Caff Graph
Houston Luzader

App Description

Every day in the United States, each person consumes an average of 165mg of caffeine a day, that is upwards of 53000kg of caffeine being consumed each day. That's about 40 cars worth of weight in caffeine being consumed every day.

Caff Graph is a daily tracker for all of your caffeinated needs. Many Americans are so busy with their life that they never even consider that they may be addicted to caffeine. With Caff Graph you can become more cognizant of how much caffeine you are consuming each day. Additionally, you can see your growth as you can set a goal to decrease your daily use of caffeine and be liberated from a caffeine addiction.

The project is hosted at the following GitHub repository: ttps://github.com/HoustonLuz/CaffGraph

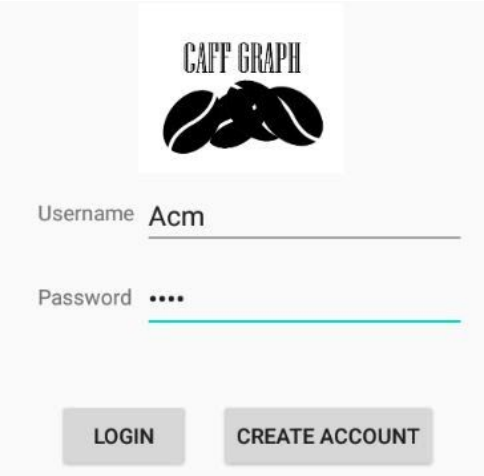
App Features

- Remembers if there is a user already logged in and skips the login screen if so.
- Stores username and password in database handled by UsersCP.java.
- Stores hashed passwords instead of raw text.
- Automatically calculates and adds caffeine in milligrams from selection of choices such as coffee and tea.
- Has the ability to manually enter a custom amount of milligrams instead in case of energy drinks.
- Stores daily caffeine intake in a database handled by IntakeCP.java.
- Remembers amount of caffeine in Current Caffeine Overview if app is closed.
- Shows all daily intakes.

UI Design

Caff graph contains three main activity UIs.

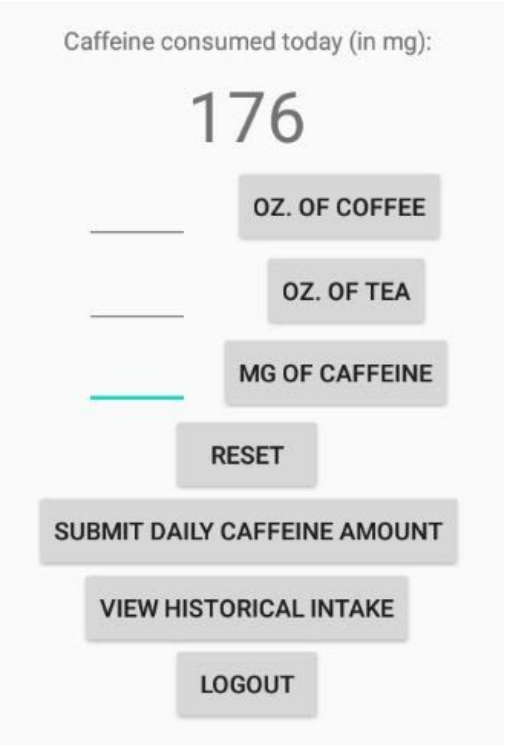
1. Login Screen



The login screen has the logo of the app as well as two fields for a username and a password. The create account button saves the current username and a hashed string of the current password. The Login button checks if the Username exists and then checks if the field after the hash is the same as what is stored in the database.

This UI contains two TextViews, two EditTexts,, two Buttons, and an ImageView.

2. ***Current Caffeine Overview***



The Current Caffeine Overview is the main activity of the app and if a user is remembered on the device, will be the launch screen. On this screen we are able to enter ounces of either tea or drink and, based off of the caffeine in milligrams per ounce obtained from google, automatically add the amount of caffeine in said drink to the counter. In the case that the user is drinking a beverage that is not tea or coffee, they are able to read the nutritional information and manually add in the milligrams of caffeine to the counter.

This UI contains two TextViews, three numerical EditTexts, and seven Buttons.

3. ***Historical Caffeine Overview***

The Historical Caffeine Overview shows all the intakes of the logged in user that they have submitted to the database. The ListView is populated automatically when switching to this activity.

This UI contains a ListView and a Button.

Code Design

- LoginActivity.java: Class handling logic for checking against the database, as described in UsersCP.java, for correct password and username as well as for inserting into the same database. Also handles logic for hashing passwords.
  - onCreate(), login(), createAccount().
- MainActivity.java: Class handling logic for incrementing caffeine counter as well as inserting into the database as described in IntakeCP.java. Also populates an ArrayList of Caffeine intakes submitted from the currently logged user.
  - onCreate(), ouncesCoffee(), ouncesTea(), mgCaffeine(), submitDay(), viewGraphAct(), resetDaily(), logout().

- GraphActivity.java: Class handling logic for populating ListView with the ArrayList from MainActivity.java.
  - onCreate(), goBack().
- UsersCP.java: Content Provider that sets up a database with columns: id, username, and password. This content provider has the same method list as IntakeCP.java.
- IntakeCP.java: Content Provider that sets up a database with columns: id, username, date, and amount of caffeine in milligrams.
  - onCreate(), delete(), insert(), update().

APIs

CaffGraph used no APIs.

Challenges

Using a custom content provider proved to be a tad too difficult for me to figure out on my own, so I settled on a simpler method of not representing the data as an object. Similarly, it seemed much easier to make a few things static as opposed to passing them through intents, I do not know if this is acceptable code style.

I did not realize that I had never populated a list through a query before this project, but the lecture notes were extremely helpful. Since I chose to just store all the entries in concatenated strings, under time pressure I had to cut a lot of proposed functionality relating to analysis because I did not have the time to implement the logic to parse these strings to isolate the mg per day.

Additionally, I could not find any good documentation on using a singular content provider for both databases but I worked around this by having two different content providers that practically handled each database separately.

Deviation from proposal/Incomplete Features

I was not able to implement the spending per day of the user on caffeine, the common statistical analysis of intake of caffeine, the tracking of when the caffeine was consumed each day, or the ability to set a goal that would color different entries in the ListView based on how close the user was to their goal on that day.

Spending per day, common statistical analysis, and the ability to set a goal that would change text color were probably possible to implement if I were given more time. While programmatically drawing a graph or finding patterns in the users time of intake were outside of the realm of my singular abilities for this project at this point in my experience.

Lessons Learned

I never realized that hashing passwords would be so easy. Cryptography has always been very abstract and terrifying to figure out with the sheer overhead of the maths involved. But while the LogCats looked scary, the validation and hashing algorithm were easy to implement and work like a charm.

I found myself working on this project into the early hours of the morning without even realizing it. I have found that I like creating apps as a pass time because it can be rather relaxing. The process feels very iterative and as my dream job is to be a software engineer, I can really appreciate the feeling of building and perfecting tangible UI-based software.

I really should have made a continuous stream of communication with my partner, I did not think it possible that they would not respond to me even when it was as late as a week until the deadline of the project being due. I think that had either of us made the effort to keep a stream of communication and progress from group creation to submission, the app would've been more fleshed out and good-looking.

Perhaps I should have made the images included in this report a bit smaller so there isn't so much white space in, but I think the report looks better instead of the alternative.

How About This

I wish this class would've been in person, I can't imagine how many more people would've done better and been more involved in the class had that been the case. Other than that, I think that the professor was one of the more sociable professors in my experience as well one of the best in terms of making the content seem easy. Granted Android Studio is an amazing piece of software, Professor Rodriguez taught so relatably that he made Mobile Programming seem like putting blocks in a mold.