Math Fun Facts

An Application for the Android Platform

**Developed For:**

Professor Francis E. Su

Benediktsson-Karwa Professor of Mathematics

Department of Mathematics

Harvey Mudd College

**Developed By:**

Anders Chaplin Aidan Cheng

Claremont McKenna College Claremont McKenna College

Ali El Abridi Anant Jaitha

Al Akhawayn University Claremont McKenna College

Introduction

Professor Su, our client, is the President of the Mathematical Association of America. As a math enthusiast, he enjoys reading thought provoking math fun facts. Some time ago, Professor Su had the idea to launch an app for sharing fun facts and brain teasers with fellow math enthusiasts. Prof. Su launched his own website and an iPhone app to match, loaded chock-full with a collection of math fun facts which he had accumulated over his years of mathematical endeavors.

With products spanning two of the three most commonly used technological device platforms today, Prof. Su wished to conquer the final frontier, bringing the joy of mathematical fun facts to the Android platform. HMC’s CS 121 Software Development course for Spring of 2016 facilitated the needs of Professor Su, bestowing upon our team this exciting project for the duration of the semester.

Core Features/Functionality

In essence, the app has the following features:

1. Users can see random fun facts and generate new random ones when they’re done with the one they’re reading right now.
2. Users can rate fun facts based on how much they enjoyed it.
3. Users can see the facts they have rated, sorted by their ratings.
4. Users can select a difficulty for a fun fact and see a list of fun facts based on this filter.
5. Users can select a subject the for a fun fact and see a list of fun facts based on this filter.
6. Users can enter a search term to see fun facts that have the search term either in the title of the fun fact or in its keywords. A list of all such fun facts is generated.
7. Users can select a fun fact from the lists generated as talked about in features 4-6. The files are displayed using their titles and color coordinated based on their difficulty.

Architecture

**File Storage**

Untitled Diagram (1) (1).png

All of the facts available to the user are stored locally, as simple HTML files. Upon opening the app, the main activity calls MathFunFactCollection, which in turn iterates over the HTML files and stores the data and metadata of those files into ParserMathFunFact objects. We are interested in displaying metadata such as the title of the fact, the subject, the difficulty,and the keywords of the fact. Our Collection object is a static object, because we only ever need to parse the fun facts once, and all further references to the Collection are to the same object. Our Collection object provides much functionality to the rest of the app as the Collection has methods used to filter out facts according to the user's preferences. With specific method calls, the Collection returns a filtered ArrayList of ParserMathFunFact objects which can then be manipulated further. The only time the Collection doesn't return an ArrayList is when we want to retrieve a randomly selected fun fact - the first tab that the user sees upon opening the app. We also created ParserMathFunFact to be very flexible objects, we could create one at any time just by passing in the filename that had to be parsed. This solved a huge issue which was figuring out how to pass data between fragments. The main way Android does this is by putting objects into a bundle, and then passing that bundle along to the next fragment. However, the bundle could only store primitive types and arrays of those types. The way we worked around this was by only passing in the filenames or arrays of filenames of the facts that the user was filtering by. So when we actually needed to display a fact's content, we would simply create an entirely new ParserMathFunFact object with the filename, and then retrieve the content when the fact gets parsed. An aside should be made quickly towards how our app handles the persistent data from rating facts. We found that it was best to store ratings inside JSON files, as having to overwrite the fun fact text files would have been messy at best, and not very robust. Further, we did not want to change the way our client generates his files so we wanted to leave his files untouched. We instead relied upon Android's built in JSON api to manipulate ratings, as each app has its own space allotted for JSON data that no other process can touch and can only be accessed with Android's built in-method.

**Fragments**

Fragment.png

It's worth discussing fragments within its own section mainly because our app relies on them so heavily. At it's core, Math Fun Facts only ever invokes one activity, the MainActivity. This is because in order to be in line with our tabbed user interface, we cannot call entire activities without obscuring the tab bar UI. Thus, we had to instead call fragments, which are essentially mini activities that can be configured such that they do not obscure the tabs. We call these fragments using a CustomAdapter that we had made ourselves. The Adapter is called by the MainActivity, and it calls different tab fragments when the user clicks the tab. We also used fragments so that the UI plays nicely with the back button such that when you click the back button, you do not exit the app entirely, but go back to a previous fragment that you were interacting with before. For example, if the user is in a tab of a fun fact that he/she selected from a list of Combinatorics facts, clicking the back button will take him/her back to the list of Combinatorics fun facts. To this end, we utilized a root fragment to serve as a container for the subsequent fragments that could be called. This is because when we call child fragments on top of parent fragments, the child fragments must look for another fragment to replace. We could not replace the parent because the fragment would then disappear, and the user wouldn't be able to navigate back to it. Essentially, we created a dummy fragment where we didn't care if it was replaced or not.

When the user clicks a tab, the root fragment automatically populates itself with the appropriate tab layout, so that the user never has to interact with the root. But because we had multiple tabs, we had to create a new root fragment for every tab. While we expected there to be some sort of overhead, performance was minimally affected. Though we had to create individual root fragments, we found that we didn't need to create separate fragments for the list of filtered facts or for displaying the singular fact. The list of filtered facts, or the ListMFF fragment, sorts facts by the parameter that was passed in, whether it be difficulty, subject, rating, or search keyword, and displays them by the title. For this, we used a ListView that is populated by the ArrayList returned by MathFunFactCollection, depending on which method we called.

The user can click an item in the Listview, and it would display only one fact. For this, we use the DisplayOneMFF fragment, and from this fragment the user can rate the fact. The rating goes up by half-increments, and sends a signal to the JSON writer to set the rating of that fact to the user's preference. So when the user navigates back to the favorites tab, he or she can see the list of facts displayed in descending order according to the rating.

**User Interface**

For this app, we decided upon a tabbed layout, so that the user can swipe or tap seamlessly across different filtering preferences. Following standard design heuristic principles, we wanted the user to 1. Always know where to locate and how to use the navigational functions and 2. Always have a clear and easy method of “escaping” or “exiting.” But instead of placing the tab bar at the bottom of the screen, seen in the iOS app, we changed the placement in accordance with the best practices of the Android platform. Android’s design guidelines suggest that the tab bar be placed at the top of the screen, below the action bar. We also placed a floating action button at the bottom right of the first tab, so as not to obscure the navigational bar, and to provide easy access via thumb- or finger-tap. This floating action button yields random fact retrieval/generation, but does so in a visually pleasing way. The alternative would have been to insert a fixed-position button, intruding on the webview and distracting from the content of the app.

Since our tab bar implementation was based off of Android Material Design concepts and libraries, it was only fitting to select a color scheme which also was derivative in nature of Material Design. By choosing a bright, radiant shade of green, we were able to set stage for nice contrast in the accenting deep blue. The contrast of the two colors was enough to be distinguishable, but not so much as to distract the eye from the content at hand.

Keeping the color scheme confined to the cool end of the spectrum also protected identity from the ListMFF fragment’s contents, which include the display of a filtered list of facts. ListMFF colors each fact according to the difficulty - yellow for easy, orange for medium, and red for hard - keeping in line with the original iOS app’s difficulty-level color scheme. The yellow-orange-red model is a nice progression intuitively connected to the increasing difficulty.

Team Performance

Originally, we had hoped to get an MVP going with a few of the features we have right now. We wanted to show random fun facts on a button click, sort fun facts by difficulty and subject, and create a tabbed layout so the user can swipe among these features.

We were able to accomplish all these features along with including rating of fun facts and listing fun facts by these ratings. Further, we were able to implement a search bar that goes through fun fact names and keywords to identify fun facts that contain the search term.

Our client initially gave us his iOS app to model our app after. He wanted all the features in it and more. However, he told us that while he hopes we can complete it in the given time, it might be too big a task for us to commit to. All he asked us to do was to make something that could be put on the Google Play store.

This vision has stayed constant throughout the semester with us changing priorities numerous times but never changing our end goal. For the most part, we have delivered this product.

Challenges

The Math Fun Fact are stored in text files instead of being aggregated in a database, and ,therefore, not easy to retrieve. We had to build a parser that goes over all the files, extract at runtime all the data stored in those files, and separate the metadata (title, subject,difficulty,description) with the rest of the math fun fact to be displayed. Another challenge was that the rating was not incorporated in those files, and the client didn’t want to use any remote storage to avoid constant cost fee for a server and its maintenance. Moreover, we couldn’t add a field in those files because it would change the usual structure with which the client adds new math fun fact to the app compared to the one from the iPhone version. Hence, we had to separate the rating storage from the math fun fact files. One way of doing it is to use a database for it, but knowing that the only data to be stored was the pair key values (Filename, Rating), it was too costly to change and restructure a lot of classes to incorporate the usage of a database such as SQLite. We agreed on using a JSON file that would be stored locally in the environment of each user where his rating would be stored.

Limitations

We wish we had more freedom as developers to act the way we did. We would have preferred using databases to store the information. The format the data was stored in was not the most efficient for us. It wasn’t the most intuitive in the format of the text and the format of the file names. We would have also hoped to start at a fast pace with some Android experience to begin with.

Future Work

It would be interesting to see the following features implemented in future iterations of the app:

1. A free and a paid version of the app. The free version will display ads and have limited fun facts while the paid version will get rid of apps and will also make all facts available. In this regard, the additional divisions between the free and paid versions can be explored further.
2. You can find ways to differentiate between read and unread fun facts.
3. The listing of fun facts currently covers an entire cell of the fun fact title based on the difficulty. This could be changed to show it in a cleaner manner. Different UI elements can be tried in this regard.
4. You can find good ways to refresh the fun facts. Maybe like a shake, a pull down, etc.
5. Prof. Su has links in a coded format in his fun facts. The codes are supposed to link to a search on fun facts. This feature needs to be implemented.
6. There should be a way to share the app through various messaging platforms.
7. In-app advertisement
8. A walk-through tutorial. For example, the first time the user opens the app they can see a tutorial that walks them through the app.

Appendix

**Phase 1**

Planned User Stories

1. As a user, I want to be able to scroll, so I can read the entire fun fact
2. As a user, I want to see a math fun fact in the app, because I enjoy reading fun facts about math
3. As a user, I want to see the title of the math fun fact, so that I know which fact I'm reading.
4. As a Developer, I want to create a Parser that allows me to view the files as a Webview
5. As a user, I want to press a button so I can read another fun fact
6. As developers, we need to learn about best practices for making views and content scrollable, so that a user can scroll to see a fun fact
7. As a user, I want to press a "Random" button and see a random fun fact, so that I can discover new math fun facts in an engaging and enjoyable way
8. As a developer I would like to parse the files to extract the "meta-data" like 'Title', 'Description', 'Categories', and 'Difficulty' from the rest of the contents.

Completed User Stories

We completed user stories 1 - 2, and 4 - 7.

Incomplete User Stories

User stories 3 and 8 were incomplete but progress was made on them.

**Phase 2**

Planned User Stories

1. As a Developer, I want to create a Parser that allows me to view the files as a Webview
2. As a user, I want to see a dynamic navigation bar at the top of the app, so that I know which view I'm currently in
3. As a user, I want to see a highlighted tab button corresponding to my current view, so that I know which view I'm looking at
4. As a user, I want to see and select facts by difficulty, because I might prefer easy, medium, or hard math fun facts.
5. As a user, I want to select fun facts by subject, so I can pick a fun fact from my favorite types of math
6. As a developer, I would like to loop through all the files to organize the files based on their metadata
7. As developers, we need to learn more about what our client wants us to display for each fun fact

Completed User Stories

We completed user stories 3 and 8 from Phase 1 and completed 1 - 2 and 4 - 6 from this phase.

Incomplete User Stories

User stories 3 and 7 were incomplete from this phase.

**Phase 3**

Planned User Stories

1. As a user, I want to have a tab bar with buttons, so that I can navigate to different views of my app
2. As developers, we need to research and establish a strategy for parsing each file, so that we can separate the text and images from each fun fact
3. As a user, I want to see fun facts colored by subject, so that I can recognize differences in a listview
4. As a user, I want to be able to search for a fun fact, so that I can quickly find the math fun fact that I am thinking of.
5. As a user, I want my favorite fun facts to be persistent, so I don't have to re-rate fun facts I like
6. As a user, I want to see math fun facts that I've rated highly appear in my list of "favorites," so that I can quickly navigate to math fun facts I love
7. As a user, I want to rate my fun facts, so that I can indicate how much I like each math fun fact I read

Completed User Stories

We completed user stories 3 and 7 from Phase 2 and all user stories from this phase.

Incomplete User Stories

None