# Background:

62,277 sports apps exist on the google play store, however they all share one common denominator; complexity. Picture this scenario, a powerlifter in the gym just set a new personal best on his bench-press, he either; writes this down in a notebook, or uses the leading weightlifting tracker Jefit. Paper can be lost. Jefit takes the creation of a new workout routine, adding a new exercise and starting a workout just to add a single record. So many choices and options leave the powerlifter over encumbered and frankly for the powerlifter such a complex process is a waste of their time. Enter Sherbert. Sherbert is the definition of an efficient app. Simply load up the app, click your lift and enter your record. Sherbert is the convenience of a notebook updated to the digital era.

Sherbert was conceived after a gym partner suggested my pre-workout supplement tasted like “anabolic sherbert”. As an amateur powerlifter I noticed a gap in the market whereby I could cater to this small cliché. Powerlifting comprises 3 specific lifts; squat, deadlift and benchpress, for someone specifically training in this sport an app tracking just these three lifts would be leaps and bounds more efficient to use than Jefit. Other creators in the play store have tried to hit this mark of a lightweight app designed for just powerlifting, however as these apps have been updated they’ve gotten lost in their ways and added unnecessary bulk to their app. An app confuscated with features may look great to a user surface level, but to repeat users these features go unused and make apps such as “Powerlifting Journal” a less than pleasurable experience.

UI design and functionality go hand in hand with a good user experience. Although this module focuses heavily on implementation rather than design it should still be noted that good UI rooted in HCI principles plays a large part in the functionality of an app. However even with good UI design an app that has no features is useless. It is therefore imperative you find a balance between having a well-designed UI and functioning application. This is the short coming of most apps on the sports category, apps are either clunky to use however have great features or a breeze to use but lack features users want. With this midpoint in mind I developed Sherbert to be not only easy to use but to resonate with its target demographic of powerlifters as a one stop shop for gym use.

With this I believe Sherbert, if added to the app store would have immediate consumption. Powerlifting is a developing sport, however also encompasses Olympic weightlifting in the form of the Clean and Jerk, an Olympic movement which encompasses all three powerlifts to train. For this reason, I also included clean and jerk recording to the app. I believe the app is easy enough to use for beginners but includes enough functionality for seasoned lifters alike. One of the main draws of the app is its base level façade of simplicity; the home screen depicts your records and gives you an exact value of what you need to beat. However, if you dig deeper hidden functionality not only secures data securely but adds a level of longevity to the app. The app itself it designed with the user in mind and to fill a specific gap in the market, however could easily be adapted to accommodate for other sports or the average gym goer.

# Implementation:

Creating Sherbert was nothing short of a challenge. The biggest challenge I found during implementing the app was a lack of debugging features in Android. Logging was temperamental on my machine and many times I had to resort to saving variable values to a shared preference file just to see what they contained.

Implementation first started at the login activity. The login activity acts as a kind of gate for the app, you either sign into google and supply your details to the app to or you cannot access it. Whilst it could be argued this limits accessibility it is a necessary design choice if further into production cloud based storage for each user profile is implemented.

The login activity uses Google’s single sign in API for users to log into the app and provide it with some basic information of who they are. This process is asynchronous and user sign in sessions are cached for future use, also coming in very handy if the user is offline. The activity “login” itself is designed using custom colours, .xml drawable shapes and a bespoke logo to stick to the idea that UI is a big part of good app design. The activity features a single button for google sign in so the user has no other option.

Upon successful sign in result whether it be cached or new sign in the user’s details must be saved for use in other activities. Shared preferences are xml files useful for saving configuration details or provide a store for small data for use between activities. However, anyone with root permission can access these xml files and so are incredibly insecure. When saving user data security should be the number one priority and so shared preferences could not be used as is. I myself have a keen interest in cryptography and by extension encryption. I chose to use an asymmetric cipher, namely RSA to create a pair of keys then use the inbuilt android keystore to store and retrieve these keys at a later date. Inbuilt java security libraries (javax.crypto) are used generate the keys which then encrypt the retrieved strings containing the user’s name (displayname) and a URL to their google profile picture. Whilst these values could be retrieved from memory as the app executes the encryption obfuscates any data stored about the user. These now encrypted strings are stored in a shared preference file named “userData” and a toast feedback popup informs the user they have signed in successfully.

This complicated process is encapsulated in a try and catch block with multiple catch parameters for every point of failure in encryption. If an exception is thrown an error is logged (viewable in logcat) with the exception shown to debug. Even if the process of encryption fails the user is still taken to the home activity via a new intent and a placeholder username and icon is shown instead. The process of creating a new key pair to add to the keystore is kept in an if block to check if the key pair already exists for the user and if not to create one. The Google sign in API itself checks for connectivity in the OnConnectionFailed method and logs any errors. The sign in button itself uses an on click listener to call the sign in API. The API parses its data itself and no further parsing is used.

The largest problem I found implementing this was the encryption of the shared preferences file. There are no inbuilt methods to encrypt an xml file itself and so I was forced to encrypt individual strings. I could have used a third party library for this process such as “SecuredPreferences” however for 2 strings I did not see the need. The main problem I encountered was with the padding parameter of the encryption. The strings would encrypt fine but would not decrypt in any activity. After exploring why this could be such an issue with padding whereby the decrypted string would be one block less than the original I found I had duplicate variable names as the process repeats twice. This took quite a while to find as the strings were still encrypted and saved to shared preferences as normal. This was also not helped by a lack of debugging features.

Home2 is the second activity a user encounters and is the hub of the app. Home is named home2 as from rebuilding the app when downgrading from android studio 3.0 to 2.3 the original home activity code would not compile. Home itself features a multitude of elements and functions.

Starting from top is the custom toolbar. The toolbar features the bespoke logo and two icons, an open drawer icon and a cog. The cog opens the settings activity. Contained in here is decrypted displayname and a logout button. The logout button creates an instance of the google api client and signs the user out. This also clears all data from the shared preferences file to increase security and minimise loss of data.

Open drawer opens a side navigation drawer for navigating the app. Inside this draw contains the name of the activity and an icon beside to represent it. The menu uses values from a menu xml file (activity\_home2\_drawer.xml) to populate the menu and each value is clickable. Upon clicking an open the user is taken to the corresponding activity. This is done using a switch case with the id of each menu element being the switch. This easy to use menu gives the user direct navigation to any part of the app and further increases user experience. Clicking the cog wheel triggers an onlick listener to take the user to the settings activity via new method.

Next is the user portrait and username. This was implemented to make the app a more personalised experience to the user. The aforementioned keystore is accessed to retrieve the private key using the same alias as the login activity to identify the correct key pair. The encrypted string is then loaded from shared preferences ran through a decryption cipher to produce the original google display name of the user. This then set as the value of a placeholder username. The next is the user portrait. Firstly, the encrypted url of the user’s icon is read in from shared preferences and decrypted using another cipher. Then a third party library named Picasso is used to download the image asynchronously and display it to the image view placeholder. I chose to use a third party library here over an asynchronous class as Picasso caches the image as well as accommodating for network changes during download. This allows the app to operate offline further and provides error handling for network state.

Following this the user’s personal bests for each lift are displayed. This is done by calling a method in each lift’s respective database helper class that uses a raw SQLite query to find the max value in the weight column for each table. These calls happen onCreate() of the activity so if the user adds a new record and returns to the home screen it is displayed. Each of the lifts pictured can also be clicked to access the specific activity relating to it. The largest problem I faced in implementing this home screen was the issue described earlier in the encryption as well as working on the design. The design itself was prototyped on paper and online and a colour scheme was created.

Unfortunately, prior to writing this report I broke the way location data was created and as a result completely broke my RESTful API. I was originally having to open up google maps to get an instance of a stored location then using that data to get latitude and longitude. However, I changed this to be a locationlistener that updates location if no data is found (to circumvent the maps issue) or if the user moves. This in theory should work however only returns the lat and long value 0.0 which is rejected by the google places API.

Prior to this however a permission check is carried out in which the user has to give permission to use their FINE\_LOCATION. If accepted the user continues if denied the app returns to the home activity. Once the latitude and longitude is found at current location an uri builder is initialised to build the api request. The uri builder builds the http address of the API and accounts for changes in location. The finished url is built as a string and used in a call to the asynchronous class to parse the returned json.

This class takes the url and calls a separate class file named gethttp. Gethttp takes the input http url, creates a http connection and based on the response downloads the data. If the response code is 201 (created) data is added to a string a buffered reader to read and string builder to write. Errors are managed using catch blogs and logged. The finished string is returned back to the asynchronous task.

This json data is then stored in shared preferences file for use offline and JSON objects and arrays are initialised. An iterating for loop loops for the length of the results iterating by each object of results and for the array values matching “name”, “rating” and “vicinity” these are then stored in a hashmap (a type of array for storing key value pairs). This is surrounded in a try and catch block that logs exceptions as errors. On the end execute block the listview to display gyms is brought into context and a list adapter is used to display these using string arrays to the list view.

This activity by far was the hardest part of implementation of the app. I had so many problems with the google places API and its documentation which were ultimately resolved after I discovered a google maps api key is separate to a google places API key. The next problem I faced was the structure of the JSON api as it was multilevel. I had originally planned the activity show the gyms via marker on the map however parsing the JSON proved to be a massive issue alongside google places not having great documentation. I finally decided to reduce the complexity of the output seeing as the json is still parsed and displayed. Change the fetching of location created a knock on effect as I thought the problem lied within the parser and so I made further changes to it this in turn unfortunately broke the activity. The API in question wasn’t my original choice of api. I had originally planned to use Wger; an open source fitness tracker whereby I could look up exercises, however to my disappointment all results were in German and so not appropriate.

I do think however the highlight of the app is each lift’s own activity. Each activity uses SQLite to store information from the user for each lift. The SQLite helper for each class is initialised on create whereby a table is created with version number. Each sql helper has 3 methods; gethighestval (for use on the home activity), getallinfo and addrecord. Getallinfo uses a raw query to select every record from the table in question. This returns a cursor containing all data. This is called on the onclick listener for a button named view all and creates a new popup displaying every record. Add record takes the current text of the textentry fields and inserts them into the current table. A prompt appears telling the user if the data is added or cannot be added.

These methods employ some interesting error handling. If the returned cursor contains nothing for viewall an error dialog is created telling the user to add some data. The method for inserting data itself returns a Boolean. Inside the method result is declared of type long (a single object of long type). The insertion function itself contained in SQLhelper returns -1 if the record could not be inserted therefore we can filter by using an if statement that if the long type is = -1 data was not inserted and so returns false. An if statement in the calling method checks the Boolean and so if it returns true and data is inserted textentry fields are cleared of their values and a prompt is displayed. A prompt is also displayed if text could not be inserted.

Personally I found these activities quite easy to implement due to previous experience with sql. The raw query syntax made sense to me and the only part I found difficult was finding a way to display all database values. I had originally wanted to display in a listview however settled on an alert dialog as the activity would have become too cluttered and hard to use. A potential workaround would be creating a new activity to show every value. I chose to use SQL for this over shared preferences as I could easily query a table rather than parsing strings and an sql table is a more convenient way of storing data for future use if adding functions like graphing or extrapolation. I chose not to encrypt these tables as the information contained was not sensitive.

# References and Resources:

## Resources:

<https://play.google.com/store/apps/details?id=je.fit&hl=en_GB>

Jefit – the number one workout app on the google play store.

<https://play.google.com/store/apps/details?id=com.powerliftingjournal&hl=en_GB>

Powerlifiting Journal – a specific powerlifting tracker app.

<https://stackoverflow.com/questions/27921515/sign-out-from-google-and-facebook-in-android-application>

Used to integrate the google sign out function in another activity (settings)

<https://www.androidtutorialpoint.com/intermediate/google-maps-search-nearby-displaying-nearby-places-using-google-places-api-google-maps-api-v2/>

Get location data and search nearby places using the google places api. Also shows how to parse JSON but is slightly outdated.

<https://github.com/obaro/android-sqlite-sample/blob/master/SqliteExample/app/src/main/java/com/sample/foo/sqliteexample/ExampleDBHelper.java>

Inspiration for the database helper classes.

<https://www.youtube.com/watch?v=T0ClYrJukPA&t=155s>

Video tutorial on implementing SQLite to android

<https://www.androidauthority.com/use-android-keystore-store-passwords-sensitive-information-623779/>

How to use RSA and android keystore to store keys. Android ssl not needed and will break decryption.

https://yuvislm.wordpress.com/2012/09/10/google-places-api-and-json-parsing-in-android/

A further resource on how to parse google places api as the official documentation is not great.

Mobile Computing Workshops

## References:

Friesen, J. (2014). Learn Java for Android development. 3rd ed. [New York]: Apress, pp.287-312, 763-801.

A great book for learning java and android development. Used the chapters on API’s and databases.

Clifton, I. (2013). Android user interface design. Upper Saddle River, NJ: Addison-Wesley, pp.47-79, 112-116.

Used to learn basics of design and how to use drawable xml elements.