Burim Derveni, Klaus Nuredini, Ernest Pambuku, Zachary Bryceland
March 15th, 2016
CS 275: Web and Mobile App Development
Project Documentation

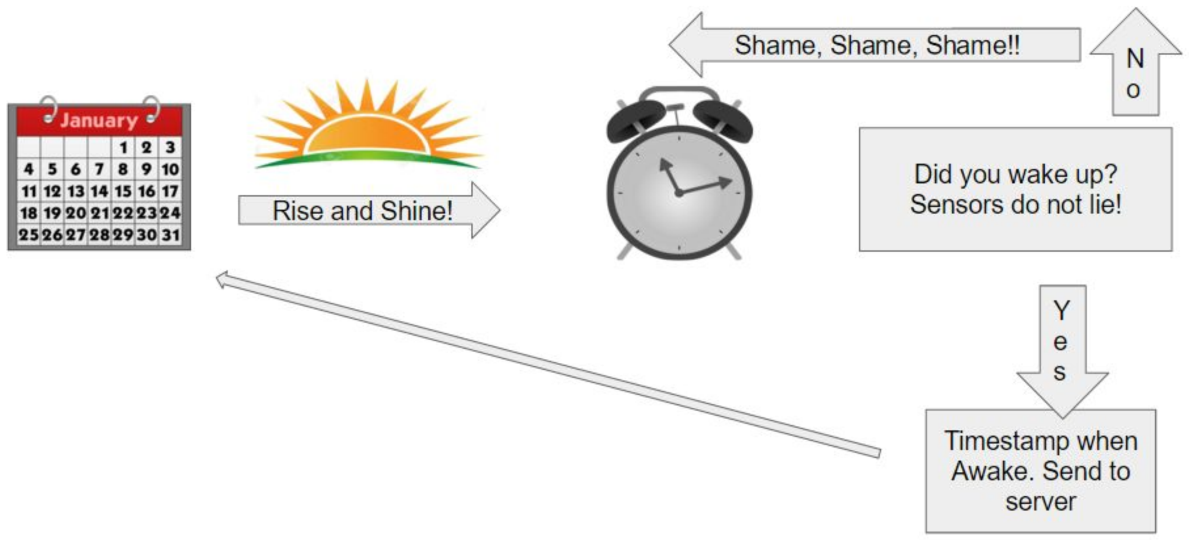
January
Rise and Shine!
Shame, Shame, Shame!!
No
Did you wake up? Sensors do not lie!
Yes
Timestamp when Awake. Send to server

Fig. 1 Flowchart showing the intended functionality

Project Description

Our project, Morning Minion, acts as a persistent alarm clock that wakes users in accordance with their calendars. The goal is to ensure that the user wakes up an hour before his first event of the day, regardless of whether he wants to get out of bed. Morning Minion pulls event data from Google calendar and triggers an alarm an hour before the first event found. Edison sensors in the user's bed provide data used to determine whether the user has actually woken up; if not, the alarm will continue. The intended functionality of the project is shown in Fig. 1.

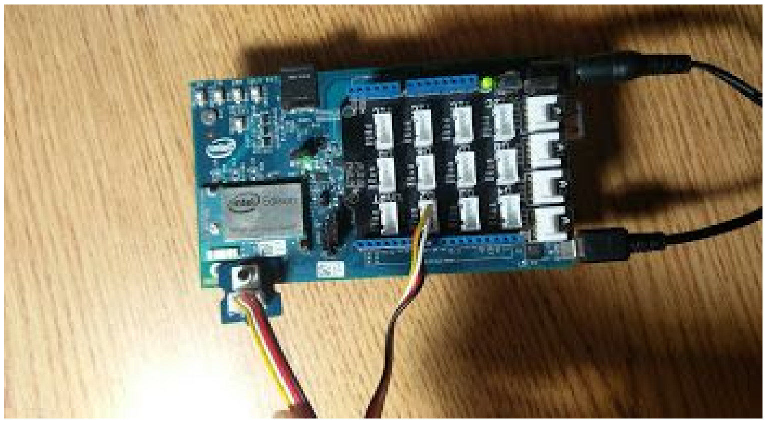

Fig.2 Edison board used to detect user motion

Project Features
This project features a fully functional Android application with a service backend that calls Google's web service client. The Android application requires user authentication to access Google calendars. Cloudmine is used as an intermediary between Edison sensors (pictured in Fig. 2) and user interfaces and is polled to provide data regarding whether the user has woken up.

Usage
To use the app, the user places a pre-configured Edison board under his bed to sense his movement in the morning. Once placed, he can sign into the Morning Minion app using his Google account. The app will show his first morning event data and act as an alarm clock the next day. The alarm is triggered an hour before the first event provided and will continue to sound until Edison sensors detect that the user has woken up.

Design and implementation details

The application uses oAuth to access the end user's Google calendar data. JSON data is pulled from their calendar in an asyncTask and the first event of the following day is retrieved. Provided that the user opens the app every day, it will continue to pull each first event, propagating as time passes. The alarm time will be set by default to be 1 hour before the first event start. This information is sent to a database in cloudmine that stores the alarm times. The edison sensor on the other end, will be querying this very database 4 times a day, 6 hours apart to see if a new alarm has been set. If it finds a future alarm it will get its time, and wait until the alarm time comes. When the alarm time comes, the edsion will start sending data to another database (hardware one). A python script and temboo choreos are used, because the CM API was not working.

When the alarm time comes, the alarm music will be triggered. The app will read from the hardware database using Temboo choreos until it reads a 0 value from the device. In that case, it will stop playing the alarm and it will take the next day's event, starting the whole process from scratch. So, the music will play until the Edison sensor provides data indicating the it should stop.

The interval at which the Edison sends to the database and the android app reads from the same database can be changed. For demonstration purposes we have lowered the time to 3 seconds on each side. This uses all of our Temboo choreos so it will normally be around 1 min in each side, with a max latency of 2 mins. We think that for a person to be properly awake, he/she needs to listen to an alarm for a good amount of time.

The app displays the event for which the user will be waked up for. It does not give the user any options besides that. We have a settings activity, where the user can choose what he does in the morning so the alarm time can be customized. However, due to time constraints, we could not implement it to the whole project.

Fig. 3, and Fig. 4 below describe the code flowchart for both the Android side, and also the hardware side.

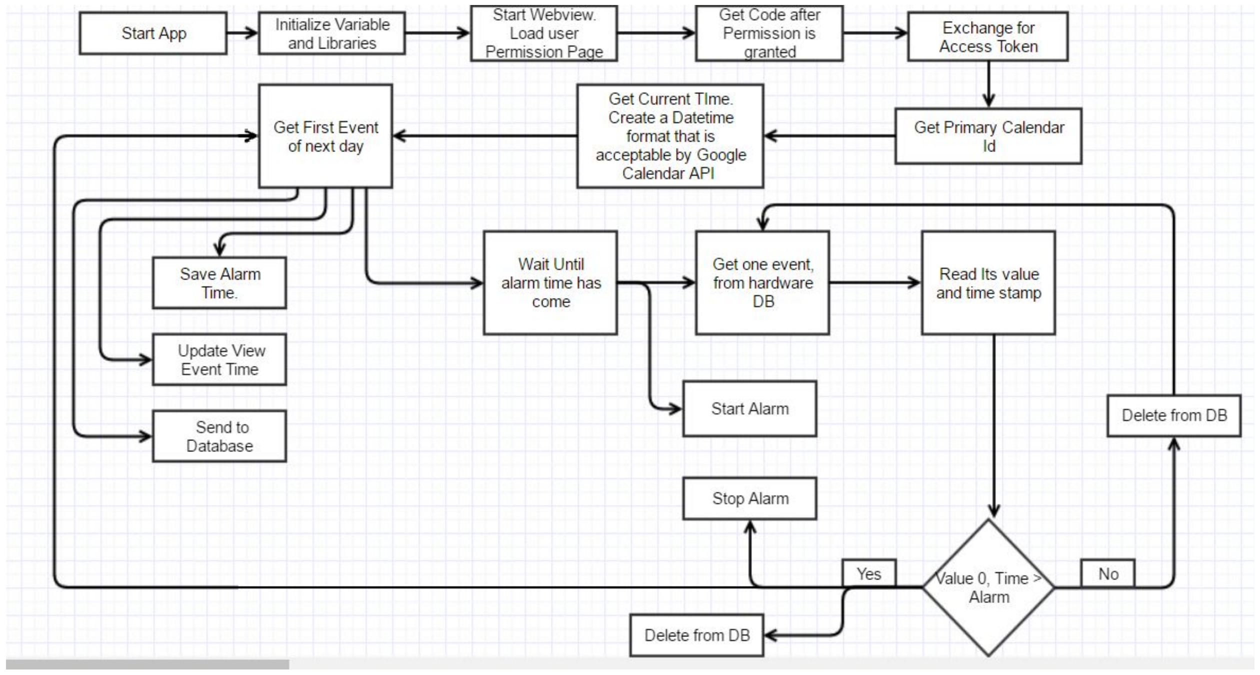
Start App
Initialize Variable and Libraries
Start Webview. Load user Permission Page
Get Code after Permission is granted
Exchange for Access Token
Get Primary Calendar Id
Get Current TIme. Create a Datetime format that is acceptable by Google Calendar API
Get First Event of next day
Save Alarm Time.
Update View Event Time
Send to Database
Wait Until alarm time has come
Get one event, from hardware DB
Read Its value and time stamp
Start Alarm
Delete from DB
Stop Alarm
Yes
Value 0, Time > Alarm
No
Delete from DB

Fig. 3 Android App Flowchart

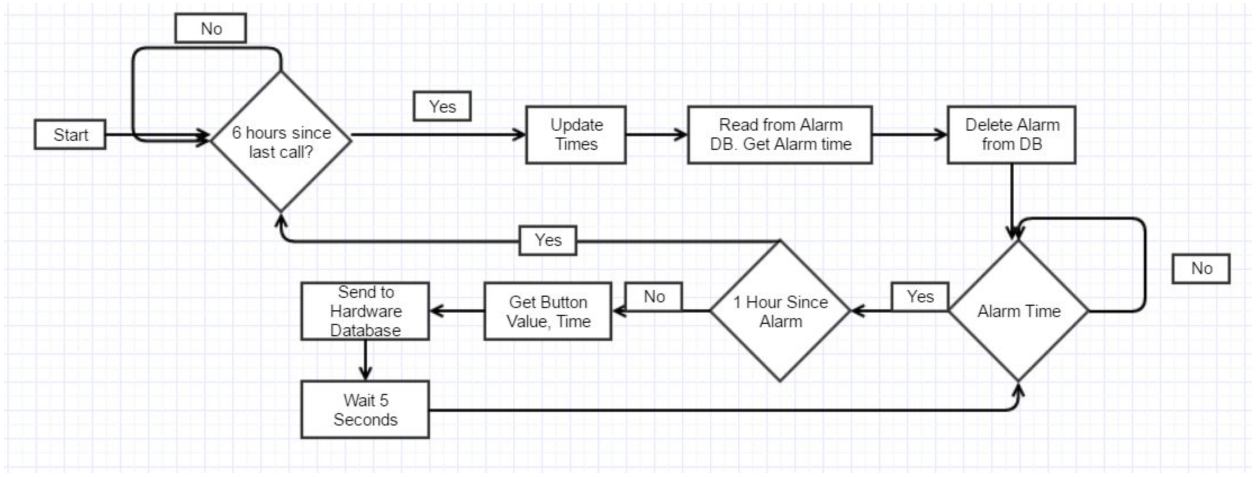

Fig. 4 Hardware Program Flowchart

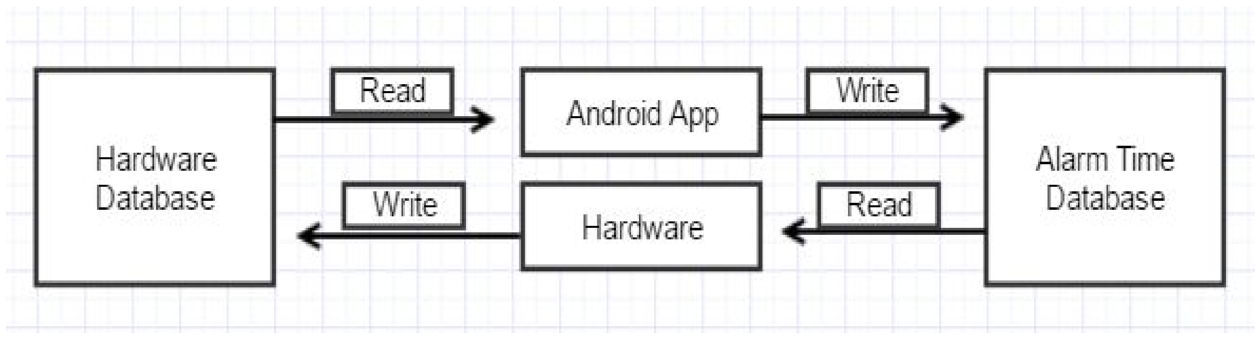

Fig. 5 Database Communication

Development Log
We were having trouble accessing MainActivity objects and methods inside the BroadcastReceiver class, AlarmReceiver. First, we had this problem, because the class was written in the MainActivity file and we had to make it static. Since it was static, it would not access objects of the MainActivity class and so, we had to create AlarmReceiver as a new class in a new file. From there, we used a MainActivity instance to access the needed methods.

Another problem we ran into was that we could not start the music at the specific time we needed to. This was because the MediaPlayer was created inside the onReceive method in AlarmReceiver and it was started there. What we did was - we created the MediaPlayer in the MainActivity and accessed it from the instance that is mentioned above. The time that we would want to start was the time of the alarm and was taken in the DownloadCalendarAsyncTask, in its doInBackground overridden method. To access that time after the doInBackground was done, we did it in the PostExcecute() and then we passed that time to the MainActivity. From there, we passed it to the onReceive method, where the sound starts.

One of the problems was also incorporating the settings activity and layout to the whole project. We tried to do this, but it was actually mostly because of the time constraints that we could not implement it. The activity by itself worked just fine, though. We could get data in it, by creating a class called Settings that implemented CompoundButton.OnCheckedChangeListener. We overrode the OnCheckedChange method in there, which we used for the setOnCheckedChangeListener() for each object in the MainActivity.

One of our main issues was the authentication part, along with calling and getting data from the calendar API. We first tried really hard to implement the Google Sign In using google play services property to authenticate with the user and then to prompt the user to click Allow for giving us permission to his/her calendar data. After a lot of work, we were still not able to implement it because we could not get an access token.

We also tried using an AccountPicker, and account manager to do oAuth. We were able to get the account info that the user provided, but when using account.GoogleAuthUtil.getAccessToken() we got an unknown sources error. We made sure that the app SHA1, and package name were the same as the google developer console info, but the error persisted.

After these tries we started trying doing oAuth using professor Mongan's method and his sample in the lecture slides and resources on course page in BbLearn. The code that Prof. Monga. provided would skip reading the return code from first step of oAuth in the `shouldOverrideUrlLoading()` method. At the end, we managed to get this to work fine and we were able to get data from the calendar.

We did have some issues with parsing the data, mostly moving back and forth with the date formats, but we figured it out, after trying different formatting methods.

There were issues with CloudMine as well. I could not get it to work in a simple java program using the SDK. Also when I tried to make web calls to the API through python, I would get a invalid key error. I could not resolve this error so I decide to use temboo.

Temboo also gives some errors occasionally, outputting the same error. It outputs an invalid key error after having run done the same choreo in the same script. We are just catching an exception in this case.

There were also problems in getting the hardware and software to work together, mainly because of the two parts not being synchronized. Introducing waits in both parts of the code solved this.