# CI660 - Fitness Application Concept Document

By Jake S.T. Ward

## Elevator pitch

My proposed project is a fitness-based android application that utilizes sensors to track footsteps and records the data for the user's fitness regime. The app will be developed using Android Studio for Android 12 (API level 31) and will be compatible with a wide range of devices. The application will also utilize data communication using BroadcastReceiver and threads/Handler. The UI design will make use of material design to be simplistic and user-friendly. The application may implement speech recognition.

## Rationale

Physical fitness plays a critical role in maintaining a healthy lifestyle, and many potential users struggle to stay on track with their fitness goals due to lack of motivation. My proposed app aims to address this issue by providing a user-friendly, efficient, and accessible platform to track fitness goals. My target users are individuals of all ages who want to track their daily fitness activity without the need for an external device, such as a fitness tracker.

Current fitness apps available on the market have varying features, and some are often complex, requiring users to navigate through various options to get the data they need. My proposed app will be designed with a simple user interface that focuses on the essentials, making it easy for users to access and understand the information they need.

## Application features

The app will utilize the sensors available on Android devices to track the user's footsteps and record the data for their fitness regime. The app will also allow users to set goals and targets for daily activity and provide them with notifications when they meet their set targets. The app will include features such as a dashboard to track daily activity, a history tab to show progress over time, and potentially a profile tab to view personal information. The app will use steps taken and profile information to list calories burned and all data will be stored after the date changes.

The app will also make use of Material Design for the construction of UI elements such as activity views/layouts, buttons, in-app notifications, etc. This allows for an easy-to-use interface due to an inherently minimalistic and flat design. The app could also feature the use of an SQLite table to store information effectively and potentially voice recognition to enter information like step goals. These extra features would be added if there was an excess in development time and the product was both stable and finished early.

In terms of competitor research, I have analyzed several apps such as Google Fit, Fitbit, and Strava. These apps have a wide range of features, but they can be overwhelming for some users. My proposed app will be designed with simplicity in mind, making it easy to navigate and understand.

Google Fit is a health and fitness tracking app that allows users to set fitness goals, track their activity, and monitor their progress over time. The app integrates with other popular fitness apps and devices, such as MyFitnessPal and Android Wear, to provide users with a comprehensive view of their health and fitness data. Google Fit also features personalized coaching and tips to help users reach their fitness goals.

Fitbit is a popular fitness tracker that helps users monitor their physical activity, sleep, and nutrition. The app tracks steps taken, calories burned, and distance traveled, as well as other metrics such as heart rate and sleep quality. Fitbit also allows users to set fitness goals, log their meals and water intake, and connect with friends for added motivation and accountability.

Strava is a social fitness app that allows users to track their workouts and share them with a community of fellow athletes. The app features detailed GPS tracking for running and cycling, as well as other sports such as swimming and skiing. Strava also includes features such as personalized training plans, leaderboards, and challenges to help users stay motivated and engaged in their fitness routines.

Overall, each of these apps has its own unique set of features and benefits for users. However, as I briefly mentioned, they can be overwhelming for some users. My proposed app's emphasis on simplicity and ease of use could be a significant selling point for people who are looking for a straightforward, user-friendly fitness app.

## Technical overview

My proposed application will be developed using Android Studio for Android 12 (API level 31). The app will utilize the BroadcastReceiver and Handler APIs for data communication between the app's components. The app will use sensors available on Android devices to track the user's footsteps and record the data for their fitness regime.

Utilizing the BroadcastReceiver and Handler APIs for data communication is efficient because these APIs provide a mechanism for inter-component communication in Android applications, allowing different parts of the application to communicate with each other. The BroadcastReceiver API allows an application to listen for broadcast messages from the system or other applications, while the Handler API allows an application to send and handle messages and runnable tasks.

By using these APIs, the application can receive and process messages and updates in real-time, without the need for constant polling or manual updates. For example, if the user changes a setting or their fitness goal, the application can use a BroadcastReceiver to receive the update and immediately adjust its behaviour, without the need for a user-initiated refresh. Similarly, by using the Handler API, the application can schedule tasks to run in the background or on a separate thread, ensuring that the user interface remains responsive, and the application remains performant. For this reason, all heavy-loaded tasks/functions should be delegated to workers as good practice and handler.post can be called to delegate tasks/functions to the main thread.

A drawback is that giving priority to broadcasts appears to no longer be supported in the latest android version (as of Android 14, SDK 33) despite the official documentation not being updated. While this is inconvenient to many developers, it would not prevent any of the listed features from being implemented.

Sensors can be used to track steps in a fitness-based Android application by utilizing the accelerometer sensor that is present in most modern smartphones. The accelerometer is a motion sensor that measures the acceleration forces acting on the device. It can detect changes in movement and orientation, including the user's footsteps.

To track steps using the accelerometer sensor, the application needs to monitor the changes in acceleration and calculate the number of steps based on the patterns detected. There are different algorithms and techniques to achieve this, and some of the most commonly used techniques are:

1. Threshold-based step counting
2. Peak detection method
3. Machine learning-based step counting

To track steps using sensors in the fitness app, I could make use of the SensorManager class and the TYPE\_STEP\_COUNTER sensor. This sensor provides a count of the number of steps taken by the user since the last device reboot. This can be used to count/track the user's steps over time. The basic Android API provides support for this sensor meaning I won't be needing any additional libraries to use it. However, I will need to request the necessary permissions in the app’s manifest file and handle the sensors appropriately with java. A filter may also be a good addition, to remove any potential noise in the sensor data and improve the accuracy of step tracking.

To track the user's daily calorie burn I'll need to implement an algorithm that takes into account the user's activity level, age, weight, and other relevant factors. There are several different algorithms that can be used for this purpose, but one most popular option is to use the Harris-Benedict equation.

The Harris-Benedict equation is a widely-used formula for estimating daily calorie needs. It takes into account a user's basal metabolic rate, which is the amount of energy required to keep the body functioning at rest, as well as their level of physical activity. To calculate BMR, I'll need to know the user's age, weight, and height. This could easily be set in the confines of a profile tab or alternatively set and edited in the settings tab. I could then use the user's steps to calculate the calories burned and potentially use the BMR to suggest the users additional calorie needs.

Android 12 ensures that my application is compatible with a wide range of devices, as Android 12 has a higher adoption rate compared to earlier versions. This means that my application can reach a larger audience and provide a consistent user experience across a range of devices. The SDK version must be at least above 29 to make use of the material design libraries to create the UI.

I may also incorporate libraries such as Google Play Services to support functionality such as location tracking, Google Maps integration, and push notifications. I will utilize various design patterns such as Model-View-Controller (MVC) to ensure a clean and modular codebase.

Data can be stored in either the device's internal storage or an SQLite database, depending on the amount of data being collected upon implementation.

Internal storage is a private storage area that is only accessible to the application. This storage option is ideal for storing small amounts of data, such as user settings, preferences, and other small text files. It is easy to implement, and there is no need to set up a separate database. To store data in internal storage, the Android framework provides several APIs such as SharedPreferences, which allow for persistent key-value storage, and the File API, which allows for reading and writing of raw data to the internal storage.

An SQLite database is a self-contained, file-based database that is accessible by the application and can store a larger amount of data. SQLite is the most used database system for Android applications, and it is easy to set up and use. To store data in an SQLite database, the application needs to create a database and tables to store the data. The application can then insert, update, and retrieve data from the database using SQL queries. The SQLite database API provides several helper classes and methods to create and manage the database and perform CRUD (Create, Read, Update, and Delete) operations.

In a fitness-based Android application that tracks footsteps, storing data in an SQLite database may be a better option as it allows for more complex data storage and retrieval, and can handle larger amounts of data. However, if the application only needs to store a small amount of data, storing it in the device's internal storage may be more efficient. The task can be carried out in either of these ways regardless.

To implement SQLite database into this project I would first need to create a SQLiteOpenHelper class that will handle the creation and management of the database. This class would then extend the SQLiteOpenHelper class provided by the Android SDK. In the constructor of the SQLiteOpenHelper class I would need to provide the name of the database, the version number, and any other necessary configuration options.

Then I would have to create a class to define the structure of the database. This class would then extend the SQLiteOpenHelper class and would have methods to create the tables, insert data into the tables, and query the inserted data from said tables. This class would also define the columns of the tables and their data types.

To interact with the database after this setup, I would make use of the SQLiteDatabase class provided by the Android SDK. This class provides methods to execute SQL queries, insert data into the database, update data, and delete data. I could also use the Cursor class to retrieve data from the database, iterate over the rows returned by the query and populated the history tab with the contents.

I found that to simplify the management of the database, I could use an ORM library such as Room. Room is a library that provides an abstraction layer on top of SQLite and allows me to define the database schema using Java classes. This would make it easier to write queries and reduce the amount of code required to manage the database.

Alternatively, Firebase could be used for data storage in the fitness app in place of SQLite. Firebase is a mobile and web application development platform that provides a suite of backend services, including a tree-structured cloud database, real-time data synchronization, and user authentication. One of the key benefits of using Firebase is that it provides a fully managed and scalable cloud database that is accessible from anywhere and can be easily integrated into any Android app.

A Firebase Database can easily be set up via its web application. To integrate Firebase into the app, I can make use of the Firebase Realtime Database SDK, which is a client library that provides a set of APIs to interact with the Firebase database. I can easily add the Firebase SDK to the app by including the necessary dependencies in its Gradle file. Once I've added the Firebase SDK I can initialize the Firebase database in the app and start using the API to read and write data to and from.

Another advantage of using Firebase is that it is backed by Google, which means that I can count on ongoing support and updates for the platform. Google has a reputation of providing high-quality tools and services for developers and Firebase is apparently no exception. In addition, Firebase is an industry standard for mobile and web app development, with many popular apps using Firebase as their backend infrastructure.

In summary, while SQLite is a viable option for data storage in our fitness application, Firebase provides a more modern and scalable solution with ongoing support and industry-standard status. By using Firebase, I could take advantage of its cloud database and real-time synchronization features.

When designing an Android application, one of the key considerations is the user interface (UI) layout. An effective UI layout can significantly improve the user experience.

Making use of ConstraintLayout would be an optimal decision for the design of the fitness application UI. It is a flexible and powerful layout manager available in Android Studio that allows us to create complex and dynamic layouts by setting constraints between the UI elements and views. It has great support from Google, with vast documentation allowing for a fast learning curve and wide adoption with developers. This implies that it will also be supported for the foreseeable future. It can be easily implemented through adapting the manifest of the application and specifying the layout type of the chosen activity.

A key benefit of using ConstraintLayout is its flexibility. With ConstraintLayout you can easily create UI layouts/views that are responsive to different screen sizes and orientations. This means that the same layout can be used on different-sized smartphones and tablets with automatic adjustment. This is especially important in the modern day, where smartphones are widely adopted and users access applications on a variety of devices with different screen sizes.

Another advantage of using ConstraintLayout is its ease of use. The layout editor in Android Studio provides simple yet in-depth support for the ConstraintLayout, making it easy to design and build complex layouts with a drag-and-drop interface. This will allow me as the sole developer to spend less time on UI design and more time on other aspects of the app development process.

In addition to its flexibility and ease of use, ConstraintLayout is optimized for performance. It reduces the number of nested views required to create a layout, which can improve app performance and reduce the likelihood of layout errors. This is especially important for applications that require a high degree of interactivity or are resource intensive.

The aforementioned “Material Design” libraries are incredibly useful for the implementation of a modern design into the UI of an application. Material Design can be defined by its combination of minimalism and flat design. It is based on the idea that the UI should mimic paper. This can be seen through the conventional use of sliding transitions between activities and shadows emanating from elements that are layered above others. Other modern application design conventions are included in material design such as buttons and interactable elements displaying ripples to visually confirm the interaction was successful.

Another reason for utilizing a minimalistic UI is to prevent frame dropping. The UX (user experience) is more easily upheld with a simplistic design, however, if frame dropping is encountered then profile HWUI rendering can be used to identify when and where the bottleneck is occurring. This would be a useful tool during development to help ensure efficiency across devices. Different devices can be emulated for testing, or an Android device can be connected to android studio for temporary installation.

Unless statically or dynamically bound fragments were to be used, the application’s UI must be designed with all elements available from the corresponding hosting activity. This is because Android applications are limited to displaying only one activity at a time. I personally am averse to using fragmented design because of its overcomplicated nature that I've found from experience. If the application were to at any point need more than one activity onscreen, then I would make use of statically bound fragments due to the difficulty of communication between dynamically bound fragments.

Configuration changes can be handled manually through alterations to the application’s manifest. This way the application can be set so as not to reset itself upon a change in screen size or orientation.

The stack is a crucial component in android application development; however, it is also a large source of errors for new developers. Memory leaks, performance issues and crashes are common issues when an application is using its default launch mode. This is because it is possible to launch an infinite of the same activities without destroying previous iterations. To prevent this the launch mode can be changed in the manifest to either singleTask or singleInstance. The lifecycles of activities could also be managed through java code to prevent similar issues.

While pop-up notifications can be executed via Google Play Services, in-app notifications would usually be executed via Toast. However, Snackbar would be the new optimal solution considering it was designed to be the replacement for the outdated toast messaging system.

Snackbar is already included in the Material Design libraries so no extra effort would be required to implement it. The code required to call the functions are extremely simple and the notifications are relative to the application, unlike toast. This means that while toast messages can confuse the UX by having messages persist after the application's destruction, the Snackbar messages will disappear alongside the app. The UX is also improved by the ability to “swipe away” the generated notifications and the ability to incorporate user interaction into the messages, as they can host functions such as closing the application or redirecting to another activity.

The widely suggested method for implementing voice recognition involves using the built-in speech recognition APIs provided by the Android SDK. This involves adding the necessary permissions to the app's manifest file, creating a new activity or fragment to handle the voice recognition functionality, creating an instance of SpeechRecognizer, setting up a recognition listener to handle the recognition results, and starting the speech recognition process by calling the startListening() method on the SpeechRecognizer instance.

The main reason for implementing voice recognition in an Android app is to provide users with an alternative input method that can be more convenient and efficient than typing. By enabling users to speak their commands or input, the app can be more accessible and easier to use for a wider range of people. Additionally, implementing voice recognition can enhance the overall user experience of the app, especially if it is done in a way that is consistent with modern design conventions like Material Design.

## UI design drafts

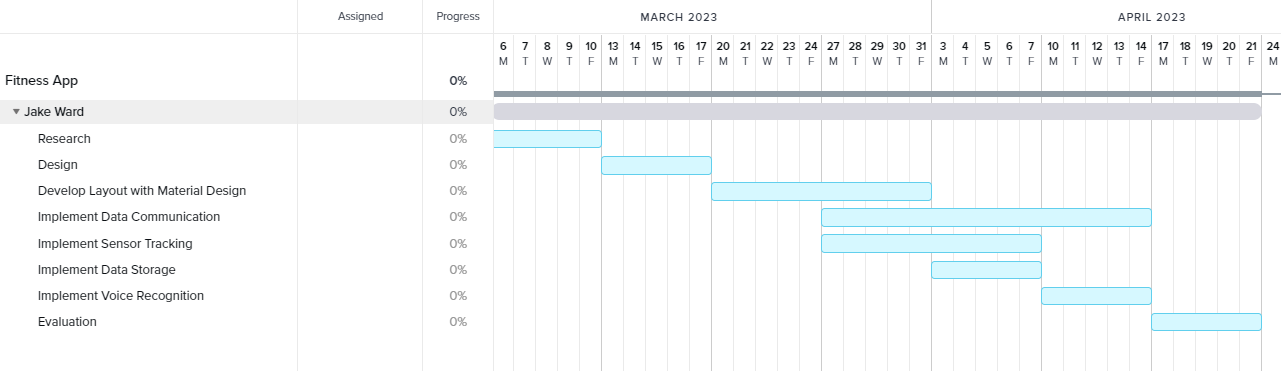
My proposed app's UI design will be simple, with a focus on ease of use and accessibility. The app will include a dashboard screen to display daily activity, a history tab to show progress over time, and a settings tab to view information and change settings.

The dashboard screen will display the user's daily activity, including the number of steps taken and the calories burned. The history tab will show the user's progress over time, including daily and weekly trends. A settings tab will be featured too with extendable elements to change between themes such as light and dark mode, display the application version and accessibility options such as changing the font size. There will also potentially be a slide-in tab on the home page where daily goals can be set using voice recognition.



## Work planning

The project will be divided into several stages, including research, design, implementation, and evaluation. The GANTT chart created using the free online software TeamGantt below shows the estimated timeline for each stage.



• Research (1 week) - research fitness apps and APIs, sensor technologies, and data communication methods.

• Design (1 week) - create UI design drafts, define app functionality, and refine requirements.

• Implementation (4 weeks) - develop the app with material design using Android Studio, implement data communication, implement sensor tracking features, implement data-storage, and potentially speech recognition.

• Evaluation (1 week) - conduct user testing and feedback sessions, evaluate the app's performance and refine features as needed.

## References

*Broadcastreceiver : android developers* (no date) *Android Developers*. Available at: [https://develo](https://developer.android.com/reference/android/content/BroadcastReceiver)per.android.com/reference/android/content/BroadcastReceiver (Accessed: February 13, 2023).

*Broadcasts overview : android developers* (no date) *Android Developers*. Available at: [https://dev](https://developer.android.com/guide/components/broadcasts)eloper.android.com/guide/components/broadcasts (Accessed: February 15, 2023).

*Handler : android developers* (no date) *Android Developers*. Available at: [http](https://developer.android.com/reference/android/os/Handler)s://developer.android.com/reference/android/os/Handler (Accessed: February 13, 2023).

*Sensors overview : android developers* (no date) *Android Developers*. Available at: <https://developer.android.com/guide/topics/sensors/sensors_overview> (Accessed: February 13, 2023).

*Overview of google play services | google developers* (no date) *Google*. Google. Available at: <https://developers.google.com/android/guides/overview> (Accessed: February 20, 2023).

*Better Performance Through Threading : android developers* (no date) *Android Developers*. Available at: [htt](https://developer.android.com/topic/performance/threads)ps://developer.android.com/topic/performance/threads (Accessed: February 20, 2023).

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.

*Android.speech : android developers* (no date) *Android Developers*. Available at: [h](https://developer.android.com/reference/android/speech/package-summary)ttps://developer.android.com/reference/android/speech/package-summary (Accessed: February 23, 2023).

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.