University of Brighton

Advanced Mobile Application Development
Reflective Report

By Jake Ward

Student Number: 20808653

University of Brighton

BSc (Hons) Computer Science

CI660 Advanced Mobile Application Development

20/05/2023

Abstract

This reflective report documents the full software development lifecycle of the CI660 project. The report discusses the UI design, advanced technical features, technical challenges faced, and how they were overcome. Challenges concerned the deprecation of features/functions, preventing the possibility of crashes, finding the best methods for communication with a database and many more. The project involved creating a fitness-based mobile application from scratch with three advanced features including sensor monitoring, database storage, and advanced UI elements. The application is designed to be user-friendly and simplistic, targeting an audience of individuals concerned with fitness. This project provided valuable skills in advanced mobile application development and project management, which are evaluated in the report. The report details evidence of the resulting fully functional application that met all requirements through thorough testing and documentation. The problems, solutions, documentation, evaluation, and other software development lifecycle sections are presented in detail below in chronological order.

Table of Contents

1. Introduction

The purpose of this reflective report is to document the project management, research, design, implementation, testing and evaluation of the CI660 project. Within is stated/discussed the UI design, advanced technical features, technical challenges and how they were overcome and reflection on the entire agile development lifecycle to display learning in the area of advanced mobile application development. This document is a companion piece to the concept document previously submitted, thus research from that document is not repeated. All topics are elaborated on in the following document.

This project concerns the development of a mobile application from scratch. The application holds three advanced features in the form of sensor monitoring, database storage with SQLite and advanced UI elements such as fragments and material design usage. The application is a fitness-based application which tracks footsteps, allows users to save their current step count in a history tab, presents a visual representation of progress towards a step goal, allows users to toggle dark mode and set their step goal in settings. The design of the application aims to be simplistic and user-friendly to set itself apart from similar applications and the target audience varies but is aimed at individuals concerned with their fitness. The application itself was created through Android Studio with Java as the main language and the entire creation is documented in this document through screenshots/figures and explanatory notes.

This project has taught me valuable skills in the area of advanced mobile application development as well as requiring skills regarding project management. I am to express these things through the following documentation and evaluation.

2. Project Management & Design

2.1 GitHub

GitHub was an all-important project management tool for the application's development. Primarily, it was essential in providing a centralized and secure repository that would be used as a backup to prevent loss in progress. Its version control would also allow simple rollbacks to previous versions to analyze old code or remedy issues introduced with new commits. The versions stored on GitHub also serve as evidence of the progress made on the project, corroborating the details in the implementation section in terms of dates and commits. It would also generate stats to monitor project development and would clearly display the user who committed changes, serving as evidence that the project was an individual project. GitHub was not the only source of the code as it was also stored on multiple devices to lower the change of losing progress significantly.

Hypothetically, if this had been a group project, GitHub's usage would also have the advantage of centralization for team members to collaborate simultaneously without commits conflicting. GitHub's issue-tracking and project management tools would also aid in managing tasks and labelling issues. Finally, GitHub would allow myself as the project manager to easily review code changes before merging them into the main branch, ensuring code quality and preventing errors.

2.2 Requirements

Requirements are used to measure a project's success and divide the agile development lifecycle into parts. In this section I've outlined the projects requirements listed from top to bottom by priority. These requirements were selected to display understanding, learning and implementation of advanced features. Development of the application would be considered as finished after the project's application met these criteria.

Must Have

1. Navigation Menu (Advanced UI)
2. Fragments: Home, History and Settings (Advanced UI)
3. Request Access to Activity Sensors
4. Step Counter (Sensors)

Should Have

5. Snackbar notification on denied access (Advanced UI)
6. Progress Bar
7. Step Goal (Shared Preferences)

Could Have

8. Dark Theme
9. History (SQLite Storage)

These requirements detailed the implementation of 3 advanced features pertaining to Advanced UI, Sensors and Storage. These requirements stayed true to the original plans for the application except for the storage choice. While firebase was the original planned method for including a history feature SQLite was instead selected after further research and consideration due to it absence of a need for internet connection to function and local storage eliminating the needs for authentication among other reasons.

Requirements were also set up to be used in a test-requirement matrix to clearly outline testing of features in a concise manner and display progress as well as measure success. This would be carried out via black and white box testing which would then be documented in an excel document.

The requirement's assigned priorities were justified as such;

- The navigation menu would need to be set up first otherwise the fragments couldn't be navigated to/between
- The fragments would need to be created second so that they could host the functionality of the app while segregating them appropriately
- The sensors wouldn't be able to be accessed with explicit permission after API 29 and so would have to be requested on the home fragment before the main feature could be implemented
- The step counter held the core functionality and so was a high priority requirement but required the first 3 requirements to be met to function
- The Snackbar notification was an important feature that needed to be implemented early in development as it ensured the functionality of the sensor by redirecting to permissions in settings
- The progress bar was less essential but displayed a visual indication of progress, improving the UX, thus making it higher in priority than some other features
- The step-goal wouldn't be integral to the application and so was lower in priority but would improve quality of life and add extra functionality
- Dark mode would be an easy to implement feature improving the UX but not integral
- Finally, history was placed as lowest in priority as it was an unessential yet complex feature meaning that it could hold up development and thus needed to be implemented last to ensure a working product was created

Completion of all requirements would mean the application successfully solved the project's problem as it would simplistically and comprehensively allow fitness tracking accurately and with an unobtrusive number of extra features to improve the UX. The three advanced features inclusions would ensure the application met the success criteria.

2.3 Deliverables

The deliverables for this project include;

- Concept Document - Previously submitted concept document in a PDF format, submitted on 3/3/23
- Application APK File - The APK file for the application, runs on Android 9 SDK 28 and higher
- Reflective Report - A single PDF containing the final report
- Project Folder - The complete project folder of my application. This includes my code, resources and everything else needed to inspect, build and run my application.

2.4 Agile Methodology

The Agile methodology was chosen for the development of the application, although development was rapid with sprints being limited due to time constraints. The Agile methodology is defined by its iterative/incremental approach to software development and the ability to respond to changing requirements throughout the development process. Agile development also places a strong emphasis

on features only applicable to working with a client or development team such as delivering value to the customer or promoting transparency and communication within the team, however, these are not applicable to this project due to the lack of a hiring custom dictating design decisions or a team of developers. Finally, the method requires incorporating frequent testing to ensure the quality and usability of the software being developed.

It was specifically chosen for this project due to many reasons, but the most significant reason was due to Agile development flexibility. Unlike other methodologies such as waterfall, the Agile development lifecycle allows for requirements and development to change throughout the entire process. This prevents developers from being restricted to initial designs, prompting the best solutions to problems to be solved immediately without having to adhere to strict initial designs. This would be especially helpful in an individual project as any straying from initial designs for the betterment of the project wouldn't require other parties to be updated, so experimentation with development would take less time and development could be more rapid.

Another prominent feature of the methodology which was appealing was the inclusion of continuous delivery. Continuous delivery means that publishing a version of the product takes place after many if not all incremental implementations. Meaning that at any given time there is an up-to-date product to share with the theoretical customer or in the worst-case scenario there is an incomplete version of the product to submit if the deadline approaches. As development would only include a one-man-team there would be no conflicts with these incremental changes.

The regular testing cycles advertised as a key feature of Agile was an equally appealing feature. This is because it ensures the best quality product has been developed by exploring all execution paths and fixing any potentially erroneous areas. These regressive tests can be a drain on development time but their insurance of quality made the choice of Agile a worthwhile investment of time. It can also be noted that the Agile development method allows developers to jump between stages of development whenever necessary, so rather than waiting for the testing phase of development, all tests can be run immediately after implementations and any errors found via testing can immediately be fixed.

2.5 Risk Analysis

Risk	Cost	Effect	Mitigation
Data loss	High	Significant impact on project progress	Regular backups on multiple devices, cloud storage and GitHub
Compatibility issues	High	Inability to function on certain devices	Extensive testing on various devices and OS versions (SDK 28 and Higher)
Security vulnerabilities	Medium	Breach of sensitive user data	Implementation of secure authentication and encryption
Performance issues	Medium	Negative user experience	Profiling and optimizing code, testing on low-end devices

User interface problems	Low	Decreased user satisfaction	Extensive testing and feedback from users
App store rejection	Low	Inability to publish app on app store	Comply with app store guidelines and policies (Theoretically, as the application will realistically be added to the Play Store)
Developer Illness or Injury	High - Low	Loss of development time	Maintain healthy lifestyle and vaccinations/boosters
Fragment compatibility	Low	App may not function on certain devices	Testing on different devices and OS versions
Snackbar notifications	Low	Users may not notice important messages	Use clear and concise messaging, consider adding sound/vibration
Device orientation issues	Low	App may not look/perform well in landscape or at different resolutions	Testing on different devices and orientations, design with constraints to be suitable for all resolutions
Step counter accuracy	Medium	Inaccurate tracking of steps taken	Testing on multiple devices regularly, chose a sensor that will track steps the most accurately
Dark theme compatibility	Low	App may not display correctly in dark mode, damaging the UX	Testing on different devices and OS versions, Test in the scenarios; Device is in low power mode, Dark mode is enabled in settings, App is launched after dark mode was enabled in settings
Shared preferences issues	Low	Settings/variables may not persist	Extensive testing and error handling for shared preferences
Database errors	Medium	Data loss or corruption in the database	Regular backups and testing, proper error handling and recovery, don’t allow nulls so

			corrupted/erroneous data can be identified

Table 1: Risk Analysis Table

2.6 Use Case Table

Use Case Name	Actors	Preconditions	Postconditions	Main Flow	Alternare Flows	Priority
Measure Step Count	User	None	All relevant information which must persist will be saved under shared preferences.	The on-screen progress bar and step count will be updated every time a change is measured in the sensor.	None	High
Enable Dark Mode	User	None	The dark mode preference in shared preferences will be updated to reflect the switch.	When the user clicks on settings and toggles the dark mode switch, the application will change to dark mode, which will persist over multiple launches.	The application is launched which the device is in low power mode	Low
Set Step Goal	User	None	The step goal preference in shared preferences will be updated to reflect the input value.	When the user clicks on settings and the step goal preference, the application will set the input value as the step goal and the progress bar will reflect this.	None	Medium
Log History	User	None	The date and current days step count is saved to the database	When the user clicks on history and then the log step count button it is added to the history with the accompanying date	None	Low
Clear History	User	None	The table in the database is wiped	When the user clicks on history and then on the wipe history button the history is wiped	None	Low

Table 2: Use Case Table

3. Further Research

Research was already carried out and noted in the concept document for the same project and so it will not be repeated to avoid self-plagiarism. However, extended research was carried out throughout development to ensure best practices as well as simply learning how to carry out tasks. Below are the detailed accounts of further research that extended further than simple bug fixes to syntax.

3.1 Device Capabilities

For the projects testing to be successful, access to a physical android device was required. As well as this it would need to have a step counter sensor, which would only be available if the device had an accelerometer and would need to run a modern version of android to ensure it would function on

modern devices and advanced features could be made use of. While already in possession of several android devices, none of them met the specification as most either lacked sensors or couldn't install the latest versions of android.

Eventually a Moto G(8) Power Lite phone was purchased for testing. This was done after heavy research into the devices capabilities. The Moto G(8) was not listed as a device capable of installing Android 12 directly from Motorola. However, it would still be able to run Android 11 which was close enough to utilize the features I would be using.

One of the reasons a physical phone was required was because of the requirement for tracking steps. An emulated device on PC wouldn't be able to test a feature which relied on sensors, meaning that the application needed to be installed on a physical device with the required sensors. The Moto G(8) was researched to find that it did support an accelerometer as well as the software-built sensor "step counter", making it an ideal purchase for testing. To double-check that the advertised sensors were available on the device, the application "CPU X" by Adalve Technologies Pvt Ltd was utilized to view details about the required sensors. Everything was functional and the application would theoretically work on the device after developer mode was activated, so the device would be the main device used for testing. Other devices would be used for testing when the opportunity arose to ensure compatibility and scalability.

Miscellaneous research into the device includes whether its version of Android could support threads, Firebase or SQLite, fragments, snackbars and more. All of which were confirmed to be true without any hindrance on the subject matters.

3.2 Reading Sensor Values on Launch

The conventional method for reading the values of sensors seemed inconvenient at times, but it was discovered through further research that there isn't a alternative method currently. The issue encountered involved inconvenience when attempting to read the step counter sensors value, as the conventional method involved reading the value inside an onSensorChanged event. It was discovered that a sensors value can only be obtained after a SensorManager has been registered and the onSensorChanged method has been implemented to receive the sensor events. The value could be extracted but not read directly from the sensor without using this process. This would mean for scenarios like displaying the step count on launch before the sensor updated, the last known value would need to have been stored elsewhere as a variable (thus a solution was devised by utilizing shared preferences).

3.3 Circular Progress Bar

The project available at https://github.com/lopspower/CircularProgressBar by Lopez Mikhael was one of the best methods to implement a circular progress bar. While this type of progress bar is available through material design, the material design circular progress bar is designed and better suited to a loading icon. Lopez Mikhael's circular progress bar is more user friendly in terms of both code and documentation, supports a large amount of customizability and is a library that can be easily implemented via Gradle. As material design was already to be used with snackbar's there was no pressure to display its usage with this element as well. I was mostly drawn to the presentation such as the easy animation as well as customization of each element, including the maximum progress, colours and scales. The equivalent in material design would be a determinate progress indicator.

CircularProgressBar also supports indeterminate mode to function as a loading symbol, although it wouldn't be used in that way for this project.

3.4 Settings Preference Screen

Although the settings screen could have been implemented simplistically through XML design using scroll views, text, input, switches and more. I decided that it would be better to explore the more conventional approach used for Android application development so as to learn and display the best coding practices. This happened to be the user-friendly preference screen and its accompanying setup.

While the specifics of the setup are detailed in the implementation section of the report under "4.8 Preference Screen/ Settings", I will detail the appealing functionality here. The preference screen will automatically save the listed preferences under the shared preferences of the application. This will always be remembered by the preference screen, thus all its set values will persist over multiple destructions and relaunches of the application. Alterations of these preferences can be overridden to add functionality to the alteration, such as switching themes. It is highly customizable like the majority of elements implemented through XML and comes within an automatic scroll view, so the number of settings will never prevent any of the preferences from being inaccessible. Finally, due to its conventional usage for settings, the visual indicators will be the same as the settings of other applications, immediately making their function recognizable subconsciously.

3.5 SQLite Database Display

While the database functionality was developed in the conventional way for SQLite (being that of creating a database helper and calling it to execute queries) the display of the table's contents was not as simple an issue. The details of the SQLite setup can be found in the section of the report labelled "4.9 SQLite Storage".

As a result of research, the conventional method for displaying the contents of a table seemed to be via use of a recycler view. However, I found this method to be needlessly complex due to it requiring a lot of unnecessary code and additional dependencies, as well as the fact that after following the documentation, the method was non-functional. Therefore, instead of wasting more time attempting to locate the issue, I decided to use a simpler approach and opted to populate a list view instead. This allowed me to display the contents of the SQLite table in a straightforward and efficient manner through array lists, without compromising functionality. I found that a cursor could be used to retrieve all contents of the table and populate the array list which in turn would populate the list view. This proved to be a much more elegant solution.

4. Implementation

In this section of the report, I will be detailing how the implementation of all features were carried out in chronological order. This displays the progress throughout the project as well as evidence of design decisions, challenges, solutions, regressive testing etc. This is to provide evidence of the entire process of development as well as providing a tutorial style walkthrough on how to build a similar application to display understanding and educate others.

4.1 Setup

First to ensure all progress would be backed up and changes could be rolled back a GitHub repository needed to be made. As such the GitHub account used to hold the repository needed to be linked to the

android studio application. This was done via creation of a personal access token with all appropriate privileges and from there the empty created project could be published to GitHub, automatically creating the repository.

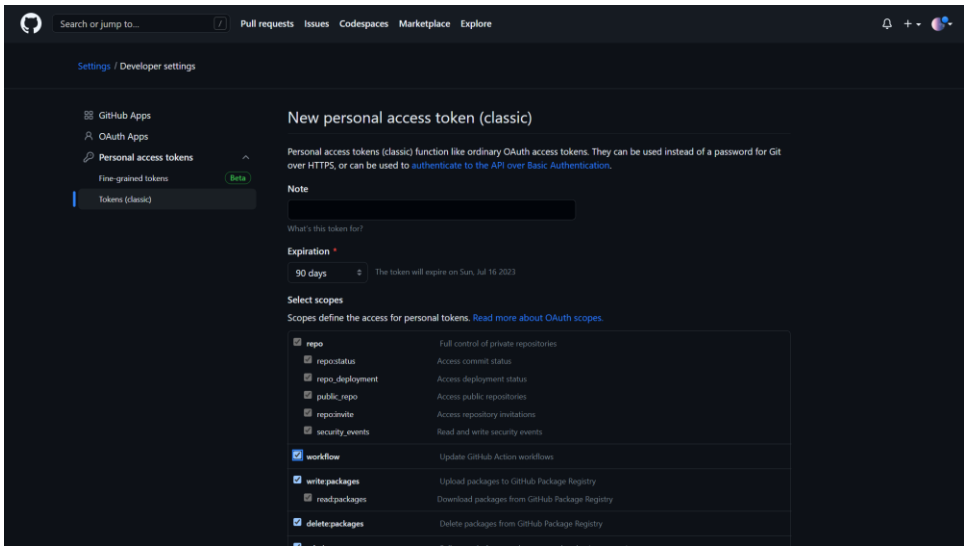


Figure 1: creating a token for access to GitHub from android studio

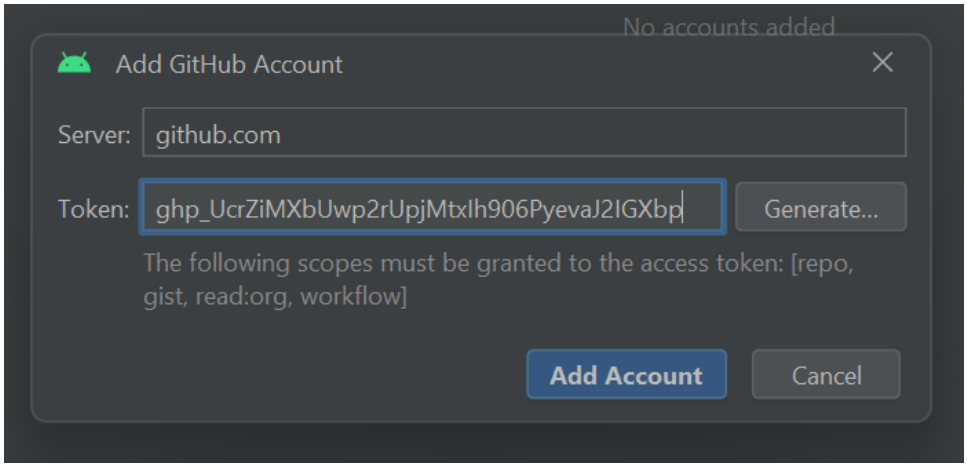


Figure 2: using token to link android studio to GitHub

The project was named after the main functionality of the application for clarity, Java was selected as the language due to familiarity allowing faster development and the minimum SDK was set as 28 so that roughly 81.2% of android devices would be able to run the software while still being able to make use of modern advanced features. The app's minimum SDK was chosen to ensure compatibility with the Moto G8 Power Lite as it was found to not be compatible with Android 12.

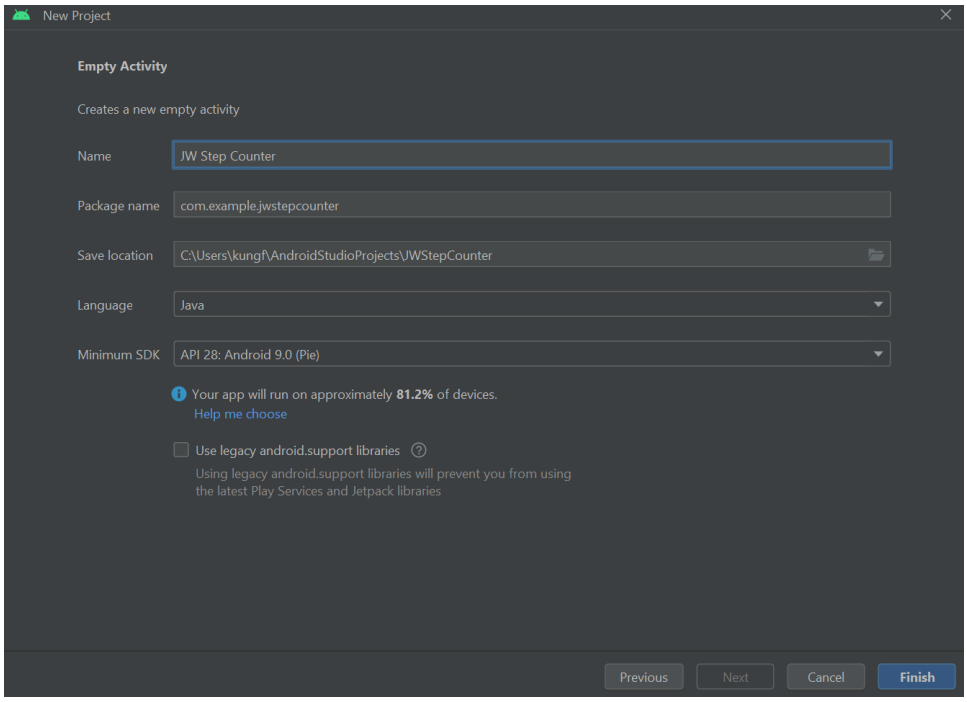


Figure 3: creating a project in android studio

4.2 Bottom Navigation Menu & Fragments

The first feature to be implemented was to be the bottom navigation menu. This had to be set up first as all the functionality was to be held within fragments which would be navigated to via the navigation menu. By designing the application with the use of fragments it would allow smooth transitions between the different sections of the application and reusable efficient components as well as simply being good practice of an advanced UI feature. By designing the app with a bottom navigation menu, a conventional method for navigation ensures users are comfortable and will likely understand how to navigate the app. It also allows all functions of the app to be instantly discoverable from every page as well as inherently holding UX enhancements such as the material design ripple effect making it clear when a click has been made.

I researched and revised the conventional way in which this type of menu could be implemented until I had learned of the correct method. This meant creating the XML for the main activity first, with a constraint layout for responsiveness on both orientations, a frame layout within to hold the fragments and bottom navigation menu below that. While the XML design view showcases the history tab as being highlighted, in execution none of them would be until one was manually set as having been highlighted.

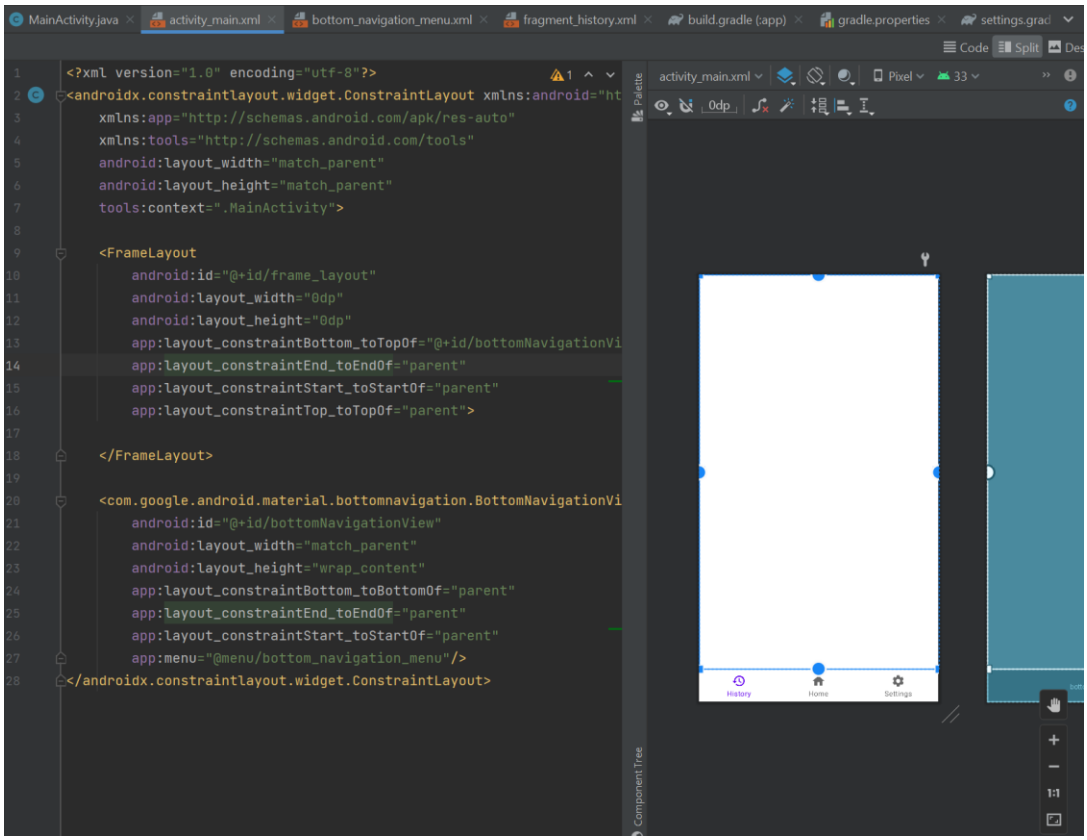


Figure 4: xml layout of main activity with bottom navigation

The next step was to create a new XML (Android Resource File) holding a menu with three items for the corresponding three planned fragments. Each would be given ids to later target in the main activities case statement as well as titles and icons which I would go onto create. I felt it would be best to have the home button placed centrally to visually display importance.

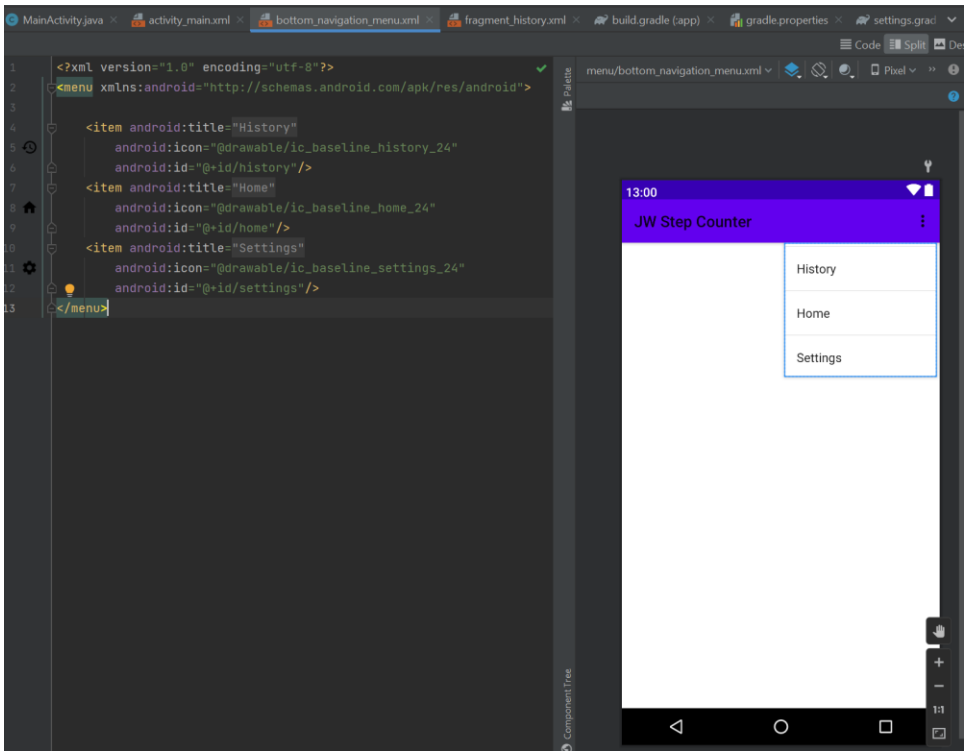


Figure 4: xml for menu element nested in bottom navigation

The icons could easily be created in android studio via clip art and were named in accordance with conventional naming schemes to make their purpose easily understandable. The icons chosen would also be the conventional icons associated with history, home and settings to avoid confusion and hopefully provide an instantaneous understanding of the fragment's purposes. The colours of the icons were set as black as they would automatically be changed by both the app's theme and if the associated fragment had been selected. The dimensions were set as 24dp to fit with the conventions of android studio thus allowing greater functionality and avoidance of unnecessary errors.

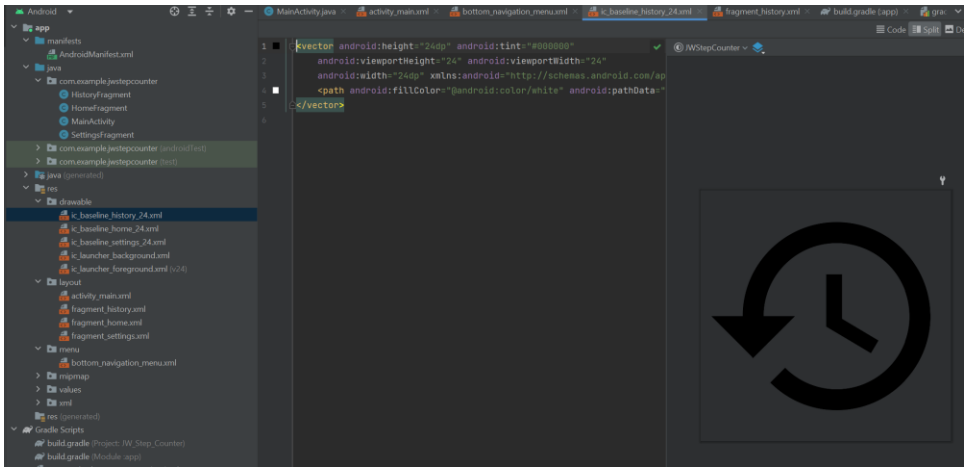


Figure 5: creation of all drawable assets for navigation

The planned fragments then needed to be created which was easily done via android studios automatic fragment creation tool. I created three blank fragments and placed a text view on screen to differentiate the fragments for aid in testing the application.

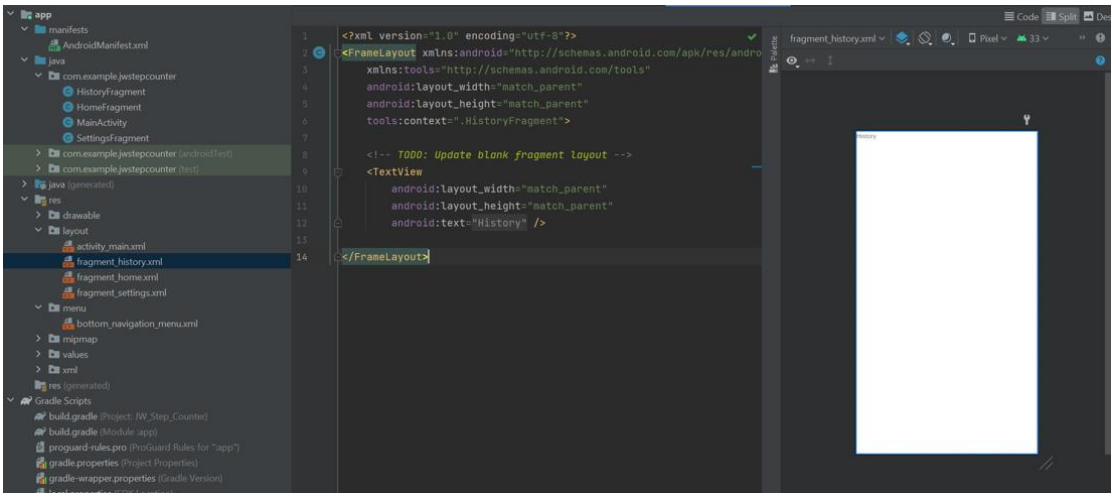


Figure 6: creation of all fragments to be navigated between

Unrelated to the current task, the target SDK was changed from 33 to 31 as I had experienced the usage of 33 being borderline experimental. An example of this was the complete removal of being able to priorities broadcasts without a replacement method advertised. So, to avoid similar issues a more stable version was chosen.

Then the task related to the bottom menu needed to be carried out in the app Gradle file. This involved enabling view binding under build features. This is required because it would simplify the process of accessing/ manipulating the UI elements and replacing the fragments held in the frame layout. It could also be used to manipulate any UI elements such as the bottom navigation menu, allowing for the selected item to be set (although this can be done without the use of view binding) as well as for a listener to be applied to the bottom navigation menu and thus all of its buttons/ items.

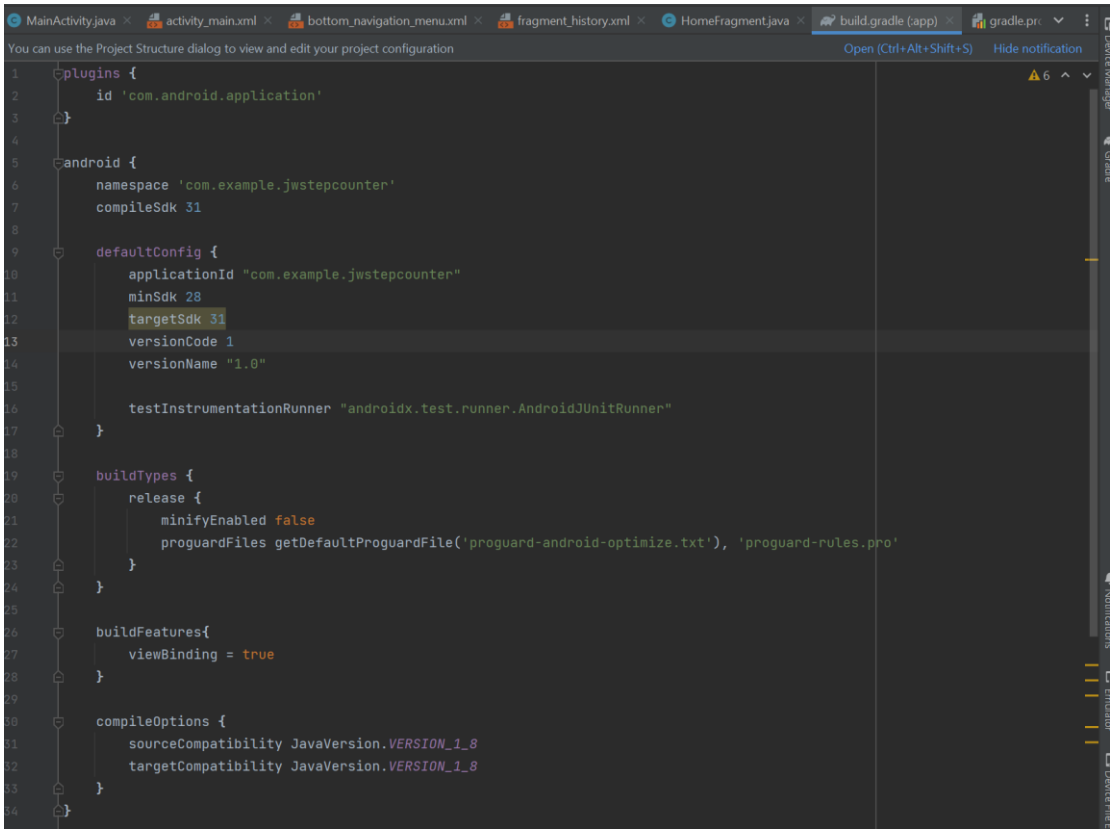


Figure 7: app Gradle file altered to use less experimental SDK and enable view bindings for fragment navigation

View binding was then used in the main activity's onCreate method to set the root view of the main activity as the route view of the binding object so that the replace fragment method can be used to display the specified fragment on screen in replacement of the frame layout. The selected item id is automatically set as the home item and a listener is placed on the bottom navigation menu using binding and a case statement calling the replace fragment method on the fragment in accordance with the button pressed.

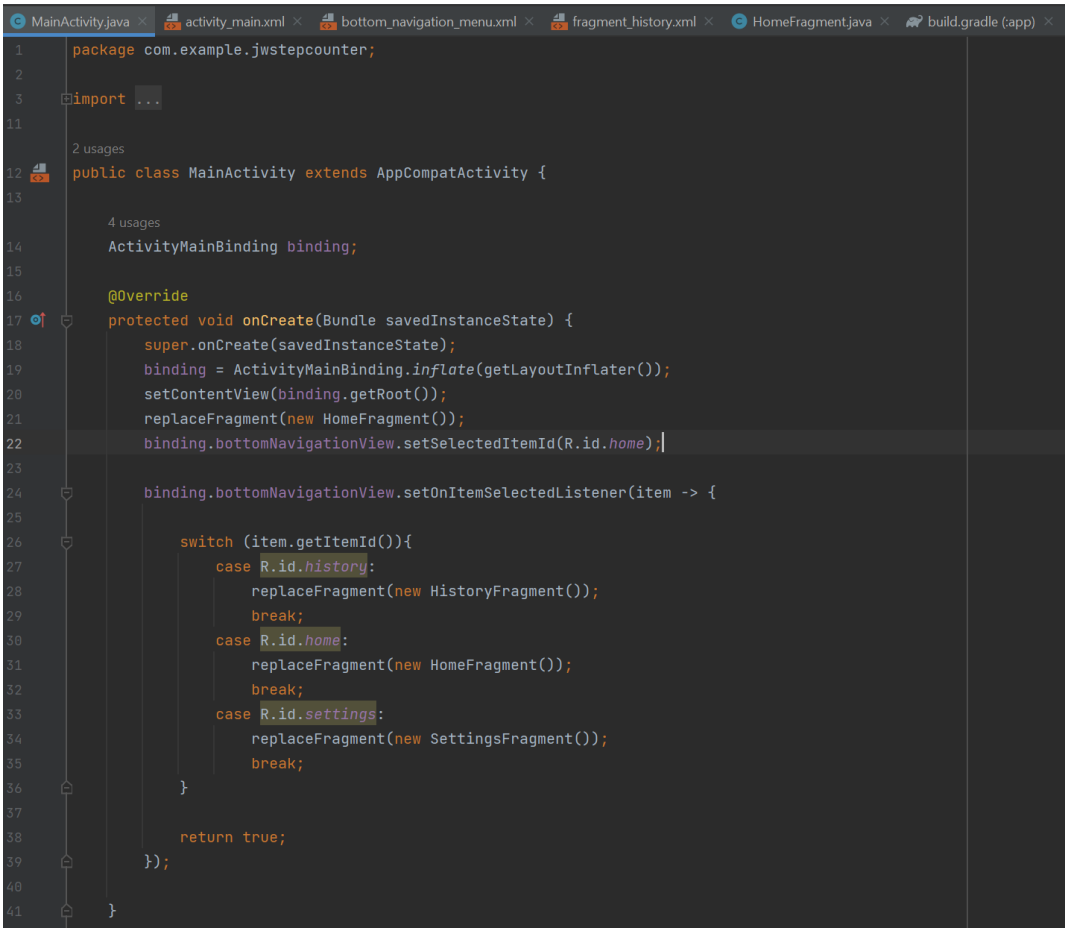


Figure 8: main activity Java to navigate between fragments

The replace fragment method designed functions by calling a fragment manager (responsible for managing all fragments in an activity), creating a transaction object to remove or replace fragments, calling the replace method on the frame layout with the fragment and committing the changes.

```
4 usages
private void replaceFragment(Fragment fragment){
    FragmentManager fragmentManager = getSupportFragmentManager();
    FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
    fragmentTransaction.replace(R.id.frame_layout, fragment);
    fragmentTransaction.commit();
}
```

Figure 8: main activity Java function to set/replace fragments

The application was installed on the factory-reset Motorola android device to find that all fragments were successfully able to be navigated to, the design was scalable over both orientations and the home

item was automatically selected as well as the home fragment automatically being displayed on creation of the application's main activity.

With a functional application complete with one advanced UI feature, enough progress had been made to warrant the upload of the content to a GitHub repository.

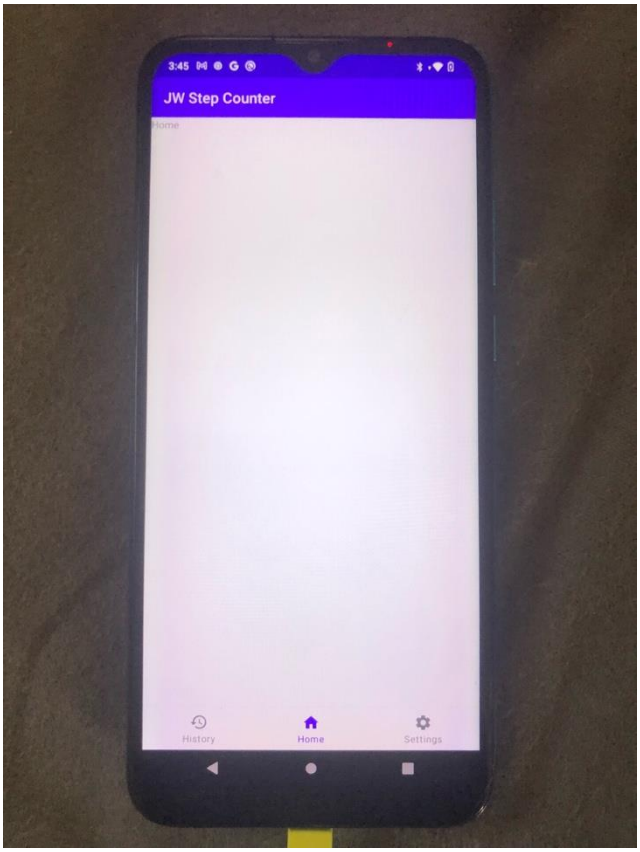


Figure 9: practical successful test of bottom menu navigation with fragments

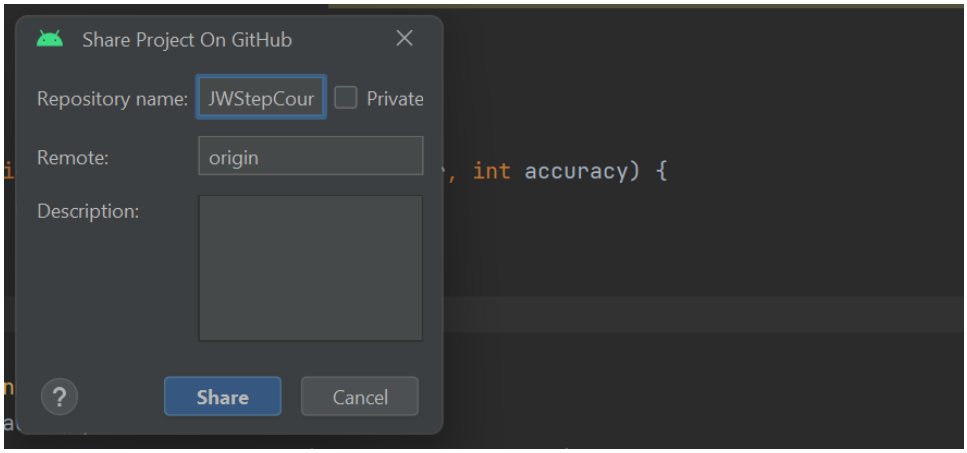


Figure 10: creating GitHub repository and pushing all commits

4.3 Step Counter & Sensor Permissions

The next set of features to implement would be the third and fourth requirements (the step counter and gaining permissions). The required permissions for accessing the activity sensors would need to be got before the step counter feature could be implemented so it was added to the android manifest. However, since API 29 permissions such as these need to be gained at runtime so this wouldn't often be made use of.

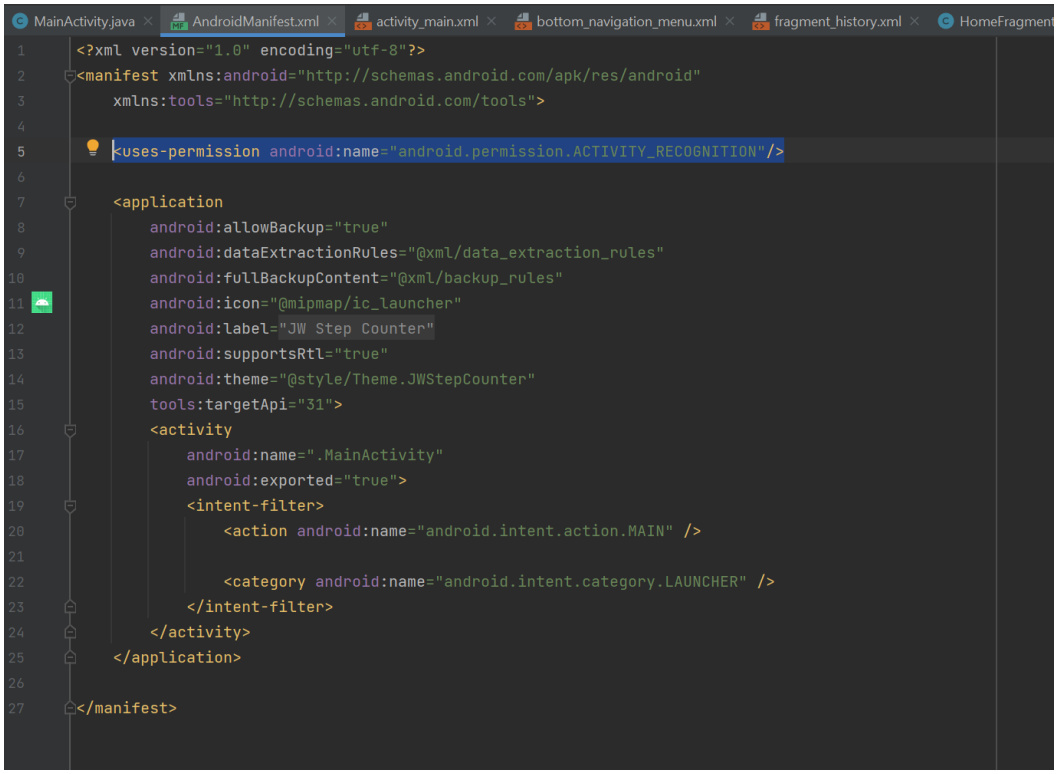


Figure 11: adding required permissions to android manifest for activity sensor access

To set-up the functionality for the step counting feature a sensor manager needed to be initialized as well as a sensor and the text view that would display the results. The blank fragments were also cleaned of all the default but unnecessary code provided leaving only the required empty public constructor. It was at this point that I made a conscious effort to include comments for any potentially confusing pieces of code as the codebase was only going to get more complex from here on and it would help reviewers of the work understand it.

All code pertaining to the step counter would be included within the Java code for the home fragment as that would be what would load first and would be the default location for the step counter. By placing it in the home fragment the step counter could be registered immediately and would be unregistered when using settings or history (because this connotes that the user isn't currently active) as well as being

unregistered when the application is closed, ensuring that battery life is saved by only handling the registered sensor listener when appropriate.

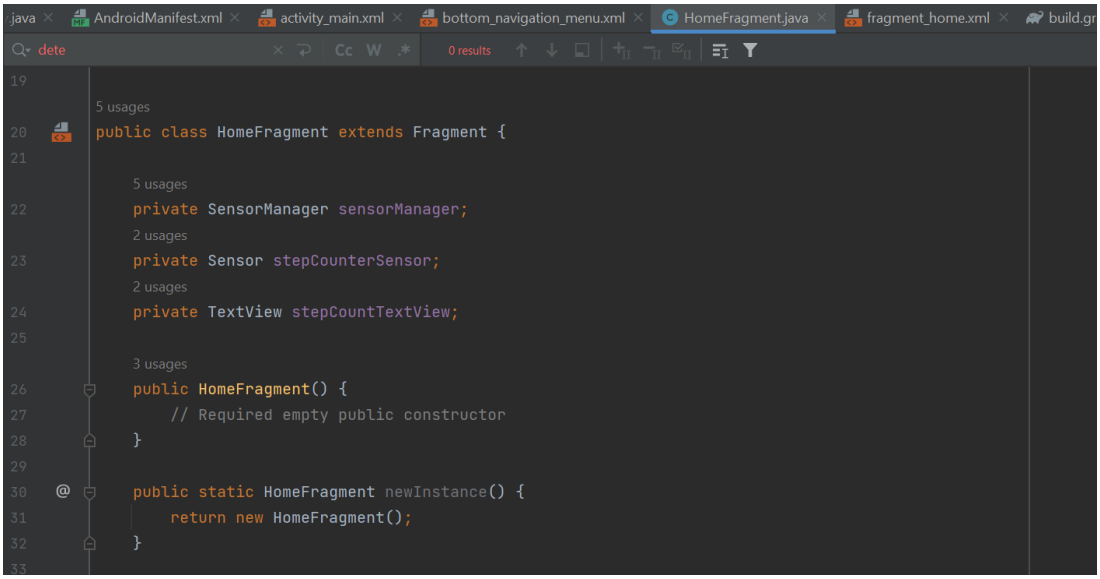
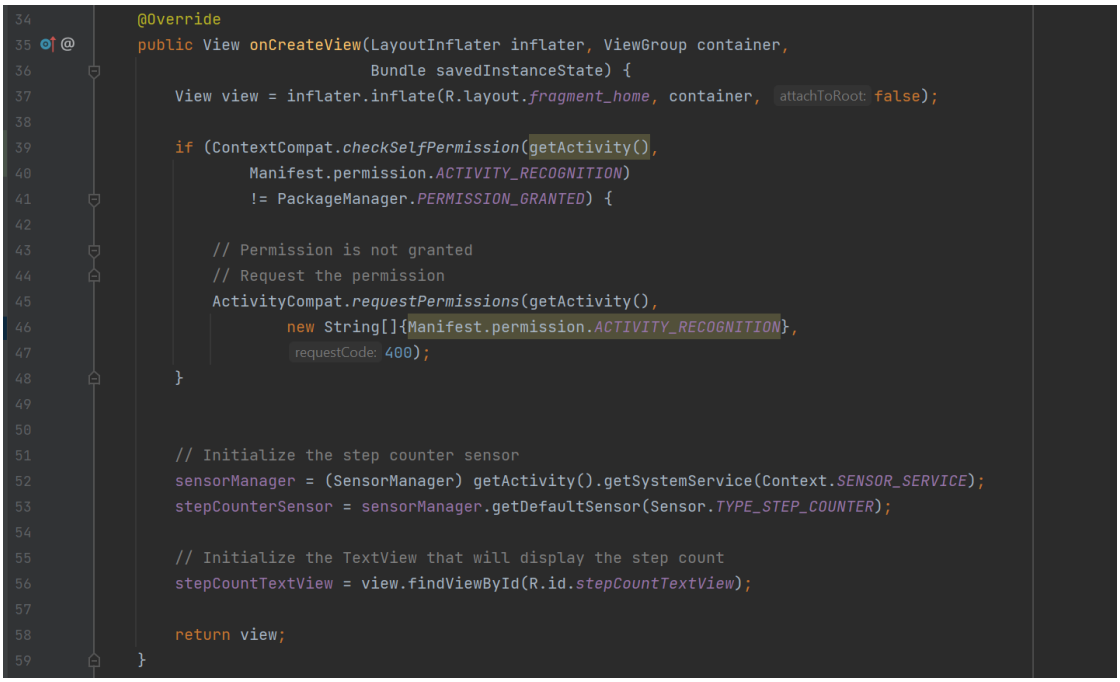


Figure 12: initializing sensor, sensor manager and step count display as well as required constructers

The onCreateView method was overridden for the home fragment to add the functionality. This overridden method and any like it would need to include the line of code to inflate the layout. As the application will be booted on the home fragment it is safe to include the code for requesting permission at runtime in this place. Permission is requested with a request code so I could override the method later. The step counter's sensor and sensor manager as well as text view were then all assigned their values, and the view would need to be returned to retain functionality of the fragment.

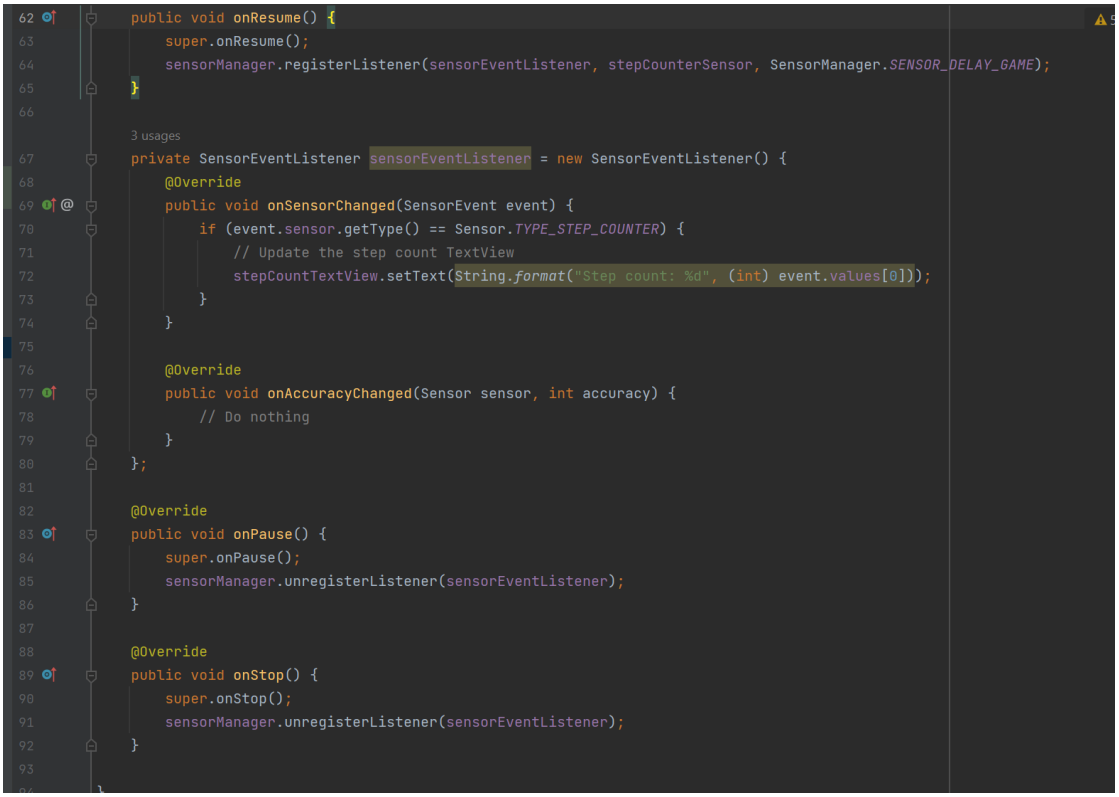


```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_home, container,  attachToRoot: false);

    if (ContextCompat.checkSelfPermission(getActivity(),
            Manifest.permission.ACTIVITY_RECOGNITION)
            != PackageManager.PERMISSION_GRANTED) {

        // Permission is not granted
        // Request the permission
        ActivityCompat.requestPermissions(getActivity(),
                new String[]{Manifest.permission.ACTIVITY_RECOGNITION},
                requestCode: 400);
    }


    // Initialize the step counter sensor
    sensorManager = (SensorManager) getActivity().getSystemService(Context.SENSOR_SERVICE);
    stepCounterSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);

    // Initialize the TextView that will display the step count
    stepCountTextView = view.findViewById(R.id.stepCountTextView);

    return view;
}
```

Figure 13: onCreateView override to check for permission and initialize the step counter and text view

The android lifecycle would then have to be managed to ensure that the sensor was used appropriately (as previously described above figure 12). The sensor manager was set to register a listener for the step counter sensor on resume and would remove it on pause or on stop (this would later be mildly changed but for the moment it was appropriate for testing the implementation of the step counter).

The event listener would be overridden to update the text view with the current step count every time a change was detected from the sensor. As onAccuracyChanged is part of the SensorEventListener interface as well as the desired onSensorChanged event, the unwanted method still needs to be overridden to satisfy the requirements of implementing the interface despite having the method hold no functionality.



```
62    public void onResume() {
63        super.onResume();
64        sensorManager.registerListener(sensorEventListener, stepCounterSensor, SensorManager.SENSOR_DELAY_GAME);
65    }
66
      3 usages
67    private SensorEventListener sensorEventListener = new SensorEventListener() {
68        @Override
69        public void onSensorChanged(SensorEvent event) {
70            if (event.sensor.getType() == Sensor.TYPE_STEP_COUNTER) {
71                // Update the step count TextView
72                stepCountTextView.setText(String.format("Step count: %d", (int) event.values[0]));
73            }
74        }
75
76        @Override
77        public void onAccuracyChanged(Sensor sensor, int accuracy) {
78            // Do nothing
79        }
80    };
81
82    @Override
83    public void onPause() {
84        super.onPause();
85        sensorManager.unregisterListener(sensorEventListener);
86    }
87
88    @Override
89    public void onStop() {
90        super.onStop();
91        sensorManager.unregisterListener(sensorEventListener);
92    }
93
```

Figure 14: registration and unregistration using the android lifecycle as well as updating the fragment with the step count after a change detected from the sensor

However, it was at this point that I started testing the new implementation and discovered a logic error. Because the text view would only update upon a change in the step sensor and there is no way to get the step count from the sensor outside of the listener, there was no way to display the current step count on screen without first walking far enough for the set count to update.

As well as this the step count sensor would always display the step count since the last time the device was booted up, not from the day and it couldn't be changed to display the step count for the day, which clashed with the initial designs of the application.

The solution to this issue would be storing the sum of all previous day's step count somewhere (preferably shared preferences for ease of access) and then recalling it, using it to minus from the current step count to reset for the day. Also, a stored figure in shared preferences could be used to automatically display the last remembered figure onscreen instead of 0 until the sensor updates (which is important due to the long wait times between updates).

Other than this the implementation was a success. Figure 15 displays the step count accurately being displayed on screen and figures 16 – 18 display the permission successfully being granted at runtime. Testing revealed an expected issue in terms of the latency of using a step count sensor as opposed to a step detector sensor. Both step count and detector sensors are not real sensors but rather algorithms that emulate sensors. The step count sensor updates less frequently than the step detector sensor, resulting in a delay in step count readings. While this can be an issue the step count sensor provides a

much more accurate count of steps taken over a longer period of time in contrast to the step detector sensor, which is much less accurate but with little latency. I made the conscious decision to use the more accurate sensor and so expected the slow update of the step count. While the android developer page estimated the latency of the step count sensor to be around 10 seconds and the step detectors latency to be around 2 seconds, the issue was in actuality worse than predicted. Black box testing of both sensors revealed a latency averaging a minute or more for the step count and 40 seconds for the step detector. These results may have been native to the device as other devices ran it slower or faster, but the application was designed to be used over long distances, so the choice in sensor stays validated.

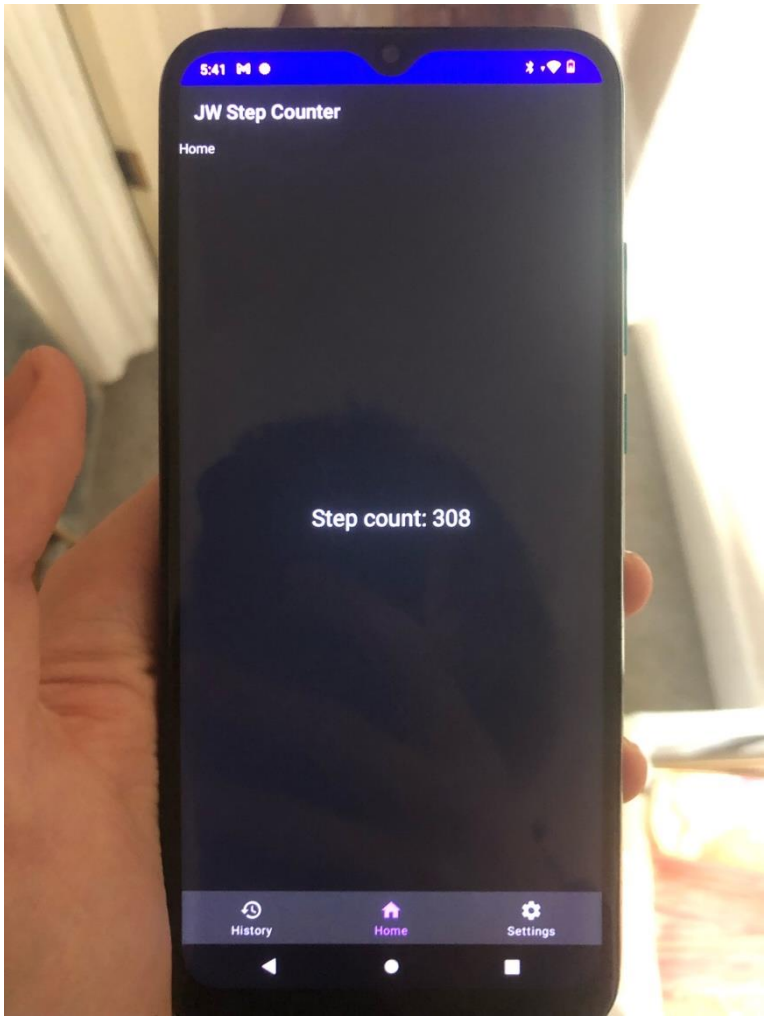


Figure 15: successful test of step counter functionality

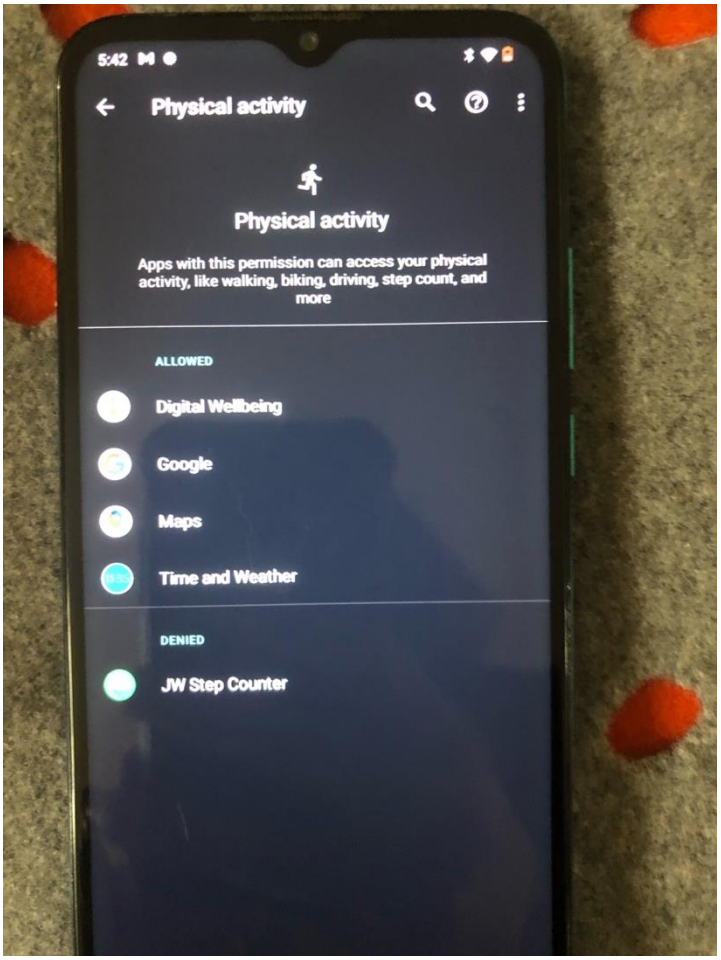


Figure 16: removing permissions from app

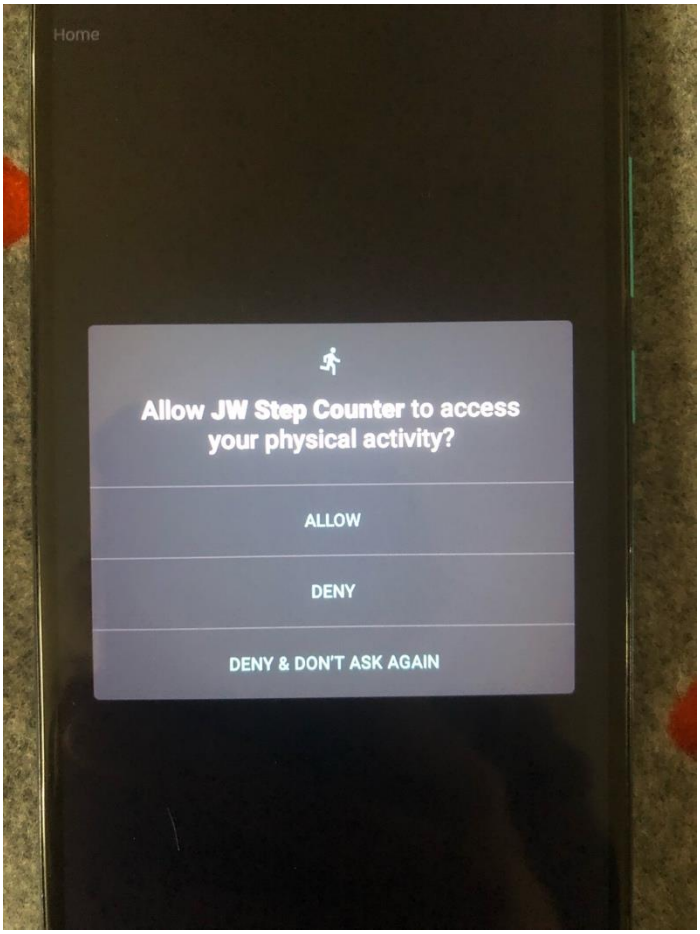


Figure 17: app successfully requests permission

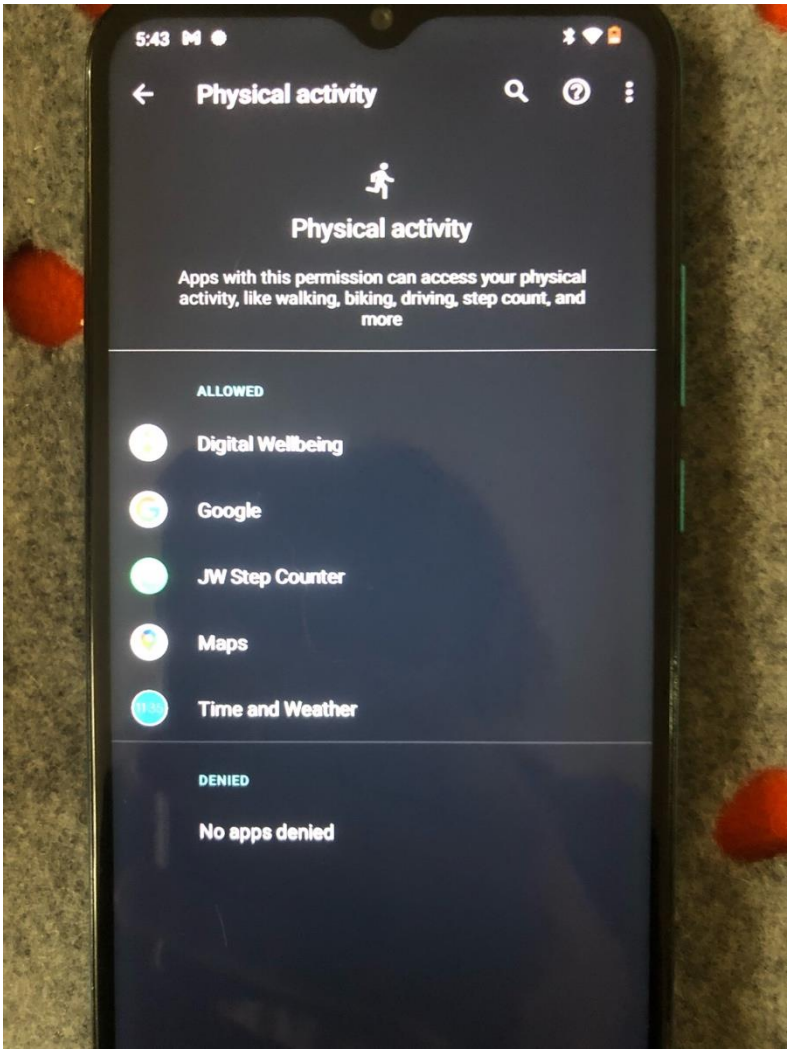


Figure 18: permission is granted again

To avoid the loss of any progress the new implementation was pushed to GitHub along with any changes that accompanied it. This practice would have to be enforced after every new update to backup progress and gain all the other advantages of version control such as roll backs and casual/passive documentation.

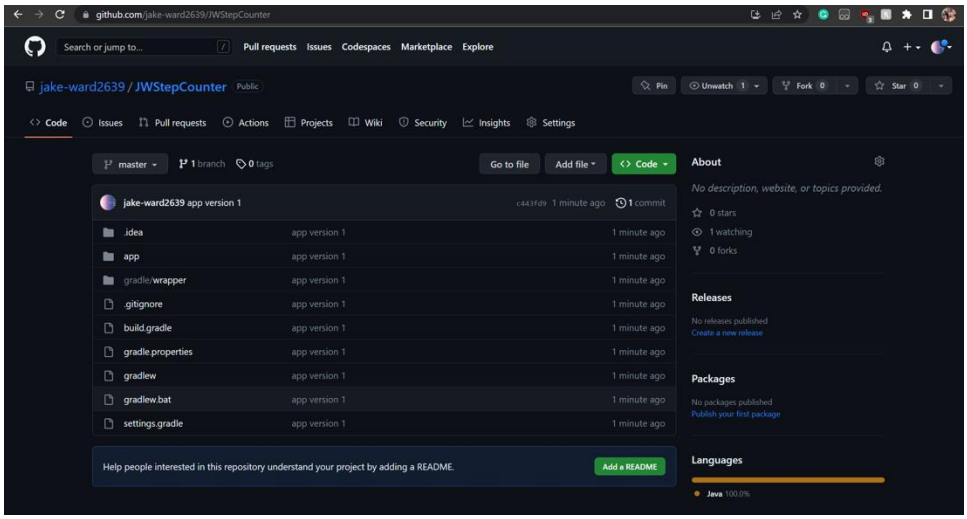


Figure 19: changes pushed to GitHub repository

4.4 Shared Preferences

From here on figures mostly consist of changes made to the code with the old code positioned on the left-hand side and the new on the right. This is because it better visually demonstrates changes, highlighting the focus of the figures and the current feature's implementation.

To create the changes previously mentioned to reset the steps per day the following implementation was made. Shared preferences were defined with values for the last known date, current step count and the old step count. A function was created to check the date to see if it had changed by using the android devices calendar. If the current date was not found to be the same as the date stored in shared preferences (or the default value of a zero-length string) then the current date would be saved under shared preferences as the current date and the old step count would be saved as the old step count plus the current step count. The current step count would be set in shared preferences every time a change in the sensor was detected and would be calculated by using the step count from the sensor minus the old step count. However, to account for the fact that on a reboot the sensors value will revert to zero and avoid errors, if the sensors value is ever less than the old step count then the two step counts in shared preferences are reset.

By using this logic, the app would be able to accurately track the user's steps for an infinite number of days. Every day, the app would check the current date against the last known date stored in shared preferences. If the date has changed, the app would save the old step count as the sum of the old step count and the current step count and reset the current step count to zero for it to be updated by the sensor listener.

The onDestroy method was used to replace the onPause method for unregistering the sensor so that if the screen of the device went black due to time out then the sensor would still be tracking steps (as would be the most useful for long distance walking).

Figures 22 – 24 demonstrate the use of shared preferences functioning successfully on the first day, second day and after a reboot. After all of this, the changes were committed and pushed to GitHub to back up the progress.

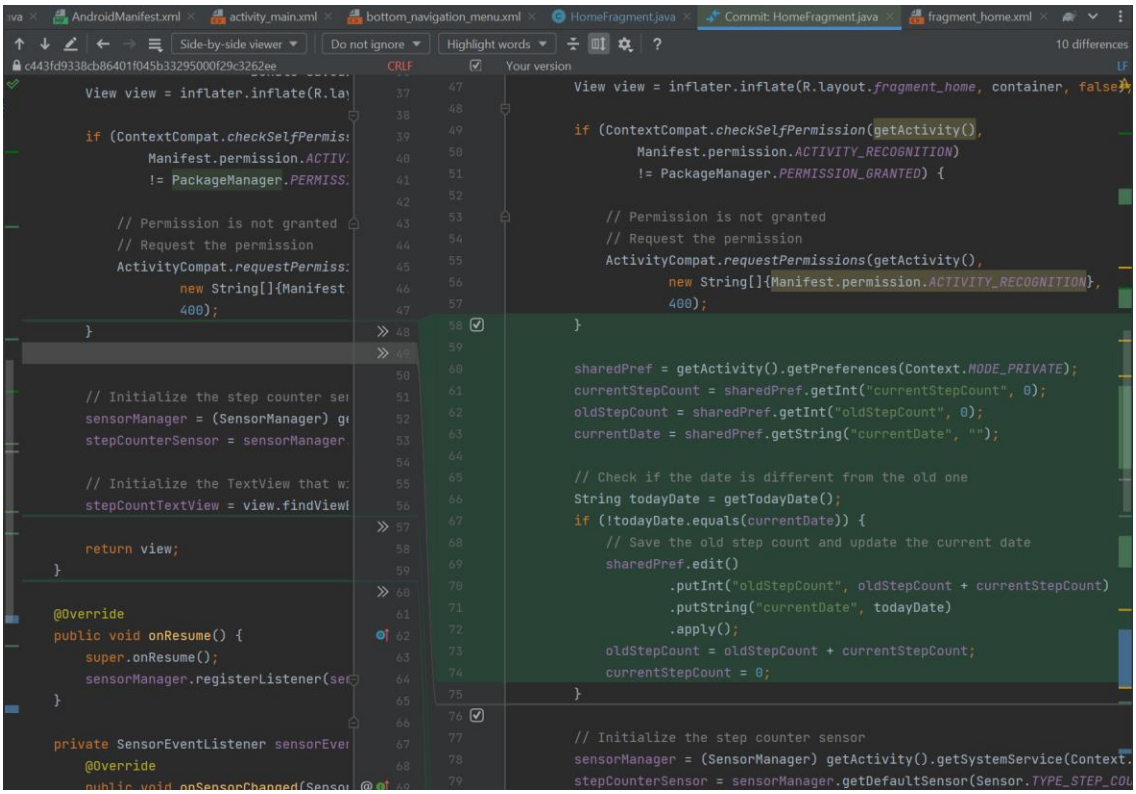


Figure 20: changes to on create of home fragment to get/set shared preferences to calculate daily step count

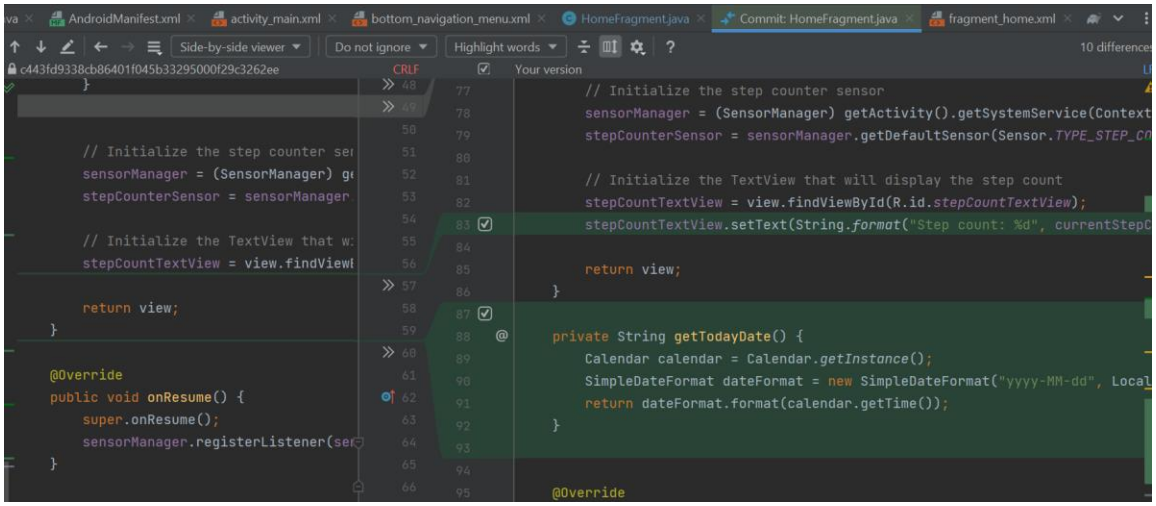


Figure 21: accurate step count on fragment/app load and get date method

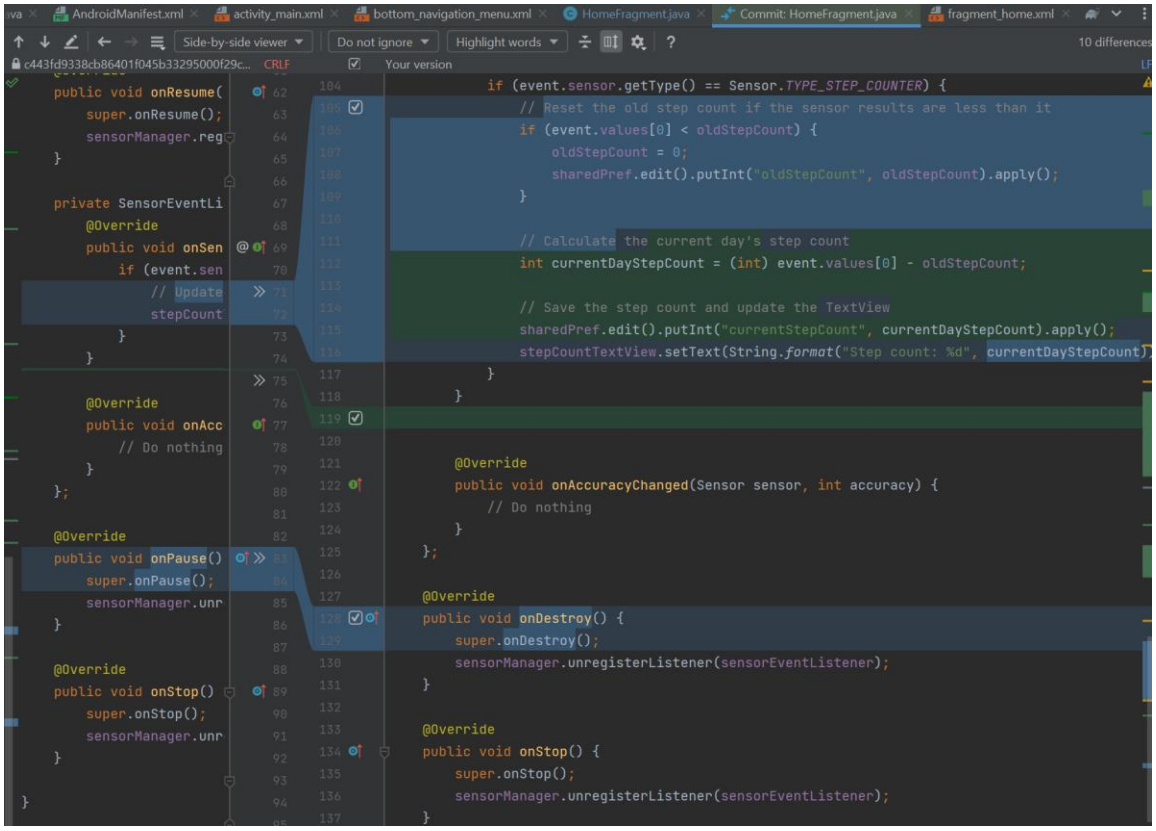


Figure 22: lifecycle handling changed pause to destroy, calculation and display of daily step count while accounting for restarted device

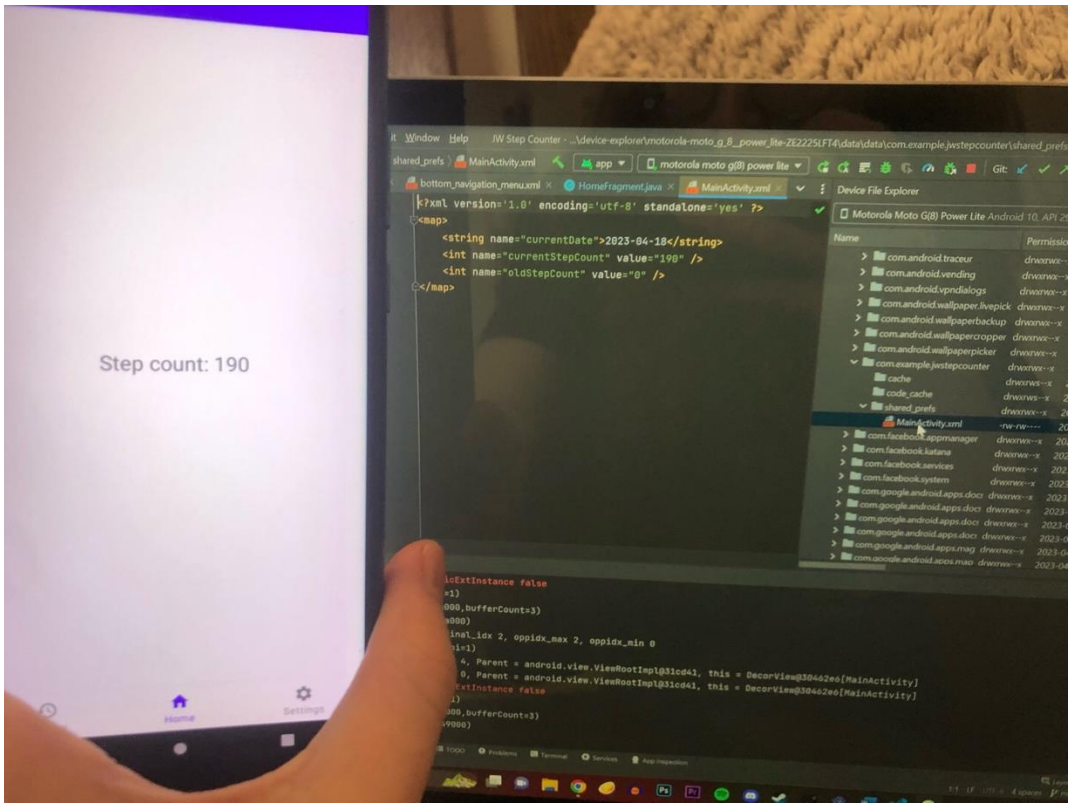


Figure 22: functional practical test of shared preferences usage

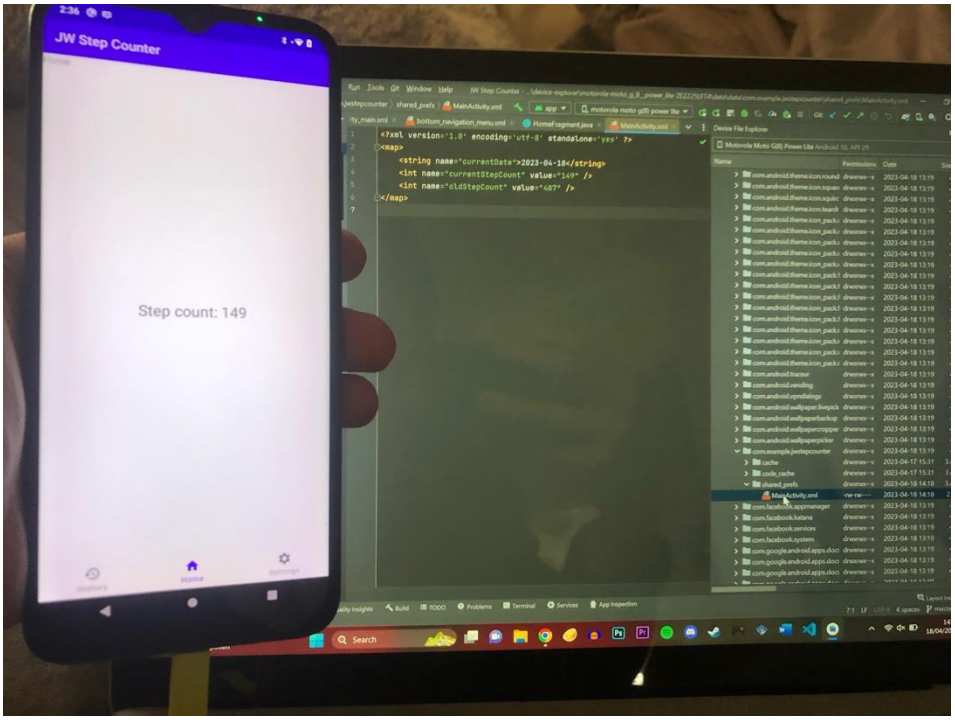


Figure 23: functional practical test of daily step count calculation

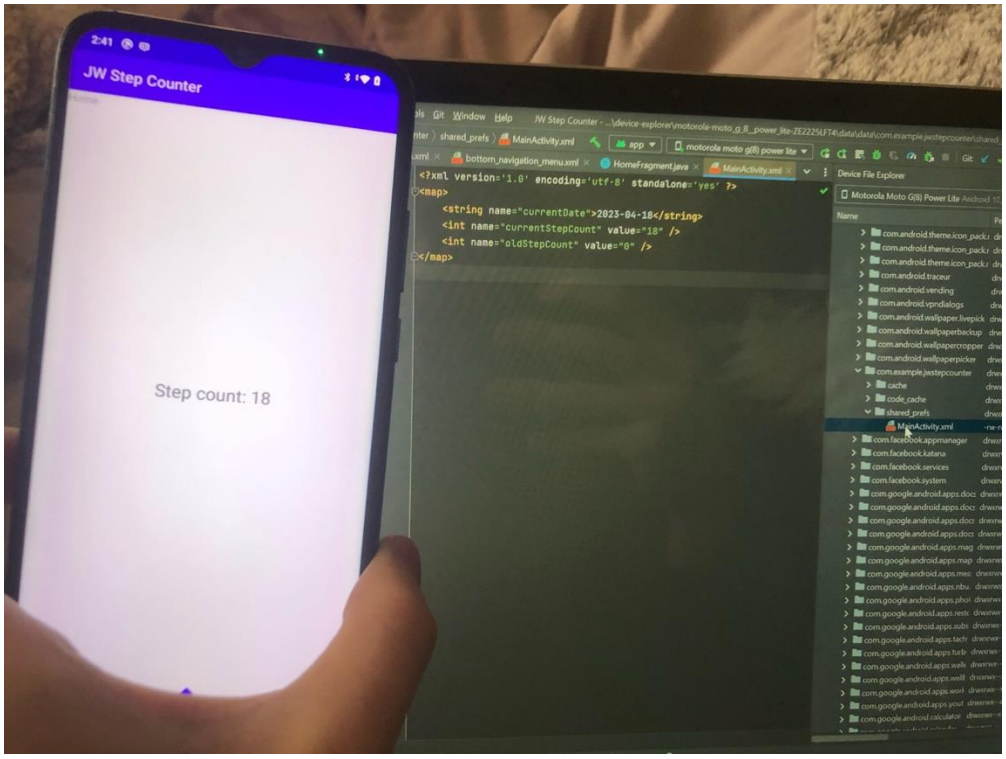


Figure 24: successful reset after device restart to avoid errors

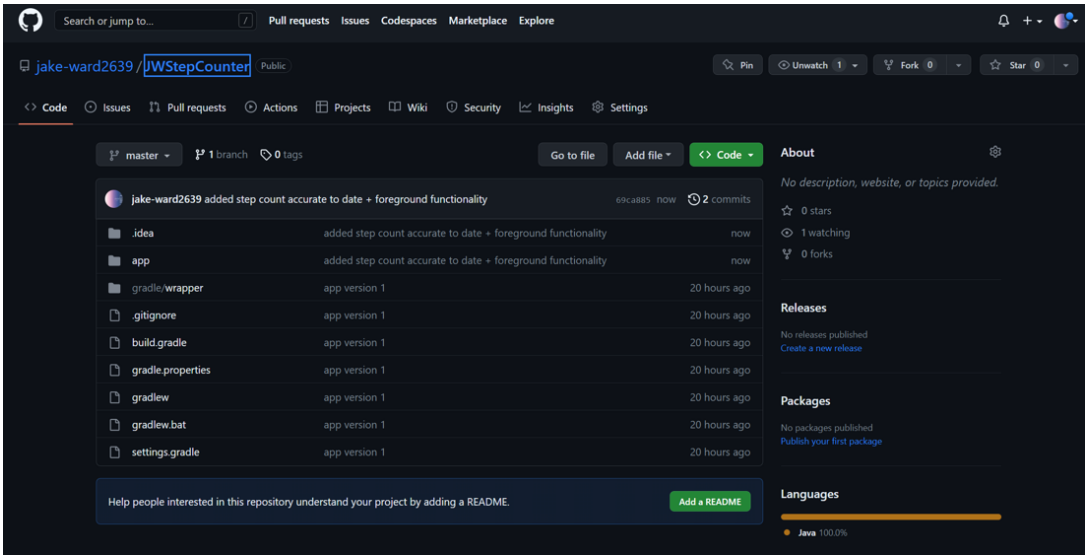


Figure 25: step counter functionality updated on repository

4.5 Snackbar on Sensor Access Denial

After having added two types of advanced features in the forms of advanced UI and sensor access, the next step was to handle the scenario in which a user didn't give permission access to their activity sensors.

I found that the method I was currently using to request permission at run time (onRequestPermissionsResult) was recently deprecated and so couldn't be overridden. Thus, I had to change the method to the usage of an ActivityResultLauncher, which could be written to handle a denial scenario.

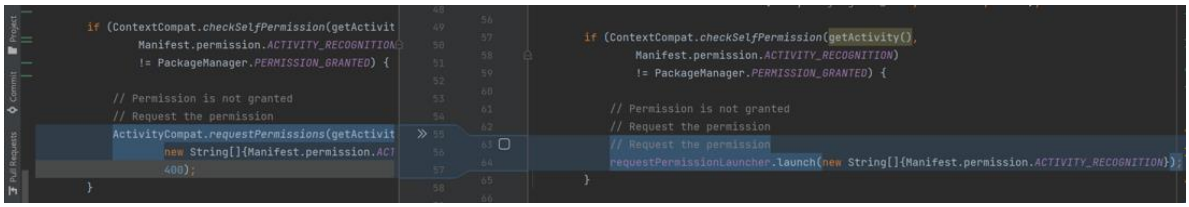
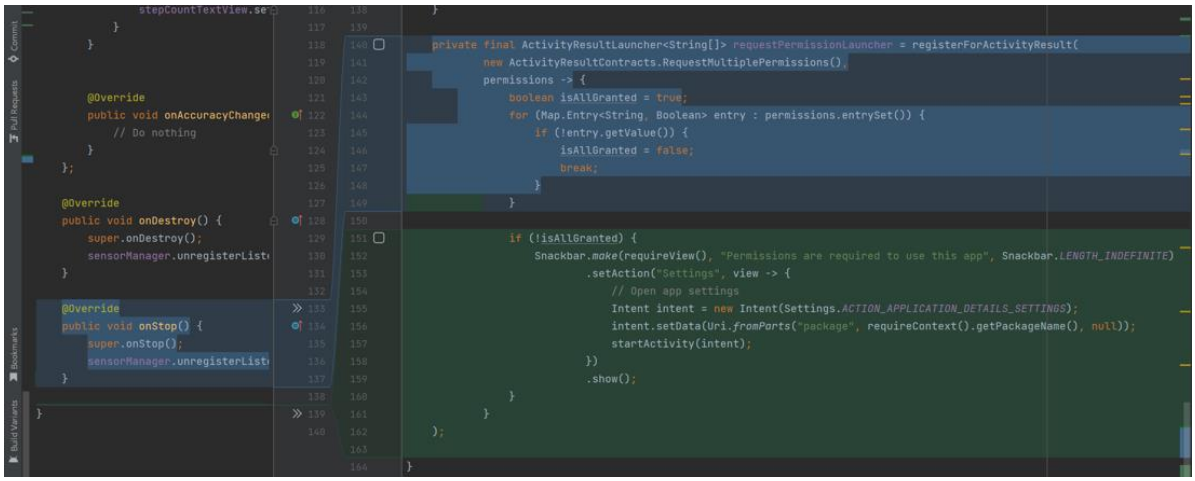


Figure 26: method for asking permission changed to ActivityResultLauncher

When a user did not allow access to the sensors required for the application then they would need to be notified about that fact that the app wouldn't function which could easily be done through a toast message. However, material design enables the use of snackbars, which can both place a more user-friendly notification on screen and hold a button that redirects the user directly to the permission which needs to be enabled. The snackbar was set to display for an indefinite amount of time because it can be swiped away by the user to remove it and it is important information, so we don't want the user to miss it. This implementation was simple due to the large amount of documentation surrounding material design and requesting permission as both are common practices. By adhering to material designs conventions, it makes the app consistent with what the user is likely used to and by keeping the error message short with a button to open the settings of the app it informs the user while giving them the most simplistic route to solve the issue.

After the snackbar and action had been set up the onStop unregistration of the sensor was removed it may have led to the step count not being updated if the user switches to another app or locked the device while the app was still running. The applications sensor monitoring would now only unregister on destruction.

Figures 28 – 30 display the application requesting permission, being denied, displaying the snackbar and the snackbars action directing the user to the app's settings to resolve the issue. It should be noted that permissions aren't asked upon every application load, tests only displayed the requests after permission had been denied again in settings.



```
        stepCountTextView.se
        }
    }

    @Override
    public void onAccuracyChange
        // Do nothing
    }
};

@Override
public void onDestroy() {
    super.onDestroy();
    sensorManager.unregisterList
}

@Override
public void onStop() {
    super.onStop();
    sensorManager.unregisterList
}
}
```

```
}

private final ActivityResultLauncher<String[]> requestPermissionLauncher = registerForActivityResult(
        new ActivityResultContracts.RequestMultiplePermissions(),
        permissions -> {
            boolean isAllGranted = true;
            for (Map.Entry<String, Boolean> entry : permissions.entrySet()) {
                if (!entry.getValue()) {
                    isAllGranted = false;
                    break;
                }
            }

            if (!isAllGranted) {
                Snackbar.make(requireView(), "Permissions are required to use this app", Snackbar.LENGTH_INDEFINITE)
                        .setAction("Settings", view -> {
                            // Open app settings
                            Intent intent = new Intent(Settings.ACTION_APPLICATION_DETAILS_SETTINGS);
                            intent.setData(Uri.fromParts("package", requireContext().getPackageName(), null));
                            startActivity(intent);
                        })
                        .show();
            }
        }
);

}
```

Figure 27: permission request handled to display a snackbar with a link to settings upon denied request and onstop unregistration removed

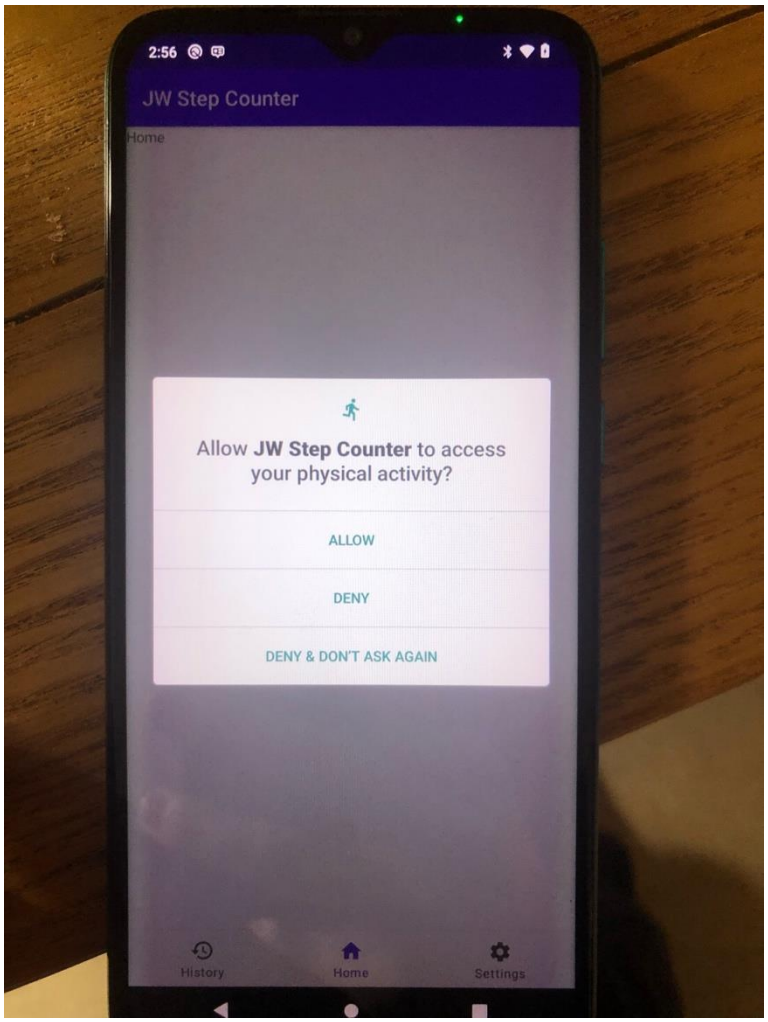


Figure 28: snackbar demonstration (request before denial)

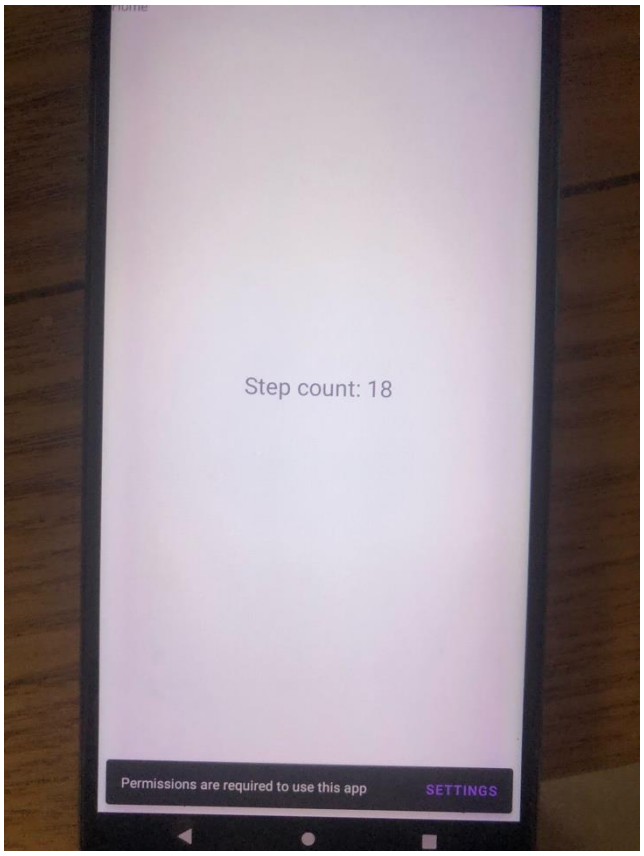


Figure 29: snackbar demonstration (message displayed after denial)

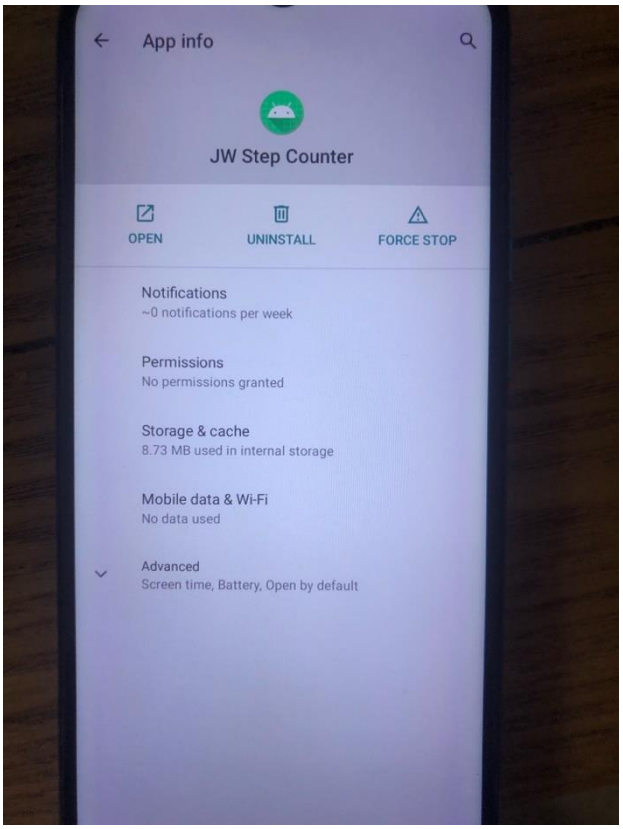


Figure 30: snackbar demonstration (after settings redirect)

Once again due to a significant development having taken place and to ensure responsible project management, the changes were all committed and pushed to GitHub to function as a backup.

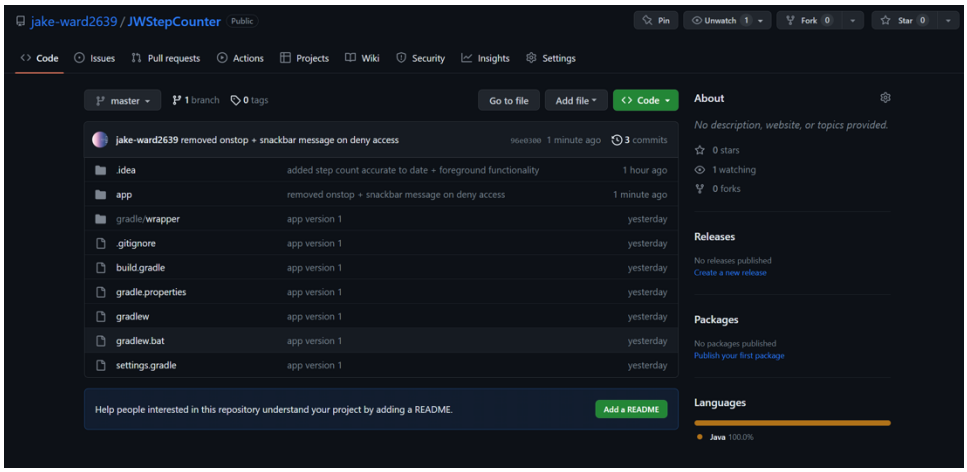


Figure 31: changes regarding snackbar and onstop committed and pushed to GitHub

4.6 Launcher Icon

As it was a relatively simple and automated feature of android studio, I implemented an application icon to replace the default one. It was designed in photoshop simplistically as its only function was to prove that the icon could be changed. It was then used to generate a new launcher icon where android studio would automatically replace all the relevant icon files with the new ones. This was promptly tested, found to be successful and pushed to GitHub. The icon could be improved but the proof of concept is there.

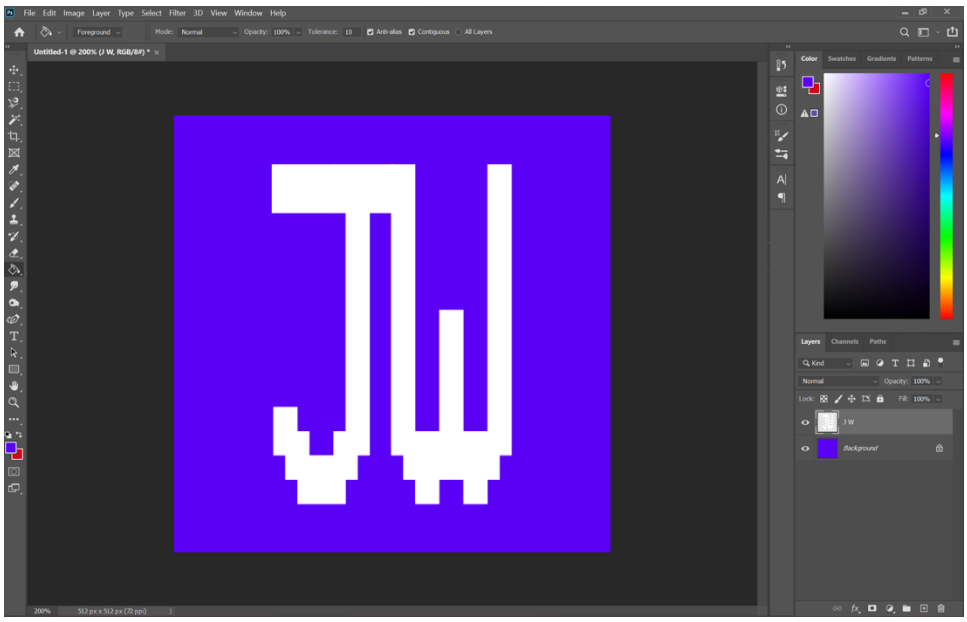


Figure 32: icon designed in photoshop

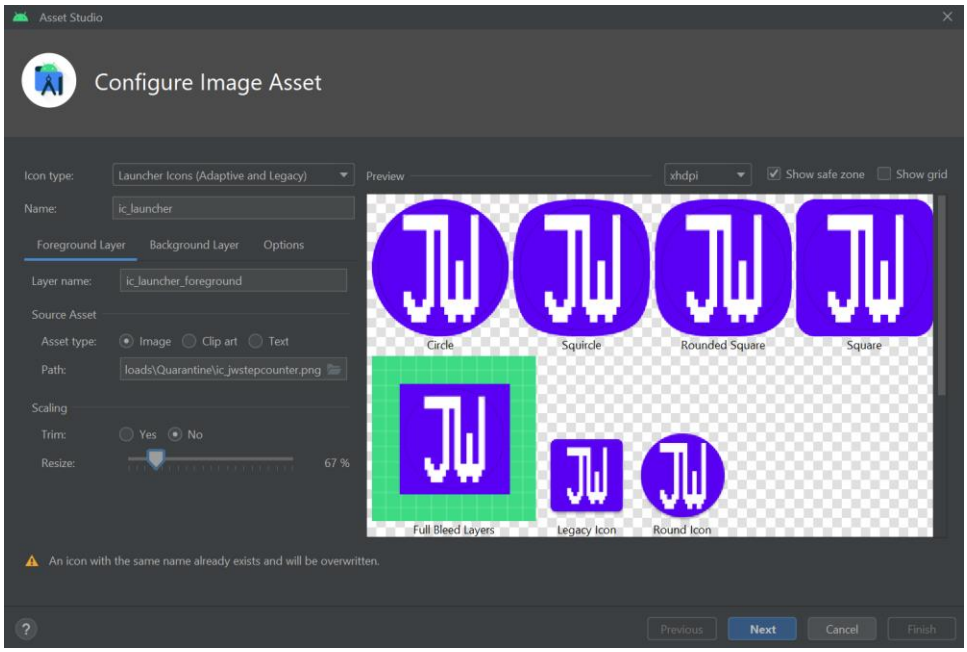


Figure 33: icon added to replace default icon via android studio

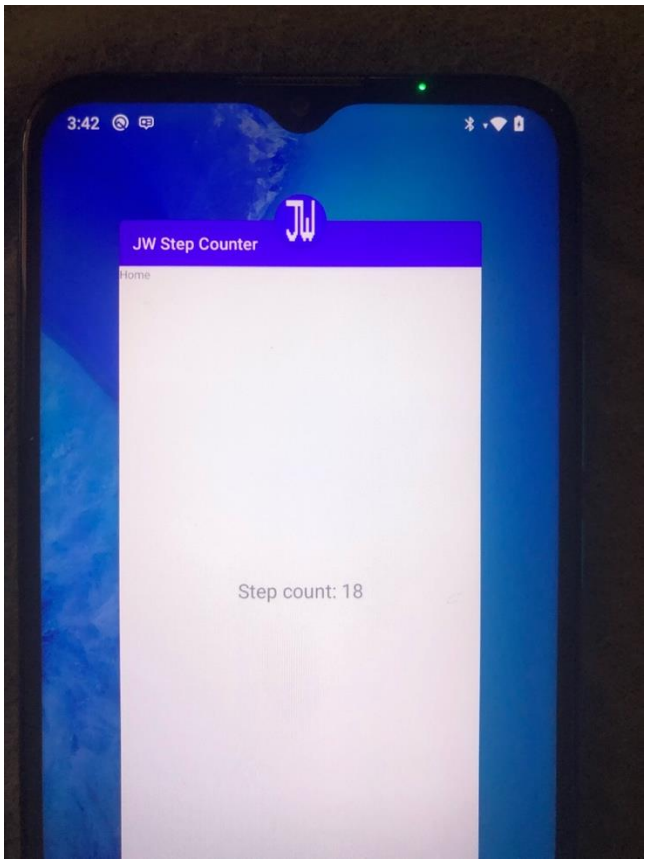


Figure 34: icon updated in app and on device

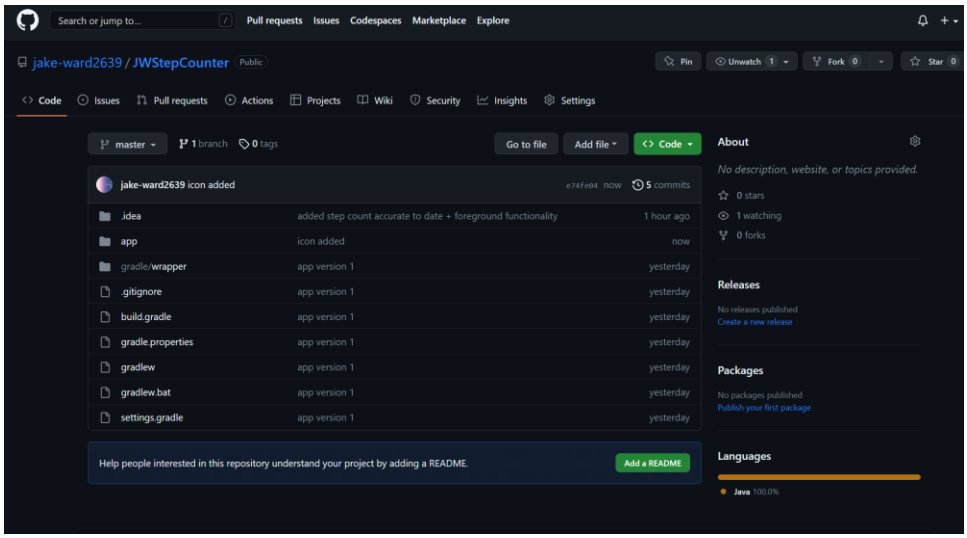


Figure 35: icon changes committed and pushed to GitHub

4.7 Circular Progress Bar

By researching similar applications for an appropriate progress bar, I came across the open sources circular progress bar developed by Lopez Mikhael which can be found at https://github.com/lopspower/CircularProgressBar. By integrating this progress bar, I could easily adhere to the initial draft designs I had made for the concept document and have all the functionality of a normal progress bar such as setting the progress, the maximum amount of progress the colours, sizes and etc. Once its inclusion was decided upon its implementation was added to the build grade file. It should be noted that while I didn't develop the progress bar it functions the same as a regular android provided progress bar and so shows no less learning. Additionally, it displays the ability to adapt other open-source projects into another project.

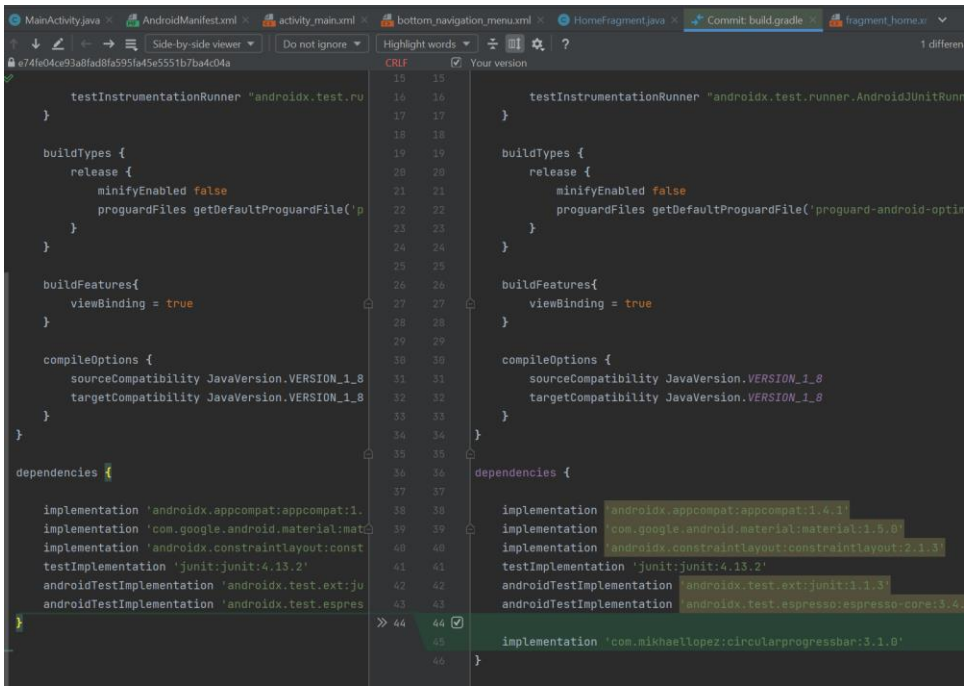


Figure 36: dependency added for circular progress bar implementation

The text view identifying the home fragment was removed due to it irrelevance with the current app and to enforce the simplistic design. The circular progress bar was then implemented in the XML for the home fragment by consulting the documentation and adapted for the application. Once again like all other elements, it was constrained after sizes and colours were selected to ensure the application would function at both orientations and on different dimensioned devices.

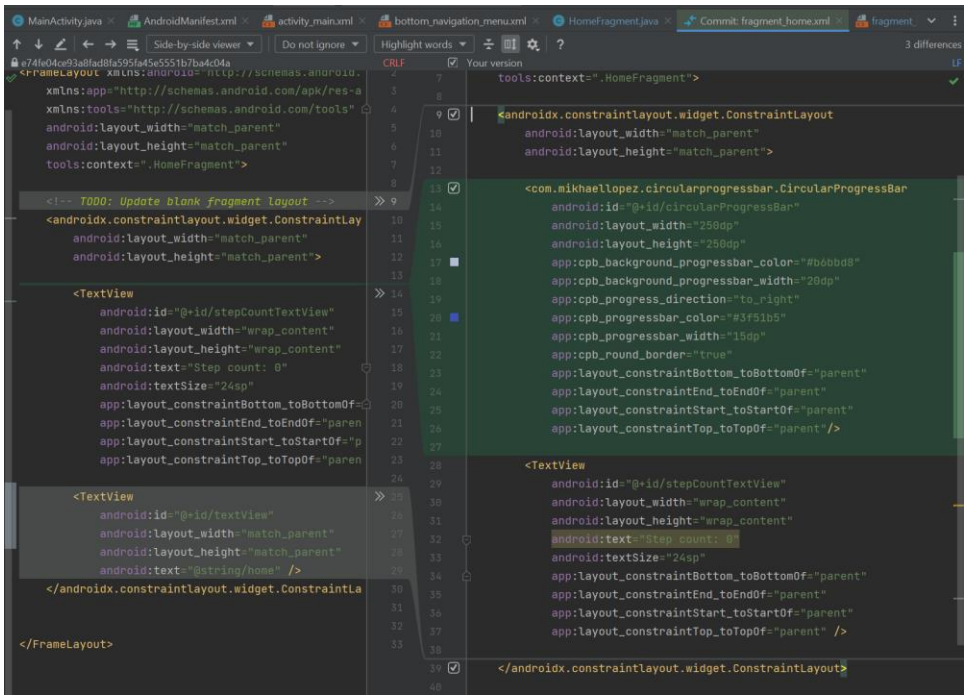


Figure 37: circular progress bar added to home fragment xml

The circular progress bar would then need to be updated every time the text view displaying the step count would be updated (on creation of the home fragments and on detected update from the sensor). So, it was set to be initialized on creation of the home fragment alongside the maximum progress being set and the current progress being set. As offered by the progress bar, to improve the UX mildly the progress was updated with animation lasting a second. The maximum progress was set using a value that would be set in shared preferences but with a default of 200 so it could still function without a step goal having been set. The changing progress line would be repeated below the other text view updated in the sensor listener.

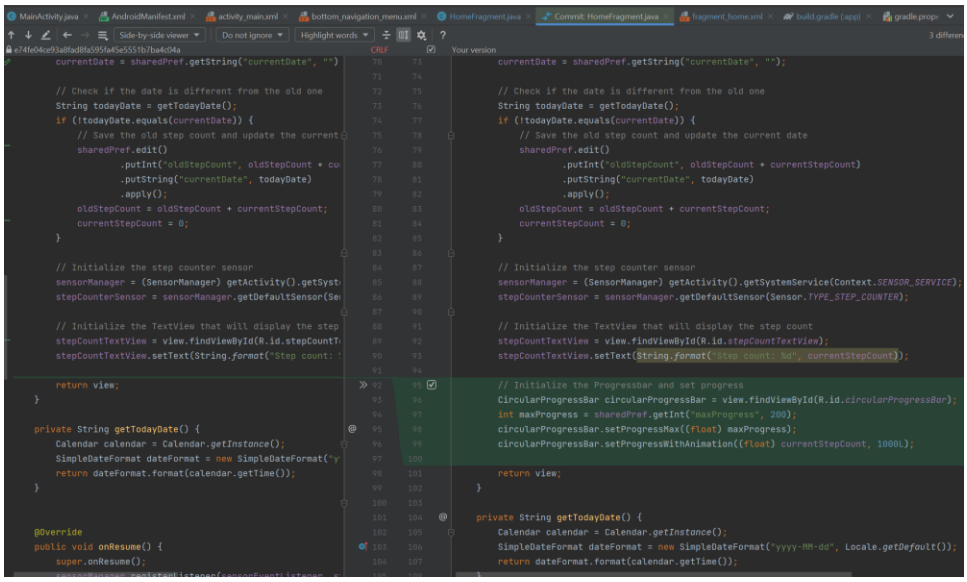


Figure 38: circular progress being set in home fragment java

```
private SensorEventListener sensorEventListener = new SensorEventListener() {
    jake-ward2639 *
    @Override
    public void onSensorChanged(SensorEvent event) {
        if (event.sensor.getType() == Sensor.TYPE_STEP_COUNTER) {
            // Reset the old step count if the sensor results are less than it
            if (event.values[0] < oldStepCount) {
                oldStepCount = 0;
                sharedPref.edit().putInt("oldStepCount", oldStepCount).apply();
            }

            // Calculate the current day's step count
            int currentDayStepCount = (int) event.values[0] - oldStepCount;

            // Save the step count and update the TextView
            sharedPref.edit().putInt("currentStepCount", currentDayStepCount).apply();
            stepCountTextView.setText(String.format("Step count: %d", currentDayStepCount));
            stepCountProgressBar.setProgressWithAnimation((float) currentDayStepCount,  duration: 1000L);
        }
    }
```

Figure 39: circular progress bar being updated upon step count being updated

This feature was promptly tested to find it was successful. After white box testing colours and length of animation were also validated and so the changes were committed and pushed to GitHub. The only bug that required fixing was due to a syntax error, where the wrong variable name was used to call the progress bar in the sensor listener. This was reverted to the correct variable and the application started working as intended.

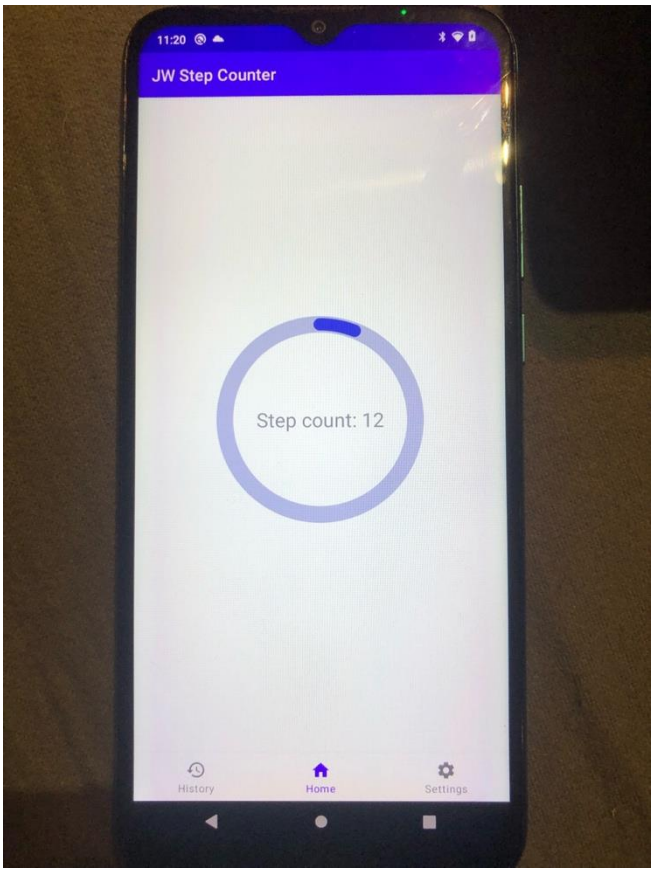


Figure 39: practical test of circular progress bar with default max value

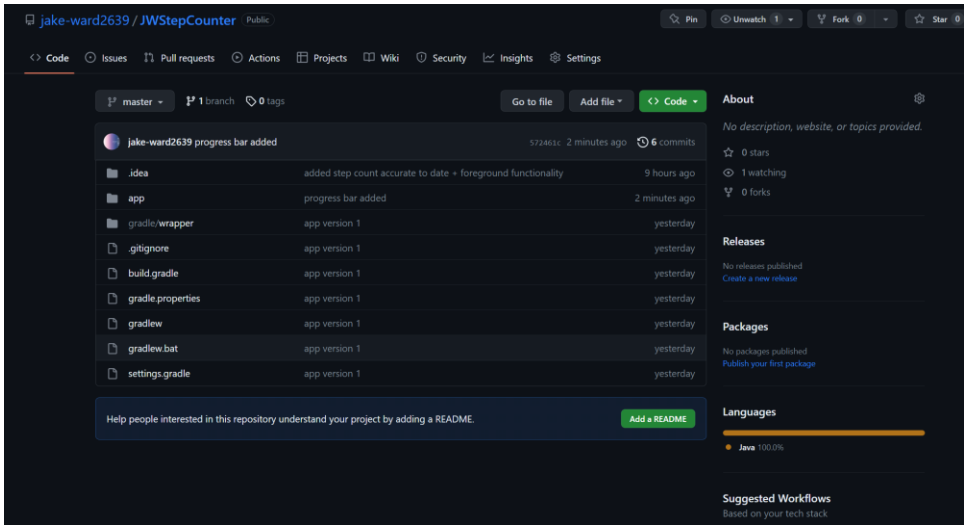


Figure 39: circular progress bar functionality pushed to repository

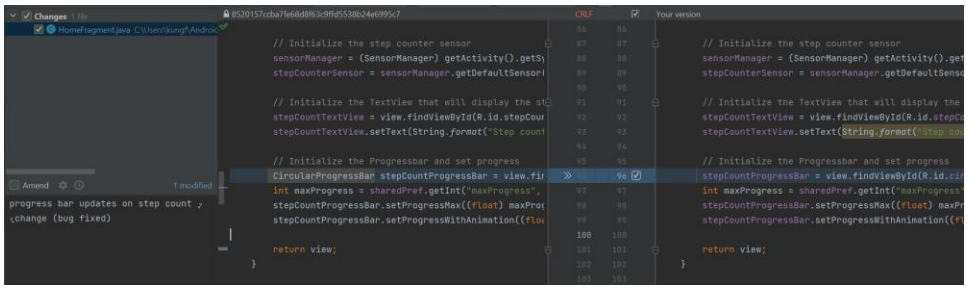


Figure 40: bug fix allowing progress bar to be updated upon new step count detection

4.8 Preference Screen/ Settings

After extended research it was decided that the conventional preference screen would be the best way to set up the functionality of the settings fragment. This would require a new implementation to be added to the Gradle build file.

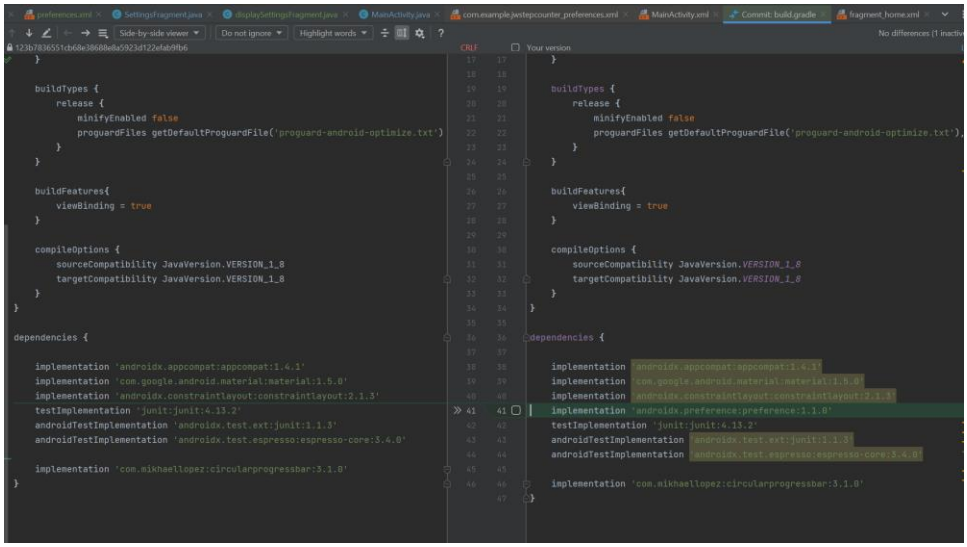


Figure 41: adding preference implementation to dependencies for settings

A new frame layout was defined in the XML file for the settings fragment and the identifying text view was removed due to irrelevance. Another XML android resource file was then created for the preference screen which would populate the frame layout. Preference screen elements can hold many types of preference objects but I decided upon a switch preference for the dark mode requirements and a edit text preference for adding a step goal to shared preferences.

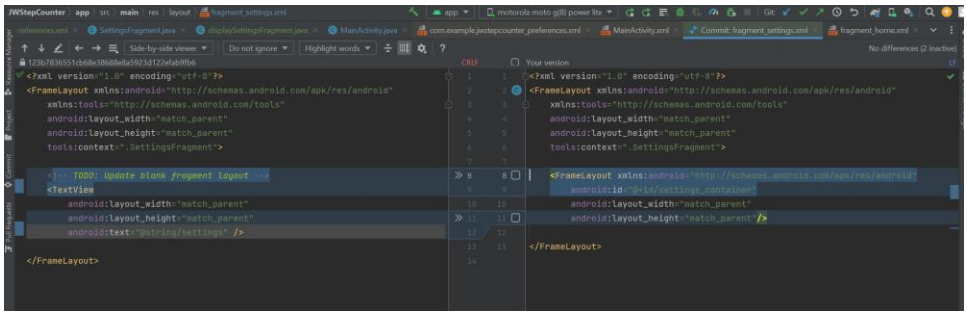


Figure 42: adding frame layout to settings fragment to hold preferences

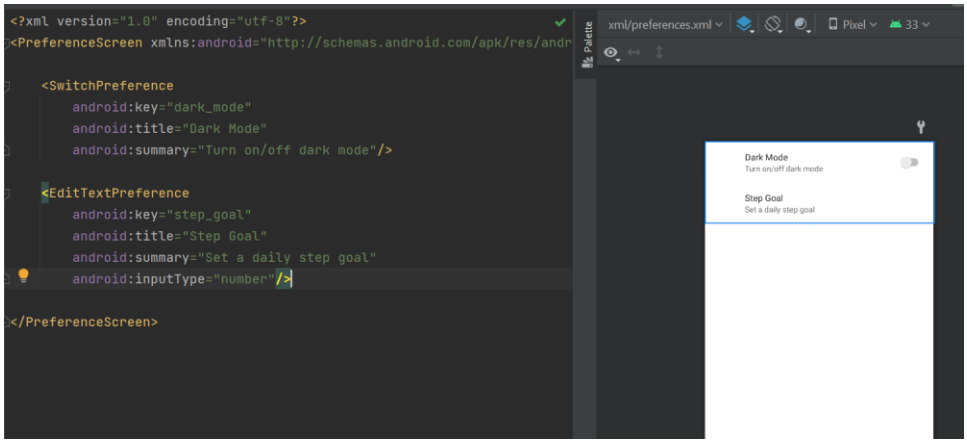


Figure 43: xml for preference screen with switch and edit text preferences

The default code for the fragment was scrubbed of all unnecessary code and code was added to the onCreateView method to replace the frame layout on the fragment with the preference screen. This was also done with a form of fragment manager similar to the initial control of fragments in the main activity. Child fragment manager needed to be called in this case as it was a fragment within a fragment, while it could have been designed otherwise, this method holds no detriment to the code, and it avoids having to alter old code, as well as allowing for settings to contain more than the preference screen if was so desired.

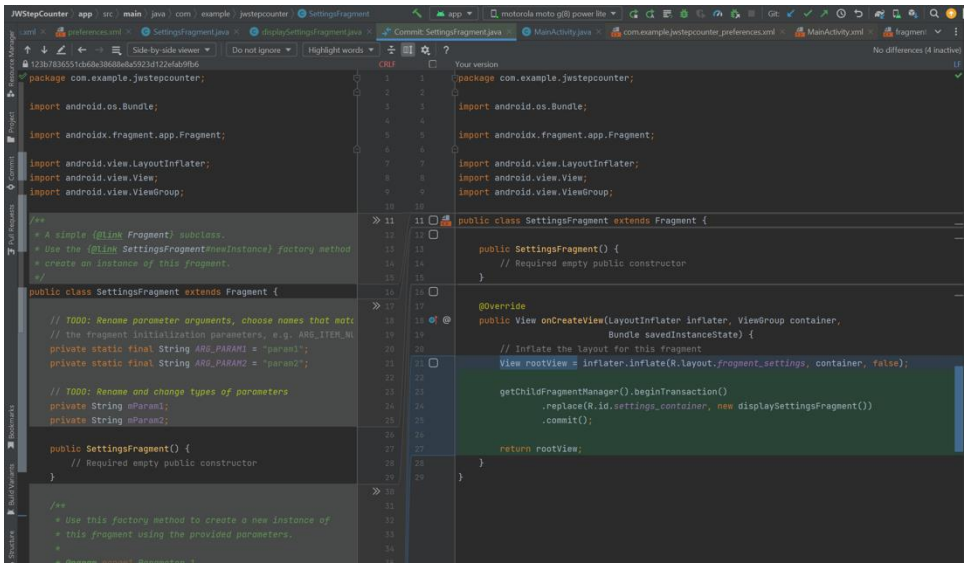
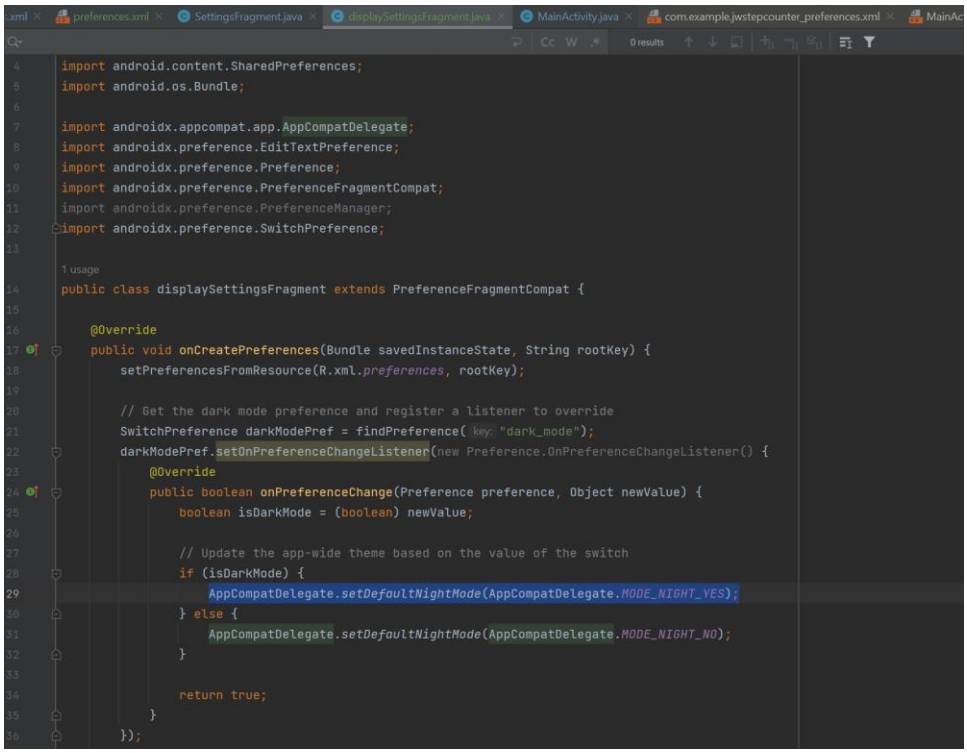


Figure 44: settings fragments java to host preferences in frame layout

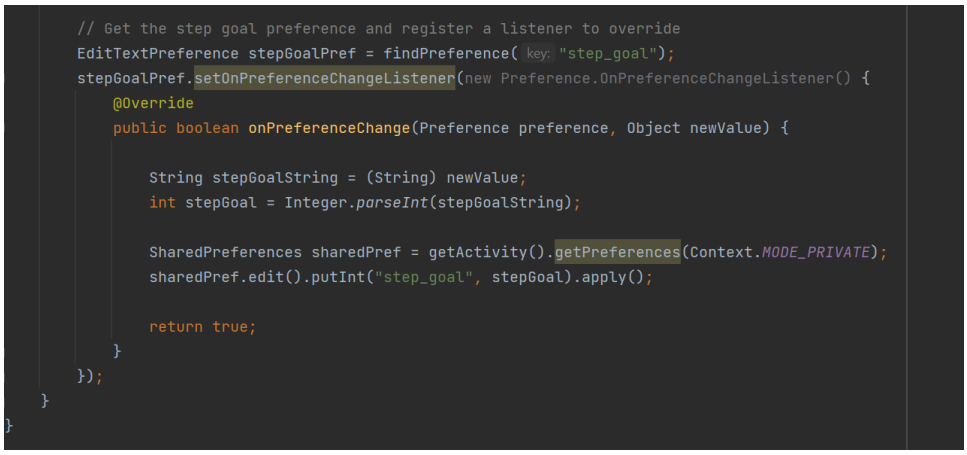
The code called to replace the frame layout couldn't be the XML file itself, so a new Java file was created which would extend PreferenceFragmentCompat to act as a preference fragment and set the preference resources as the predefined XML file. OnCreatePreferences needed to be utalised in this case and listeners were placed on the two defined preference elements (as would have to be the case with any further settings).

Code was added to the listener for dark mode to change the theme in accordance with the switches value and functionality was added to the step goal preference to save it under the activities preferences so it could be access by the previously written code to define the maximum value of the progress bar.



```java
import android.content.SharedPreferences;
import android.os.Bundle;

import androidx.appcompat.app.AppCompatDelegate;
import androidx.preference.EditTextPreference;
import androidx.preference.Preference;
import androidx.preference.PreferenceFragmentCompat;
import androidx.preference.PreferenceManager;
import androidx.preference.SwitchPreference;

1 usage
public class displaySettingsFragment extends PreferenceFragmentCompat {

    @Override
    public void onCreatePreferences(Bundle savedInstanceState, String rootKey) {
        setPreferencesFromResource(R.xml.preferences, rootKey);

        // Get the dark mode preference and register a listener to override
        SwitchPreference darkModePref = findPreference( key: "dark_mode");
        darkModePref.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                boolean isDarkMode = (boolean) newValue;

                // Update the app-wide theme based on the value of the switch
                if (isDarkMode) {
                    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YES);
                } else {
                    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO);
                }

                return true;
            }
        });
```

Figure 45: preferences java (dark mode listener and functionality)



```java
    // Get the step goal preference and register a listener to override
    EditTextPreference stepGoalPref = findPreference( key: "step_goal");
    stepGoalPref.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
        @Override
        public boolean onPreferenceChange(Preference preference, Object newValue) {

            String stepGoalString = (String) newValue;
            int stepGoal = Integer.parseInt(stepGoalString);

            SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
            sharedPref.edit().putInt("step_goal", stepGoal).apply();

            return true;
        }
    });
}
```

Figure 46: preferences java (step goal listener and functionality)

Preferences automatically get saved by using this method so on creation of the home fragment I added code that would check the preference of theme and adjust it accordingly. However, the preferences aren't saved in the preferences of the main activity, so the step count was included in both preferences, to keep all relevant information in one place but to still automatically remember the previous step count in the settings fragment. After these changes were made, they were all tested to expected success. Figure 48 and 49 show the separate shared preferences files in the test device that had been altered and the result displayed themselves onscreen as predicted.

There was however one unsolved error. According to all current documentation I could find, limiting the edit text preference in the XML to numbers should prevent users from being able to input letters or symbols, however in practice this wasn't the case. Many other methods were explored to fix this issue, but no solutions were found. To prevent wasting anymore development time the feature was left in its current state with a bug fix planned for the future. The step goal feature works fine otherwise and will close the application if the input isn't a number, preventing the rest of the application from breaking. The changes were added to the GitHub repository to back up progress.

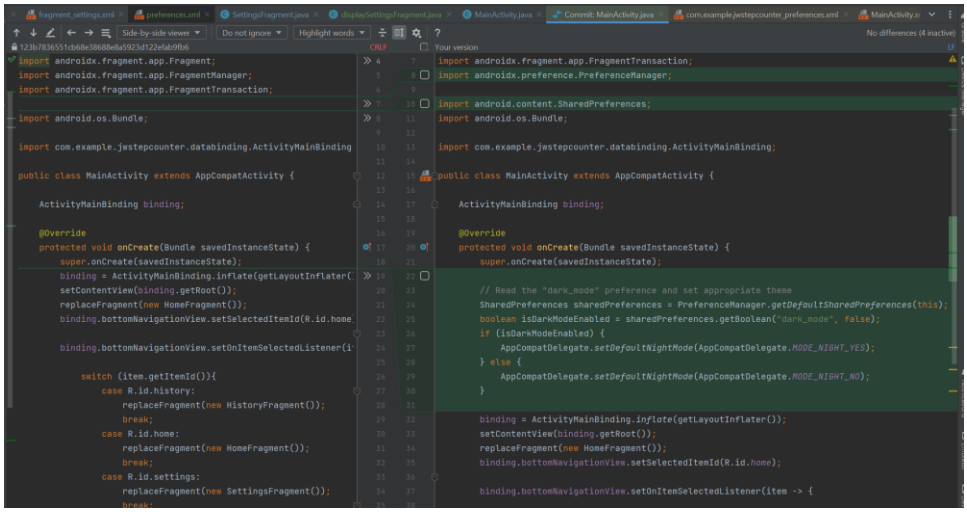


Figure 47: setting the theme on app launch according to shared preferences

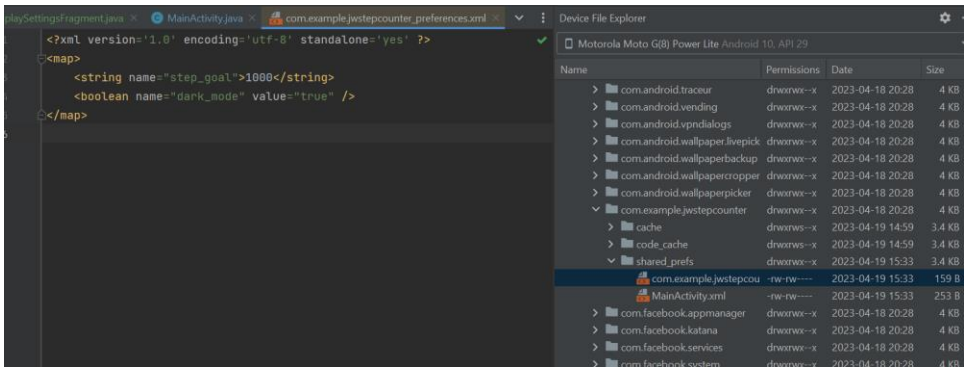


Figure 48: shared preferences of app

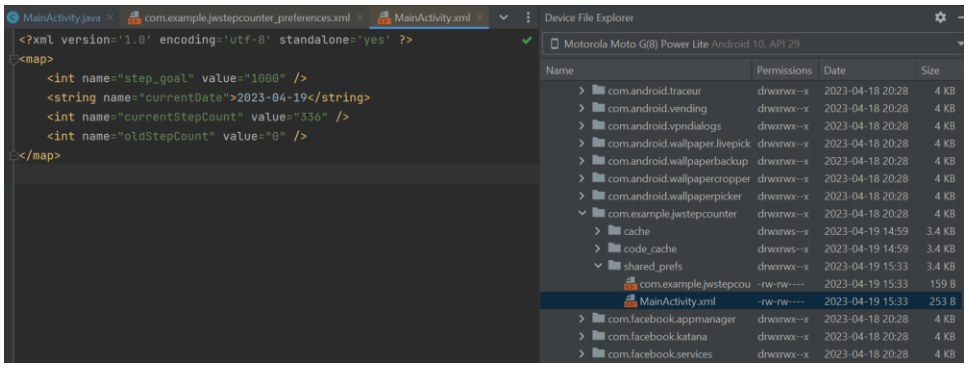


Figure 49: shared preferences of activity

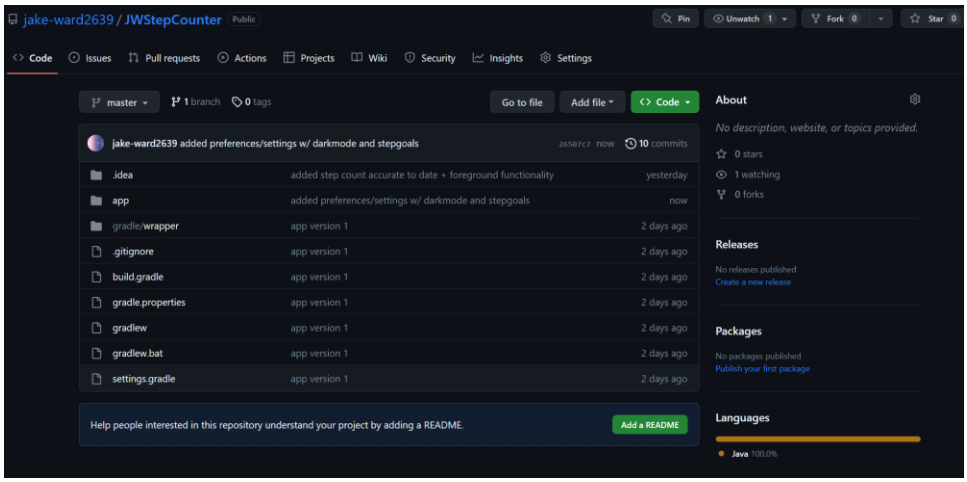


Figure 50: preferences (dark mode and step goal) pushed to repository

4.9 SQLite Storage

The final feature to implement would be a storage method for use in the history fragment. While this was initially planned to be done via firebase due to their modern design, conventional use in the industry as well as lack of complexity to set up, it was dropped in favour of SQLite due to familiarity with SQL and the fact that the storage can be done locally, eliminating the need for a login system to differentiate users and allowing users to have offline access. Offline access was the driving reason behind this decision as a fitness app would likely be consulted on location in hard-to-reach areas that would likely lack a sufficient signal. This would ensure the application could use all functionality at all times while also allowing me as the developer to display more skill in programming as SQLite has a reputation for complexity.

To set up a connection with an SQLite database, a database helper class would need to be created first. In this database helper the database name, version, table name, id column, date column and step count column would all need to be defined. These variables would then need to be used in the onCreate method to create the database through a SQLite query. Code would also be needed for the onUpgrade event so that the new table created would be dropped if the same table already existed. Variables that

would be desired from the database would need to be made public so that the history fragment could use them in its queries.

It may be confusing to others who are only familiar with SQL why _id was used for the primary key column name instead of simply id. In the given code, the id is set as _id because it is the common naming convention for the primary key column in an SQLite database table. _id is used by convention in Android as the primary key column name for CursorAdapters to bind data to views and uniquely identify each row in the table. By using this naming convention "AUTOINCREMENT" can be utilized to ensure that each row is uniquely identifiable. Using _id as the primary key column name instead of id also enables the use of other Android convenience methods like the CursorAdapter and its ability to obtain results from the database.

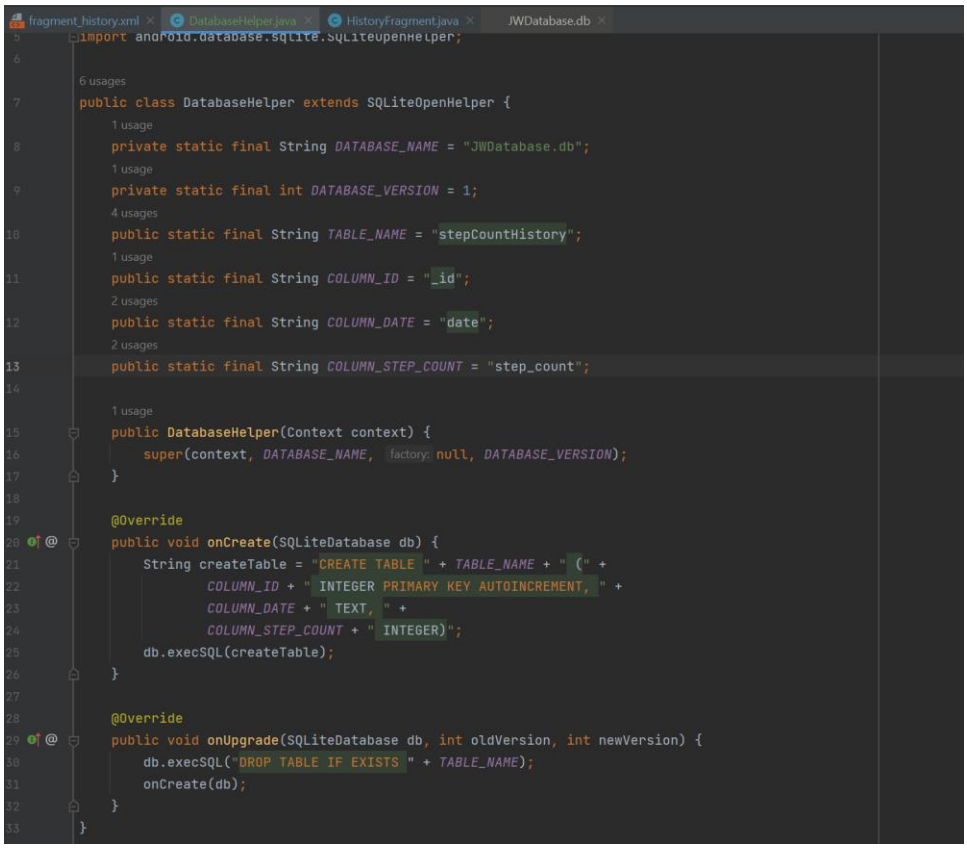


Figure 51: database helper java code to create database and table

Like the previous two fragments, the unnecessary generated code was removed. It was decided that the best way to demonstrate storage was in real time and it would also prevent the user from having to wait until the day had ended to log their results. So, a button was added to save data to the database, and another was added to wipe the table in case it overwhelmed the user. A recycler view was added as it was found to be the most conventional method of displaying results, however later research would reveal its unnecessary complexity causing it to be replaced with a list view instead.

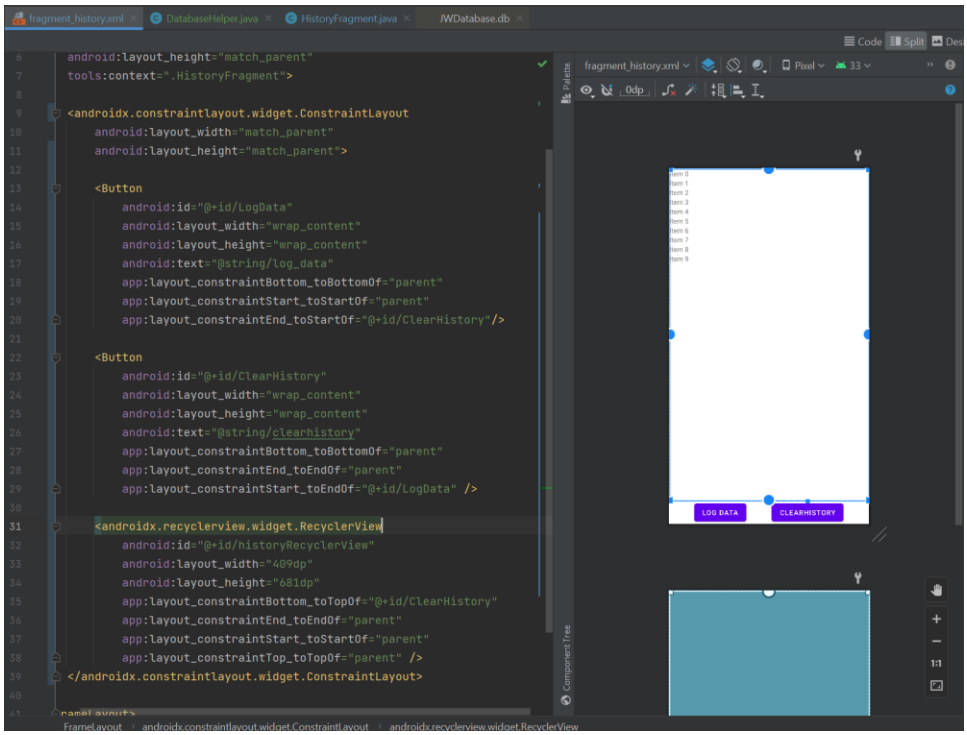


Figure 52: xml for history fragment

The fragments Java file was then designed so as to use the database helper to access the table or create it if necessary. All XML elements requiring monitoring or manipulation were initialized alongside shared preferences so the current step count could be accessed without directly using the sensor and the database helper was also initialized.

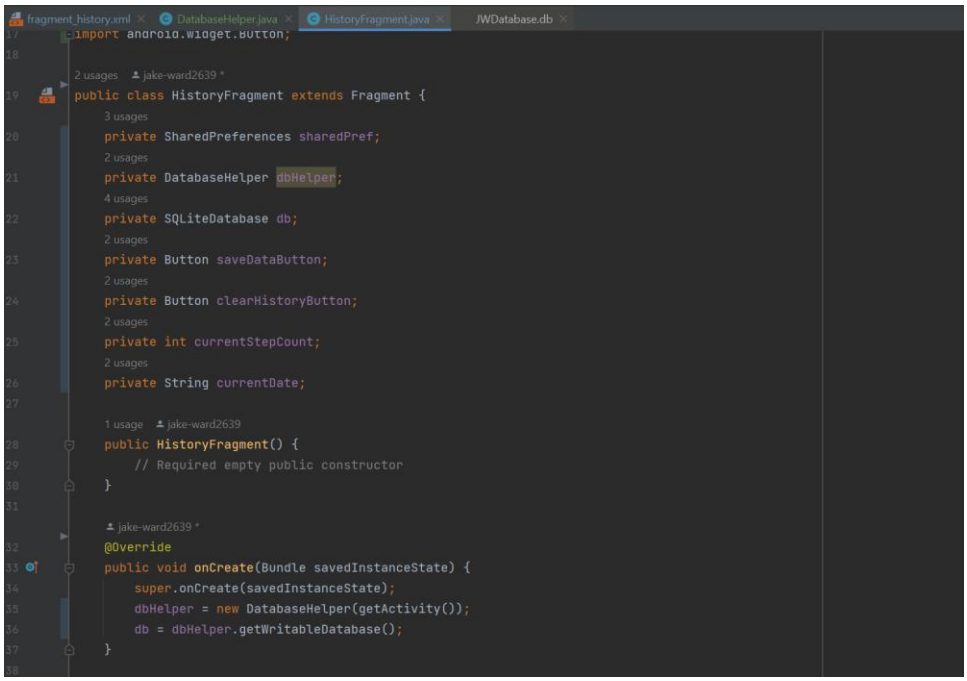


Figure 53: history fragment java utilizing database helper class

To ensure everything would work the insert and delete query functionality needed to be set up before displaying the results of the table. TODO statements were added everywhere in the code that the displayed results would need to be updated in accordance with the table. A listener for the save button would take the current date and step count from shared preferences and insert them as a new row in the table. A similar listener was added for the clear history button which would simply wipe the table. A required closing of the database would be carried out on destruction of the fragment. This would allow a user friendly and easy to test UI demonstrating storage with SQLite after the displaying of results had been implemented.

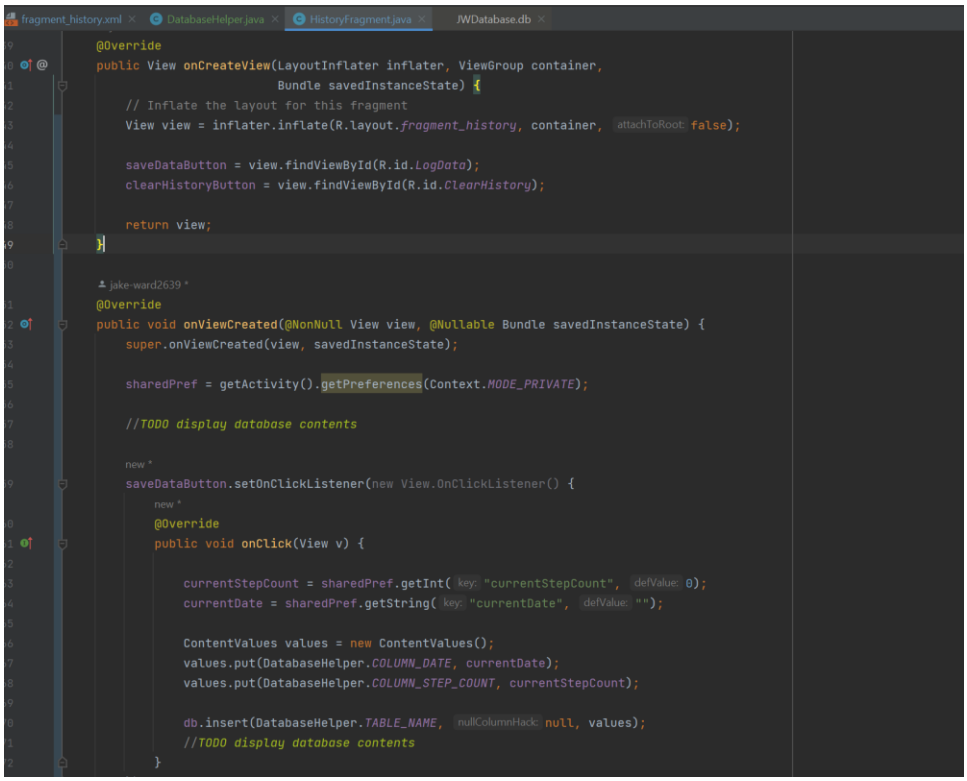


Figure 54: history fragment java handling insert queries with TODO statements for displaying table contents

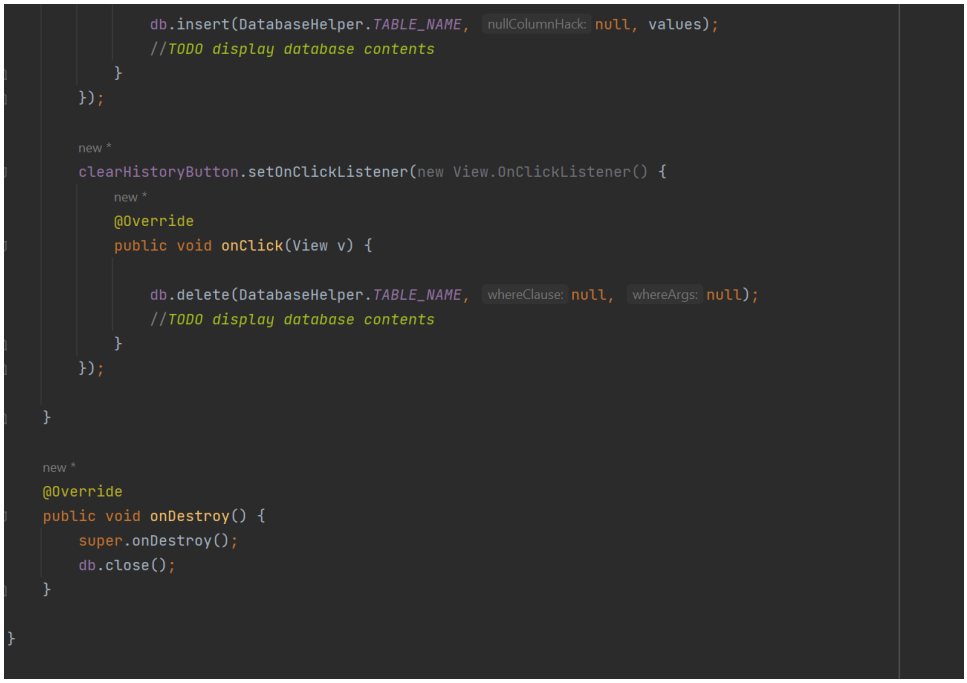


Figure 55: history fragment java handling delete query and closing the database connections with TODO statements for displaying table contents

Without the planned display of results, the database could still be viewed from the phone's/device's files using the device manager in android studio. An extension was added to android studio to allow the reading of database files and the additions to the table were both visible and accurate, proving the code to be functional.

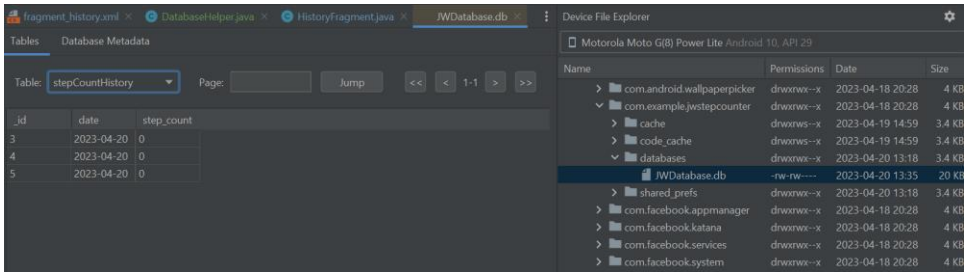


Figure 56: successful test of database/table creation with inserted data

As previously stated, the recycler view would require an unnecessarily complex method to display the results within, so the arguably better presented and more user-friendly list view was used instead. It was added to the XML file for the fragment, contained above the buttons so that it would never be hidden behind them and the element had scrolling enabled so that the results could still all be viewed if their height was longer than the dimensions of the phone.

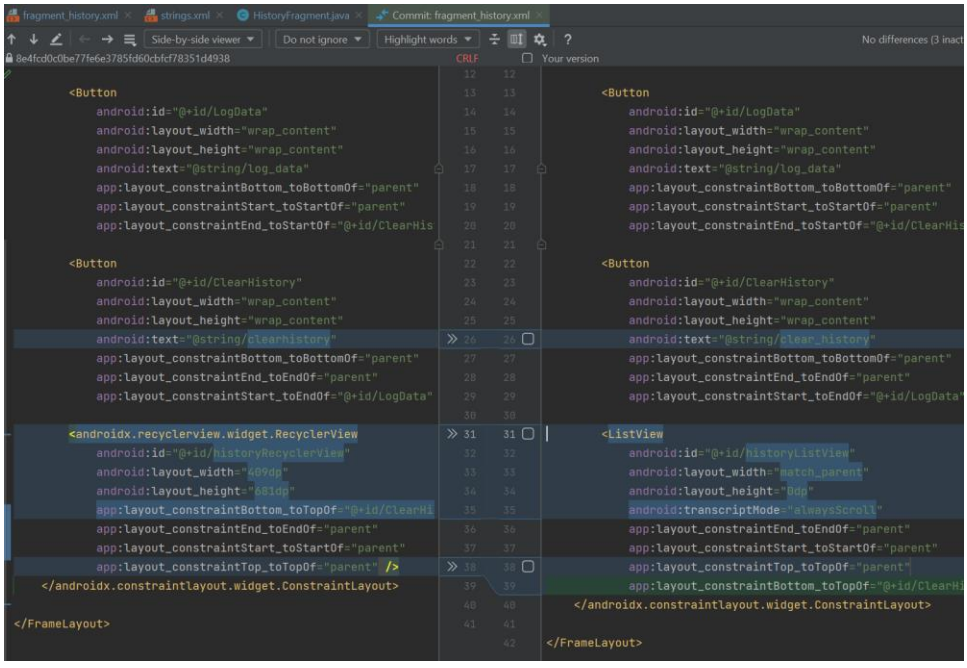


Figure 57: replacing recycler view with list view

The method I was designing to display results from the table onscreen wouldn't require and differentiation between uses so all TODO statements were replaced with a call to the method with only the view required seeing as how it wouldn't be defined in the method otherwise.

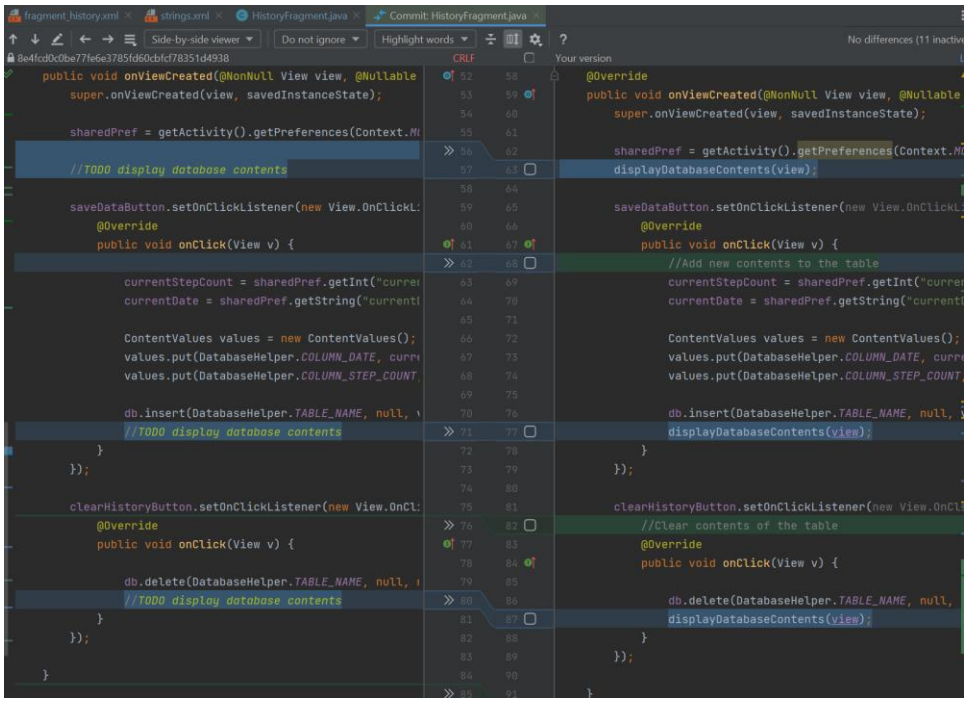


Figure 58: calling method for displaying table contents in list view

A cursor and array list were defined to get the results out of the database and populate the list view. The cursor was set to move through every result in the table and add them to the array list one by one until the final entry had been reached. The cursor would then be closed and the list view would be populated via an adapter and the contents of the arraylist.

```
private void displayDatabaseContents(View view) {
    // Retrieve data from the database
    Cursor cursor = db.query(DatabaseHelper.TABLE_NAME, null, null, null, null, null, null);
    ArrayList<String> dataList = new ArrayList<String>();
    if (cursor.moveToFirst()) {
        do {
            String date = cursor.getString(cursor.getColumnIndexOrThrow(DatabaseHelper.COLUMN_DATE));
            int stepCount = cursor.getInt(cursor.getColumnIndexOrThrow(DatabaseHelper.COLUMN_STEP_COUNT));
            dataList.add(0, date + " - " + stepCount);
        } while (cursor.moveToNext());
    }
    cursor.close();

    // Display data in the ListView
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(getContext(), android.R.layout.simple_list_item_1, dataList);
    ListView listView = view.findViewById(R.id.historyListView);
    listView.setAdapter(adapter);
}
```

Figure 59: method for displaying table contents in list view

Figures 60 – 62 display this method functioning with results added, being clear and new results being added again. The table would be displayed whenever a change had occurred to it. However, it was found that it wasn't as user friendly as predicted due to the latest results being added to the bottom of the list view rather than the top. This meant that the latest results would likely eventually be off screen which would prevent the user from seeing immediate results. As such, many methods were attempted in the XML file to flip the order, but the decided solution was to simply add results to the array list at the start rather than the back. After everything was found to be functional, the implementations were optimized for the whole application, and it was backed up to the GitHub repository with all the requirements having been met. The final task was to thoroughly test the application once more and bug-fix where appropriate.

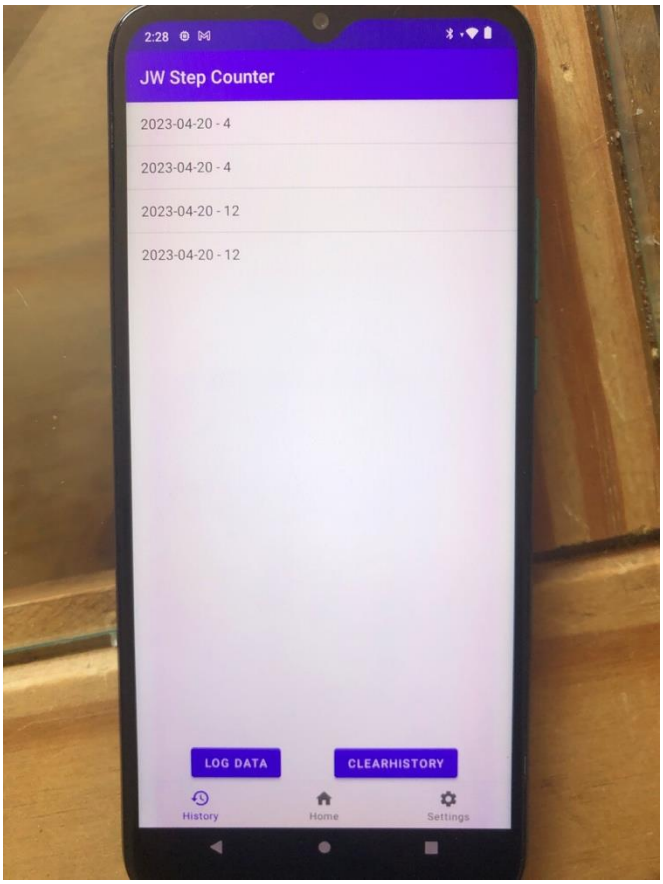


Figure 60: successful test of logging and displaying history

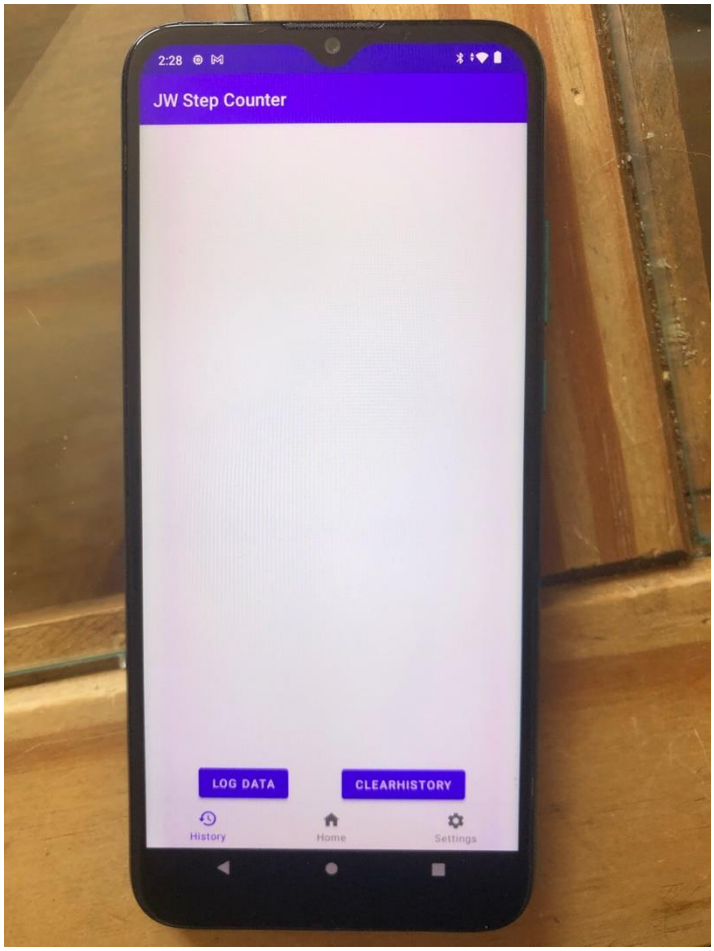


Figure 61: successful test of clearing and displaying history

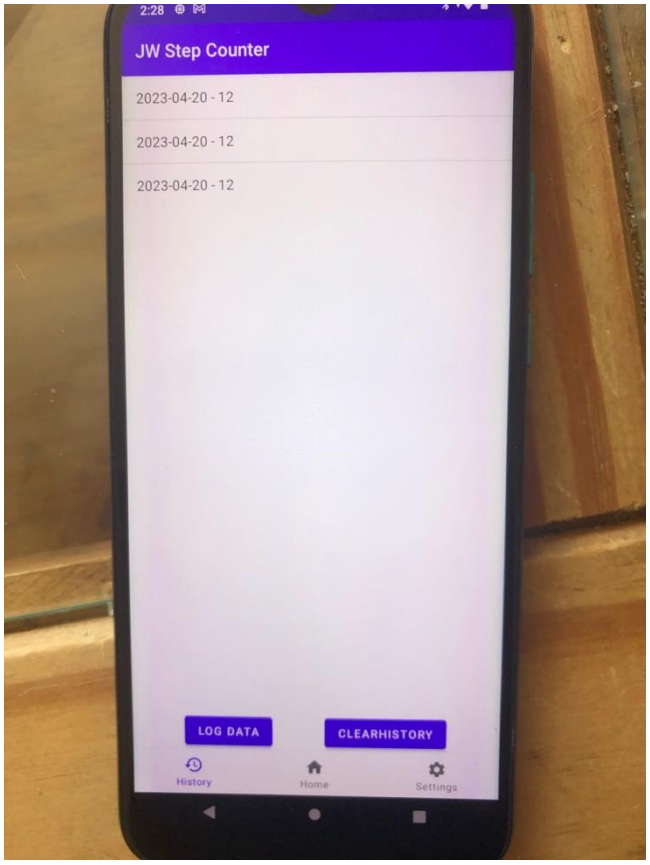


Figure 62: successful test of logging and displaying history after clearing

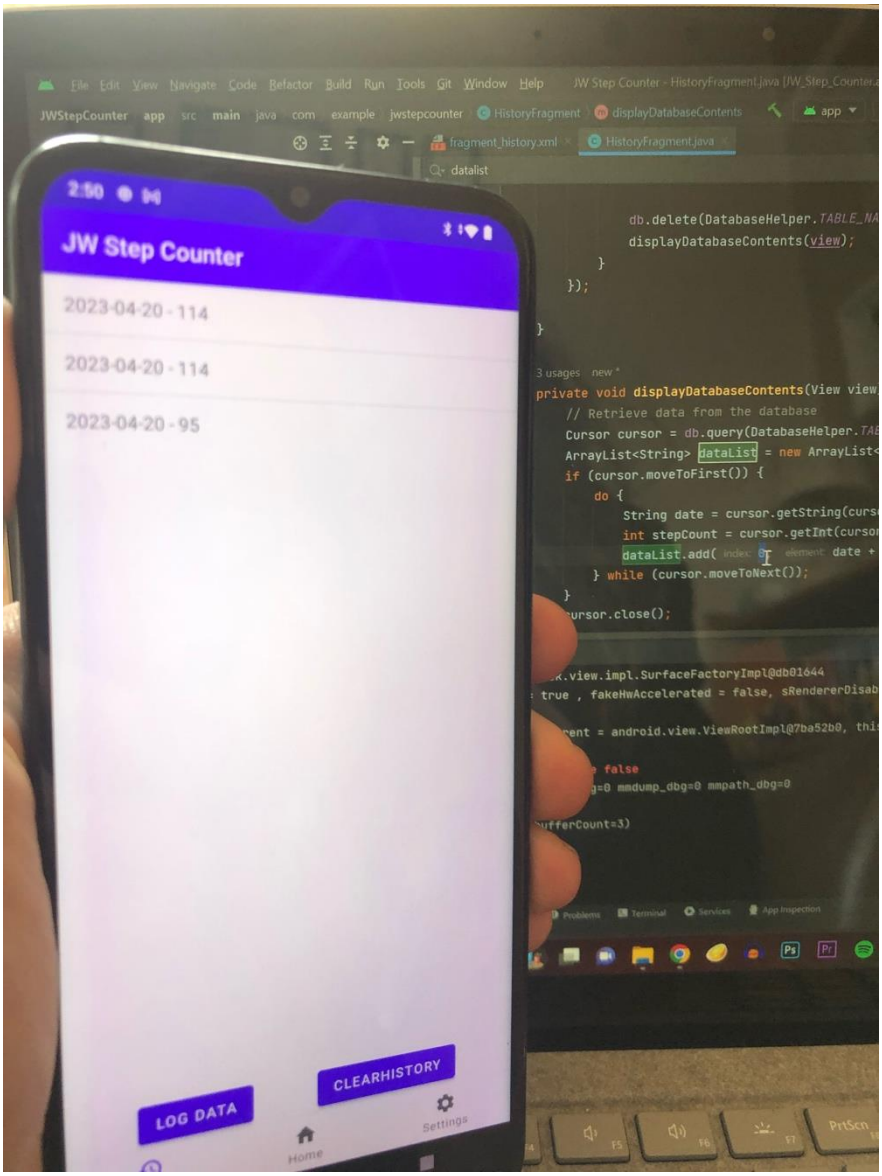


Figure 63: successful test of logging and displaying history from top to bottom by adding to array list with index 0

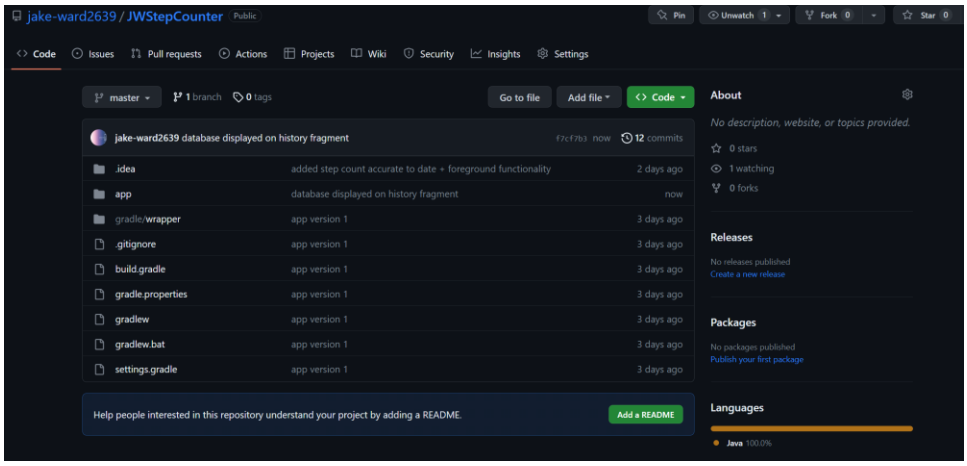


Figure 64: pushing fully functional app to GitHub repository

4.10 Bug Fixes

With extra development time the bug regarding the step goal edit text preference could be addressed. With the poor documentation I was forced to browse developer forms for a solution. Through stackoverflow a solution presented itself as it seemed my current solution stopped functioning sometime around 2019. This solution involved programmatically setting the input type of the preference on binding. A test with the code seen in figure 65 caused the correct keyboard to display and disallowed symbols (as can be seen in figure 66).

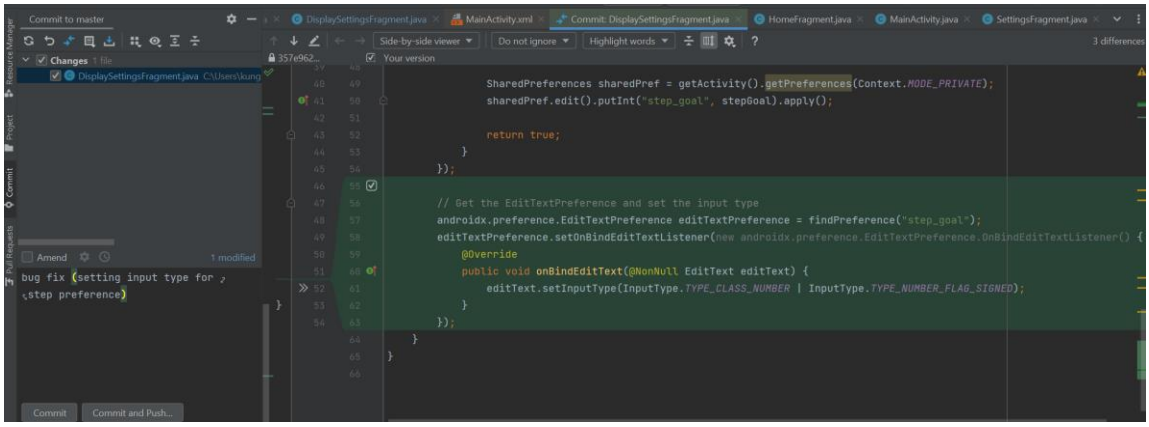


Figure 65: pushing fully functional app to GitHub repository

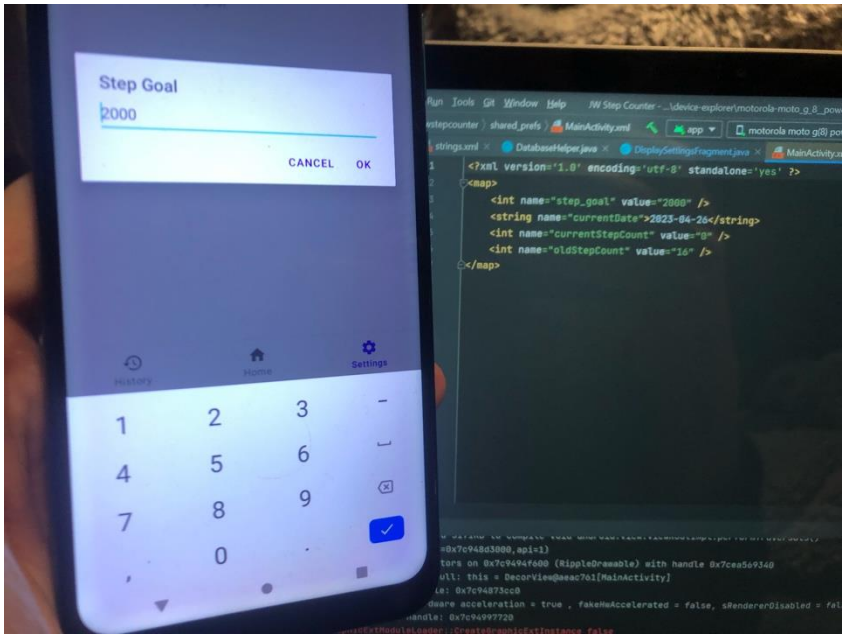


Figure 66: pushing fully functional app to GitHub repository

5. Testing

Testing was an incredibly important part of the project as not only does it ensure stability of the application, but it also functions as the success criteria for it. As such I documented all tests in a test-requirement matrix which can be seen in the below table. The table features the tests in chronological order of their execution. There were 4 types of test which were not mutually exclusive;

- Black Box testing – Often defined as when the tester has knowledge of the internal workings of the product, this is the label for all tests done by myself as the sole developer to ensure that the functionality implemented functions in the expected capacity.
- White Box testing – The antithesis of Black Box testing, White Box testing has the testers in the dark about the internal workings of the application. This is the label given to all tests carried out with potential users, used to ensure completeness by checking if other users may find erroneous or unintended execution paths missed by myself as the developer.
- Regressive tests – Tests that were redone whenever code was altered which could have theoretically affected the code that the test was testing. The goal is to ensure that new additions do not affect previous working functionality. For more efficiency in the report, these tests have only been written about once, but it should be noted that they were tested and then retested every time the code was significantly altered.
- Progressive tests – Labels tests that mark new functionality and the testing done to ensure that it is working as intended.

Tests labelled with conflicting labels only means that the same test was carried out in those different ways. For example, a test labelled as both white and black box means that the same test was executed with the developer who knew about the applications functionality but also with testers who did not know. Also, a test labeled as both regressive and progressive means that when it was first implemented it was progressive but would become a regressive test that would be retested frequently as well.

As a reminder to help navigate and understand the table, here are the preestablished requirements with their IDs which were used in the test-requirement matrix;

Must Have

1. Navigation Menu (Advanced UI)
2. Fragments: Home, History and Settings (Advanced UI)
3. Request Access to Activity Sensors
4. Step Counter (Sensors)

Should Have

5. Snackbar notification on denied access (Advanced UI)
6. Progress Bar
7. Step Goal (Shared Preferences)

Could Have

8. Dark Theme
9. History (SQLite Storage)

Requirement & Test Type	Test Description	Test Objective	Test Steps	Expected Result	Actual Result	Pass /Fail
N/A Black Box Progressive	Backup process is in place	Ensure GitHub is Connected to Android Studio to setup the backup process	Create Application in Android Studio, publish it to GitHub using account linked via token, consult GitHub	Generated Application is found in a repository under the same name of the project and account	Same as Expected	Pass
1, 2 Black Box White Box Progressive Regressive	Fragments are Navigatable	Ensure the bottom menu bar hold the correct icons, names, selected icon and navigates to the correct fragment	Launch application, check selected icon, click an icon, identify text on screen, identify selected icon, repeat for all icons	The selected icon is always the same as its associated fragment	Same as Expected	Pass
3 Black Box White Box Progressive Regressive	Granting activity sensor permission	Ensure that the step counter can access the activity sensors	Open the activity sensor and click yes if a pop up appears	If the device is running an API level above 29 then access will not automatically be granted, and a popup will appear asking for permission	Same as Expected	Pass
4 Black Box White Box Progressive Regressive	Measuring step count	The step count is accurately measured through the sensor of the same name	Open the application, walk several steps taking note of the amount until the step count on screen updates, then compare the sums	The results will be similar and thus accurate	Much greater latency than expected	Pass
7 Black Box White Box Progressive Regressive	Shared preference usage (launch)	Ensure shared preferences displays the last known value on launch	Launch application, take enough steps that the step count text view changes, close	The text view will be the same as when it was closed on launch	Same as Expected	Pass

			the app, reopen it, observe text view			
7 Black Box White Box Progressive Regressive	Shared preference usage (new day)	Ensure shared preferences displays the step count per day	After the previous test close the app and change the date on the device, restart the application and observe the text view and shared preferences (viewable through the device manager)	The text view reflects the current step count (which will be zero on launch but updated when a change is detected in to sensor). The old step count will reflect the sum of the previous days step counts and the date will reflect the current date	Same as Expected	Pass
7 Black Box White Box Progressive Regressive	Shared preference usage (reboot)	Ensure shared preferences resets after reboot	After the previous two tests restart the device, then launch the app	Shared preferences step count is reset back to zero and the text on screen is as well	Same as Expected	Pass
3, 5 Black Box White Box Progressive Regressive	Snackbar notification on denied access	Make sure an indefinite notification is displayed on denial of permission that can be swiped away and can open the settings to change permissions	Deny permissions to the application in settings, open the application, click deny on the pop up, swipe away notification, reopen application but follow the link on the notification instead	Notification can be swiped away, will stay for an indefinite period and when click will take the user to the settings of the application to enable permissions	Same as Expected	Pass
4 Black Box White Box Progressive Regressive	Measure step count with app paused	Ensure that the step count sensor will still function in the likely scenario that the screen turns off due to time out	Open the application and suspend the app for a while, resume the app and review the step count after it updates, repeat the process with the screen closed instead	Sensor will have still been registered and so the step count will still be accurate. It will not unregister until the application is destroyed of a fragment other than home is navigated to	Same as Expected	Pass
N/A Black Box White Box Progressive	Custom Icon	Ensure the custom launcher icon is applied	View the app on the menu screen of the device, open the app and suspend it	The custom icon will be visible in both scenarios rather than the default android app icon	Same as Expected	Pass
6 Black Box White Box Progressive Regressive	Circular progress bar reflects progress	Ensure the progress bar accurately shows a visual representation of the progress defined in the text view	Open the application and take steps until the text view updates	The progress bar will fill up accordingly with an assumed step goal (max progress) of 200 steps	Same as Expected	Pass
7 Black Box Progressive	Accounting for new day + opened with no steps taken + reopened	To detect a logic error in the functionality of the step count and shared preferences	After a new day open the app, close it, then reopen it and observe the text view	It will execute the same way as in the test named “Shared preference usage (new day)”	As suspected the previous days step count is displayed until it updates. To fix this the current step count in shared preferences is reset to 0 on a new days detection not just on reboot.	Fail

8 **Black Box** **White Box** **Progressive** **Regressive**	**Dark mode on launch and in settings**	**To ensure dark theme is used in the appropriate situations**	**Open the application on low power mode, switch back to normal mode, enable dark mode in the apps settings, close and reopen the application**	**Dark mode will be enables on low power mode or when enabled in settings**	**Same as Expected**	**Pass**
7 **Black Box** **Progressive**	**Step goal is saved to shared preferences**	**To ensure a step goal is set in shared preferences**	**Open the application, navigate to settings, click on the step goal, enter a step goal, consult shared preferences via the device manager**	**Shared preferences will have the step goal recorded**	**While the step goal is saved to shared preferences, the keyboard isn't limited to numbers, if anything other than a valid number is entered then the application crashes**	**Fail**
6, 7 **Black Box** **White Box** **Progressive** **Regressive**	**Circular progress bar is affected by step count**	**To ensure a step goal is set in shared preferences and it is used to set the maximum progress of the progress bar**	**Open the application, navigate to settings, click on the step goal, enter a step goal, return to home and observe the progress bar (if step count is zero then take steps until it updates)**	**Shared preferences will have the step goal recorded which is then in turn used to define the maximum progress for the progress bar**	**Same as Expected**	**Pass**
9 **Black Box** **Progressive**	**SQLite database creation, insert and delete queries**	**Proof of concept that the SQLite database storage will function**	**Open the application, press the buttons to add entries to the database, consult the device manager to find the database, click the wipe database button**	**A table will have been created under the applications database with the entries, they will be cleared upon the second button being clicked, a new table will not be made if it already exists**	**Same as Expected**	**Pass**
9 **Black Box** **White Box** **Progressive** **Regressive**	**History tab functions**	**Ensure the history from the database is visible in the history fragment**	**Open the application, press the buttons to add entries to the database, click the wipe database button**	**The entries will display and persist on screen, over multiple days, until the wipe history button is clicked**	**Same as Expected**	**Pass**
7 **Black Box** **White Box** **Progressive** **Regressive**	**Step goal keyboard is limited to numbers**	**Ensure removal of the bug from the step goal function in settings**	**Attempt to enter an invalid step goal**	**Keyboard is limited to characters that would create a valid float, nothing entered can crash or break the application**	**Same as Expected**	**Pass**
N/A **Black Box** **Progressive**	**All regressive tests function after SDK 33 is enabled**	**To ensure that the less stable version of android didn't cause any issues**	**Restart Gradle, check that no additional errors are there, install on a device running SDK 33, execute all previous tests marked as regressive**	**All regressive tests function after SDK 33 is enabled**	**Same as Expected**	**Pass**

Table 3: Test-Requirement Matrix

6. Evaluation

6.1 Mobile User Interface Design

As mentioned in the concept document, one of the selling features of the application to differentiate itself from the rest of the market was its simplistic UI design. This is due to the fact that most fitness-based applications overcomplicate their applications, thus forcing an unwanted learning curve onto the user. I believe a simplistic but useful UI design was created through the following practices;

- **Conventional Design** – By using conventional design for the application, it lessens the burden on the user of learning how to use the app. This is because they will be likely to have already encountered similar features and will understand how they work from memory. A prime example is how a bottom navigation menu was used to navigate between tabs and the icons chosen to represent the different tabs are the conventional icons for history (an arrow and clock), home (a house) and settings (a cog). Through their use, the user can instantly see and understand where different parts of the functionality of the app lay and what to click to navigate to them. I think the layout of the app fit conventions to be as clear as possible, with the main functionality being at the centre of the initially loaded page while the other two tabs display from top to bottom to show that they are not the main focus of the app. The progress bar effectively displays the amount of steps left until the goal has been reached as well as the amount of steps so far which is also clearly written on screen.
- **Material Design and Advanced UI Features** – I believe the snackbar notification was a worthwhile feature to have included because it can be forcibly removed by the user via swiping, meaning that it does not intrude upon the rest of the application, but can stay indefinitely to notify the user. Using fragments instead of activities helps enforce the idea that the user is still within the same application, which is not always clear when having switched activities. Little animations also help improve the UX and subconsciously express functionality, such as the ripple effect when clicking a button, the growing curve connoting that a list cannot be scrolled anymore and the animation of the progress bar upon load.
- **Scalability** – By using constraints on all UI elements it made sure that the UI design would be scalable for different orientations as well as different device sizes.
- **Themes –** By allowing a dark theme based on device and app preference the user can choose whichever theme they prefer. Many users often opt for a dark theme as it is not as obtrusive at night, but a light theme should still be included because the app is likely to be used outside, prompting light mode to be used to increase visibility.

Through all of these practices and design choices a feel that the UI successfully displays the functionality of the application in a clear and concise manner. Focusing attention on the important features without any errors and with a combination of both inherent and purposeful animations improving the UX.

6.2 Advanced Mobile Technical Features

As previously mentioned there were three main advanced features added to the project in the form of sensor monitoring, SQLite storage and advanced UI design. Evidence of all claims can be found in screenshots in the implementation section of this document as well as the app itself submitted alongside the report as both the codebase and an installable APK file.

The advanced UI design entailed dynamically bound fragments tied to a bottom navigation menu. The advantages of this include the ability to switch between different views/fragments of the app easily and quickly, as well as the ability to maintain the state of each fragment separately, allowing for more efficient memory management. Additionally, the use of a bottom navigation menu allows for intuitive and familiar navigation for users as it is a widely adopted conventional design across many mobile apps which has in turn shown the design to be effective in providing intuitive navigation for users. While not made use of in this particular instance, in addition to familiarity and efficiency in memory management, the use of dynamic fragments could also be useful for future additions due to their flexibility and reusability meaning they could be used across several activities while fitting different UI responsively. The use of dynamic fragments also supports potential future development due to the fact that it modularizes code, making it more maintainable over time However, in some ways I regret using dynamic over static as static fragments have a much less complex implementation and dynamic fragments are destroyed and created constantly making lifecycle management a long-winded and strenuous task (especially when working with values that need to be unassigned for efficiency, like sensor listeners and database connections). This task was accomplished without error, the only critique of a design decision could be that the home fragment needs to be open to continue counting steps, however, it was designed this way because users are unlikely to use the other fragments for long periods of time or while walking/running, meaning that this shouldn't negatively affect users. The reason this is the case is because the sensor listener cannot be left open as it would drain the battery and so is unassigned on the destruction of the home fragment.

The sensor monitoring itself was the main feature of the application, but its implementation came with a few issues. The step count sensor was implemented to measure the users step count and display it on the home fragment. The value of the step counter sensor couldn't be read outside of the sensor event listener and so it was placed in the home fragments java code, causing the aforementioned issue of the step count not being monitored when other fragments are open. However, in hindsight if the functionality had been placed within the main activity's java code, then the apps reading of the step count wouldn't be reliant on the home fragment being open. If this design choice was made however, then other issues would have arisen such as monitoring in the fragment when to update the on screen step count, or alternatively how to target a dynamic fragment to change its step count seeing as how dynamic fragments do not have set IDs (an issue experienced in the previous year's app development module hence its avoidance). Additionally, as previously mentioned there was a dramatic latency that was much larger than expected between updates of the step count. Although, the sensor was found to be accurate as advertised so this wasn't a major issue. Overall, the features implementation was successful, however in the future it may be useful to entertain the alternative solutions to this kind of implementation. For future developments it may also be worth attempting to design a step counting algorithm based off a hardware sensor like the accelerometer to avoid the latency issues, however that decision may come at the cost of accuracy. The process of sensor monitoring itself was easy to implement and would defiantly be considered for future projects. This feature also forced learning on the subject of permissions due to android's increased security in the area requiring permission to be given at runtime to access sensors.

Finally, the implementation of an SQLite database was executed without issue, allowing users to log their history at will to the database to be consulted at a later time or date. The storage was originally going to be done at the end of each day or the beginning of a new day, however it was decided that it

would be better design to allow the user to choose when they wanted to log their history by adding the functionality to a button, because they may want to abstain from saving it to their history and users might not want to wait for a new day to see their results in their history. In addition, by designing it in this way it would be easier to test as results are instantly added to the database. The database was also originally going to be implemented via firebase as opposed to SQLite due to its conventional usage at an industry standard and its easy implementation. But when it was realized that this would then require a method of uniquely identifying each use and the history fragments functionality would then become dependent on an internet connection not likely to be available while the user is exercising, SQLite was selected as the more suitable method.

Overall, the advanced features added to the mobile application project were successfully implemented, and provided valuable insights and lessons for future development. The use of dynamic fragments in the UI design allowed for flexibility and modularity, while the step count sensor monitoring and SQLite database implementation provided useful functionalities for the users. However, there were also some limitations and trade-offs that were encountered during the implementation, such as the latency issue with the step count sensor and the complexity of managing the android lifecycle with dynamic fragments. The implementations of these features have taught me specifics such as how to implement each feature as well as overall lessons such as the importance of planning out features before implementation, the need for alternative solutions in case of unexpected circumstances, managing the android lifecycle and practicing functionality with efficiency on devices.

6.3 Testing Strategy

When evaluating the testing of the application I find that while it was extremely in-depth and a near constant practice it could have been documented more effectively. The tests could have featured recordings of tests as well as having dated entries about every time a test was executed. However, this was not the case for two reasons, one being that the devices used for testing were not commonly owned by myself and so recording would be an overstepping of boundaries and the other more impactful reason being that this would be a massive time sink. The application by nature would involve walking around executing long tests to ensure accuracy and so many tests were executed so frequently that development time would have been dramatically affected. So, for project management reasons testing was reduced to the declarations of the different types of tests carried out so that documentation wouldn't waste as much time.

There different type of tests actually carried out ensured that testing was comprehensive, covered all areas and that the application would be easy to use for the consumer. Over 15 different people were consulted for black box tests however they cannot be named as a lack of anonymity would cause ethical issues. Tests were separated between black and white box to ensure the every execution path was explored to prevent errors and regressive tests were constantly carried out every time a new feature was added just to make sure that no code/functionality had been negatively affected.

These methods were proven to have been successful on small occasions such as checking that functionality had indeed been implemented in the predicted way but displayed undeniable usefulness when uncovering the aforementioned error concerning the step goal inputs keyboard not being restricted.

6.4 Project Management & Monitoring

Obviously due to the fact that this was an independent project there was no need to manage a team of people however it was still important to manage and monitor the project's progress to ensure that it was completed within the given timeframe and for general practice.

GitHub was constantly used to backup progress and gain the advantages of version control. In this way no progress could be easily lost and progress could be rolled back if necessary. GitHub also allowed the tracking of progress through named commits and statistics. In an environment with more than one developer the process would have involved different branches being merged into the main branch, which would have prevented clashes in implementations, but in this scenario, it wasn't necessary. Statistics can also be gained from GitHub to measure the progress of the project in terms of codebase, issues, and pull requests. Additionally, if GitHub was to be used in a real world scenario with multiple developers, then said developers can easily review and comment on each other's code, and suggesting improvements or fixes. This helps to ensure that the code is of high quality, and reduces the chance of errors. GitHub is robust enough that it was and remains a reliable source to back up the progress of the application to, although just to be careful the applications codebase was also stored on multiple physical devices. All changes were committed and pushed at the end of every new but tested implementation.

A Kanban Board created on Trello was also used to measure progress. The kanban board helped with project management by providing a visual overview of the project's workflow, allowing me to note and easily view what tasks are in progress, what tasks are pending, any deadlines, and what tasks have been completed. I found this feature to not be suitable for such a small development task but with more developers and future development I can understand how it can increase efficiency in the workflow.

A Gantt Chart was initially made to map out the schedule. This helped with project management despite being a team of one because it provided a clear and comprehensive overview of the project timeline and allowed the prioritization of tasks to be mapped out and displayed. The Gantt chart was used to map out dependencies between tasks, helping avoid unnecessary delays in development

The requirements and subsequent requirement-test matrix were used in project management to measure success and ensure that the final product met the desired specifications. By defining clear requirements and creating a requirement-test matrix, I was able to track progress and ensure that each requirement was being met as the project progressed. This would also cause preplanning of features preventing error later on in development. Then the requirement-test matrix provided a clear and demonstratable framework for testing and validating the final product through its predefined requirements. Additionally, these requirements were prioritized according to dependance on one another but also importance to the project, helping to ensure that a successful project was developed even in a worst-case scenario.

Similarly, a use case table was developed to pre-plan the development of the application to avoid logic errors causing major issues during development. Use case tables let me explore all potential executable paths although they are not always accurate as some aspects of development would be unpredictable in the early stages.

The final topic worthy of reevaluation when it comes to project development is risk analysis. A risk analysis table was created to explore the possibility of negative impacts on the project so that plans to

avoid, negate or minimize costs could be put in place. While nothing substantial went wrong, I feel it was appropriate to take these measures just in case the worst were to occur.

6.5 Reflection on Process

On the whole this project and its development have demonstrated a great many learning points. The obvious learned skills are the best practices and construction of the advanced features mentioned above in chapter 6.2, as well as revision of the construction of basic android applications. But further than that this project has taught unexpected lessons that will be useful for future development.

A large subject that will be taken note of in the future will be the advantages and disadvantages of using fragments with dynamic binding as opposed to static binding. This was again discussed in chapter 6.2 of the report but to summarize, dynamic fragments should be used in a scenario where the UI needs to be modified at runtime or when the application needs to support multiple screen sizes and orientations, whereas statically bound fragments should be used when the UI is fixed and known at compile-time. While statically bound fragments may have been more appropriate for this type of project in hindsight, their use from an educational standpoint I warranted due to their complexity.

It was learned that a sensors accuracy and delays are not always guaranteed and can vary dramatically depending on the device making use of them. This means that picking a sensor for your purpose requires a joint between research and experimentation to narrow down the best fit for the task.

I was made aware of more useful features such as shared preferences which allowed persistence between fragments. Their usage and potential were researched extensively and because shared preferences functions like a repository with the application folder it can be customized to hold different information in different areas to aid in organization.

Android documentation was not always up to date and so it was learnt that the best practice was to consult multiple sources before forming an opinion on what was possible and what was best practice.

I utilized Gradle integration of an outsider project to aid in developing a feature. So, while this feature was only a small implementation which could have easily been replaced with materiel design, it has demonstrated how useful unofficial open-source material from the community can be and how it could be utilized in the future.

Extensions were used in android studio helping especially with the development of the SQLite database, as android studio cannot read SQL files by default. This proved that reputable extensions can improve the design process dramatically if made use of in the correct fashion.

Project management skills were tested and in the future documentation will be more vivid so that testing will be more representative of the large amount of time that it took in the schedule.

I acknowledge that there are strengths and weaknesses alike present in the application built, however there is no single weakness that cannot be remedied at a later date and the strengths far outweigh the weaknesses. Below I have listed the various strengths and weaknesses of the application in an effort to demonstrate this.

Strengths of the Application:

- Simplistic and Recognizable UI – The UI design uses conventional and thus familiar design for navigation, allowing users to switch between different views/fragments of the app easily. The design is also visually appealing and easy to use alongside subtle animation providing a positive user experience.
- Fragmented UI – The UI design makes use of dynamic fragments which are created and destroyed through navigation. This helps to maintain the state of each fragment separately, allowing for more efficient memory management.
- Accuracy with Step Counting – The step count sensor was implemented to accurately measure the user's step count and display it on the home fragment. The sensor was found to be accurate, providing users with reliable information about their physical activity and progress.
- Visual Representation of Progress - The app provides a visual representation of the user's progress towards their step goals in the form of a large progress bar. This allows them to easily track their activity over time and helps to motivate users to reach their step goal.
- Step Goals – The app allows users to set step goals and track their progress towards those goals. This provides users with a clear target to aim for and helps to motivate them to stay active. The ability to customize step goals also provides flexibility for users to set goals that are appropriate for their fitness level and lifestyle.
- Dark Theme – Dark mode helps with viewing in the dark without straining the user's eyes as well as saving battery by using less bright colours. It can be turned off and on in the app but will also automatically be enabled if the device supports a low power mode.
- Save To and Clear History – The application is able to store a step count alongside the date so the user can track their progress in terms of a fitness regimen. This functions without reliance on access to the internet and can store a large amount of information. It can be cleared at any time if the user desires to.

Weaknesses of the Application:

- Limitations of Sensor – There are a couple limitations of the sensor. The step count sensor used has a much larger latency than it’s contemporary, the step detector sensor. However, this is the tradeoff for accuracy. The other limitation is that due to the current design of the application, the sensor listener is only registered while the home fragment is open, ideally this would be altered in the future, however, the application was designed around the concept, with no reason to leave the home fragment while pursuing fitness goals.
- Step Goal Keyboard – While the step goal’s text input field displays a keyboard used for numbers and disallows the input of other figures, it still leaves some characters visible. While not a fatal flaw, the keyboard displaying symbols which do nothing when clicked is bad for the UX as it connotes error.

The application has been built with future development in mind by providing a modular and scalable architecture which allows for easy integration of new features and improvements. Additionally, the code has been well-documented and organized, facilitating maintenance and further development by any other potential future developers. Below are listed a few of the future developments that could be carried out as well as potential methods of executing them.

Future development would include:

- Smartwatch – Simply put, the application could be redesigned in android studio to be suitable for smartwatches as a fitness application would likely be ideal for that type of device despite the niche market. This would only involve the detection of the type of device to implement which can be done by obtaining the device model on manufacturer provided by the Android SDK.
- Improve Step Count – The sensor currently only has a registered listener while the application holds the home fragment open. This could be changed to run until the application is closed by creating a service that runs in the background and continuously tracks step count sensor. This service could then broadcast the step count data to other components of the app through a messaging system akin to the LocalBroadcastManager. This may however warrant investigation before implementation as monitoring at all times will significantly reduce battery life. Therefore, an alternate path could potentially be to display information about how the application works or having a switch to turn the step count listener on or off.
- Limit Keyboard Further – The keyboard displayed when the preference for the step goal is being set is already limited to only apply to numbers. However, it can be limited further to only include the numbers themselves. This can be done by overriding the edit preference method and a method previously found on stackoverflow to create a custom keyboard. This feature would not be high in priority as the application already disallows these extra symbols displayed from being entered
- Calculate Calories Burned – It would be a relatively simple addition to calculate calories burned as the settings could have all relevant data entry points required (such as sex, age, weight, etc.) and then the step count could be used alongside it in an algorithm to calculate and display. This task was only not carried out due to time constraints as this task would involve a large time sink for what is essentially a very simple feature.
- Firebase Conversion – The largest potential future update would involve adding a login system so that data could be logged online, replacing or acting the same as the SQLite database. This way history could persist over devices and leaderboards could be implemented.

The applications repository can be found at https://github.com/jake-ward2639/JWStepCounter

References

Clifton, I.G., 2015. *Android user interface design: Implementing material design for developers*. Addison-Wesley Professional.

Hagos, T. and Hagos, T., 2018. Android studio. *Learn Android Studio 3: Efficient Android App Development*, pp.5-17.

Moroney, L. and Moroney, L., 2017. The firebase realtime database. *The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform*, pp.51-71.

Lister, C., West, J.H., Cannon, B., Sax, T. and Brodegard, D., 2014. Just a fad? Gamification in health and fitness apps. *JMIR serious games*, *2*(2), p.e3413.

Muntaner-Mas, A., Martinez-Nicolas, A., Lavie, C.J., Blair, S.N., Ross, R., Arena, R. and Ortega, F.B., 2019. A systematic review of fitness apps and their potential clinical and sports utility for objective and remote assessment of cardiorespiratory fitness. *Sports Medicine*, *49*, pp.587-600.

Zhou, M., Mintz, Y., Fukuoka, Y., Goldberg, K., Flowers, E., Kaminsky, P., Castillejo, A. and Aswani, A., 2018, March. Personalizing mobile fitness apps using reinforcement learning. In *CEUR workshop proceedings* (Vol. 2068). NIH Public Access.

Mew, K., 2015. *Learning Material Design*. Packt Publishing Ltd.

Roza, A.M. and Shizgal, H.M., 1984. The Harris Benedict equation reevaluated: resting energy requirements and the body cell mass. *The American journal of clinical nutrition*, *40*(1), pp.168-182.

Aditya, S.K. and Karn, V.K., 2014. *Android sQLite essentials* (pp. 88-90). Packt Publishing.

Chang, Y., Hong, L., Su, G. and Leng, X., 2018, March. Design and Development of Mobile Academic Platform in Universities. In *2018 International Conference on Mechanical, Electronic, Control and Automation Engineering (MECAE 2018)*. Atlantis Press.

Mikhael , L. (no date) *Lopspower/circularprogressbar: Create circular progressbar in Android*, *GitHub*. Available at: https://github.com/lopspower/CircularProgressBar (Accessed: April 24, 2023).

Hongman, W., Xiaocheng, Z. and Jiangbo, C., 2011, October. Acceleration and orientation multisensor pedometer application design and implementation on the android platform. In *2011 First International Conference on Instrumentation, Measurement, Computer, Communication and Control* (pp. 249-253). IEEE.

Au, K.W.Y., Zhou, Y.F., Huang, Z. and Lie, D., 2012, October. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security* (pp. 217-228).

Wilson, J., 2013. *Creating Dynamic UI with Android Fragments*. Packt Publishing Ltd.

Murphy, M.L. and Murphy, M.L., 2010. Using Preferences. *Beginning Android 2*, pp.213-224.

Allen, G. and Allen, G., 2015. Using Preferences. *Beginning Android*, pp.327-342.

Vogel, L., 2010. Android sqlite database and contentprovider-tutorial. *Java, Eclipse, Android and Web programming tutorials*, *8*, p.5.