Table of Contents

Project Description

The current tools that help people learn vocabulary suffers from the problem that learners lose interests and motivation as time elapses. Hence we produce a solution to this problem by adding in excitement of competition into the learning process. We propose an online game that employs the idea of friendly competition. A set number of questions will be tossed out and the two players would race to answer them; their score would be based on their answers' correctness and speed.

Process Practiced

We followed XP Programming by doing development in iterations and doing pair programming. Because we were not ready to build the final complete sketch at the very beginning of semester, avoiding test-driven development was our decision.

Our team is divided into 3 pairs: Boyang & Chi, Shiqi & Yifan, Lingzi & Qingsong. We didn't rotate pairs since we didn't find the need to do so. The group development went on well, and we tried to divide the job evenly within the group.

The programming process we followed gave us great benefit in the whole developing process, as we realized all the designed and necessary use cases on time. One thing we could improve is our refactoring strategy. In each iteration we filled ourselves with assignments to new features, but we didn't put refactoring our previous designs on our time allocations.

Architecture

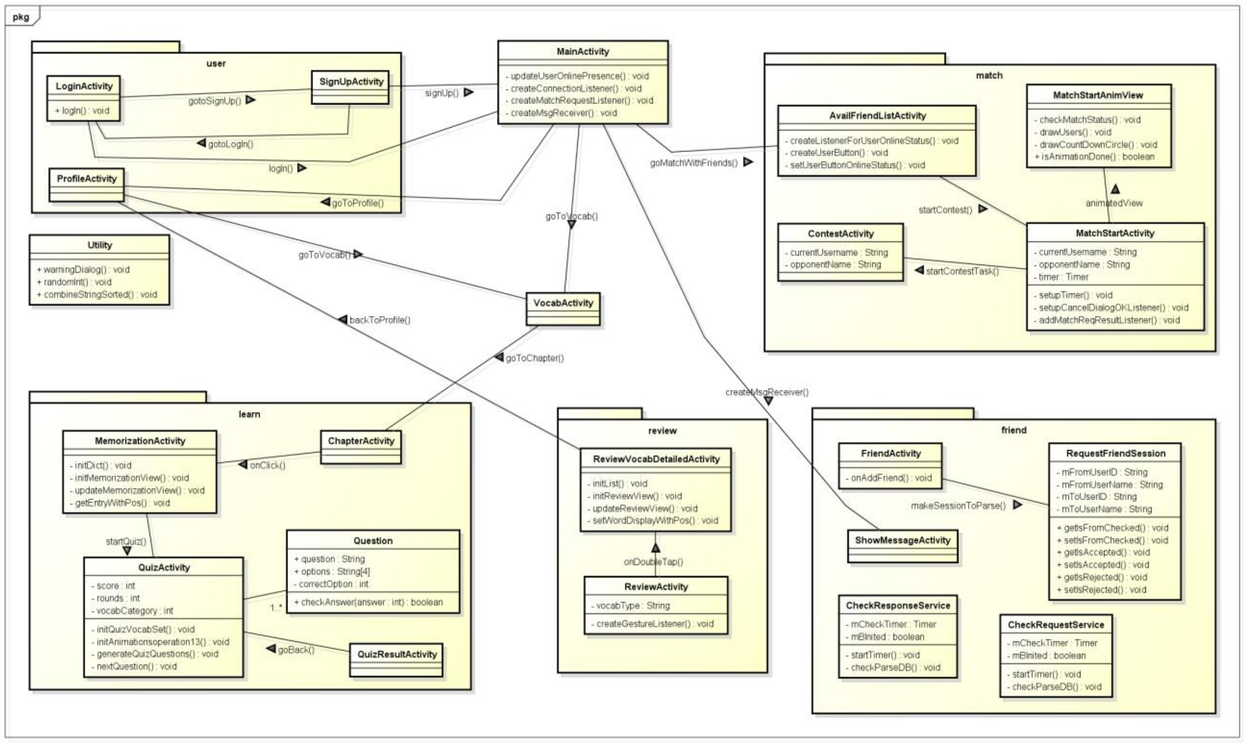
Overview

The Android app is built using Parse and Firebase libraries as the backend. Parse acts as a database and Firebase functions as a real-time storage. We stored user information (name, email address, learning experience) and vocabulary data on the Parse cloud server. The competition data, and meta-data indicating user's online and matching status are stored in Firebase.

The project source is broken down into 7 packages:

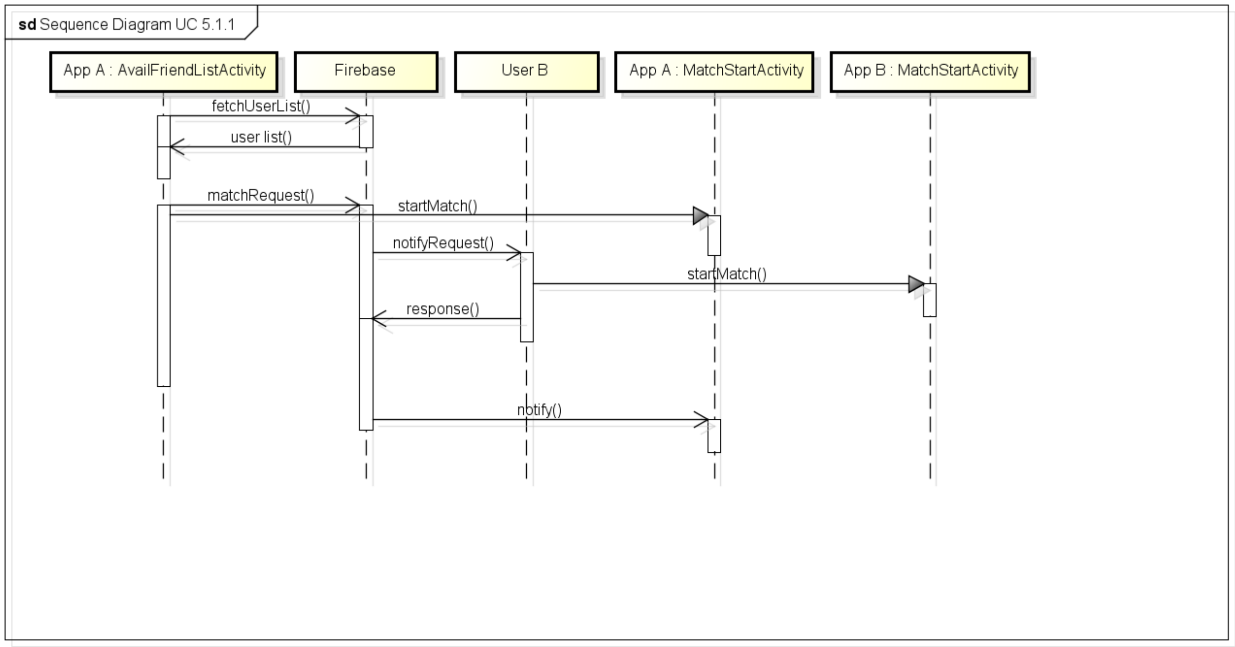
- `definitionAPI` package contains the sources used to fetch definition for vocabularies.
- `user` package contains login/signup implementations that fundamentally make up our user system.
- `friend` package contains logics for friend requests and responses.
- `learn` package contains sources dealing with our app's learning system (i.e. view vocabulary by chapters and the quiz feature for each chapter).
- `review` package implements the logics for user to review difficult vocabs.
- `match` package contains logics dealing with matching, including match request/response and real-time match competition.
- `postquestion` package implements the feature that allows users to contribute problems and admins to review/approve them.

Also we have some general classes under our main package that is useful to all classes in our project. For example we have a `Global` class containing useful constants to enable reuses across classes. Also we have `FirebaseSingleton` class which employs the Singleton design pattern to ensure we have an efficient use of backend Firebase references. Besides these, the `MainActivity` for our project which contains navigations to all other project features is also placed under this package. Below is the UML class diagram for important classes in our project.

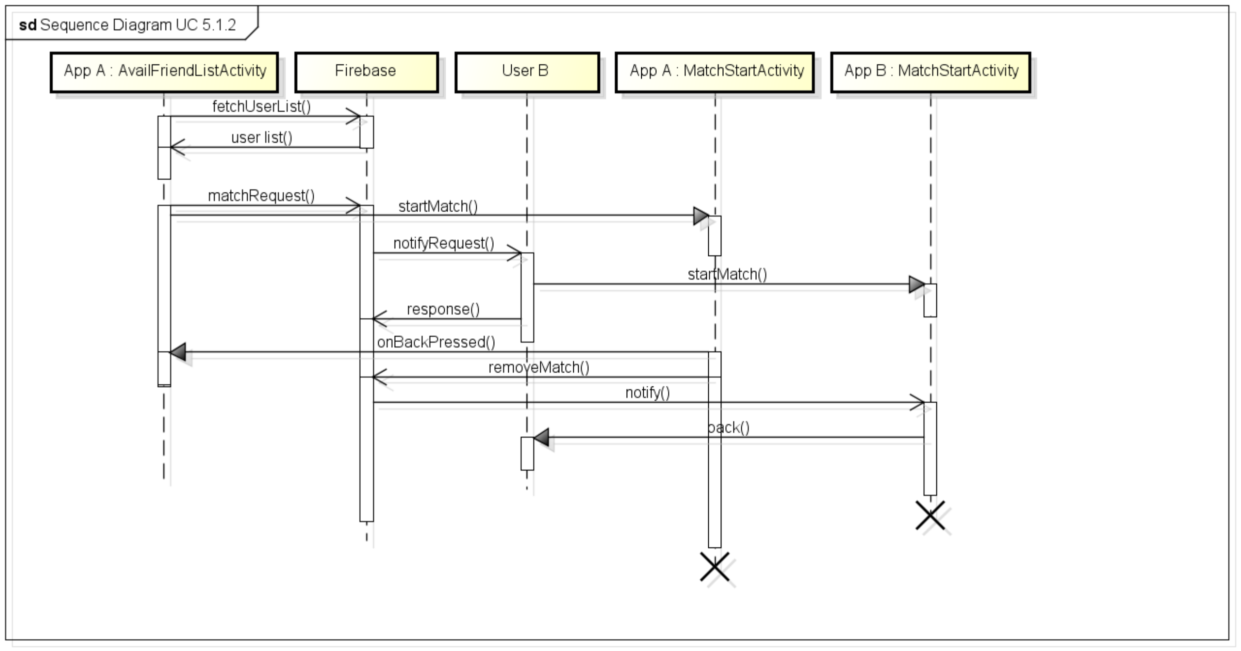


Sample Sequence Diagrams

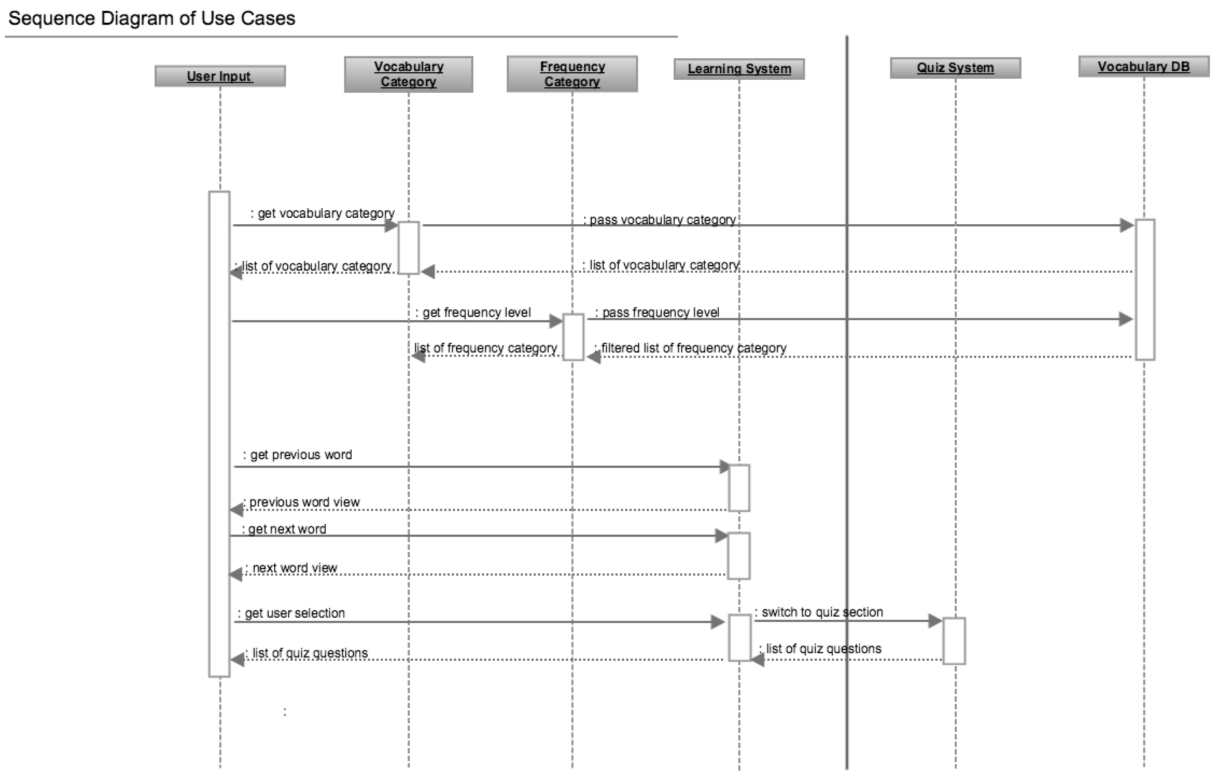
Sequence Diagram for Initiating a Match Request - UC 5 (Success)


sd Sequence Diagram UC 5.1.1
App A : AvailFriendListActivity
Firebase
User B
App A : MatchStartActivity
App B : MatchStartActivity
fetchUserList()
user list()
matchRequest()
startMatch()
notifyRequest()
startMatch()
response()
notify()

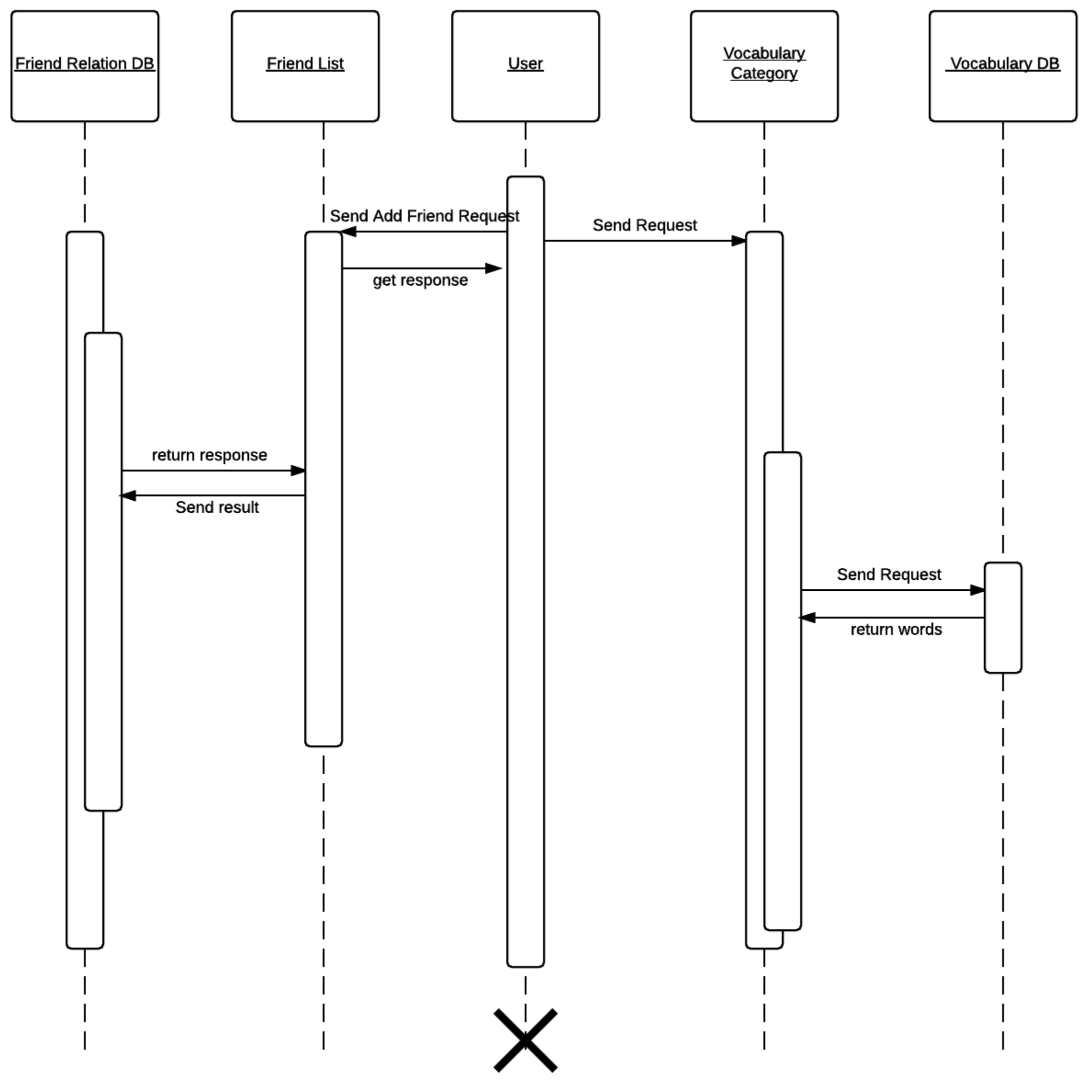
Sequence Diagram for Initiating a Match Request - UC 5 (Fail)



Sequence Diagram of the Learning System - UC 7, 8, 16



Sequence Diagram for Review System and Friend System - UC 9 & 10



Requirements & Specifications

Use Case 1 - Login System

Specification

This use case provides the users a system to signup an account and login with passwords correctly encrypted.

Design

- **LoginActivity.java**
  This `Activity` class handles the login logics and interaction with the *login.xml* interface.
- **SignupActivity.java**
  This `Activity` class handles the signup logics and interaction with the *signup.xml* interface.

Use Case 3 - Facebook Friend Connection

Specification

This use case focuses on send invitations to Facebook friends. When user logged in next time, they will get a notification.

Design

- **FacebookLoginActivity.java**
  This class is responsible for the whole Facebook events, including tracking tokens, access profile, access friends, and display results
- **FriendsInvitesFragment.java**
  This class provides a dialog for logged in Facebook user to send Facebook friends' invitation requests to this game.

Use Case 5 - Online Matching with Friend

Specification

This use case focuses on the competition matching with available friend online.

Design

- **AvailFriendListActivity.java**
  This `Activity` class handles the friend list display and interaction with the *availablefriendlist.xml* interface.
- **MatchStartActivity.java**
  This `Activity` class handles the logistics of the start matching event and interacts with *matchstart.xml* interface.
- **MatchStartAnimView.java**
  This `animation` class handles the animation of the start matching event, giving reasonable amount of time to handle the matching delay.
- **SelectMatchingTopicAcitivity.java**
  This `Activity` class handles the animation of the start matching event, giving reasonable amount of time to handle the matching delay.

Use Case 7 - Vocabulary DB design

Specification

This user case designs of backend database that stores all of the vocabularies as well as user information.

Design

- **Parse Database: User, GRE, GMAT, SAT, TOEFL**
  The above tables are the backend Parse tables that stores all of the user information they filled in at register and all the vocabulary sets of different difficulty level.

Use Case 8 - Learning System

Specification

This use case implements the learning vocabulary functionality and allows the user to choose from specific vocabulary category, frequency category and then word chapter list to start learning.

Design

- vocabpage.xml
  This `XML` file implements the layout for vocabulary category list (GRE, TOEFL, etc.).
- chapterlist.xml
  This `XML` file implements the layout for frequency category list (high, medium or low).
- freqchaplist.xml
  This `XML` file implements the layout for partitioned word lists in numerical order.
- memorizationpage.xml
  This `XML` file implements the layout for the learning system page including the word content, word meaning, a speaker icon for pronunciation, previous and next buttons, and a button to go to quiz.
- **VocabActivity.java**
  This `Activity` class handles the vocabulary category list display and interaction with the *vocabpage.xml* interface.
- **ChapterActivity.java**
  This `Activity` class handles the frequency list display and interaction with the *chapterlist.xml* interface.
- **FreqChapActivity.java**
  This `Activity` class handles the partitioned word list display and interaction with the *freqchaplist.xml* interface.
- **MemorizationActivity.java**
  This `Activity` class handles the learning system display and interaction with the *memorizationpage.xml* interface.

Use Case 9 - Review Vocabulary

Specification

This use case implements the review vocabulary functionality and allows the user to have a look at the words he/she has made mistakes on in the past quizzes, and be able to enter the review system to self-check the review words.

Design

- review.xml
  This `XML` file implements the layout for review words list.
- reviewchart.xml

This XML file implements the layout for review system page including the word content, word meaning hidden beneath a circle, progress bar, "I know" button, "I don't know" button, and "finish" button.

- **ReviewActivity.java**
  This Activity class handles the review words list display and interaction with the *review.xml* interface.
- **ReviewVocabDetailedActivity.java**
  This Activity class handles the review system display and interaction with the *reviewcahrt.xml* interface.

Use Case 10 - Add Friends

Specification

This use case focuses on users could add game friends by search their name on the list.

Design

- **CheckRequestService.java**
  This Service class lets the app to check the friend request and return to the users.
- **CheckResponseService.java**
  This Service class lets the app to return the friend request result to the users.
- **FriendActivity.java**
  This Activity class lets the user to choose the user and send add-friend request with the *friend.xml, row_friend.xml* interface.
- **ShowMsgActivity.java**
  This Activity class lets the users to choose accept or reject this friend-add request with the *showmsg.xml* interface.
- **RequestFriendSession.java**
  An extension of ParseObject that makes it more convenient to access information about a given Meal.

Use Case 11 - Words arranged by frequency

Specification

This user case implements the functionality that is able to calculate the frequency of a given word has shown up in a kind of exam.

Design

- **FindFrequentListFromPDF.java**
  This Java class extracts word strings from PDF files which are past exams. Then it calculates the frequency of all word stored in the database based on these past exams and store then back into the database.
- **Word.java**

This `Java` class defines a word. It contains the string of a word, the definition of this word and it's frequency of showing up on previous exams.

- **FrequencyTest.java**
  This `Java` class tests the functions implemented in FindFrequentListFromPDF.java

Use Case 12 - User-Created Quiz Questions

Specification

This use case focuses on users could post DIY quiz questions depending on the vocab list and Admin could check these quiz questions. Another specific function is User-created Quiz test.

Design

- **AddPostActivity.java**
  This `Activity` class let the user to choose vocab list with the *addpostvocablist.xml* interface.
- **CheckPostActivity.java**
  This `Activity` class let the Admin to check the user-created questions with *checkquestion.xml* interface.
- **PostQuestionActivity.java**
  This `Activity` class let the user to submit crated questions with the *postquestion.xml* interface.
- **ShowUserQuizActivity.java**
  This `Activity` class let the users to choose the vocab list to do the quiz with the *usercreateduizlist.xml* interface.
- **UserQuizActivity.java**
  This `Activity` class let the users to do the quiz with the *userquizmain.xml* interface.

Use Case 14 - Well Designed Interfaces

Specification

This use case implements all of the major layouts that the app needs to achieve its functionalities

Design

- **login.xml**
  This `XML` file implements the layout for a user to login into the app system using his/her username-password combination.
- **signup.xml**
  This `XML` file implements the layout for a potential user to fill in his/her information to sign up as a user.
- **chapterlist.xml**
  This `XML` file implements the layout for a logged in user to choose a chapter of selected vocabulary type to start learning process.

- **vocabpage.xml**
  This XML file implements the layout which is the main learning page. It contains the content and definition of a given word. It also contains buttons for a user to switch to different words and enter a quiz.
- **mainpage.xml**
  This XML file implements the main layout that will show up right after a user has successfully logged in. There will be several buttons for the user to select different activities.

Use Case 16 - Quiz System

Specification

This use case provides the users a system to test their proficiency in the vocabulary lists he/she has learned.

Design

- **QuizActivity.java**
  This `Activity` class handles the quiz problem generation logics and interaction with the *quizmain.xml* interface.
- **QuizResultActivity.java**
  This `Activity` class takes charge of quiz result generation and interaction with the *quizresult.xml* interface.

Use Case 17 - Word pronunciation and online dictionary

Specification

_______This use case implements word pronunciation, provides word definition API for others to use, and has the functionality of searching definition for any word.

Design

- **AllData.java**
  This class is responsible for converting from `JSON` to Java object
- **DataObject.java**
  This class is a data type for vocabulary object from Parse database, which contains word creation time, definition, objectID, update time, word itself, and frequency.
- **DefinitionAPI.java**
  This class contains an implementation that uses dictionary API to fetch definitions.
- **DefinitionEntities.java**
  This class is a class contains an `ArrayList` of `DefinitionEntity`
- **DefinitionEntity.java**
  This class contains word definition, and its contributor
- **DefinitionHTTPS.java**
  This class is responsible to get definition results from API by using `HTTP` client

- **WordListDefinitionFulfiller.java**
  This class is responsible to generate a new `JSON` file which contains original information as well as the definition got from API.
- **HttpAsyncTask.java**
  This class is responsible to get definition results from API and display the results from `HTTP` client
- **OnlineDictionaryActivity.java**
  This class is responsible to search any word for their definition by API

Reflections

Chi Zhou (chizhou3)

Reflection

This is my first-time learning Android and also my first time collaborating in a team of 6 people. Studying Android helped to clear up my understanding of the mobile app infrastructure. Overall, I think I liked the process and communications our team practiced in the development of this project.

Learning & Improvements

Probably the first thing I learned the hard way is that it's hard to divide chunks of big tasks into smaller ones, especially when the components have dependencies with each other. This is one of the things we could very well improve upon.

Another thing that we could improve is our testing strategy. Since we used libraries for our backend, we utilized provided asynchronous database calls for communication, but that has given us a very difficult time in testing. We learned in the end that Espresso has built-in support for asynchronous fetches done using `AsyncTask`, but we don't have time to change all our async behaviors using `AsyncTask`, so this is also something that we could improve upon.

Besides these, this project has been a challenging learning experience. We had to work with some random undesirable behaviors Espresso has (such as having to add "\n" to each input in a textfield else it has a risk of breaking on inputs). Also we needed to consciously write codes that work for both sides during a real-time match.

Future Plans

If we have the chance, maybe we would like to tweak our UI to be more elastic on different Android layouts. Also we could possibly port our app to IOS, which would be another great learning experience to study the differences between the two major mobile OS.

Boyang Chen (bchen11)

Reflection

This was the first time I worked on android development. I paired with Chi on several major parts: SignUp and Login System, Quiz System, available Friend Matching and Learning System Layout. I really get the chance to know what Android platform is and how the whole system operates. I still got a lot to catch up, but this should be a very good start.

Learning & Improvements

I learnt the Design pattern in 428 class and applied some of the concepts into our project development. Two popular Backend Server, Parse and Firebase, are being used during the process and I felt very happy to know such convenient method. And I touched the overall Android development process throughout the semester. I felt that this great chance means a lot to me.

Future Plans

Expanding on the matching functionality; set up an IOS App using the same backend in the future time maybe.

Yifan Wang (ywang123)

Reflection

In this project, I learned how to design, implement and refactor project. We follow the extreme programming methodology. At the same time, I coordinate with my pairs to finish up the features of word frequency generation; Login system interface design; Word definition API; Online dictionary; Facebook Friends Invitation.

Learning & Improvements

This is the first time to write an Android application. I learned things on https request and response when doing online dictionary, and I also learned how to communicate between Facebook API, Parse database, and Android application.

Future Plans

Refactor test cases, probably using Mockito in the future
Keep working on Facebook and Parse connections

Shiqi Zhou (szhou14)

Reflection

During the winter break, I took Coursera android development course taught by Professor Angrave and I am so motivated about doing Android. As a result, I decided that I should join an android team and do an android project. During the whole semester I have been cooperating with my teammate and learned a lot from everybody. The tasks I accomplished for this project including front-end interface and backend database design, word frequency, word chapter division as well as vocabulary pronunciation.

Learning & Improvements

Parse Database; Android Espresso Testing; Android Development; UI Design; Extreme Programming.

Future Plans

Keep working on Android Development and get familiar with more useful APIs.

Qingsong Qi (qqi3)

Reflection

I've never worked on Android development before but I am really interested in exploring this field and that is why I chose Android as my CS428 project. During this semester, I became rather familiar with Android IDE and am able to adopt my java programming knowledge to accomplish coding tasks. The part I worked on including quiz design, friend matching function design and several smaller functionalities.

Learning & Improvements

Parse Database; Android Espresso Testing; Android Development; UI Design; Extreme Programming.

Future Plans

Keep learning on Android Development.

Lingzi Liu (lliu15)

Reflection

I've always wanted to learn mobile programming and this experience is definitely a good start. In this Android App project I paired with Qingsong and we implemented the following four features: learning system, review system, user-customized quiz questions and friends system. We experienced a hard time in learning how every component is structured and connected and how the IDE, Android Studio, works at first, but the work becomes much easier and more fun as we get more familiar with all the strong tools we've got.

Learning & Improvements

I learned the Android App Development as a whole, and more specifically, UI design, Activity implementation, and UI and Unit Testing. I also learned the Agile development process including iteration plannings and code refactoring, and most importantly team work.

Future Plans

Bring better UI design to our App, extend the unfinished features and keep learning more about Android App development.