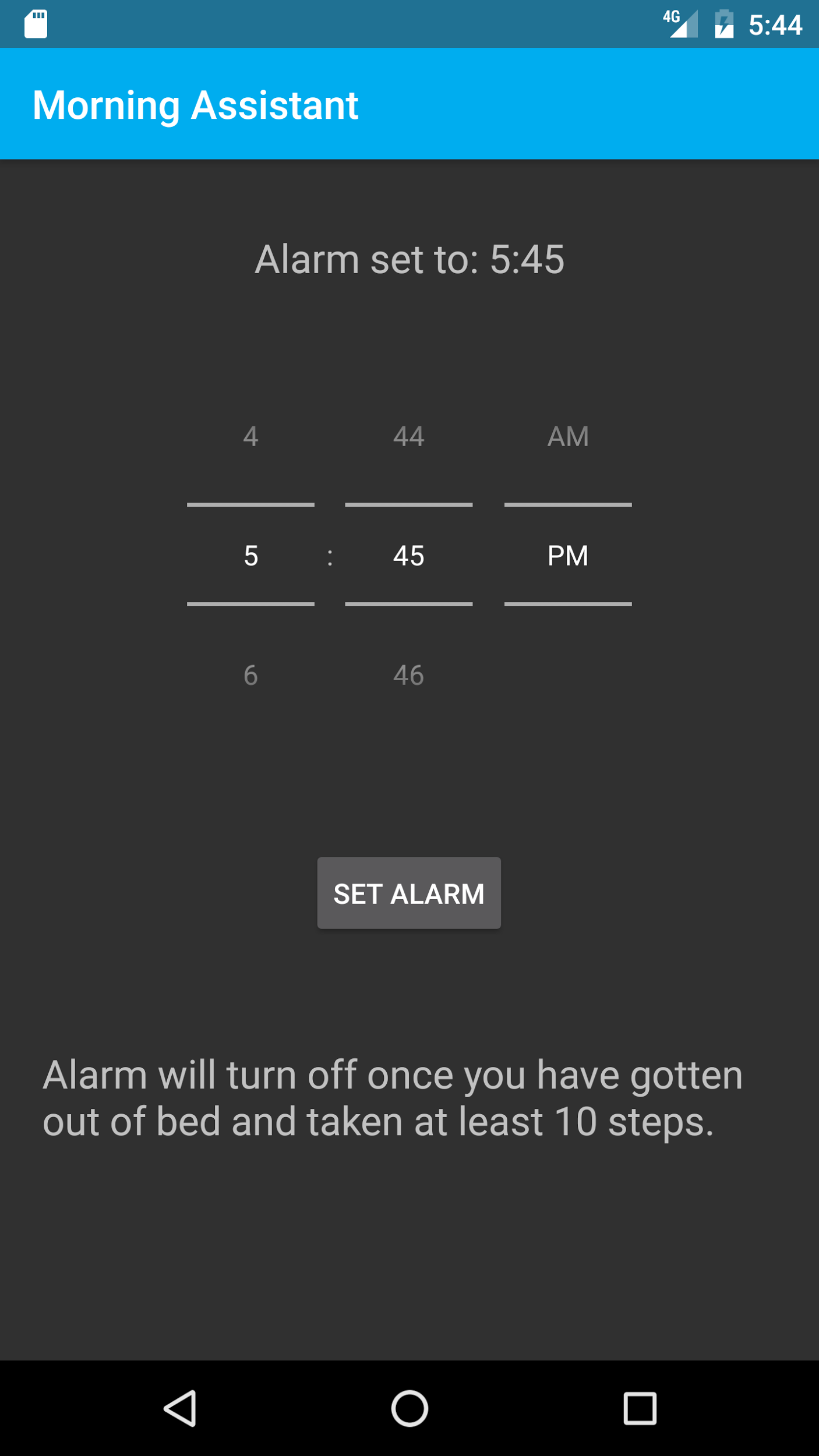
Project Documentation: Morning Assistant

By Devin Chen, Logan Landrum, Andrew Stoycos, Nick Thomas

When we first got together, we wanted to develop a morning assistant application that helps a morning averse user get through the early, painful hours of the day. The application would be an alarm clock that would require the user to move a set distance before turning off, somewhere ideally close to 10 or so steps. If there was time, we would add in a task manager wherein the user would input his/her tasks for the day and their schedule for the day, and the application would map tasks to free time slots in the user’s day.

## UI Design



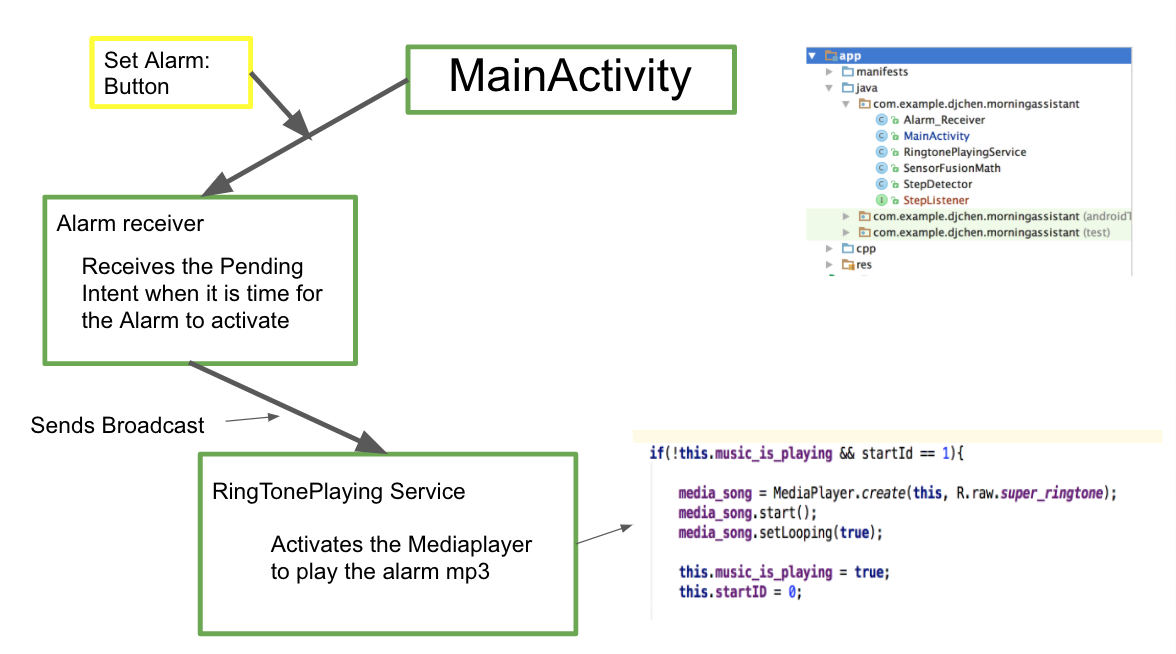
**Figure 1:** UI of Morning Assistant with alarm set

Construction of the App began with building the basic UI within activity\_main.xml folder of Android Studio and as seen above in Figure 1. This was one of the more straightforward portions of the app construction because the majority of the code was pre-made, requiring simple drag and drop of the various elements onto the design layout. A dark theme was chosen because most alarms are set before one goes to sleep and as a result the user is in a dark room. Therefore, a light theme would have been harsh on the eyes in that low-light environment. Additionally, the theme used was ideal because it kept the UI minimalist, allowing for easier use. The app required three different types of UI elements: the timePicker, a button, and two different textview boxes.

There were two timePickers that we could choose in the default list available, the spinner used for the app as well as a dial. We chose the spinner since it was a more user friendly and easier to pick the particular time. It also allows for easy time input by clicking on the numbers between the grey lines, for both the hour and minutes, and typing in whichever time you want. The timepicker’s value was stored in a Calendar object within MainActivity.Java to be later utilised by an alarm manager.

The Textview boxes were placed at the top and bottom of the UI with the instructions for use located at the bottom and an alarm status update located above the timePicker. The status update was placed at the top of the app because it held important information such as the set time for the alarm and whether or not the alarm was off. This information should be easily seen and accessible so we placed this information at the top. The other Textview box with the instructions was placed at the bottom because it felt like a natural place to put this information. Similarly, the “Set Alarm” button was placed below the timePicker because of the natural flow of the eye.

## “Set Alarm” Button



**Figure 2:** User interface “Set Alarm” Button

For the “Set Alarm” button press to be received an onclick listener function was built in MainActivity.java and the first thing it did was store the time selected into a premade Calender object. There was also a bit of error checking built in at this step, specifically if the time selected was during the following day it would increment the DATE of the calendar by one day. Also the times, with default set to a 24-hour clock, were changed to a 12-hour format and had to be converted to strings so that the textview object could display the selected alarm time. Then an Android Intent was converted into a pending intent which eventually activate another class, alarm\_manager, once the time specified time supplied by calendar was matched by the system. Alarm\_manager then wakes up the system, and sends the intent(my\_intent) to another object, Alarm\_receiver. Alarm\_receiver, recognized the intent and extra strings attached to it, either “on” or “off” and also created a new service\_intent for the RingtonePlayingService. This service intent was given the string from the original intent, and then run through the startService function which allowed the code to move into the RingtonePlayingService class. This class is where the code decided if the alarm was being set or if it was time for the alarm to be turned off. Here the extra string that was originally attached to the intent in MainActivity.java was put in a variable called state and is either “On” or “Off”, and then the variable startID was set accordingly. The following if statement then see if the music is playing and what value is stored in startID. If the media player is not running and the “On” button was pushed the media player will then be set to start and loop the chosen mp3 file via the system's internal speakers. The next functionality portion of the app is what separates us from the rest, the step counter, although proving very difficult, ended up being one of the most interesting parts of out app.

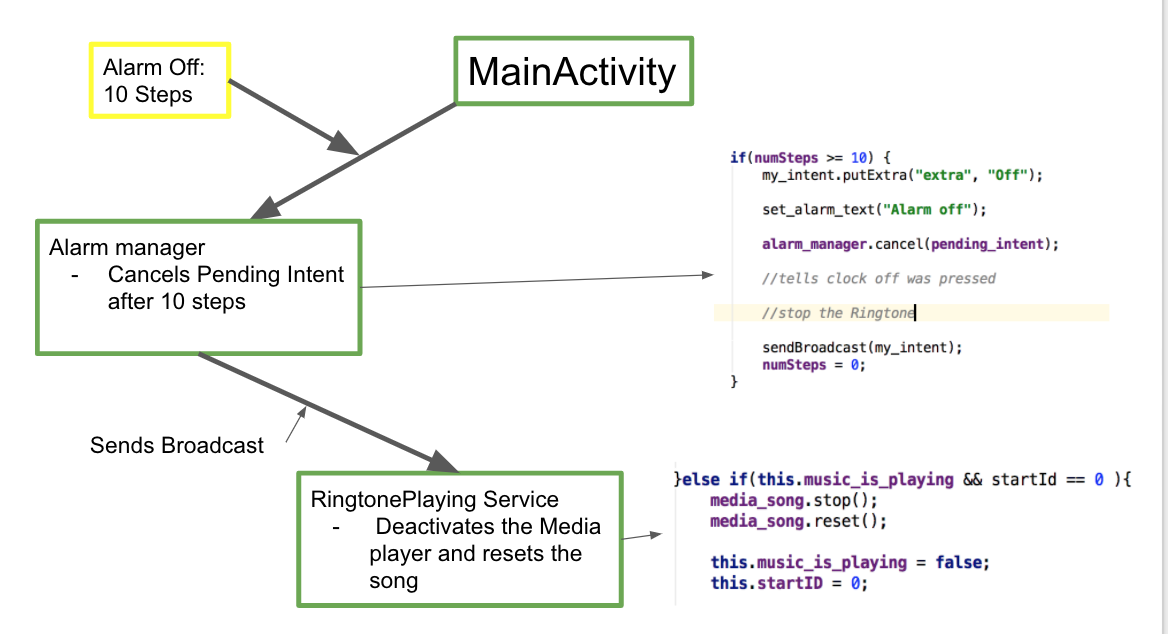
## Location API

For counting the steps, we attempted to implement a GPS API into the application. It was ascertained that 10 feet was a displacement of 0.0000247 degrees. Using guides provided by Android (<https://developer.android.com/training/location/index.html>) and YouTube (<https://www.youtube.com/watch?v=rEvwVf-qbDs&t=2s>), we implemented the Google Play Location Services API set to access “fine location” and set to “high accuracy”. However, we ran into a wide array of problems. While the app was able to connect to Play Services and create an API Client object, the object was unable to connect services, which was ascertained using the isConnected() boolean function. The object would return GPS coordinates (0, 0).

After considerable frustration, we decided to abandon using GPS and instead use a pedometer, ideally in the form of an API. Next, we tried to implement the Google Fit API, which has step counting capabilities. We used the guides provided by <https://developers.google.com/android/reference/com/google/android/gms/fitness/SensorsApi> and <https://code.tutsplus.com/tutorials/google-fit-for-android-reading-sensor-data--cms-25723>. However, we ran into similar issues that we had with the Location Services API. First, it would continuously/infinitely prompt the user for permission and a Google Account to use Google Fit functionalities. Second, the API Client object would still not connect after being created.

Next, we attempted to implement built in device Step Sensors to count the number of steps taken by the user. We followed a video guide by Indragni provided in the follow link, <https://www.youtube.com/watch?v=pDz8y5B8GsE> , but the device we using to test functionality, an LG G4 H810, does not have a step sensor. Considering that step sensors are not universal in devices, it would be better to just use the accelerometer to count steps. We had been leery about using the accelerometer from the beginning because it is easy to fool the accelerometer that steps have been taken by simply shaking the device. The accelerometer can also be overly sensitive, which is another drawback in using it. Given our experience with API’s, we decided that using the accelerometer was better than nothing at all. Google has an open source pedometer on <https://github.com/google/simple-pedometer> that takes in events from the accelerometer in three-space, puts the data from the events into mathematic functions that essentially perform double integrals on the data to acquire distance or steps. We implemented this pedometer, and then all that was left was in the backend was getting the alarm to turn off after the user takes ten or so steps.

## Step Counter



**Figure 3:** Step Counter’s interaction with Alarm Manager and RingTonePlaying service

The StepListener method defined in MainActivity increments number of steps the user takes and outputs a message in the log that a step has been taken. Once ten steps have been taken, the method cancels the pending intent from the alarm. It also creates a new intent, tryIntent, which has an “Off” string tag. Then once the intent passes through the Alarm Receiver and into the Ringtone playing service, the conditions are met so that the media\_song is stopped, reset and numSteps is set to zero. This turns off the alarm and prepares it to take another command.

## Debugging

After Initially finishing all the major portions of the code, there were some pretty big bugs that had to be diagnosed and sorted out. First of all the textviews displayed the incorrect time if the minute time wasn't a double digit, i.e less than ten. To fix this we first converted the hour and minute integers from the calendar object into strings, and if the minute integer was less than 10 a “0” was concatenated into the minute string. The next issue we ran into was that if the alarm was set to before the system’s time (i.e to the next day) the pending intent would execute immediately since the calendar object’s date was never incremented. To fix this issue we compared the time stored in the calendar object to the system’s current time, and if the calendar's time was less than the system’s time the calendar's date was incremented. Lastly the alarm couldn't be used twice in a row, this was easily fixed by making numSteps 0 once the RingToneplaying service turned off the media\_player. These were just a few of the major bugs associated with the code, with more time hopefully more bugs could be found and further improvements could be made.

## Future Improvements

The initial goals with this application were to create an in depth morning assistant which would force the user to wake up and give them a display of the tasks they had to do that day in a calendar format for easy viewing. Due to time constraints, not all of this was accomplished. Further development of this app would have involved the implementation of the above features. Additionally, the use of location services API or the step counter Google Fit API would be included in the app. These feature would be more accurate and reliable than the step counter method that is used in the current app. Addition of more functionality would also be done, such as customization of alarm sounds and the timePicker. Improvements to the back-end of the code would also increase the performance of the app and its efficiency.