the database.

The web management portal will most likely be coded in the PHP language which is also a O-O language. The server-side DB that the PHP web-page will provide a UI for will be comprised with MySQL.

.II System Hierarchy (Packaging):

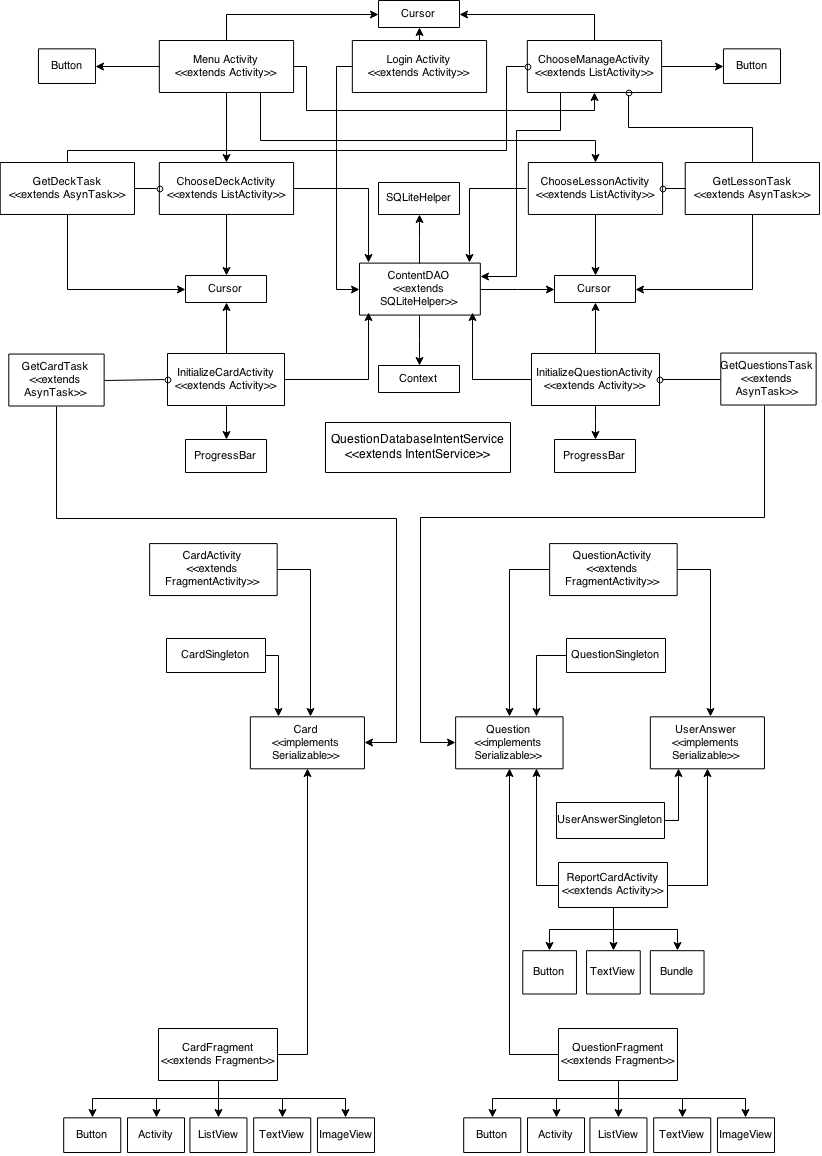
The Java developer-generated source-code is organized into four different packages. The default package contains DB connection classes and the Title Screen class. Manage package contains classes in order to fully implement the Manage avenue. Practice package contains classes in order to fully implement the Practice avenue. Quiz package contains classes in order to fully implement the Quiz avenue.

.III System IO:

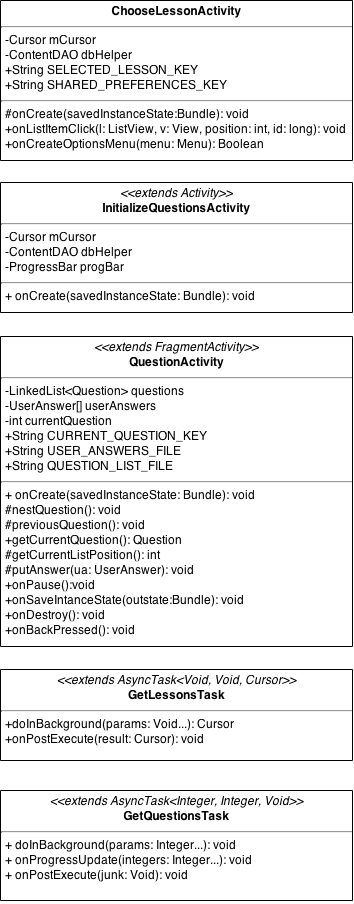
Currently, all application data will reside on the android device that it is installed on. The application requires no internet access to function. The SQLite database lies locally within the application, so access to a remote database is only required if content has been manipulated since the application was installed.

VIII. Class Design

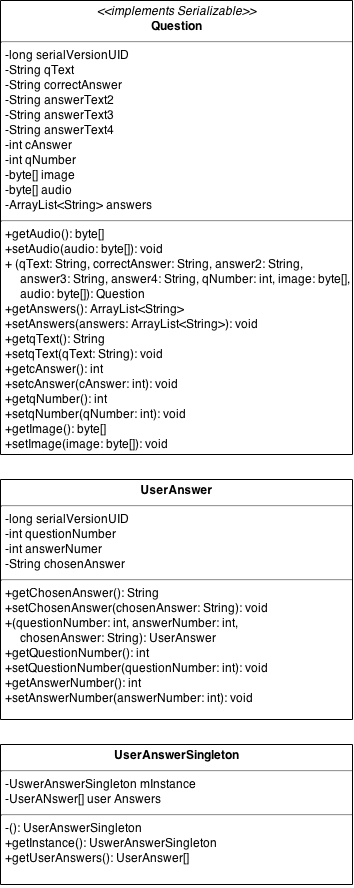
.I High-Level Class Diagram:

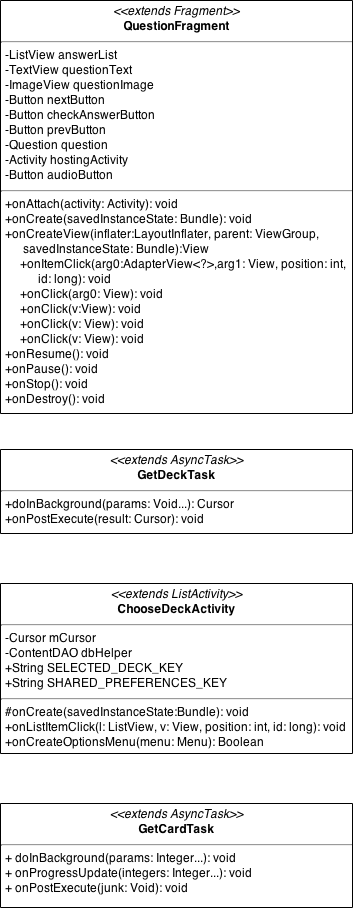
***[](https://drive.draw.io/#G0B3Y_Ai05HOYwRHF0TGotMXE4Qms)***

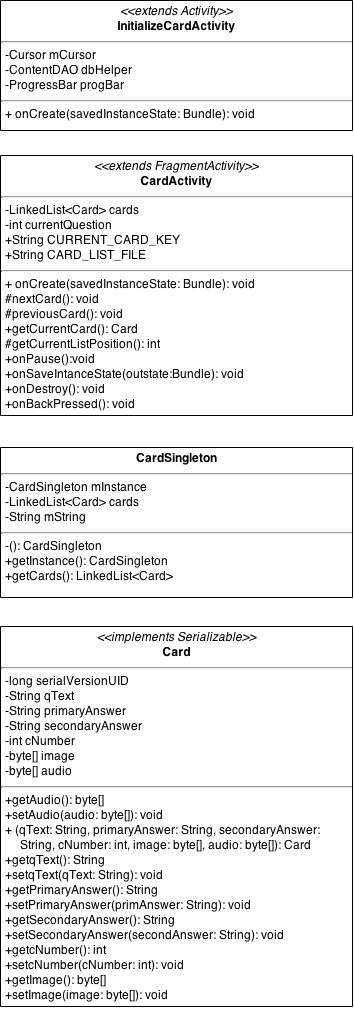
.II Detailed Class Diagram:

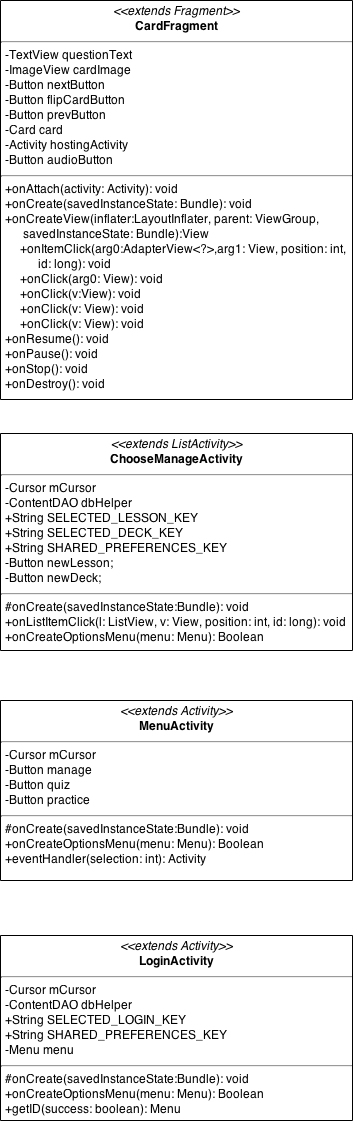
***[](https://drive.draw.io/#G0B3Y_Ai05HOYwWktISDZlVy1ubWs)***

***[](https://drive.draw.io/#G0B3Y_Ai05HOYwWktISDZlVy1ubWs)***

***[](https://drive.draw.io/#G0B3Y_Ai05HOYwWktISDZlVy1ubWs)***

***[](https://drive.draw.io/#G0B3Y_Ai05HOYwWktISDZlVy1ubWs)***

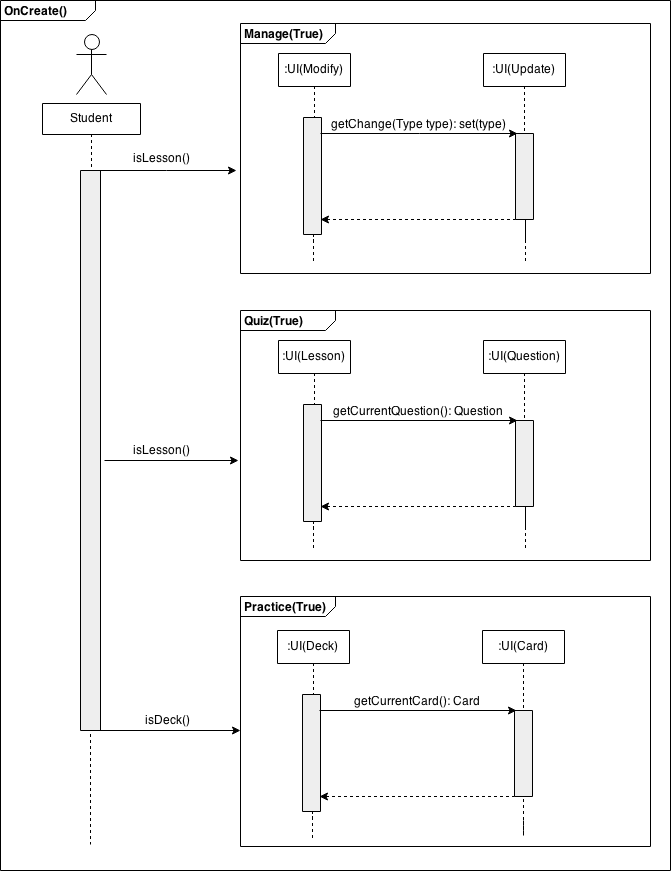
***[](https://drive.draw.io/#G0B3Y_Ai05HOYwWktISDZlVy1ubWs)***

***[](https://drive.draw.io/#G0B3Y_Ai05HOYwWktISDZlVy1ubWs)***

.III Class Explanation:

* LoginActivity is the screen a user will be brought to when initially launching the application if away for a long period of time. User logs in which will determine the credentials as a student or admin/teacher.
* MenuActivity is the home screen that is initially launched when the user logs in. Also serves as the home page that is accessible by an action bar shortcut.
* ChooseManageActivity is the first thing to appear when the manage avenue is selected. It displays the list of lessons and decks.
* ChooseLessonActivity is the first thing to appear when the quiz avenue is selected.  It displays a list of lessons which can be chosen by the user.
* InitializeQuestionsActivity gathers all of the questions for the selected lesson, displaying a spinning progress indicator in the meantime.
* QuestionActivity keeps track of the current question in the lesson set, the student’s recorded answers, and other information relevant to the session.
* GetLessonsTask returns the lesson chosen by the user so that the program knows which set of questions to retrieve.
* GetQuestionsTask is the specific entity tasked with gathering the questions for a given lesson.
* ReportCardActivity calculates and displays the student’s performance on the lesson’s question set.
* ContentDAO deals with all direct interactions between the database and the rest of the program.
* QuestionDatabaseIntentService handles internal database updates by connecting to a remote server.
* QuestionSingleton creates a container for a set of questions.
* Question defines the format of all the information involved in a single question.
* UserAnswer defines the format for a record of a student’s answer to a given question.
* UserAnswerSingleton creates a container for all of the student’s answers to questions.
* QuestionFragment sets up and organizes the components on the screen when a question is being displayed.
* ChooseDeckActivity is the first thing to appear when the practice avenue is selected.  It displays a list of decks which can be chosen by the user.
* InitializeCardsActivity gathers all of the cards for the selected deck, displaying a spinning progress indicator in the meantime.
* CardActivity keeps track of the current card in the deck set, and other information relevant to the session.
* GetDeckTask returns the deck chosen by the user so that the program knows which set of cards to retrieve.
* GetCardsTask is the specific entity tasked with gathering the cards for a given deck.
* CardSingleton creates a container for a set of cards.
* Card defines the format of all the information involved in a single card.
* CardFragment sets up and organizes the components on the screen when a card is being displayed.

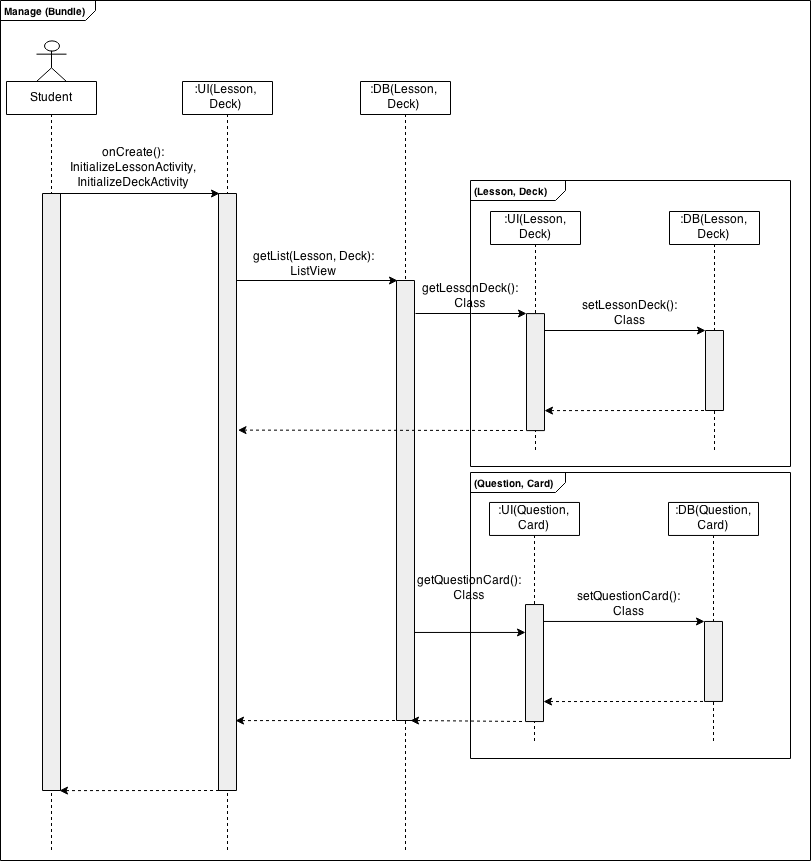
IX. Processes

.I High-Level Sequence Diagram:***[](https://drive.draw.io/#G0B3Y_Ai05HOYwU0NuN0xyempiQlk)***

.II High-Level Sequence Diagram Explanation:

The Flash-Card application is broken down into three main processes. They are, manage, practice and quiz. Once logged in, a user may enter any of the portals. Manage serves as a way for the user to manipulate the data stored on the database. Practice serves as a way for users to view the content in a non-weighted system. Quiz serves as a way for users to view the content in a weighted system that prints out a report at the end.

.III Manage Sequence Diagram:

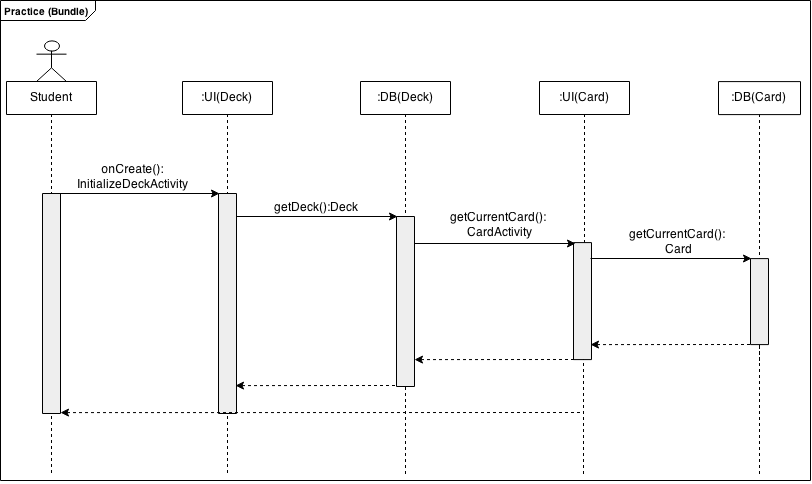
***[](https://drive.draw.io/#G0B3Y_Ai05HOYwSTE0YlEyRktDMTQ)***

.IV Manage Sequence Diagram Explanation:

There are two management portals, one for an admin and one for a general user. The general user may manipulate/delete/add any data pertaining to the practice avenue. The user is able to upload pictures and audio from their phone as well as put a primary and secondary answer. Those changes if saved will be written to the local DB which can then later be written to the server DB with an internet connection

The admin portal has all of the same capabilities as well as the ability to manage the quiz track. The lessons and questions on the quiz track are the same for everyone unlike the practice track being that of a client-side UI. The admin can manipulate/delete/add any data pertaining to both avenues. The local DB will not automatically synchronize with the server DB until the admin commits those changes. This is in case of a malfunction or human-error, the admin can sync with the server DB if changes to the local were unintended.

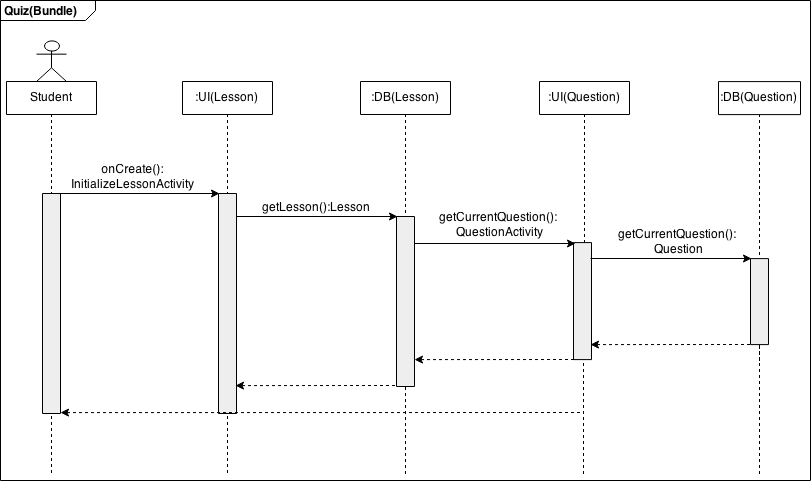
.V Practice Sequence Diagram:

***[](https://drive.draw.io/#G0B3Y_Ai05HOYwUDg5bmtGU0hoa1k)***

.VI Practice Sequence Diagram Explanation:

If the practice avenue is selected in the home screen, a user will be prompted with a list of custom and admin made decks from the local DB. A deck contains cards, which will be displayed if their parent (deck) was chosen by the user. The content will be filled programmatically from selecting fields from the Card table in the local DB.

.VII Quiz Sequence Diagram:

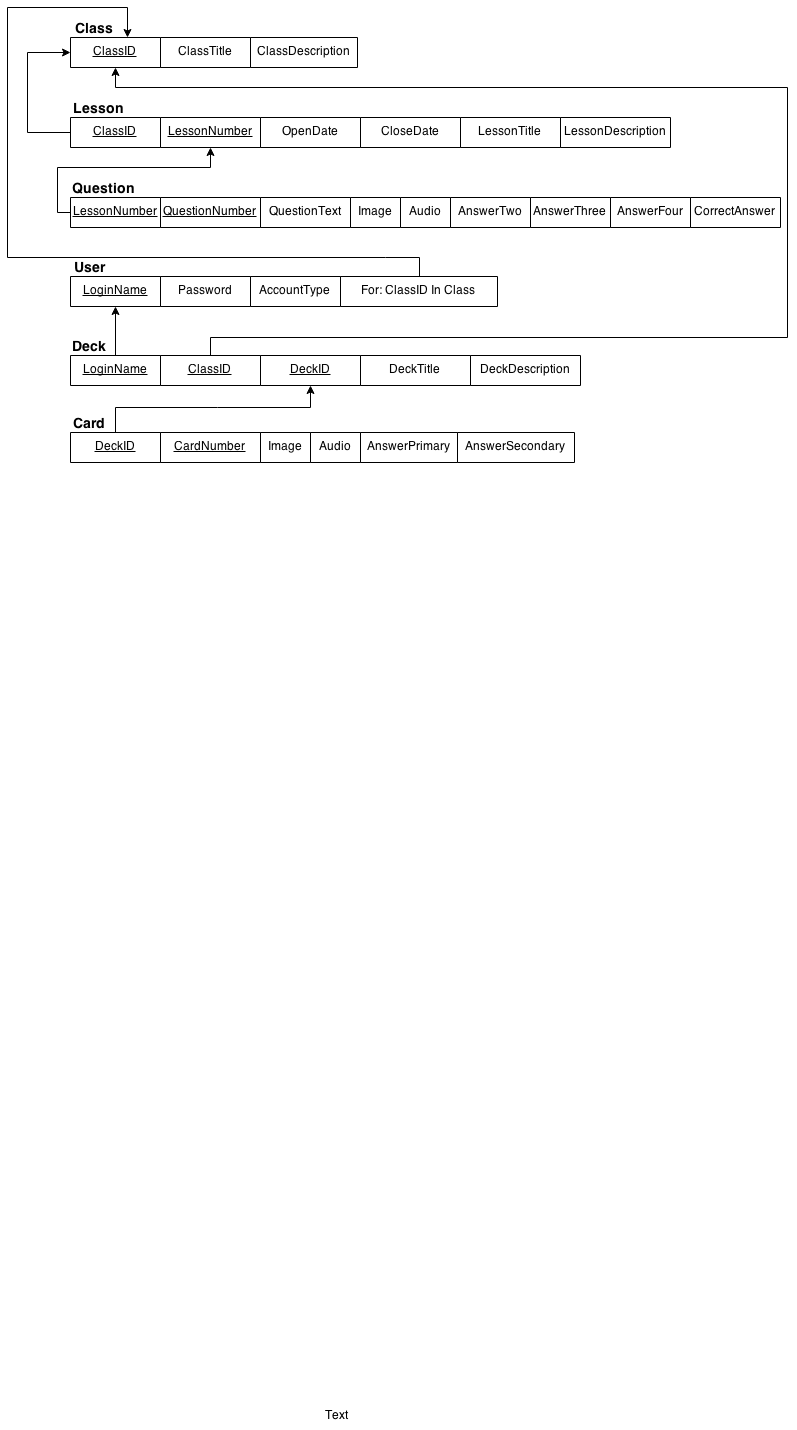
***[](https://drive.draw.io/#G0B3Y_Ai05HOYwUTFvUmVobl9WYzg)***

.VIII Quiz Sequence Diagram Explanation:

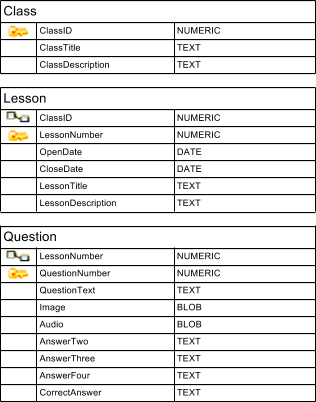
If the quiz avenue is selected in the home screen, a user will be prompted with a list of admin made lessons from the local DB. A lesson contains questions, which will be displayed if their parent (lesson) was chosen by the user. The content will be filled programmatically from selecting fields from the Question table in the local DB.

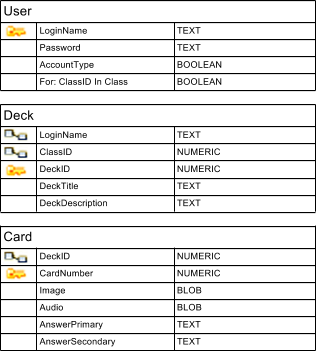
X. Database Design

.I DB Relational Schema:

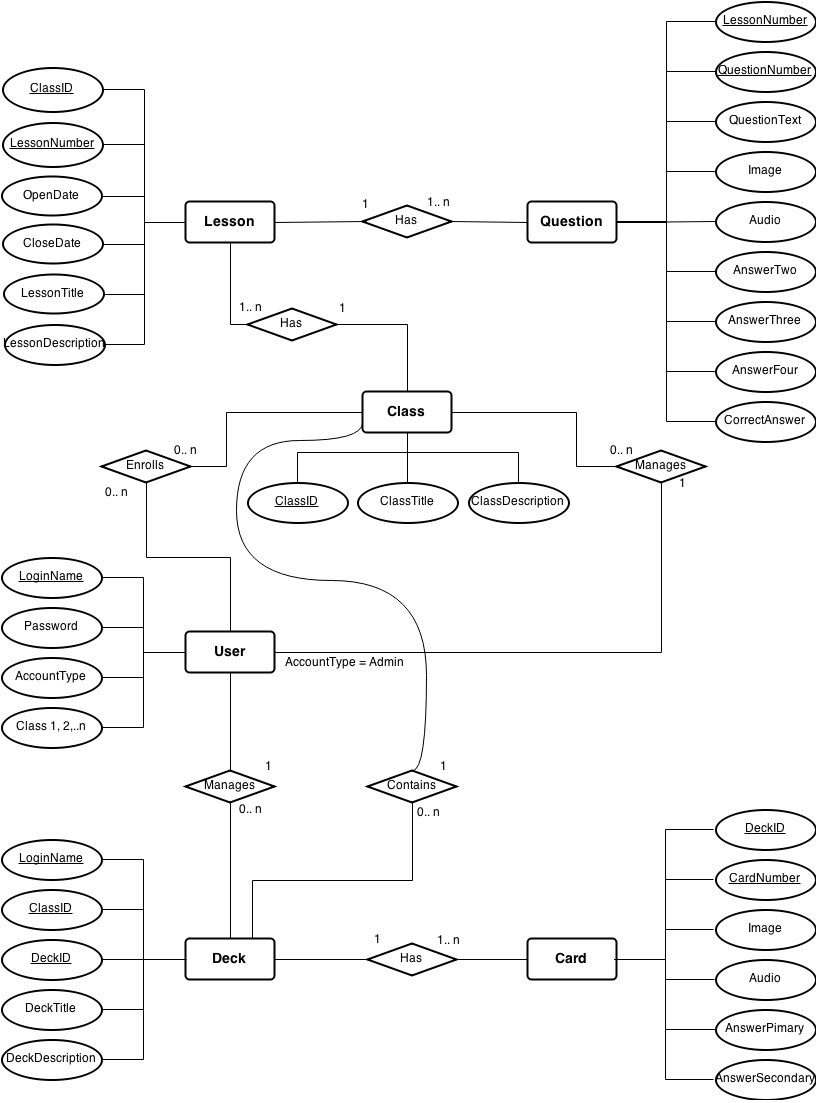
***[](https://drive.draw.io/#G0B3Y_Ai05HOYwV3BXLVNhY2dWTHc)***

.II DB Schema Definitions

******

******

.III DB E-R Diagram:

***[](https://drive.draw.io/#G0B3Y_Ai05HOYwak5KLUhiTk5NSjQ)***

.IV DB Glass-Box Explanation:

The current database consists of six different tables. The first table being a User consists of four attributes. LoginName which is type TEXT is the primary key which shall uniquely identify a user. To confirm a User, a Password attribute of type TEXT is checked. There is an AccountType, type BOOLEAN, which identifies if the User is an admin type or not. The last is the flagged Classes that the User is enrolled in.

The second table, titled Class, has three attributes. The first attribute is ClassID, which is NUMERIC and will consist of three numbers to uniquely identify a Class. The second attribute is ClassTitle which is type TEXT. ClassDescription describes the class in a TEXT format.

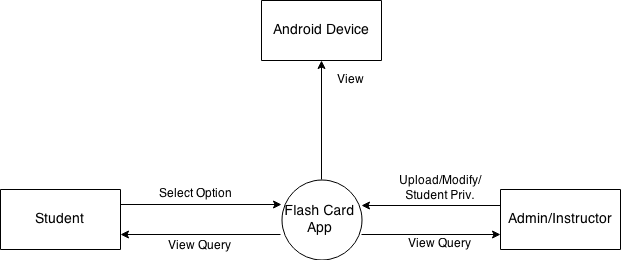
The third, titled Lesson, has six attributes. The first attribute is LessonNumber, which is type NUMERIC. The second attribute is ClassID which is also NUMERIC. LessonNumber and ClassID serve as the primary keys for Lesson, which uniquely identifies each lesson. The third attribute, LessonTitle, is type TEXT, and the fourth, LessonDescription, is type TEXT. There is also a OpenDate and CloseDate attributes which are both defined as a DATE. Each individual Lesson row has many Questions associated with it. Also, ClassID is a foreign key that is synced with ClassID in the Class table to ensure referential integrity.

The  fourth table in the database, titled Question, which has nine different attributes. The primary key of Question is a combination of QuestionNumber and LessonNumber, which are both type NUMERIC. QuestionText is type TEXT, Image is type BLOB, and Audio is type BLOB. AnswerTwo, AnswerThree, Answer Four, and Correct Answer are type TEXT. Every row in the Question table is associated with only one Lesson. Also, LessonNumber is a foreign key that is synced with LessonNumber in the Lesson table to ensure referential integrity.

The fifth, titled Deck, has three attributes. The first attribute is DeckNumber, which is type NUMERIC. DeckNumber is the primary key of Deck, which uniquely identifies each deck. The second attribute, DeckTitle, is type TEXT, and the third, DeckDescription, is type TEXT. Each individual Deck row has many Cards associated with it.

The sixth table in the database, titled Card, which has six different attributes. The primary key of Card is a combination of CardNumber and DeckNumber, which are both type NUMERIC. QuestionText is type TEXT, Image is type BLOB, and Audio is type BLOB. AnswerPrimary, and AnswerSecondary are type TEXT. Every row in the Card table is associated with only one Deck. Also, DeckNumber is a foreign key that is synced with DeckNumber in the Deck table to ensure referential integrity.

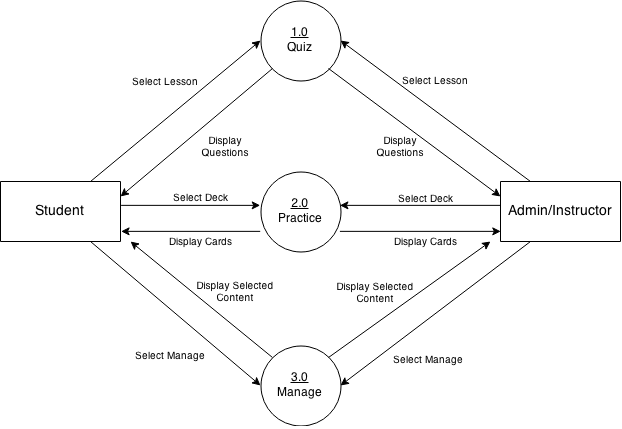
XI. Data Flows

.I Context Data Flow Diagram:  
[](https://drive.draw.io/#G0B3Y_Ai05HOYwb1Utc2pmdW40UXc)

.II Context Data Flow Explanation:

The main application will be run on an Android device. There will be two types of users (Student and Admin). A student is able to manipulate local DB data which can be synced with the server. The main function of a student however is to view the data from the local DB. It only needs synced with the server DB if an admin changes the DB. The main function of the admin is to manipulate and manage the data, and manage who can enter their class.

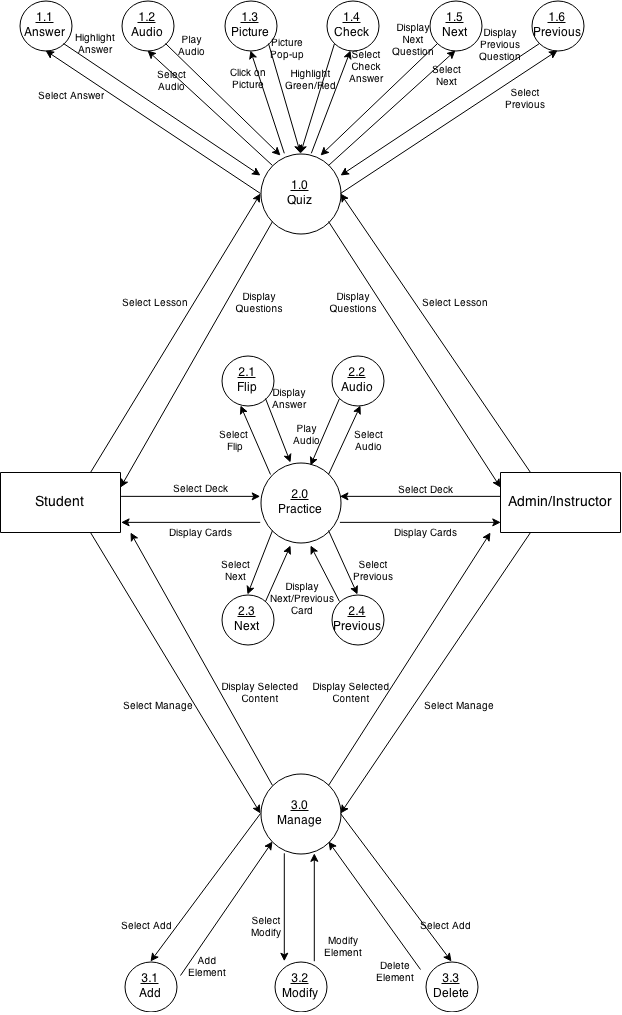
.III Level 1 Data Flow Diagram:

***[](https://drive.draw.io/#G0B8wVQUrDBEuibEM4aXJEcXlUQVU)***

.IV Level 1 Data Flow Explanation:

There are two main streams of data, that is to manipulate or view for this application. The views will differ depending on if the user chooses option 1, 2, or 3. Option 1 presents the user with admin-made content. Option 2 presents the user with admin-made and self-made data if the user made a custom deck. Option 3 presents the user with both lists of decks and lessons (lessons are able to be manipulated if and only if the user is of type admin).

.V Level 2 Data Flow Diagram:

***[](https://drive.draw.io/#G0B3Y_Ai05HOYwellST3UwUm50alk)***

.VI Level 2 Data Flow Explanation:

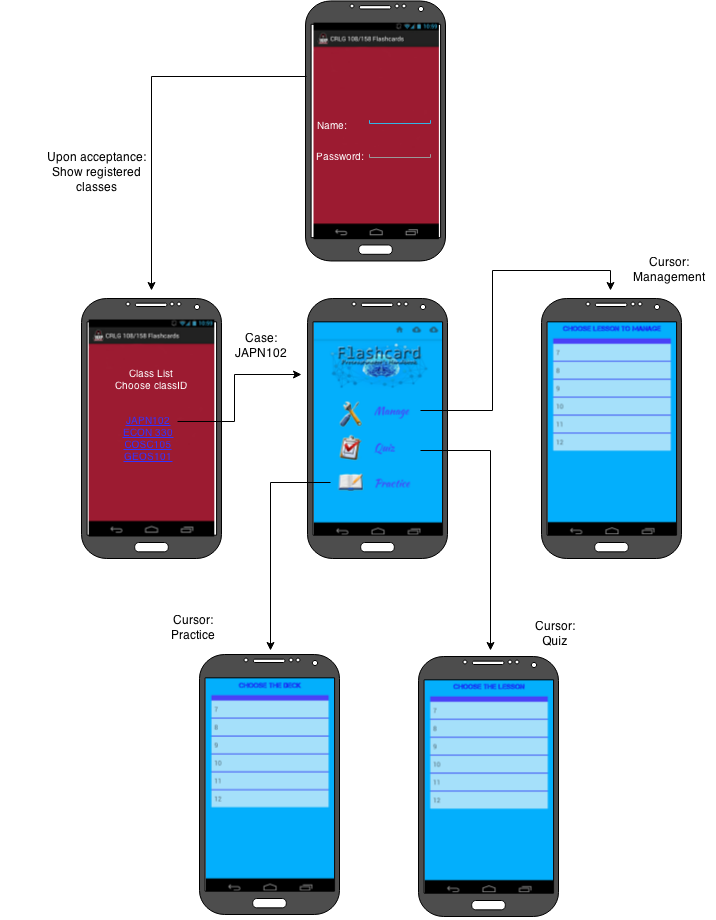
When the Quiz portal is selected, the user will be presented with a list of lessons for that class. Based upon the selection of the lesson, questions will be queued in for it. The first question will populate the screen unless there was a saved instance. Based upon the operation the user selects, will determine the interaction of the data.

When the Practice portal is selected, the user will be presented with a list of decks for that class. Based upon the selection of the deck, cards will be queued in for it. The first card will populate the screen unless there was a saved instance. Based upon the operation the user selects, will determine the interaction of the data.

When the Manage portal is selected, the user will be presented with a list of decks and lessons for that class. From here a user may edit the lesson or deck properties or even add or delete a lesson or deck. Based upon the selection of the deck or lesson, cards or questions [respectively] will be queued in for it. The list of cards or questions may be individually modified or deleted. User may also add to the list. Admin of the class may edit quiz portal data as well as practice portal data, which a student only has modifying privileges to.

XII. GUI

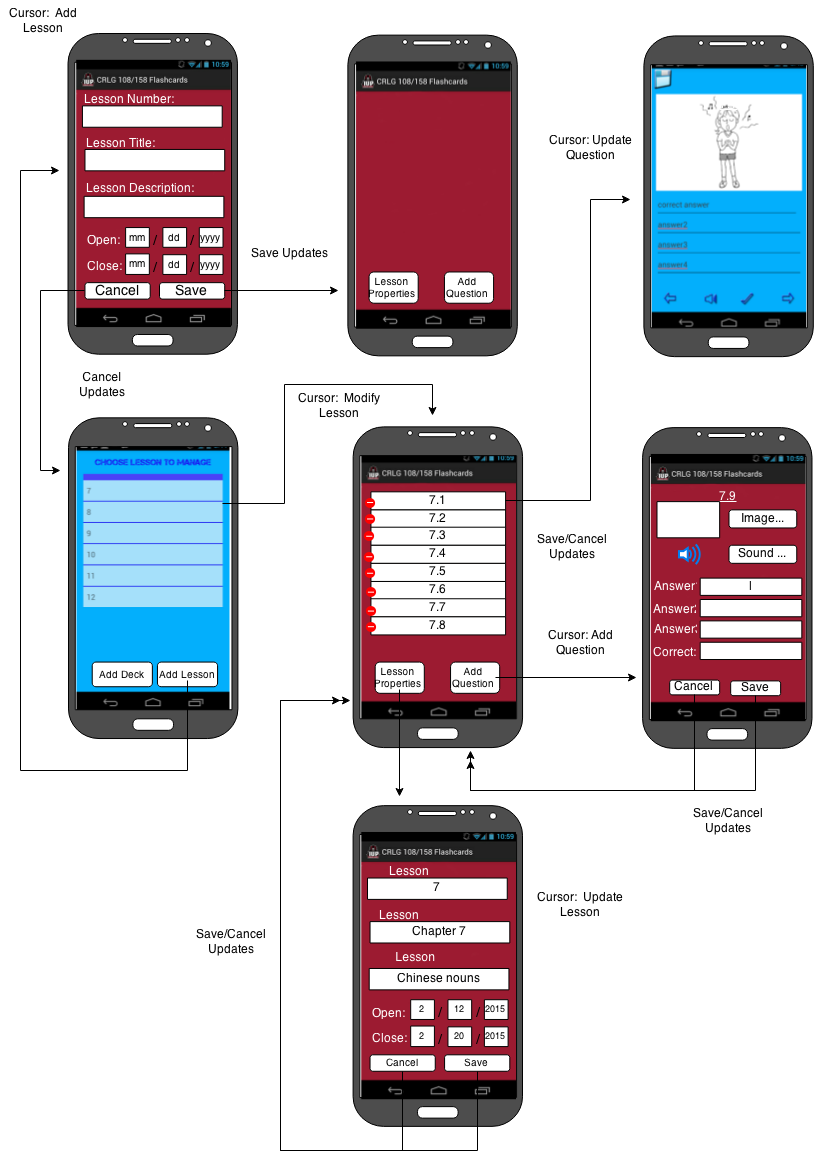
.I Login Case:

***[](https://drive.draw.io/#G0B3Y_Ai05HOYwMlM0UVdSWlRsUWc)***

.II Login Explanation:

Once opening the app you will be prompted with a login screen where the user will input his/her credentials to access the app. Once logged in the user may choose which class they want to access (only classes currently enrolled in will display). The user will then be prompted the home screen which will have a shortcut actionbar item reference, and provide the user with the three main functions. Login will only be necessary if the application has been idle for some time, or is the first time opening up the app.

.III Manage Case:

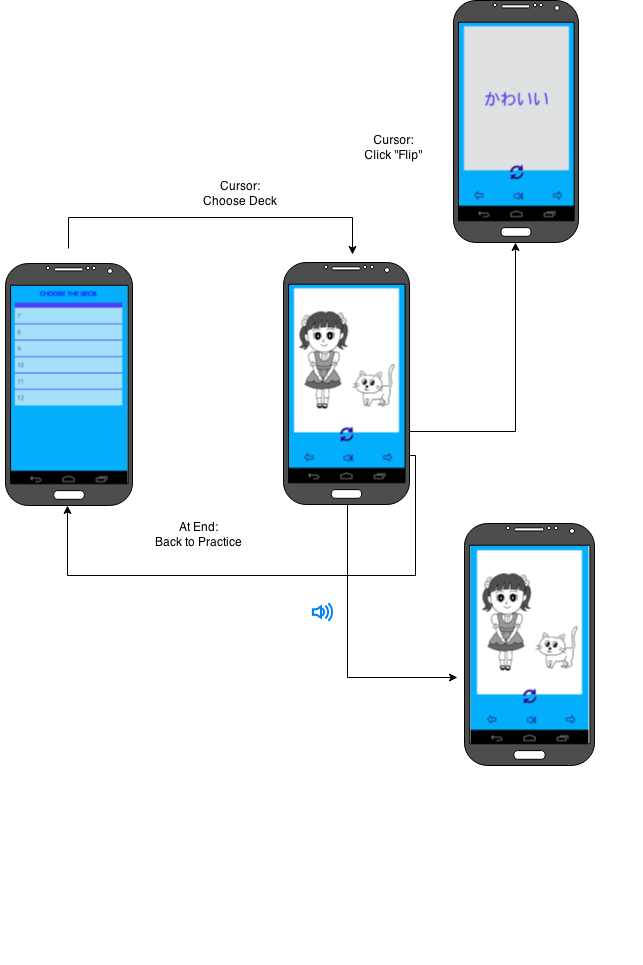
***[](https://drive.draw.io/#G0B3Y_Ai05HOYwUmNfdFFEMkw5SzQ)***

.IV Manage Explanation:

When in the management section of the app the user, if admin, has the ability to modify/delete/insert lessons and decks (students have these abilities for decks only). Since there is a general hierarchy of lesson to question and deck to card, all modifications to the parent (lesson or deck) will be done in the initial screen. Each attribute will be filled to ensure the DB will have valid content which will be required in order for the save functionality to work. If saved then the changes will be written to the DB, otherwise cancel the modification.

If a specific deck or lesson is selected, the connection to the DB will provide a view of the cards or questions [respectively], for it. A user may delete from the list from the list view, or add an item to the list from the list view. An item may be modified if chosen, in which the user will be presented all fields that are present in the DB. When the changes are done, if saved and completed, the changes will be written to the DB, otherwise the modifications will be canceled.

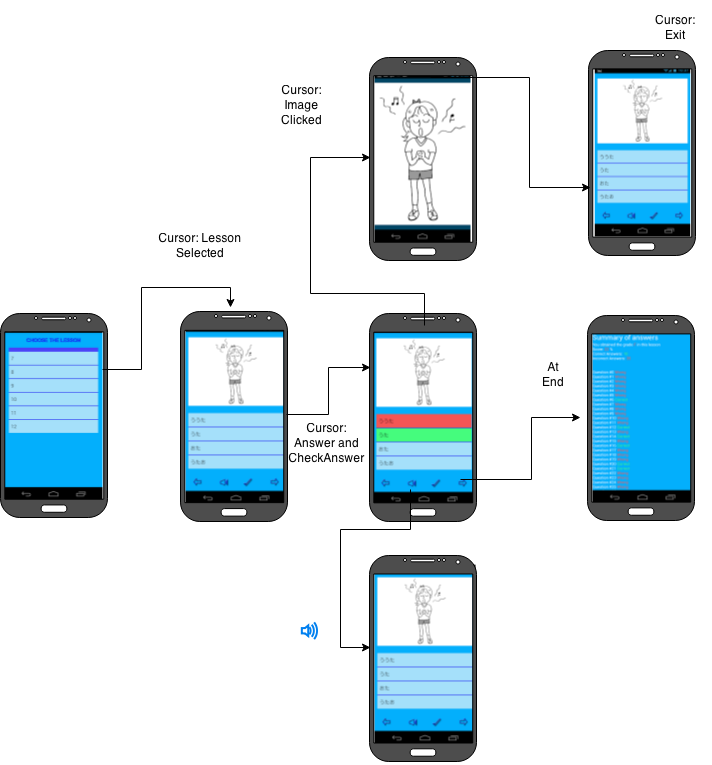
.V Practice Case:

***[](https://drive.draw.io/#G0B3Y_Ai05HOYwa0NSdE1NTWxXOXc)***

.VI Practice Explanation:

When in the practice portal, a user will be presented with a list of decks available to view. Admin-made decks will be available as well as client-made decks that were instantiated in the management portal. The first card will be instantiated given the deck, or the last saved card. Once that view is provided, a user will be shown the image, in which they may flip the card which will show them the primary language and the secondary language interpreted translations. A user has the ability to listen to the audio associated with the image to help assist the learning process. A user will traverse through the list of cards by using the next and previous buttons.

.VII Quiz Case:

***[](https://drive.draw.io/#G0B3Y_Ai05HOYwWTl5U1pZLTRmNTQ)***

.VIII Quiz Explanation:

When in the quiz portal, a user will be presented with a list of lessons available to view. Admin-made lessons will be available. The first question will be instantiated given the lesson chosen, or the last saved question. Once that view is provided, a user will be shown the image as well as the possible answers for the question. A user has the ability to listen to the audio associated with the image to help assist the learning process. Once a user is ready to answer the question, they may pick from the array of answers in which the app will highlight the answer chosen. A user will then be able to check the answer in which the app will highlight their answer red if wrong and highlight the correct answer in green. If the selection was correct then it will be highlighted green. A user will traverse through the list of questions by using the next and previous buttons.

XIII. API

.I Project to Eclipse Environment:https://lh6.googleusercontent.com/fptjtigi0p3NdVyzs3wXF6JjJZrWrI_AgGso_dX0D29PLLUdqNuXJJbmVLRytmvBAVwWMivyVrtGwgFSmvcpKdFWqfQ10kj9hhwY18RSgu7ZeM42RtAZejHZlk5zB8igX25JMQo

|  |
| --- |
| WARNING:  It is not recommended that you should choose to develop in the Eclipse                       environment. At this time, Android has been outspoken in that they stopped                       supporting Eclipse as the official Android development environment. |

1. Download Eclipse and Android SDK
2. Download SVN plugin for Eclipse.
   1. Click [Help] tab, and click on Eclipse Marketplace...
   2. Search for “Subversive - SVN Team Provider” in the marketplace
   3. Install plugin
3. Change Perspective to SVN Repository Exploring

a. In the top right hand corner of the Eclipse window is the Perspective toolbar. It

will currently be set to Java

b. Click the “Add Perspective” button, and then click SVN Repository Exploring

|  |
| --- |
| Note: If you can’t find the Perspective toolbar, you can also complete steps a. and  b. by clicking on the [Window] tab, then Open Perspective, then SVN  Repository Exploring. |

4. Adding New Repository Location

a. In the left-hand side of Eclipse will be the SVN Repositories window

b. Click the New Repository Location button. This will open a new window.

c. In the URL textbox, enter  svn://keeper.nsm.iup.edu

d. Enter credentials and Check the Save Authentication checkbox if you’d like.

e. Click Finish.

5. Checking Out a Project

a. While in the SVN Repository Exploring perspective still, click on

svn://keeper.nsm.iup.edu to view projects listed on the server.

b. Right click the FlashCard folder, and click Check Out.

c. Switch back to the Java Perspective to see the project. You can now work on the

project.

6. Commit (Pushing changes you’ve made locally to the server build)

a. In the Java perspective, right click the FlashCard project, or a specific

.java file you would like to push. Hover over Team, and then click Commit.

b. You can add a comment to the update you are adding. Checkmark whatever

items you want to update. Click OK.

7. Update Local Project

a. In the Java perspective, right click the FlashCard project, or a specific

.java file you would like to update. Hover over Team, and then click Update.

b. This will update your local files to the latest revision that is stored on the server

8. Update Local Project to an Older Revision

a. In the Java perspective, right click the FlashCard project, or a specific .java file you would like to update. Hover over Team, and then click Update to

Revision.

b. Either choose a date, or pick a specific revision to revert back to.

9. Run the Android application

|  |
| --- |
| Note: There may be a dialog box that comes up with an error while trying to find the  Android SDK. Point it to ADT package you downloaded, sub-directory: ‘SDK’ |

10. Highlight project in project view pane

a. Go to [Project] tab → Click ‘Clean...’ → Select Project → [Ok]

11. Run again, should get a built version without errors. Another dialog box will pop-up

telling you that it needs a virtual device to run off of, unless you have a VM set-up.

a. Refer to next steps if this is you.

12. [Window] tab → ‘Android SDK Manager’

a. Make sure that the ‘Show:’ checkboxes ‘Updates/New’ & ‘Installed’ are checked

true

b. Go through and check API packages you wish to run your device off of and other

packages needed to get the VM running.

|  |
| --- |
| Note: You can simply select all packages to ease your mind of installing the correct            packages, however it will make your SDK package quite large. |

* 1. → click [Install packages…]

d. Close out of Eclipse, there will be open dependencies, SDK Manager is a separate

entity so it’s okay to close out.

13. Open Eclipse back up click [Help] tab → ‘Check for Updates’

|  |
| --- |
| Note: If you want to download all packages, then follow steps 12-13 for there will be pre-  requisite and dependency packages. This takes about 3 iterations. |

14. [Window] tab → ‘Android Virtual Device Manager’

a. ‘Device Definitions’ → Select a device you want to use

b. Click ‘Create AVD...’

i.) Eclipse should auto-generate fields to satisfy a complete VM.

c. [Ok]

d. ‘Android Virtual Devices’ → you should now see your device, highlight and click

“Start” and eclipse should put everything together to where when you hit play.

e. Your Virtual Android Device should pop-up in a separate window.

i.) Once it has loaded, you can press play in Eclipse to download the

application to your device.

.II Project to Android Studio Environment:

1. Get Android Studio Install Wizard

a. Use recommended download

b. SDK comes with package

1. Run Android Studio

a. Close current project if there is a project opened

b. You should now be at the “Welcome” screen

1. Select [Configure] → Settings
2. Locate “Version Control” section

a. Click [Subversion]

i.) Un-check “Use command line client”

ii.) Click [Apply] → [Ok]

1. Go back the “Quick Start” Menu
2. Select [Check out project from Version Control] → Subversion

a. Add repository (Click Green plus sign)

i.) Enter in “Repository URL” field: svn://keeper.nsm.iup.edu

b. Choose ‘FlashCard’

i.) Click [Export] once highlighted

1. Choose your local WorkSpace directory

|  |
| --- |
| Note: There maybe dialog boxes that comes up with an error while trying to find the  Android SDK path. This is because the last person to submit changes takes record of their local path. Follow the dialog boxes Android Studio gives you, and it should point to your local SDK path. |

8. To enable SVN

a. Open the FlashCard application if not done so already

i.) Click [VCS] in top, horizontal menu bar

1. → Enable Version Control Integration

|  |
| --- |
| Note: Step 8 must be completed in order to do step 9 and 10 |

9. To push project to SVN

a. Click [VCS] → Commit Changes

i.) You can add a comments to the update you are adding. Checkmark

whatever items you want to update →  Click [OK]

10. To receive update from the SVN

a. Click [VCS] → Update Project

i.) It is not necessary to check any options → [OK]

ii.) Accept their files if prompted, to ensure validity

|  |
| --- |
| Note: There maybe dialog boxes that comes up with an error while trying to find the  Android SDK path. This is because the last person to submit changes takes record of their local path. Follow the dialog boxes Android Studio gives you, and it should point to your local SDK path. |

11. Run Android SDK Manager (can be located on top, horizontal menu bar as image icon)

a. Your SDK path may be found here at the top of dialog box

b. Make sure that the ‘Show:’ checkboxes ‘Updates/New’ & ‘Installed’ are checked

true

c. Go through and check API packages you wish to run your device off of and other

packages needed to get the VM running.

|  |
| --- |
| Note: You can simply select all packages to ease your mind of installing the correct             packages, however it will make your SDK package quite large. |

* 1. → click [Install ### packages…]

e. Close out of Android Studio, there will be open dependencies, SDK Manager is a

separate entity so it’s okay to close out.

12. Open Android Studio after download

a. Click [Help] → Check for Updates

13. Run AVD Manager (can be located next to SDK Manager Icon)

a. This is where all virtual devices are stored.

b. Delete any default devices built with install

i.) Locate device row and ‘Actions’ column

1. Click down arrow → Delete

c. Click [Create Virtual Device..]

i.) Select a device skeleton → [Next]

1. Choose API → [Next]

a. Customize to your choosing → [Next]

|  |
| --- |
| Note: We found that the most robust API to choose was pulling from the ARM\* OS.            This should be chosen for your initial device as to test all steps were done            correctly. HAXM x86 delivered APIs are the fastest, select those once confirmed |

.III External Android Device Testing:

|  |
| --- |
| Note: This is the most efficient way to test Android applications. VMs are somewhat slow            and by the time they load, are really just to make sure your app loaded and that’s it.            External device testing should be done if you have an Android device and USB cable. |

|  |
| --- |
| WARNING: These instructions were tested for Android Studio only! Although may serve as a                      guide for when in eclipse, the only difference would most likely be where to                      access the ADB terminal from. |

1. On your device

a. Connect to computer via USB

i.) Change to Camera (PTP) or Media (MTP)

|  |
| --- |
| Note: Whether your device is noticed as a PTP or MTP will be determined later on, do             not worry which one until troubleshooting. |

* 1. Navigate to Settings → Developer Options

i.) Ensure that USB debugging is checked true

|  |
| --- |
| Note: If Developer Options are hidden (not seen as a settings option), Navigate to            ‘About Phone’ and press 7 times or until dialog box comes back telling you that            “You are now an Android Developer” |

2. On computer resource (Windows 7)

a. Ensure that driver for the device has successfully been installed.

i.) Right-Click Computer → [Manage]

ii.) Select [Device Manager] in the left pane

1. Expand ‘Other device’ → Right-Click your external device

1. Browse computer for driver software → [Next]
2. Find Google USB Driver in your SDK that should have been downloaded in previous set of in *Project to Android Studio*
3. Click [Next]

|  |
| --- |
| Note: If your device is not supported by the Google USB Driver, you may find your            device support at the following site:  <http://developer.android.com/tools/extras/oem-usb.html#InstallingDriver> |

* 1. When done successfully, device will show up with its respective name in the Device Manager

c. Restart Computer

3. On Android Studio

a. Ensure that device is plugged in and USB Debugging is checked true

b. Select [Tools] → Android → Android Device Monitor

i.) You should see that your device is online

ii.) → [Close]

c. Select [Run] → Edit Configurations

i.) Navigate to “Target Device” section

1. Select your device → [Apply]

a. [OK]

ii.) Press [Play]

d. External device testing is now set-up, cable is required for the connection unless

you do the following.

i.) Ensure that device and computer are connected to same wifi hotspot

ii.) Go to a terminal (There is one on Android Studio)

1. Navigate to the \*sdk/platform-tools then type the

following commands

a. adb usb

b. adb devices (should see your device)

c. adb tcpip 5555

d. adb connect #.#.#.# <ip address>

e. adb devices (should see your device twice)

1. Disconnect USB

iii.) Click [Play], choose your online wireless device

.IV Load Local SQLite DB to SQLiteBrowser

1. Download SQLiteBrowser install Wizard

a. Go through selecting Recommended Install

1. In Android Studio navigate to Assets/ Folder

b. Double-click the DB

i.) Browse for your SQLiteBrowser executable

|  |
| --- |
| Note: If Step 2 is not working with your computer. Run SQLite executable and select            ‘Open Database’. Then navigate to your Android Studio WorkSpace and find the            DB in the Assets folder of the Android Studio Project. |

3. Be sure to click [Write Changes] if you do any modifications that you would like to take

place

.V Find Apache2 WebApp Files

|  |
| --- |
| WARNING: These instructions can only be applied if savage.nsm.iup.edu is still running. If                      not, please see tutorials in how to set-up a LAMP and/or WAMP server and push                      the server script files to the deployment directory |

1. Download WinSCP Wizard Installer

a. Run the Executable Supply the following fields:

i.) File protocol: SFTP

ii.) Host name: savage.nsm.iup.edu

iii.) Port number: 777

iv.) User name: pjsxdva

v.) Password: changemenow1

b. Click [Login]

c. Navigate to /var/www/html/FlashCard

.VI Locate MySQL DB

|  |
| --- |
| WARNING: These instructions can only be applied if savage.nsm.iup.edu is still running. If                      not, please see tutorials in how to set-up a MySQL DB in which allows full CRUD                      operations from all IP addresses |

1. For console style UI

a. Download putty client and supply the following fields

i.) Connection Type: SSH

ii.) Host Name: savage.nsm.iup.edu

iii.) Port: 777

b. Click [Login]

i.) Username: pjsxdva

ii.) Password: changemenow1

c. Run command mysql -u root -p

i.) Password: root

ii.) Type: use FlashCard

d. You may now enter SQL commands to the DB

1. For GUI DB management

a. Download MySQL Workbench with recommended settings

i.) Run Executable

1. Click [Database] → Connect to Database

a. Connection Method: Standard (TCP/IP)

b. Hostname: savage.nsm.iup.edu

c. Port: 3306

d. Username: root

e. Password: root

2. Locate DB Schemas on left pane

a. Navigate to FlashCard

XIV. Glossary

**APK –** Android Application Package; Application packaging system used to distribute and install Android applications.

**Class Diagram** – An illustration of the relationships and source code dependencies among classes drawn/written in the UML.

**CSV –** Comma Separated Values; Commonly used amongst professionals of all departments providing a clean and easy tabulated format.

**DB (Database)** – A storing of data for a system, accessible in various ways.

**DBMS (Database Management System)** – Controls the security and integrity of a database, as well as the controlling of organization, storage, and retrieval of data in the database.

**DFD (Data Flow Diagram)** – Often used as a means of overviewing the system graphically and preliminary. Provides a graphical representation of the flow of data in a system, modeling its process and aspects.

**E-R (Entity-Relationship) Diagram** – A graphical representation of objects or concepts within a system or organization, and their relationship to one another.

**GUI (Graphical User Interface)** – An interface that allows the user a graphical representation of the system with an electronic device.

**Hiragana -** the more cursive and more widely used form of kana (syllabic writing) used in Japanese, especially used for function words and inflections.

**IO (Input/Output)** – Declares any operation, program, or device that transfers data to or from a system.

**Kanji -** a system of Japanese writing using Chinese characters

**Login system -** an element in the app that allows users to login with an ID and password to regulate who can access the app.

**Relational Schema** – A logical definition of a table; the table name, and the names of each column.

**Sequence Diagram** – A representation of object interactions within a given time, showing how processes operate with one another and what is in their order.

**UI (User Interface)** –A view in which the end-user sees.

**UML (Unified Modeling Language**) – A general purpose modeling language in the field of software engineering, providing a standardized way of visualizing a system.