***Insulin App***

***George Chester, Tamira Bean***

**App Description**

Our Insulin app is designed to provide users with peace of mind and to alleviate stress and anxiety about keeping track of insulin injections as well as running calculations. Our app can be used by people with diabetes themselves to store data but can also be used by teachers and professionals. Instead of having to use a calculator and doing the math yourself, our app will handle all of it for you as well as use an API to pull carb data from common foods.

**App Features***.*

1. Login: Any user can create an account that will store their data in a local database. There can be multiple accounts.
2. Calculator: The user can fill out the form provided in the app and once they hit the calculate button it will do the computation in the background and show the user the final value. There is a button to import carbs as well if they would rather do that then type it in manually.
3. Import Carb: With this feature, the user can type in robust sentences like “For breakfast I had 2 eggs,”. When they hit the go button, our app will interact with Nutrionix API to find the total carb count for their sentence. If the user knows the carb count for some of their food, they can insert that manually here as well to add to the total value.
4. Historical Data: The user can find all their data on this page, and it will be separated by date and time. The user can also look at a graph giving a visual representation of their past insulin levels.
5. Database: A local SQLite database is used to store all the data for the users and can verify their login information.
6. Security: All passwords given are hashed using Bcrypt. Users have a right to privacy with their medical data.

**UI Design**

1. ***Login***

***Graphical user interface, application, Teams

Description automatically generated***

This activity has two EditText. One for username, and one for password. Then their respective TextViews to show the user. Then two buttons for the user to click on. One button to go to create account, and one to login.

1. ***Create Account***

***Graphical user interface, application, Teams

Description automatically generated***

This page has a TextView at the top, then two EditTexts where the “hint” attribute is used to show the user what needs to be done. Then two buttons at the bottom. One to go back to the login page, and one to create the account with the user data.

1. ***Calculator***

***Graphical user interface, application, Teams

Description automatically generated***

This page has 7 TextViews, all used to guide the user for their inputs. Then 5 EditTexts for the form inputs itself. 1 final TextView that is hidden where the final calculation will be shown in at the bottom above the buttons. Then 3 buttons. The first one is to run the calculation, and the other two are to go to the home page and the import carb page respectively.

1. ***Import Carbs***

***Graphical user interface, application, Teams

Description automatically generated***

This page has 3 TextViews. Two of them are to guide the user and one is for the total amount of Carbs. There are 2 EditTexts for user input and 1 button for each EditText, with 1 button at the bottom to head back to the calculator. Total of 3 buttons.

1. ***Front Page***

***Graphical user interface, application, Teams

Description automatically generated***

1 TextView at the top to welcome the user. 2 buttons, one to go to the historical data, and one for the calculator.

1. ***Historical Data***

**Code Design**

This app contains the following classes:

1. MainActivity.java: Class handling logic for the login page.

* onCreate()

1. createPage.java: Class handling the logic for account creation

* onCreate()

1. FrontPage.java: Class handling the logic for the front page

* onCreate()

1. dbProvider.java: Class handling the logic for the database

* MainDatabaseHelper(), onCreate(), delete(), insert(), query(), update()

1. historyPage.java: Class handling the logic for the historical data.

* onCreate()

1. importCarb.java: Class handling the logic for the API and user searches.

* onCreate(), makeRequest()

1. Quote.java: Custom class for storing API data

* Quote(), getName(), getBrand\_name(), getServing\_weight\_grams(), getNf\_totalCarbohydrate()

1. calculatorPage: Class for handling the logic for calculating insulin and user input.

* onCreate()

**APIs**

Our project used 1 API:

**Nutritionix**: This API is used to gather carb data from common foods that users type in*.* All authentication is done by the program.

**Challenges**

*This project had several roadblocks during implementation of various parts. The first struggle was with the hashing of password for security. Android’s hash libraries are apparently incomplete and we needed to download a third party library to handle the hashing.*

*Also, there were issues with creating the API integration. The system from Nutritionix is incredibly robust but going through the documentation took a substantial amount of time to understand how it could work. The JSON had a different format than we had for our assignment (which is to be expected). This caused some delays to figure out how to parse different JSON formats.*

**Deviation from proposal/Incomplete Features**

*The only deviation from our original plan was in our security features. We were originally planning on having a more robust encryption system for our database and user data. However, with android’s libraries being a bit incomplete and not having a means of safely storing private keys without bringing up other security concerns, we opted to have a third party library hash the passwords to the database.*

**Lessons Learned**

*We learned more about the complexities of security systems and the numerous pitfalls that a company or user could run into when starting to create a safe app. We also learned to not attempt to recreate the wheel, so to speak, when dealing with complex systems like hashing and salting information.*

*We also learned more about API systems and how to integrate them. The different formats for JSON brings up the importance of understanding how to parse data properly. Looking into Nutritionix’s API system, we found that API’s can be incredibly robust and handle very complex queries. This gives more freedom and flexibility to developers in what they can provide to users.*

**How About This**

*In this section, describe anything that you think would have been helpful to do as part of the class. Some examples of things you may mention:*

1. *We think a brief dive into security within android and the limitations of the current libraries would also be helpful for students to at least be made aware of.*
2. *Covering API’s earlier and more in depth could have provided more help in understanding the complexities of documentation.*