4. Development

This section outlines the development stage of the project. It details the development of the Android application (including the process of creating the user interface, database, and subsequent gamification elements of the app), as well the subsequent user studies and machine learning implementation that followed.

**4.1. User Interface**

Shortly after completing the research and planning, the development of the project commenced. This chapter begins with an overview of the app’s development. As was the case in the project plan, the user interface of the app was the first thing to be built – subsequently based on the wireframes developed on Mockingbot.com (see section x).

Android Studio was the platform used to develop the app, utilised because of its immense capabilities and array of built-in libraries – which made the design and implementation of the app much easier. Notably, Android Studio divides the app into its layout and its functionality, using XML and Java, respectively. For the layout, resources were made to represent colours, strings, styles, and dimensions that would be used throughout the app (in the Android studio project, go to the “res” folder).

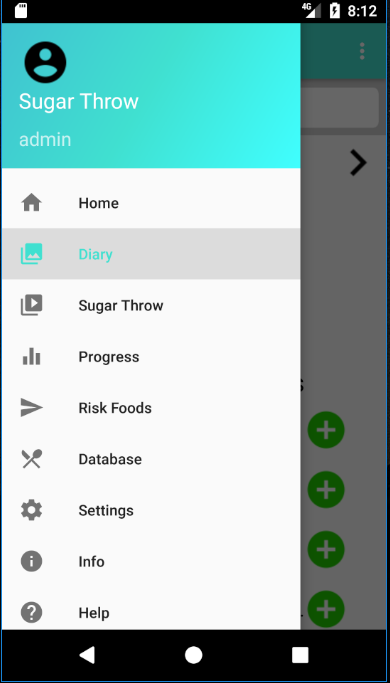
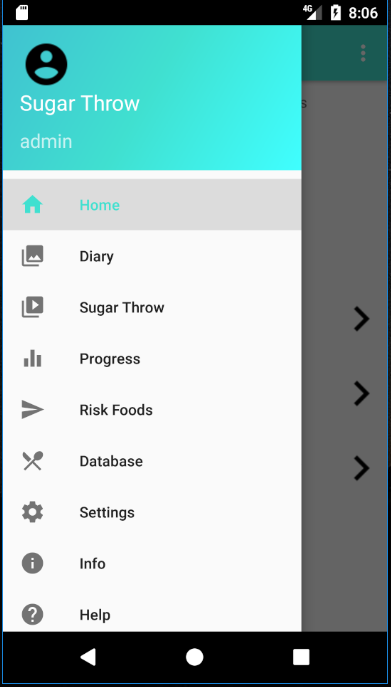
In Java, a LayoutCreator class and ViewCreator class were created for when layouts and views (i.e. TextView, ImageView, EditView, etc.) had to be instantiated programmatically, meaning the front end of the app would be created based on what existed in the Java code. This was used, for example, for generating the foods that appeared in the diary. This was necessary because it is unknown how many foods the user will add to their diary – if a fixed number of TextViews were added to the layout, this would restrict the user if they intended to add more than what was allowed. Therefore, the TextView would be created whenever a user added a diary, so that the diary could be of a variable length.

The following subchapters will go into detail regarding some of the user interface features.

***4.1.1. Navigation Drawer***

Navigation around the app was pertinent to its usability. Subsequently, a navigation drawer (or side menu) was added, with its implementation made easier by the built-in libraries in

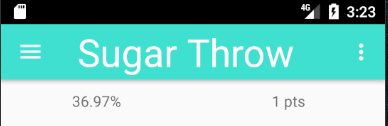
Android studio. The IDE has an activity titled NavigationDrawerActivity, which comes with an already existing side menu. This was customised to include the features present within the app, and to include the username of the user who is currently logged in (see figure x). Moreover, the items that existed with the navigation drawer could be selected (code which was editable within the activity in Java).



***Figure x****: Navigation Drawer*

***4.1.2 HUD and Toolbar***

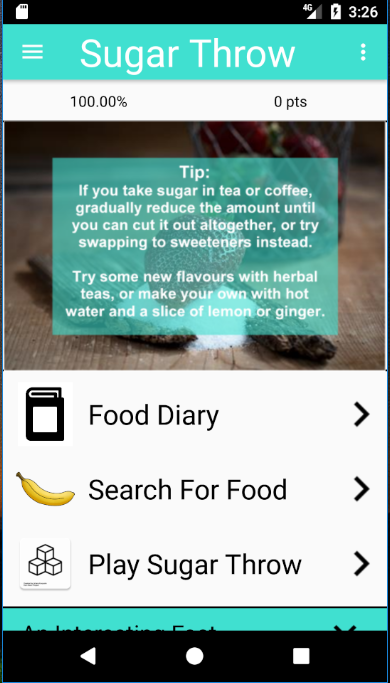
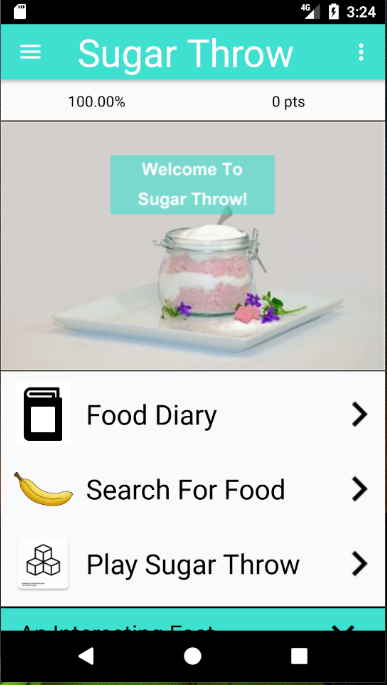
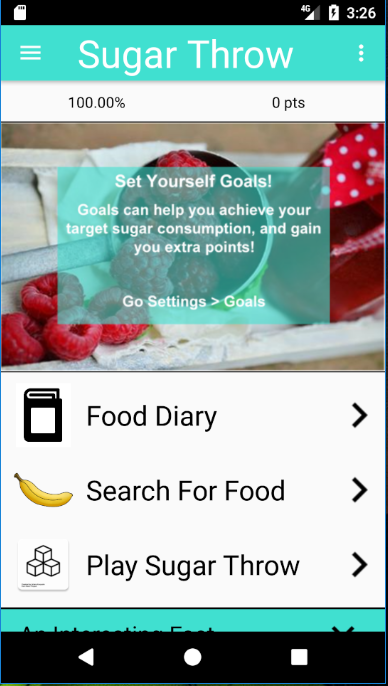
To give the user more information on the home page of the app, a heads-up display (HUD) was created. The HUD includes the amount of sugar the user has left on their daily allowance and the total number of points they have obtained thus far.



***Figure x****: Toolbar and HUD*

Another significant feature in the image above is the toolbar. Not only was this used for access to the navigation drawer (toggling the burger navigation icon), but also included the title of the activity and an options menu (seen on the right of the above image). Considerably, therefore, both the toolbar and the HUD are an important element of the user interface.

***4.1.3 ImageSlider***

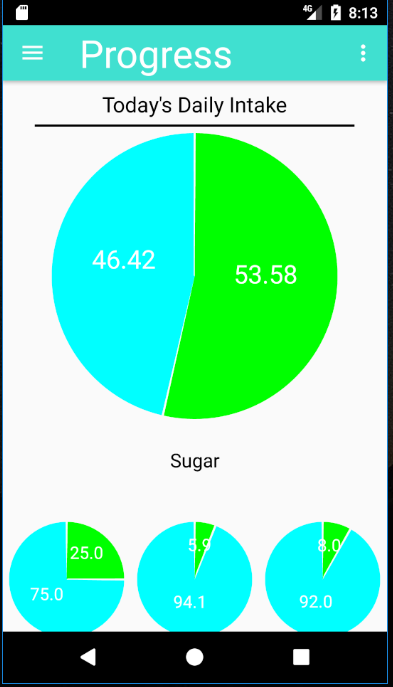
The ImageSlider class inherits the methods and functionality from the PagerAdapter class. The latter class is used to allow multiple images to occupy one space (thereby mimicking an image carousel). Subsequently, if the user swipes left or right on the “hero image” on the homepage, another image will appear (see figure x).

***Figure x****: Three Images within the ImageSlider*

***4.1.4 MPAndroidChart Library***

Much of what exists in the standard Android Package Kit (APK) was sufficient for this app. However, the APK is limited when it comes to creating charts and graphs (which were necessary for displaying the daily intake and usage streak in a graphical form). Subsequently, the MPAndroidChart library – developed by PhilJay (2017) – was added to the list of dependencies.

As highlighted in figure x, the MPAndroidChart library was used to represent data in a graphical form, i.e. pie charts and line graphs. The pie charts show the percentage intake and amount left a user has for a particular food group (sugar, salt, etc.). The line graph shows the number of foods the user has logged over the last five days. Considerably, these elements make the app less mundane and subsequently more appealing to the user.



***Figure x****: Uses of MPAndroidChart Library*

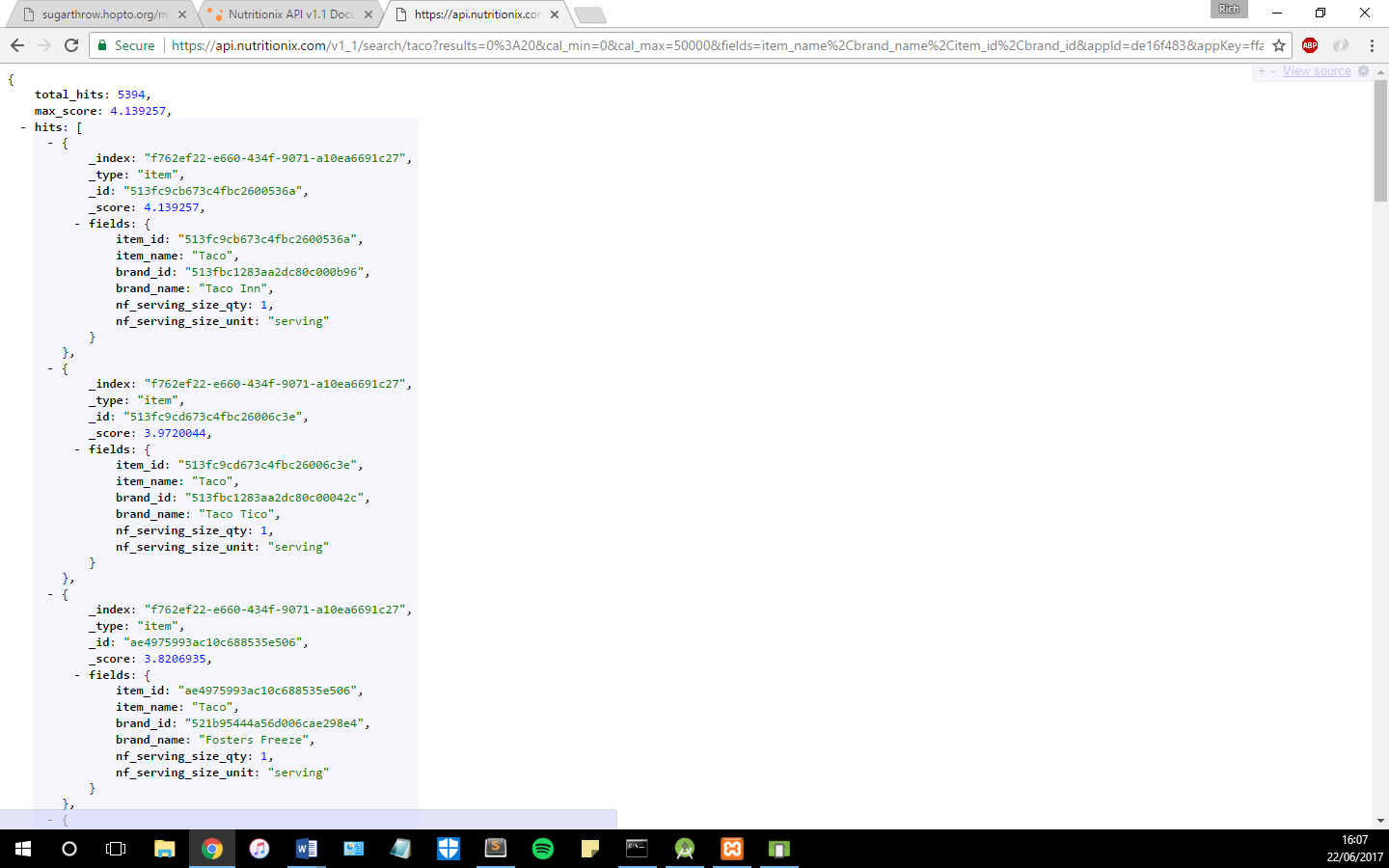
***4.2 Nutritionix API***

In order to obtain food items, and nutritional information corresponding to those items, an online food database was needed. The Nutritionix API offered free access to over 500,000 grocery items from Canada, the US, and the UK (cite). Although there are certainly more effective and richer APIs that exist online, Nutritionix offered the minimal requirements necessary for a project of this calibre. It was therefore the ideal choice.

***4.2.1 How the API worked (Volley requests)***

The Nutritionix API allows calls to be made to the database via a unique API ID and API key. The database is then queried using these unique identifiers, in which a URL is returned. At this URL, sits the results of the search, which exist within a JSON obejct (as identified below).

Then, a way to parse the contents of the webpage was needed so as to use the search results in the app. Subsequently, the Google Volley library was utilised. Volley is Google’s networking library which allows for HTTP requests to be made (cite). On response of the request, subsequent processes would occur. However, in some cases, the volley request returns an error, so is therefore given an “error listener”. This was used so that if, for any reason, there was an error in the HTTP request, the app wouldn’t crash.

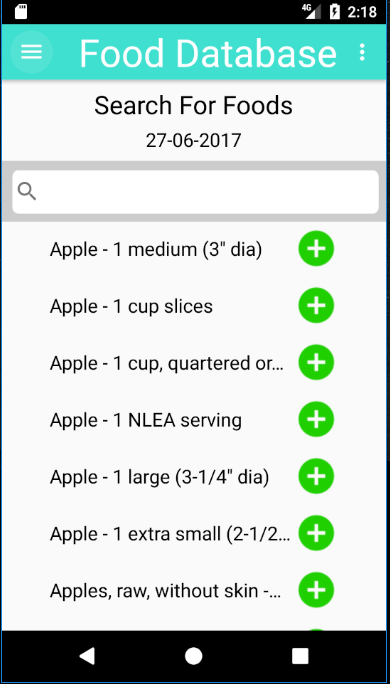


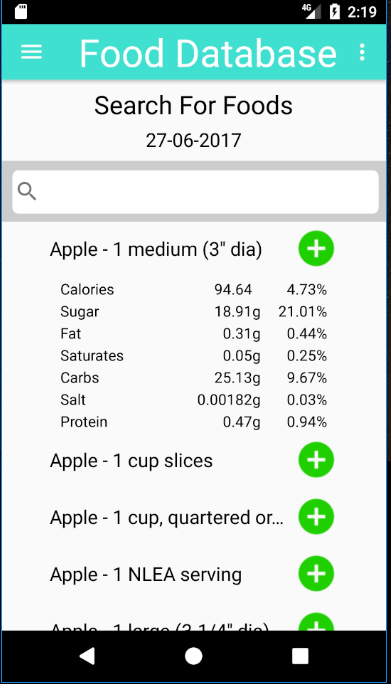
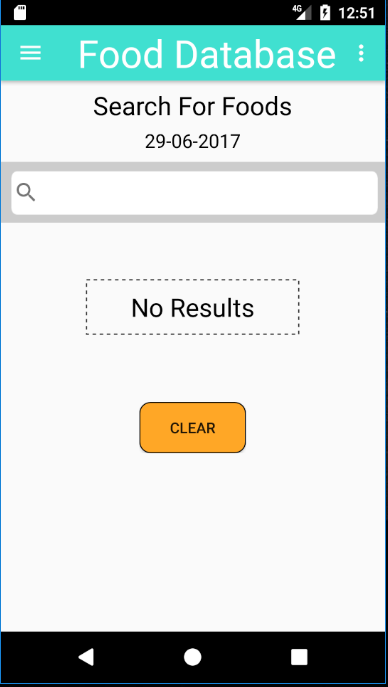
***Figure x****: JSON Object returned by API*

In the request method, Volley uses the URL provided to send a HTTP request. Once this request responds, the response is passed into a String which, in this case, is an array of JSON objects.

***4.2.2 Returning Results to the App***

The String which is returned on response of the Volley HTTP request is an array of JSON objects, provided the search returned any results. If the search did not return any results, then the app would display “No Results”, as shown in the left image of figure x. Otherwise, the results are read into a 2D array list of Strings, containing the item name, item id, and brand name of the products returned. This was done by using a JSONArray object (while the JSONArray still had hits to be read, the results would be parsed using a JSONObject, which would contain the identifiers needed in a String).





***Figure x****: Food Database Search Activity*

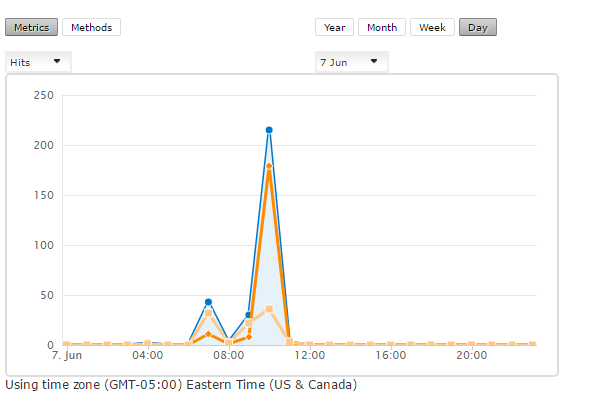
Notably, the item id needed to be recorded in order for UPC searches to occur. UPC searches are those in the Nutritionix API which include the nutritional information for the corresponding food item. These requests were similar to those used to search the API, but instead of a search term, the URL is passed the item id. Like the search request, the returned response is a JSON array, but with just one JSON object.

***4.2.3 Limitations of the API***

A significant issue with the Nutritionix API is that it limits the number of API calls an account is allowed to have per day (unless a monthly subscription is purchased). It limits the number of searches to 5000, and item views (views of nutritional content) to 200.

Search calls are used whenever a search request is made within the app, i.e. the user uses the search on the Food Database Activity or on the Diary Activity. Item view calls are used whenever a food item is added to a user’s diary, or the content is looked at when a food is clicked (see figure x).

The graph below shows the usage of the API over a day. Noticeably, the number of item views (dark orange) surpasses 200. This resulted in the API no longer being usable for that day. This wasn’t a problem when using the app for personal use, but would become a significant issue when the user studies take place. Subsequently, a method to resolve this issue was to mimic “caching” the data.



***Figure x****: The Daily Usage of API*

***4.2.4 Caching the Data***

The purpose of a cache is to store data, which is accessed regularly, away for future use (cite). Therefore, to counteract the limits on API usage, a local database was used so as to “cache” the data. It effectively stored foods that were used most often, so calls to the API weren’t necessary. The creation of the database is highlighted in section 4.3.

To mimic a “cache”, whenever a food item was added to a diary or the contents looked at, the information for that food would be stored in a local database. Then, when that food was searched for again, its contents would be referenced from the local database, therefore not making a HTTP request and using an API call. This process would prove to be vital for allowing multiple usage of the API.

***4.3 SQLite Database***

A database is pivotal to the app in order to record the user’s daily food intake. As a result, an SQLite database was created. Notably, MySQL databases are expensive (cite)…Alternatively, SQLite databases are known to be useful for embedded systems (such as apps), as they are simple, and therefore less expensive.

***4.3.1 Creating the Database***

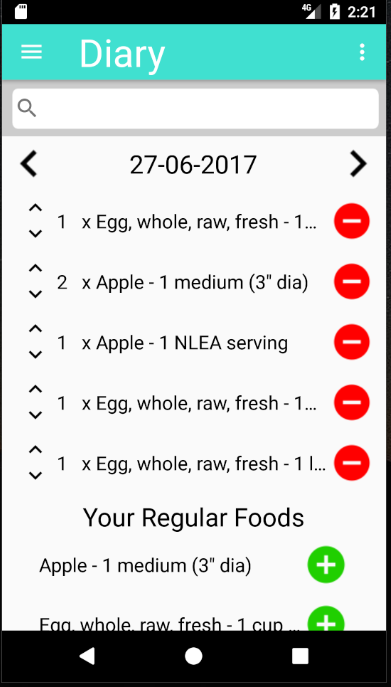
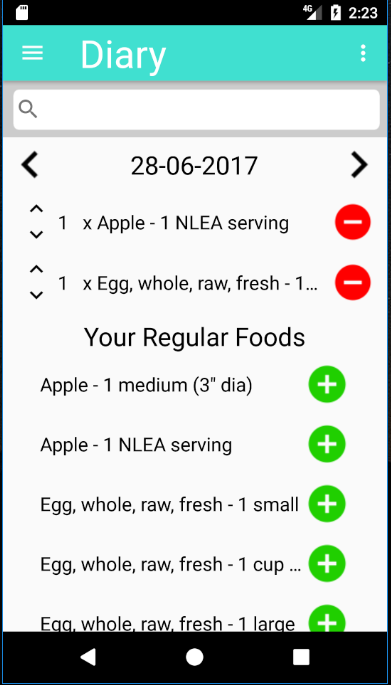
The database was created locally on the command line using SQLite3. The five tables created were “User”, “Food”, “Diary”, “Goals”, and “Sugar”. A schema is highlighted in the appendices, figure x.

The database created on the command line was copied into the assets folder of the Android project. Then, a Connector class was used to establish a connection with the SQLite database that sat within this folder. This class inherited the SQLite operations from the SQLiteOpenHelper class that exists within the Android SDK. “Connector” is a Singleton class in which the database connection is accessed by various activities throughout the app.

The “Execute” class is used for querying the SQLite database. It is passed an initialised Connector object, allowing inserts, selects, updates, and deletions to be performed on the database. The “SqlQueries” class was used to store all the SQL queries used in the app. All these queries include placeholders (“?”) so that the app isn’t vulnerable to SQL injection.

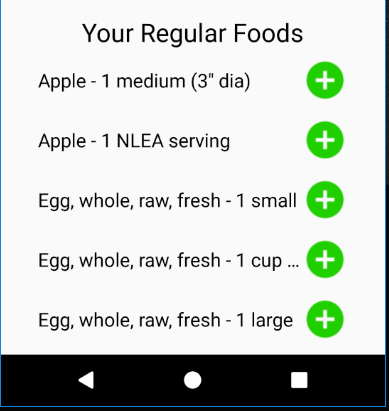
***4.3.2 Diary and Food Database***

The SQLite database is used a considerable amount throughout the app. One of the more important uses is in the Diary Activity. The database holds the foods the user has eaten on a given date (queried from the Diary table). The user can also use the arrows to the top of the activity to view diaries from other dates.



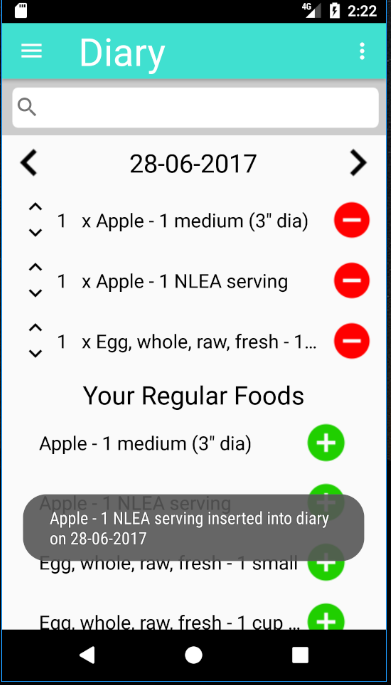
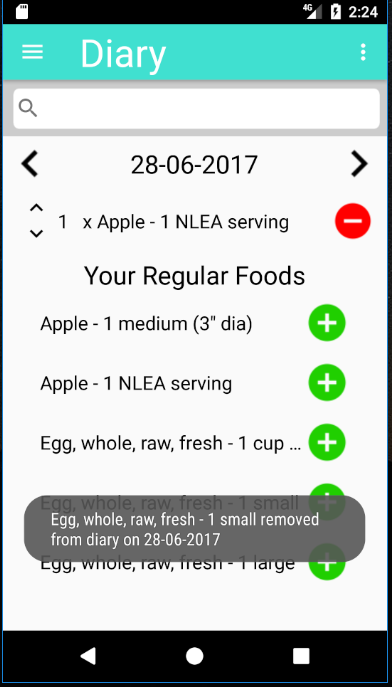
***Figure x****: Screenshot of Diary Activity*

A regular foods section is a feature that proves to exist on many dieting apps. It is often the case that a user eats certain foods more than once a week, and this section saves them having to search for the food every time they want to add it.



***Figure x****: Regular Foods Section in the Diary Activity*

Finally, the database is used within the Diary Activity for inserting / removing foods. The user can press the “minus” buttons to remove foods from their diary, or the “plus” button that sits within the regular foods section to add that food to their diary. The contents of the diary will then be updated. The user can also increase or decrease the quantity of that food by pressing the arrows to the left of the food name (see figure x).



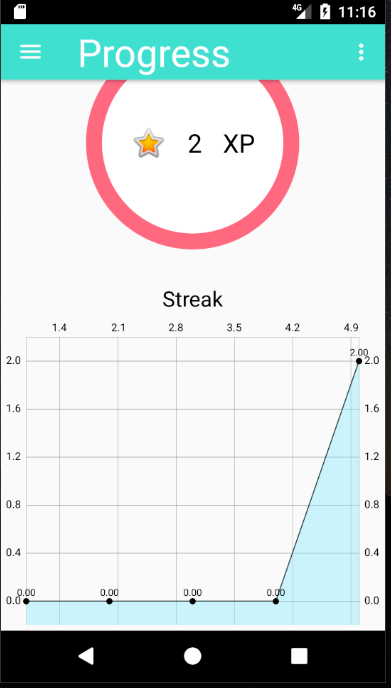
***Figure x****: Food being added and removed from Diary*

In addition, as previously noted, the contents from the Nutritionix API are cached into the local SQLite database to save on API calls. This is the main purpose of the database for this activity, bar adding foods to the user’s diary.

In some cases, the app needed to wait until the request had finished before it could process certain things. This is an instance of an asynchronous task. In order to prevent certain processes from running until other processes had finished, an Interface titled “ServerCallBack” was created, containing the method “onSuccess()”, which would return the response string once the HTTP request had completed. In the case of inserting into the local database, this prevented any null objects being processed before the HTTP request had returned any of the data.

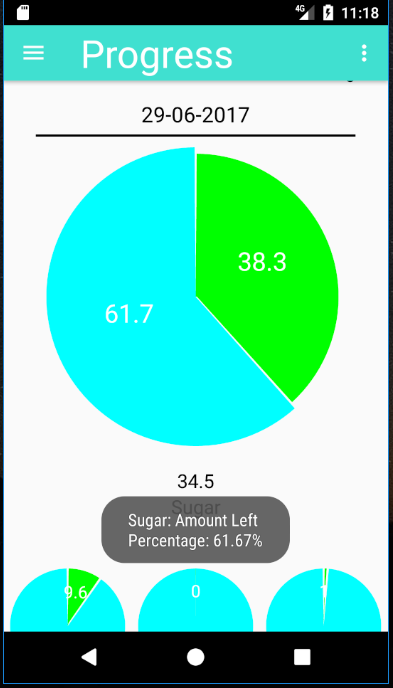
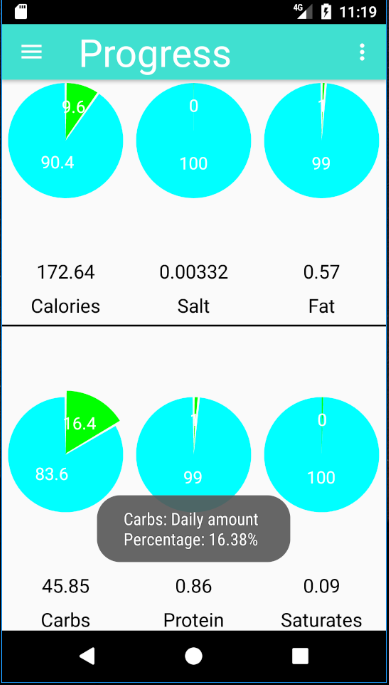
***4.3.4 Progress and Goals***

In the Progress Activity, the SQLite database is utilised for a number of components. For instance, the user’s points (obtained from the “User” table) are displayed at the top of the activity. Moreover, the daily intake per food group the user is allowed is altered based on the “Goals” table – if the user has indeed set daily goals (see figure x).



***Figure x****: Screenshot of Navigation Drawer in App*

Furthermore, the total intake and amount left is calculated depending on data that exists in the database. The database is queried for the SUM of each food group on a given date and the percentage contribution for that food group is determined within the FoodContentsHandler class. This class is responsible for any process involving the nutritional content of food; it is used considerably for giving personalised feedback (see section x).



***Figure x****: Screenshot of Navigation Drawer in App*

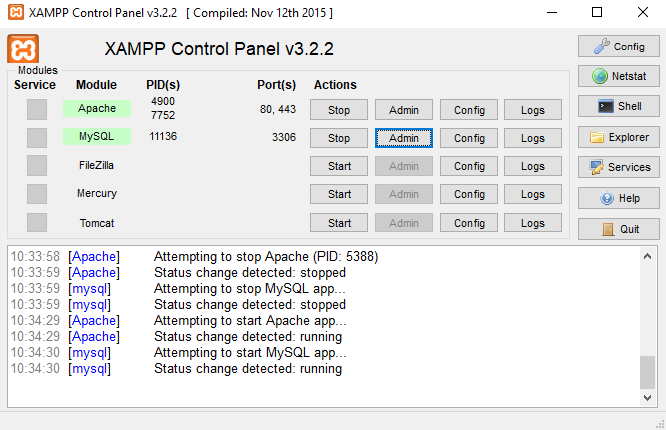
In addition, a subtle feature that exists in the app is the ability to view the Progress of different dates in the user’s diary. If the user clicks on the date within the Diary Activity, they will be sent to the Progress Activity and displayed the nutritional content for that date. If they simply navigate to the activity from the Navigation Drawer, they will be displayed their progress for the current date.

***4.4 Online Database***

Having a database that is local to a user is adamant for recording their data and determining their daily food intake. However, this data is only local to their device, making the data inaccessible to this researcher. Subsequently, a place to store the data online was needed. This subsection outlines the development of an online database that was accessible from a static IP address.

***4.4.1 Creating a Server***

In order for this data to be accessible, it needed to exist somewhere on the web. This researcher therefore had two options: the first would be to purchase a domain name and push contents onto a website; the other would be to create a home server. After much deliberation, the latter option was chosen.

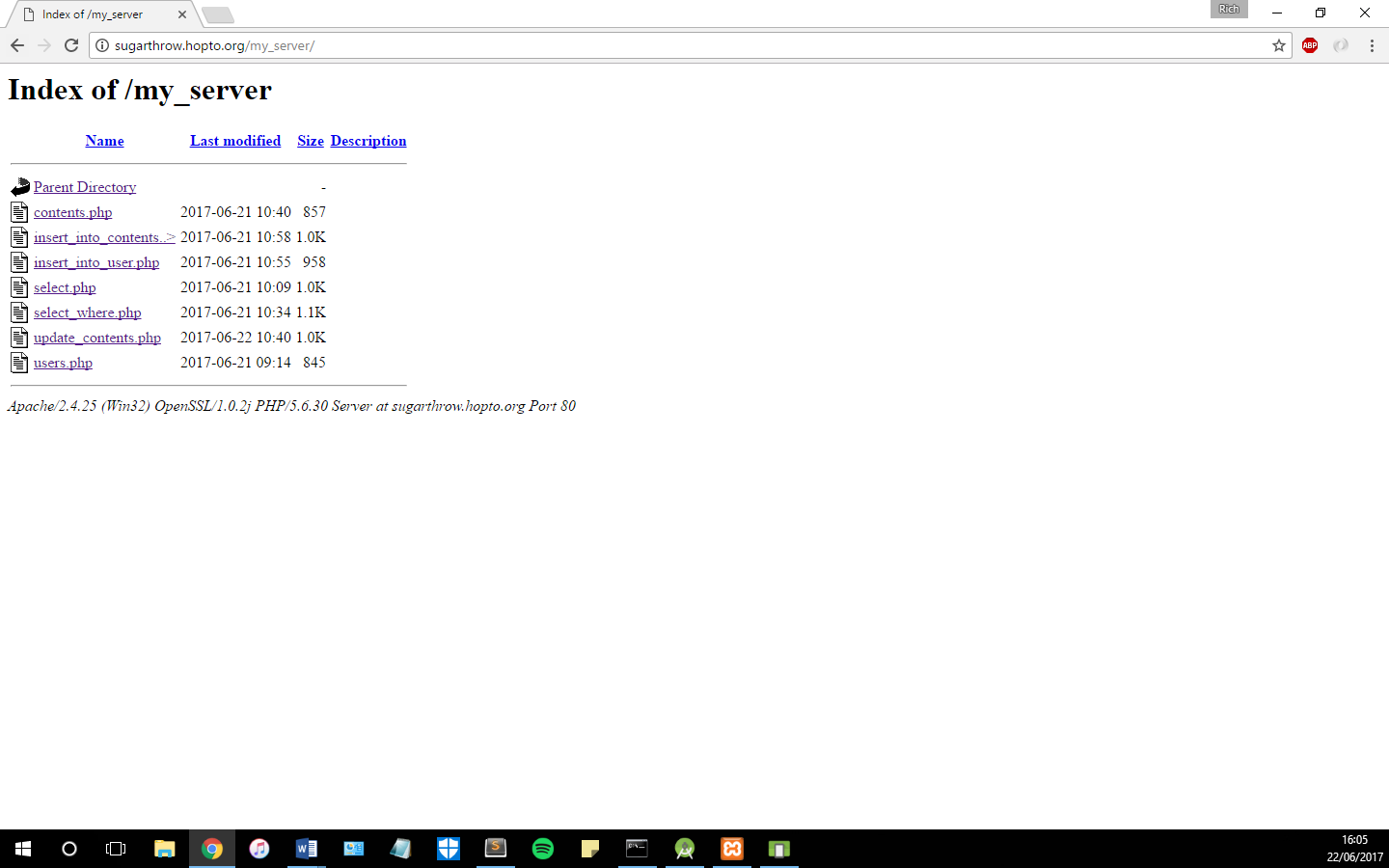


***Figure x****: Screenshot of Navigation Drawer in App*

The process began by first downloading the XAMPP server client. XAMPP is a free, open-source cross-platform web server package consisting mainly of an Apache HTTP server and MariaDB database (Apache, 2017). The inclusion of a MySQL server made this the ideal software to use for this project.

As only the user data and the total amount of food a person has eaten on a given date was required, the database consisted of just two tables, “Users” and “Contents”. The database was created using phpMyAdmin – software provided by XAMPP that handles the administration of MySQL over the web (cite).

Next, the “httpd.conf” file was edited to point to a folder titled “server” on this researcher’s home computer. Now, anything within this folder could be accessed by entering the IP address of the local network. In order for the contents of the folder to be accessed by outside users, the router needed to be configured to allow HTTP requests. Port forwarding was set up so that the computer acting as the server handled any requests on port 80.



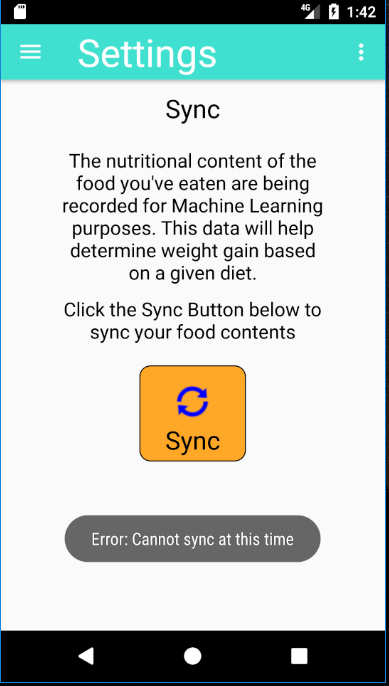
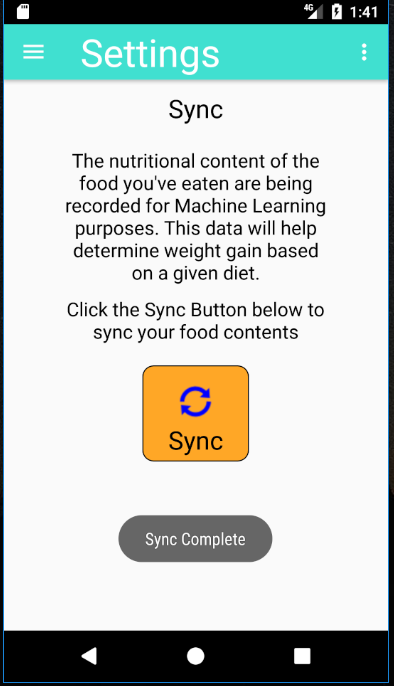
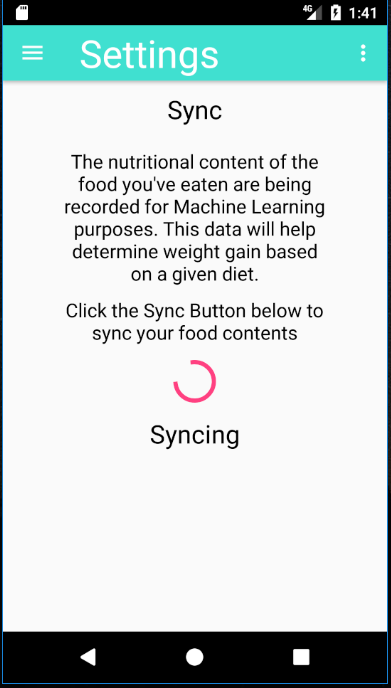
***Figure x****: Screenshot of Navigation Drawer in App*

As Windows Firewall blocks HTTP requests to direct to the IP (to prevent malicious attacks), this needed to be bypassed by adding an inbound rule to allow any requests on port 80 to be accepted. Now, the contents of the “server” folder could be accessed by any computer by entering the network’s IP.

However, IP addresses aren’t always constant; they change whenever the router is reset and then randomly allocated (cite). As a result, a dynamic domain name was needed. Subsequently, a free domain name was obtained and the Dynamic DNS Update Client was downloaded. This software links the domain name to the IP address, so that when the IP changes, the contents can still be accessed by reaching the same domain name. The server could now be accessed by going to <http://sugarthrow.hoptp.org/>.

***4.4.2 Adding Content to Server***

After everything for the server had been set up, PHP files were created to query and make changes to the online database. The “contents.php” and “users.php” files simply select all rows from the corresponding tables (“Contents” and “Users”, respectively) and display the results in a JSON Object.



***Figure x****: Screenshot of Navigation Drawer in App*

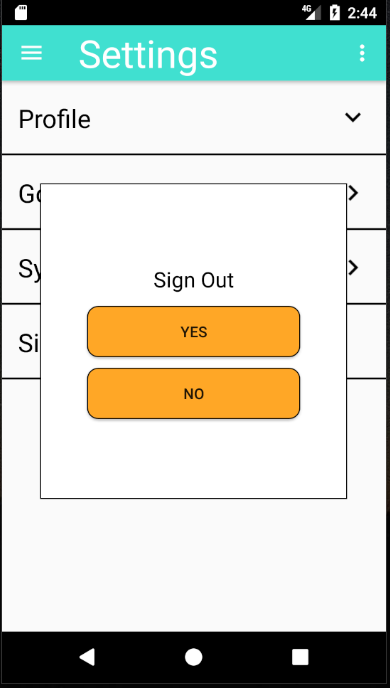
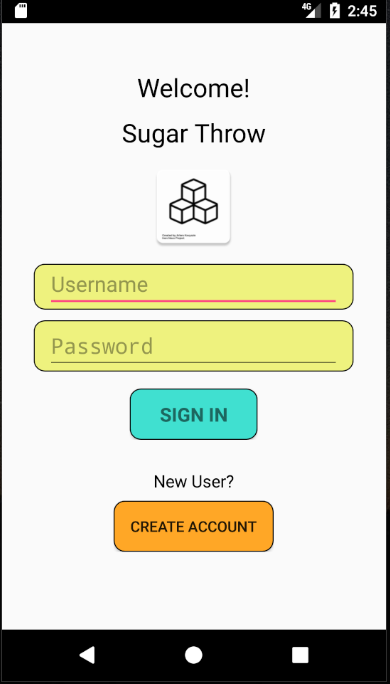
The “select\_where.php” and “select.php” files returned JSON Arrays of JSON Objects so that the data could be processed in Android studio. In some cases, these were used to query the database to see if data existed. A particular problem that arose when using these files to query the data was that if no results were returned, Android studio would throw an error. This would often be because the app would try to parse a JSON Object that didn’t exist. Subsequently, this was altered in the PHP files. If the query returned no rows, then a JSON Object would be created containing the value “null”, thereby solving the error. The “insert\_into\_user.php” and “insert\_into\_contents.php” files were used whenever insertions to “User” and “Contents” were necessary.

To allow the user to sync their data to the online database, a Sync activity was added under Settings. If the user clicks the “Sync” button, a progress bar will appear, letting the user know that the app is syncing. Considerably, as the server is just a home computer that isn’t on 24/7 (like most servers), a user could sync their data when the server wasn’t running. Thus, methods which made requests to the server had an error handler which would return a message saying “Error: Could not sync at this time”, followed by the name of the error, e.g. Network Timeout Error (see figure x).

This was used to safeguard the app and prevent it from crashing whenever the server (computer) was turned off. If the server is running, and there are no errors with the sync, then the user will be displayed a message saying “Sync complete”.

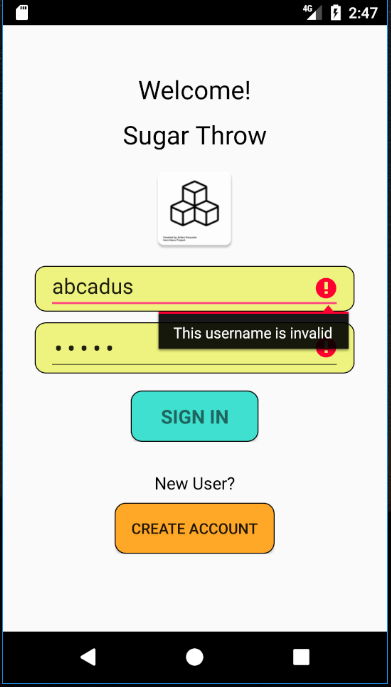
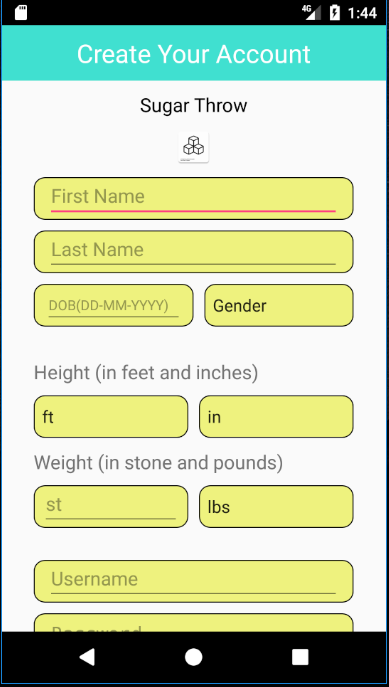
***4.4.2 Logging in and Signing Up***

The concept of multiple users on one device was made possible by having the local SQLite database. A user would be able to log out of their account and sign into another one if they chose to. Signing out of an account would then result in the Login screen appearing (see figure x).



***Figure x****: Screenshot of Navigation Drawer in App*

The database would be used to check whether the username exists and that the password corresponding to that user was correct. If any of these prove to be invalid, the user is prompted by an error (see figure x). If the user does not already have an account, they could create one by clicking the “Create Account” button at the bottom of the Login Activity.



***Figure x****: Screenshot of Navigation Drawer in App*

To sign up for an account, the user enters detail which are then inserted into the “User” table. Before the account is created, the app first checks to see if the username already exists on the online database. If it is in fact unique, then the user’s details are added to both the local and online database.

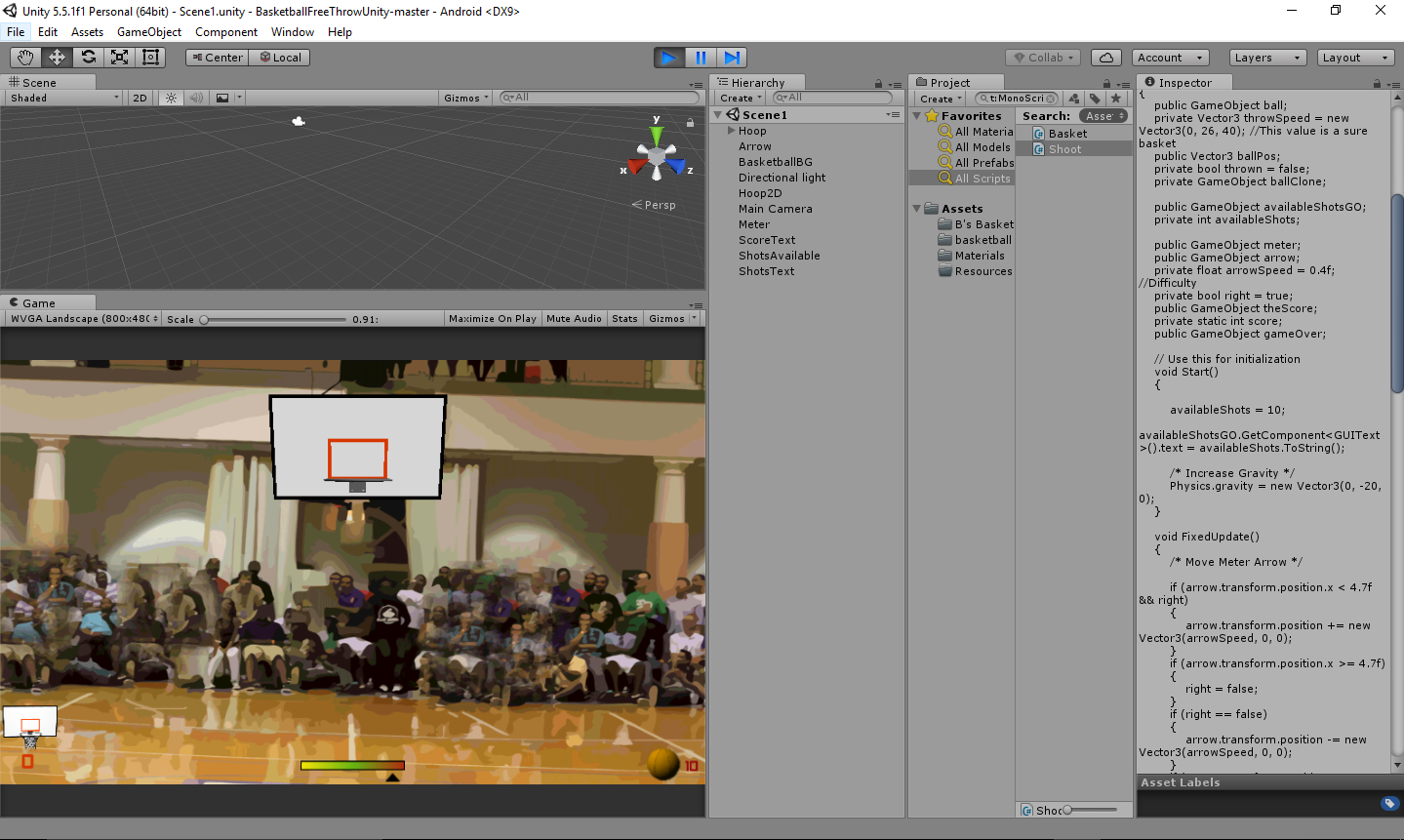
When the user opens the app, the LoginActivity appears on the screen (see figure x). To prevent users from having to login each time they opened the app, the “SaveSharedPreferences” class was created. This class is used to determine whether the user is logged in or not. If the user is logged in, they will be redirected to the homepage. Otherwise, they will remain on the login page.

***4.4.3 Hashing passwords***

For security, passwords were hashed using an algorithm developed by (cite). Storing passwords in plain text is not good practice, and is a severe security risk. Subsequently, the password entered by the user when they sign up is hashed into a String which is then stored in the database. Then, when the user wants to login using their password, the password is hashed once more and if this matches the hash that exists in the database, the user will be allowed to enter.

***4.5 Unity Game***

As previously stated, a core concern with dieting apps is that they lack the ability to keep the users engaged over long periods of time. Subsequently, this app has utilised a pre-existing free-throw game to thereby increase engagement and encourage continued use.



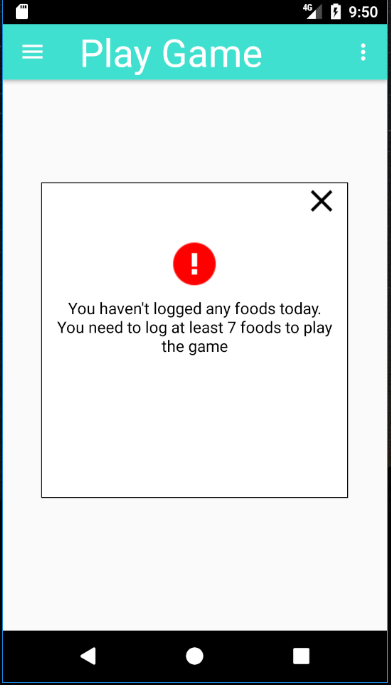
***Figure x****: Screenshot of Navigation Drawer in App*

The Unity game was created by Yanez (2014) and is a simple, basketball, free-throw game. The user is given a basket and is required to press the screen when the arrow is in the green to try and score points. The game was edited so that the number of shots available to the user could be changed and the difficulty increased.

***4.5.1 Problems with the game***

The game was built in Unity so that it could be embedded into the Android app. The Unity classes were added as dependencies to the Android project and the “build.gradle” script was updated to allow the Unity game to run.

Initially, the purpose of embedding the app was to allow the user to use the points they’d obtained as the number of throws they have within the game. The issue with this is that by exporting the game to Android, it becomes embedded, and communication between the two cannot therefore exist (cite). A way to work round this would have been to build the game inside Android, rather than embedding it form Unity.



***Figure x****: Screenshot of Navigation Drawer in App*

However, the look and functionality of the game wasn’t pivotal to the success of this project, so the time spent doing this would have been time wasted. Instead, the game is used as a means of entertainment for the users of the app, in which they can only play if they have logged foods on that day. Consequently, the concept of using the game as an engagement mechanism is still relevant.

***4.5.2 The Game inside the App***

As previously mentioned, the game is embedded into the Android app. When the user clicks “Play Sugar Throw” on the home page or navigates to Sugar Throw from the Navigation Drawer, the app will determine whether the user is allowed to play the game by looking at the number of foods they have logged for that day. If they have logged fewer than 7 foods, they will be prompted with a message (seen in figure x).

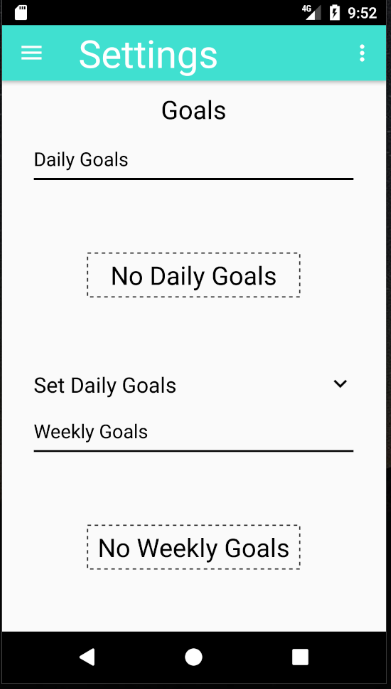
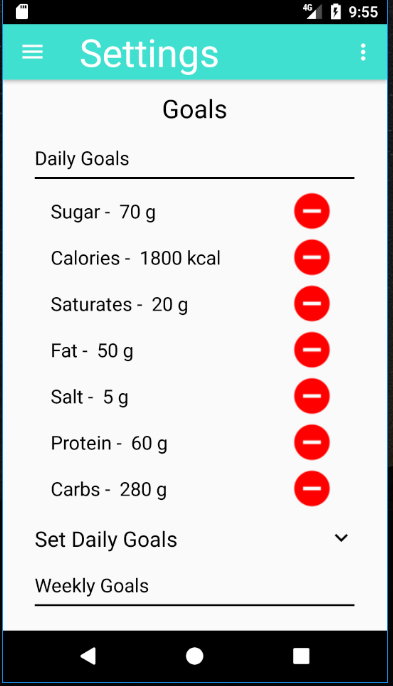
If they have in fact logged enough foods for that day, they will be allowed to play the game. The game will load and the user is given 10 shots. At any time, the user can press the cross in the top right corner to exit the game, or use the navigation drawer to pause and leave the game screen. Pressing the close button results in the application being killed. Therefore, the Unity game needed to be made a separate process from the app itself. By doing this, when the game is “killed” (exited via the cross), the entire application doesn’t close as well, just the game activity.

***4.6. Gamification Elements***

To reiterate, gamification is… (cite). The gamification elements within this app correspond to the ones discovered in the research review: goals; feedback; and self-monitoring. This subsection highlights how these elements are incorporated into the app.

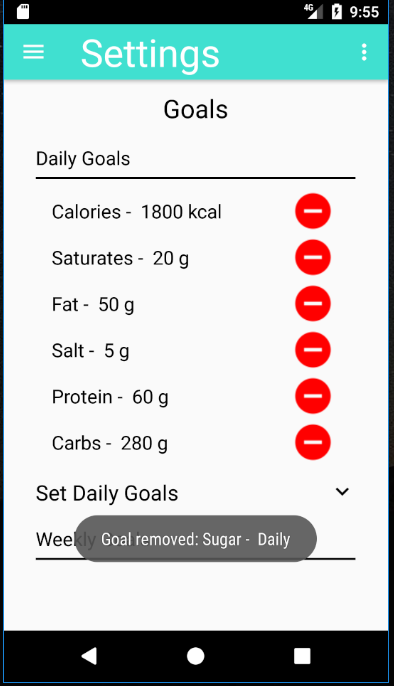
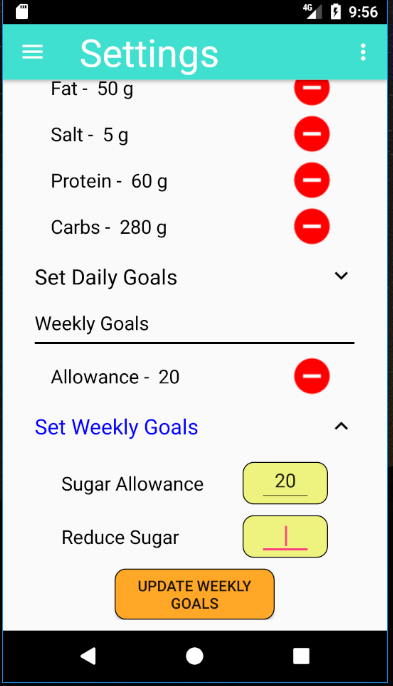
***4.6.1 Goals***

The ability to set goals is important to a gamified app. Therefore, this app contains its own activity that sits within Settings (Settings > Goals). In the activity, the user is given the option to set daily and weekly goals. The daily goals refer to the intake of each food group (i.e. sugar, calories, etc.) a person can have each day. The weekly goals refer to the intake of sugar and the percentage reduction of sugar they wish to have over a week.



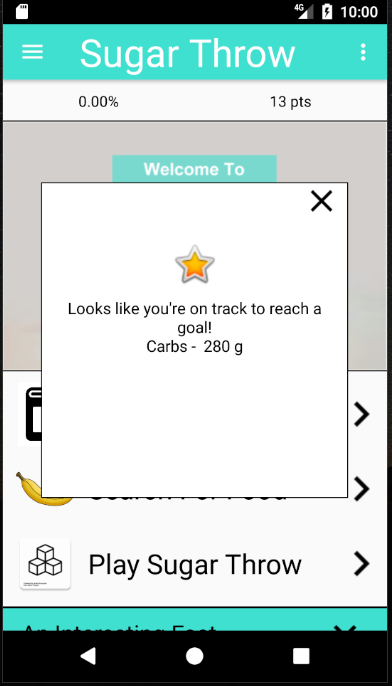
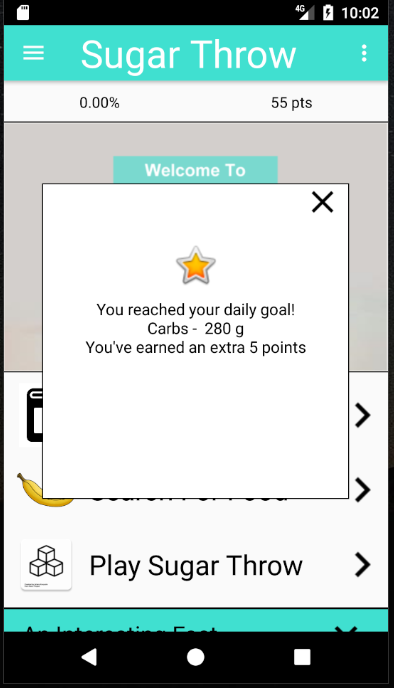
***Figure x****: Screenshot of Navigation Drawer in App*

When the user adds a goal, it appears within their “Goal Diary”. If they have no goals in their diaries, then “No Daily Goals” or “No Weekly Goals” is displayed. These goals can also be removed by clicking the “minus” button.



***Figure x****: Screenshot of Navigation Drawer in App*

The purpose of goals is to give the user a target to reach. By reaching this target, they are rewarded with 5 points, and if they exceed their goal / don’t achieve their goal then they will be deducted 5 points. This, in turn, gamifies the concept of goals and encourages the user to improve their habits.

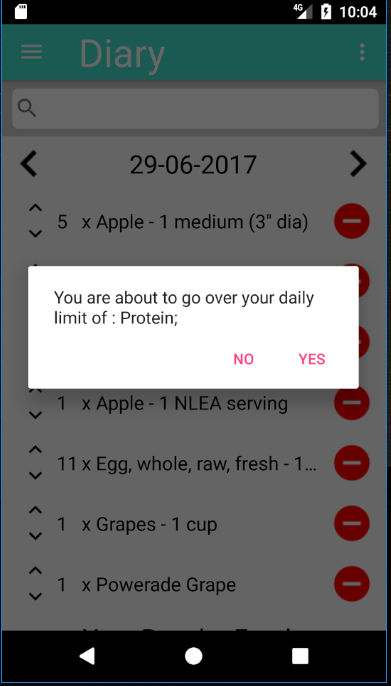
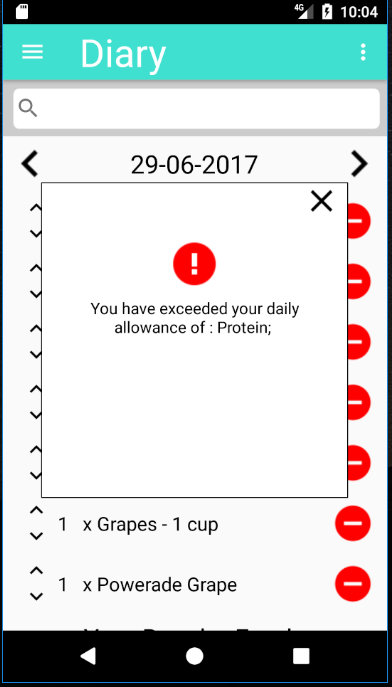
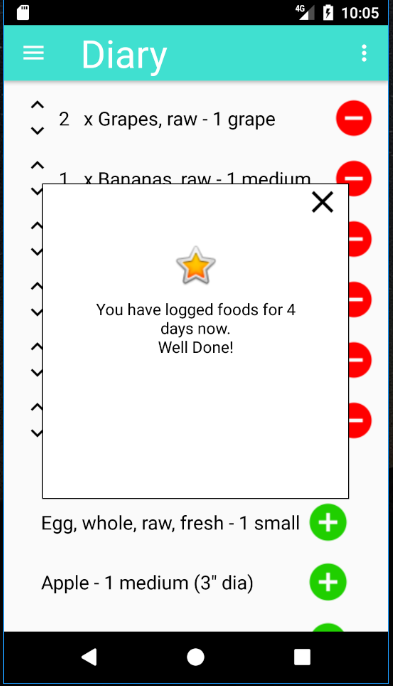


***Figure x****: Screenshot of Navigation Drawer in App*

***4.6.2 Feedback***

Feedback for the app appears in many forms. For instance, feedback popups are used throughout to indicate a number of features:

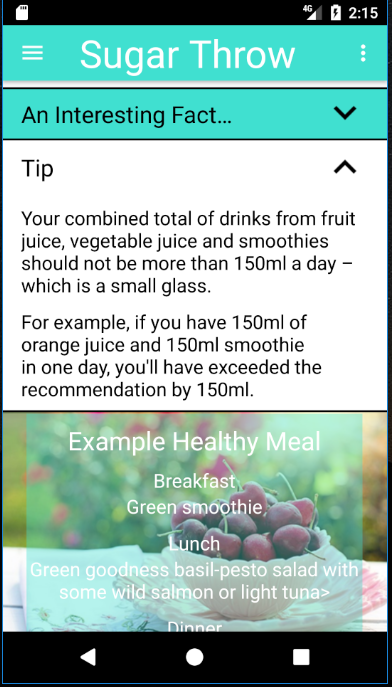
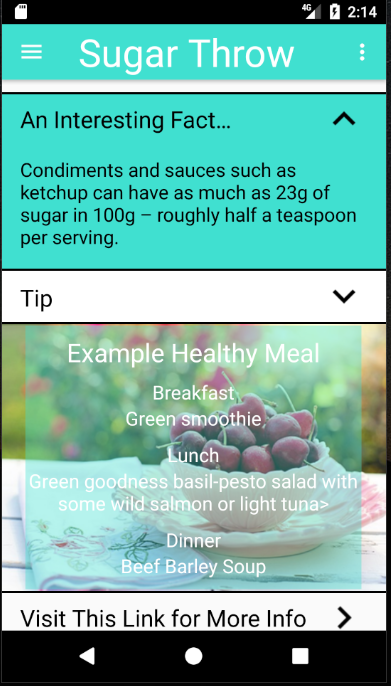
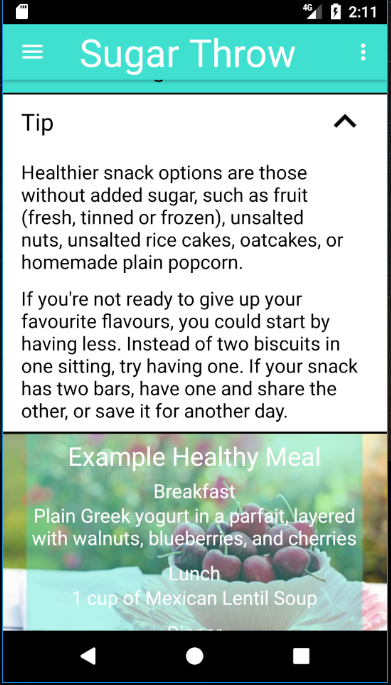
* A user has exceeded a daily amount
* A user has achieved a goal
* A user has failed a goal
* A user has logged foods for multiple days in a row
* A user is eating a lot of a particular food that is unhealthy



***Figure x****: Screenshot of Navigation Drawer in App*

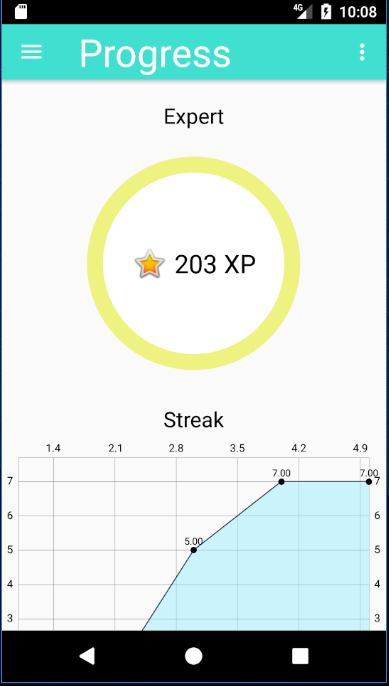
The idea behind the feedback popups is to give the user an understanding of their progress. They are given both negative and positive feedback, denoted by the image that appears atop the popup (see figure x).

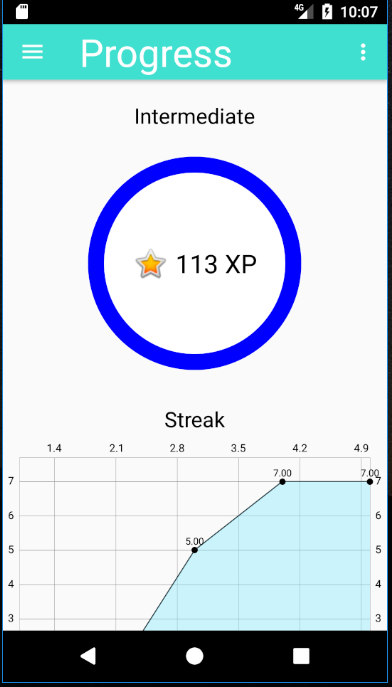
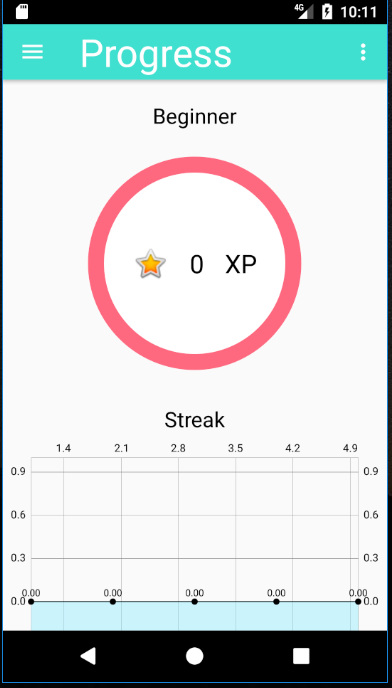
Another example of feedback within the app is the use of dynamic content. This dynamic content (which is controlled by the “DynamicContentHandler” class) changes each day to give the user a different number of health tips, facts, and meals (see figure x).



***Figure x****: Screenshot of Navigation Drawer in App*

Finally, achievements are used as yet another means to give the user personalised feedback. If the user reaches 100 points, they are given the “Intermediate” achievement. If the user reaches 200 points, they are given the “Expert” achievement (see figure x). Achievements condition users to… (cite).





***Figure x****: Screenshot of Navigation Drawer in App*

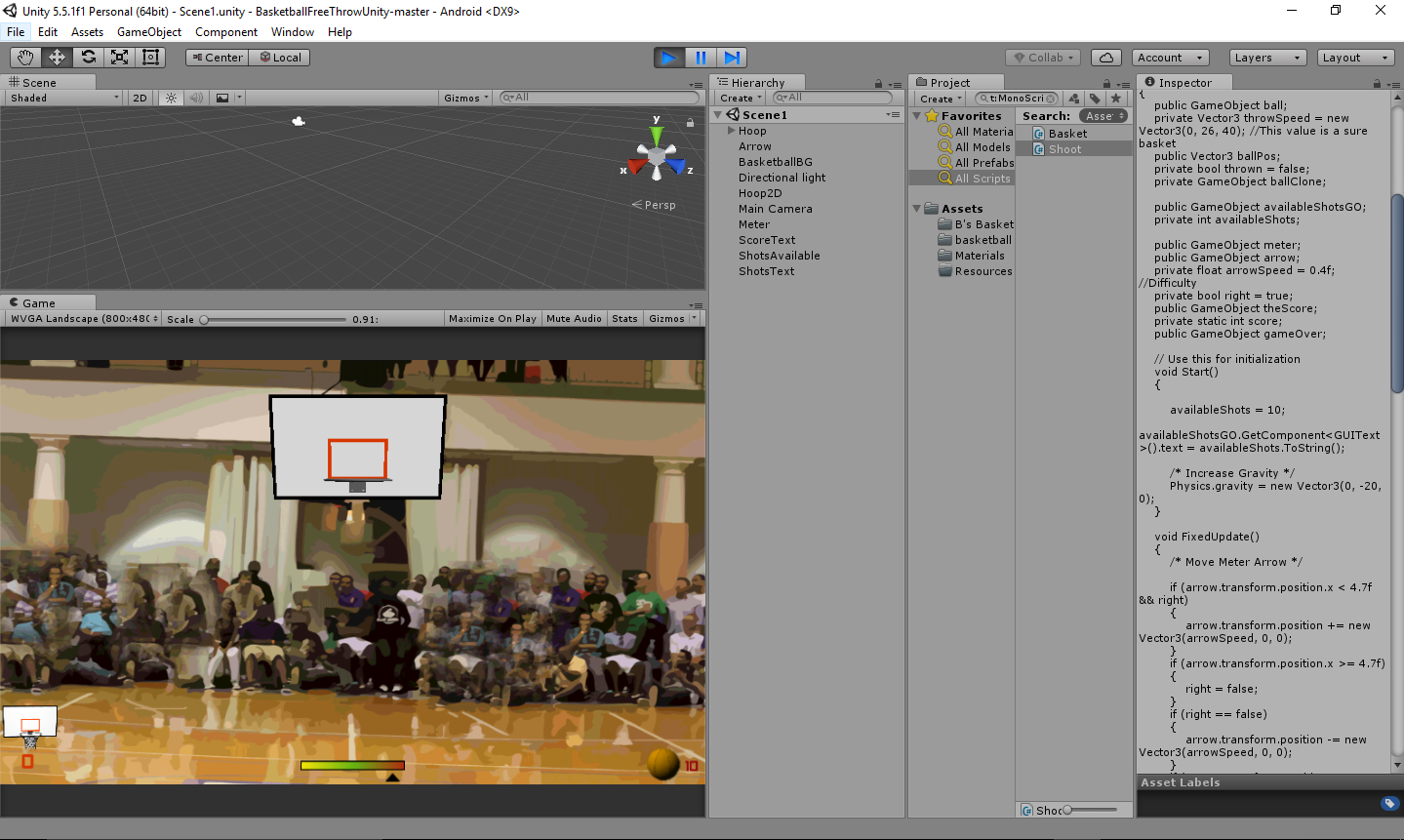
***4.6.3 Self-Monitoring***

Finally, the third gamification technique utilised throughout the app is self-monitoring. Self-monitoring is engrossed in the app; the user is required to monitor their own diets, enter their own food intake, etc.

***4.7 User Acceptance Testing***

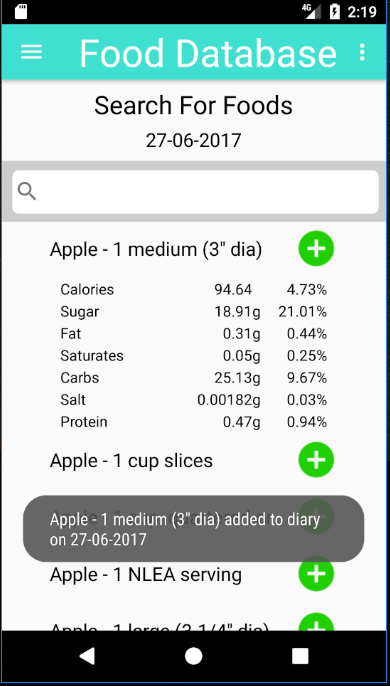
Following the completion of the user interface and most of the app’s functionality, user acceptance testing was performed. User acceptance testing is a phase of software development in which the software is tested in the “real world” by the intended audience (cite).

Two students were selected based on convenience to test the app. As this was simply an exercise to determine how usable the app was, the interviews were very informal. Notably, each student was asked to play around with the app for 15 minutes; navigating around the app, adding foods they’d eaten, playing the free-throw game, etc.



***Figure x****: Screenshot of Navigation Drawer in App*

From these user tests, a number of flaws in the app were noticed. Most noticeably, there was issues when it came to adding / removing foods, changing dates, updating goals, and anything else involving pressing a button. The users had difficulty knowing whether they had in fact pressed these buttons. Considerably, this feedback was taken on board and the app went through a subsequent stage of development. In this stage, more “Toast” messages were implemented to give the user an idea of what they were doing, i.e. when a food was being inserted / removed (as shown in figure x).

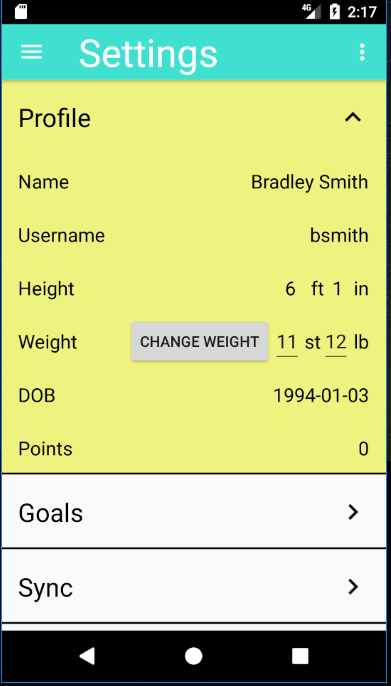
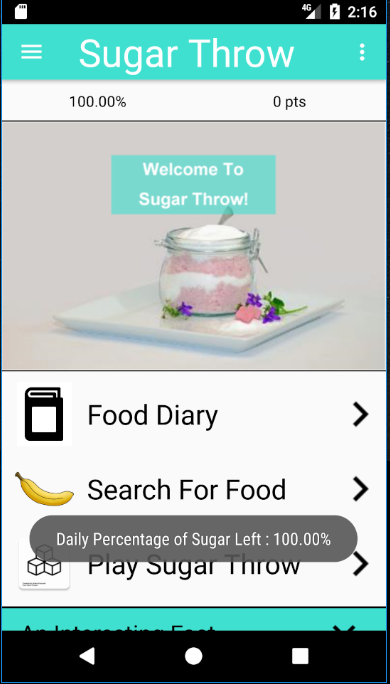
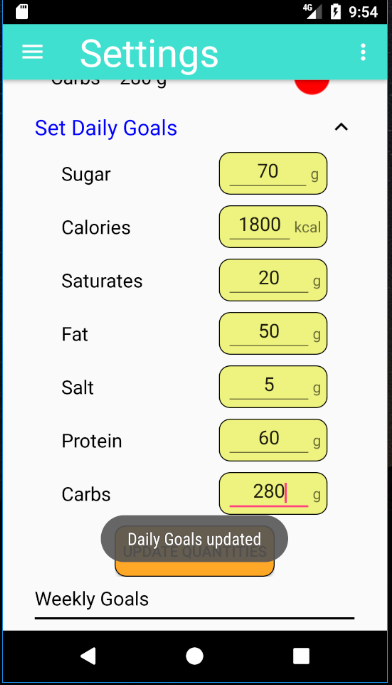


***Figure x****: Screenshot of Navigation Drawer in App*

Table x identifies further feedback given by users and how the app was subsequently improved as a result.

**Table x:** Feedback

|  |  |
| --- | --- |
| **Feedback** | **How app was improved** |
| “Display the total of each food group as well as the percentage in Progress” |  |
| “Needed to be made clear what the [HUD] was displaying” | Similarly, these “Toast” messages were used to make the HUD display more click – the user would now be able to click on the items in the HUD and given more information on what the numbers represented. |
| “Be able to change your weight in Profile” |  |



***Figure x****: Screenshot of Navigation Drawer in App*

In addition, a significant amount of time was set aside to improve the look and feel of the app. Much of the feedback from the user acceptance testing was that the design didn’t look very appealing. If the interface is not appealing, it is not engaging. This project aims to improve attitudes towards healthy eating by making an *engaging*, gamified application. Therefore, a lot of time was put into this exercise as user engagement was a key determinant of this project.

***4.8 Regression Testing***

Following the user acceptance testing, some regression tests were completed to determine whether all the functionality of the app worked properly. Regression Testing is … (cite).

Some notable bugs found during regression tests are included in table x.

**Table x**: Bugs discovered during Regression Testing

|  |  |
| --- | --- |
| **Bug** | **How the bug was fixed** |
| Search not working | The issue here was that the hash map titled “searchKey”, which was used to remove duplicates in the search entries wasn’t cleared when the user made another search. The searches would then be deleted and the user would not be able to see any results. Clearing the table subsequently solved this problem. |
| Sync not working | The Sync button that exists in the SyncActivity was meant to show a progress bar when clicked. This worked when the server was on, but not when the server was off. This was subsequently fixed by assuring that the progress bar appeared regardless of whether the server was on or off. |
| Back button not updating diary | Whenever the user pressed the back button on their Android device and this resulted them being in the Diary activity, the results in the Diary would not have been updated (if food had been added before the back button was pressed). Subsequently, a method was created in the MainActivity (and therefore inherited by all other activities) which meant that the previous activity was launched when the back button is pressed. To do this, the previous activity (as a string) was added to the Intent object that was passed to each activity. |
| On Track to Reach Goals Popup displaying wrong Goal | The feedback popup which let the user know they were on track to reach a certain goal was displaying the wrong goal, for example, if the goal was Sugar – 90g, it would show Fat – g. This was simply because the order of the Goals in the SQL query did not match the ones in the hash map. |

Once all these bugs were rectified, the app was ready to be deployed so that the user studies could take place.

5. User Studies

This section outlines the user studies that went underway following the completion of the app.

***5.1. Pilot Study***

- Developed a questionnaire and tested the user study on two people

- Split these two into those who made goals and those who didn’t

- Asked them to go away for a day, use the app, get used to its gamification elements, etc., and then got them to come back and share their experiences

- The purpose of doing a pilot study is to eliminate any problems before the actual study

- Questionnaire is seen in appendices

***5.2 User Study***

* Lowered the SDK for the user studies from Marshmallow to Lollipop, as there was some users from the sample who had older phones / OS.
* Link back to objectives:
  + 1. Identify the impact of healthy eating app on reducing sugar consumption within a sample of students over a period of 1 week prior versus 1 week after
  + 2. Identify the impact of goal-setting on producing behavioural change for half the cohort
  + 3. Analyse which gamification technique has the greatest impact on claimed behavioural change
* The process
  + Gather students using a quota sample – email sent to students, colleagues and friends to see who would be available. Had 15 responses, which were cut down to 10 based on judgement (5 male, 5 female)
  + Split students into two groups of 5 – each respondent was given a random number. 5 random numbers were then pulled out of a hat, and these were given the task to set goals, whereas the other 5 respondents would not set goals (this was split 4 male, 1 female).
  + There and then, got the students to use the app, recording the foods they ate yesterday. For those who had difficult remembering, they were asked to simply provide an overview of the foods they believe they had eaten. The app then recorded the amount of sugar that user had that day
  + Users went away for a week
  + Gather them back to analyse the results – gave them a questionnaire to fill out

***5.3 Gathering data from the User study***

* + For the first objective – went through each person 1 by 1 to see how much sugar they’d had that week. Below is an example graph showing the change in sugar for one of the respondents
  + Recorded the results for each day and determined the percentage reduction in sugar they had recorded
  + If the percentage reduction for all respondents proved to be significant – then it was inferred that the app did encourage them to reduce sugar consumption
  + For the second objective – looked at the difference in sugar levels between those who set goals and those who didn’t
  + For the last objective – used the questionnaire and used the average for the last questions as the dependent variable and then used the goal-setting, feedback, and self-monitoring questions as dependent variables. Ran a factor analysis followed by a regression analysis to determine if any of these were significant

**Table x:** the amount of sugar the students had prior to using the app

|  |  |
| --- | --- |
| **Student** | **Amount of Sugar** |
| *Student 1* |  |
| *Student 2* |  |
| *Student 3* |  |
| *Student 4* |  |
| *Student 5* |  |
| *Student 6* |  |
| *Student 7* |  |
| *Student 8* |  |
| *Student 9* |  |
| *Student 10* |  |

6. Extension Task – Machine Learning

4.5 Results of Machine Learning Tests

7. Conclusions