Final Report: App Opener

Mobile App Engineering and User Interface

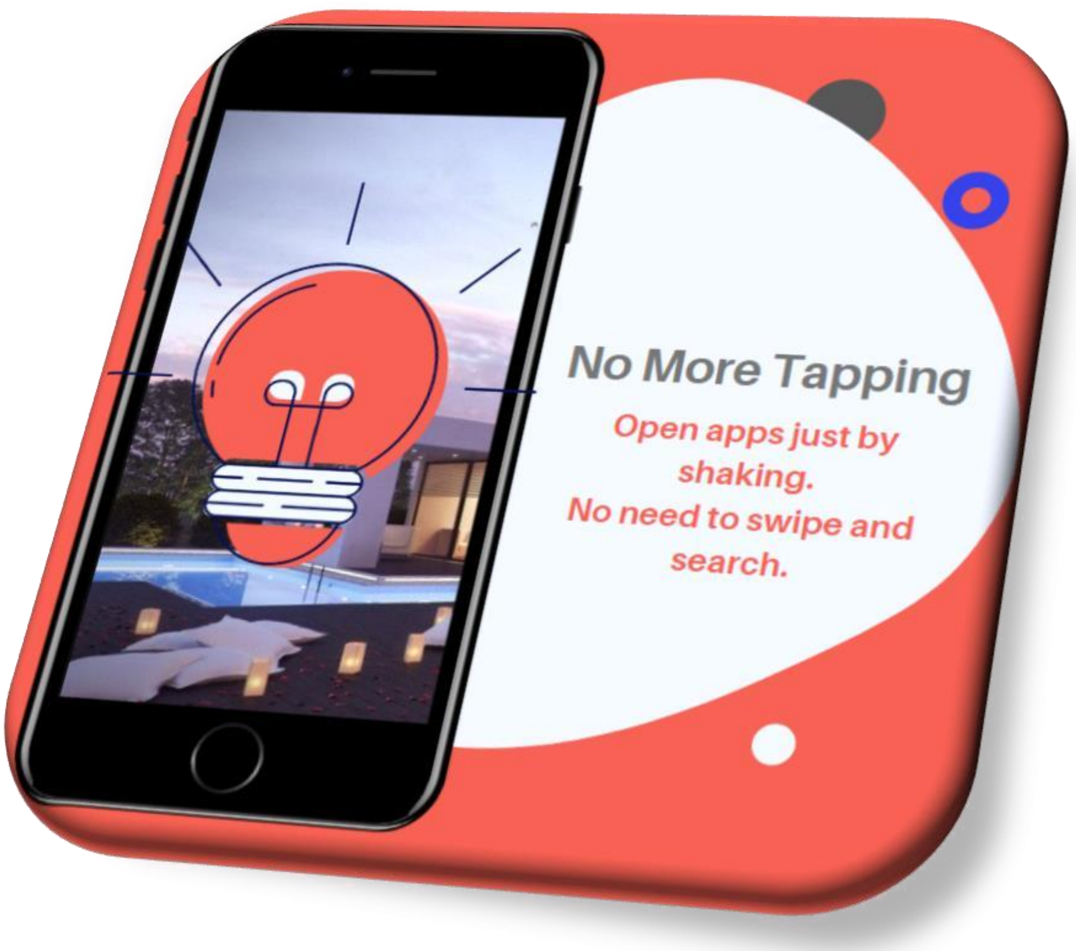
Students:

Andy Lee

Parthkumar Patel

Robert Riso

Aryeh Ness



Motivation – What is the project good for?

- The main motivation to create this project was to make it easier for the user to open apps without swiping through screens and searching for a particular app.
- Many shops and fast food restaurants give discounts if you use their app instead of ordering through the cashier. This results in way too many apps on our phones and searching for them takes about 20 seconds.
- Since there is no swiping or searching required this app can also save a lot of time.

Our Solution to solve the problem

- We implemented the main objective by using sensors on our phones that can detect the shake and open a selected app right away.
- The sensor that we chose is the acceleration sensor which will measure the shake of the phone and then open the desired application for the user.
- The location-based part allows user to choose an application and then assign locations. This allows our app to open the user selected app when the user is in close proximity to the selected locations.
- Our features implemented allow us to solve most of the features we wanted to create using our motivation.

How does it work – key components and their interaction

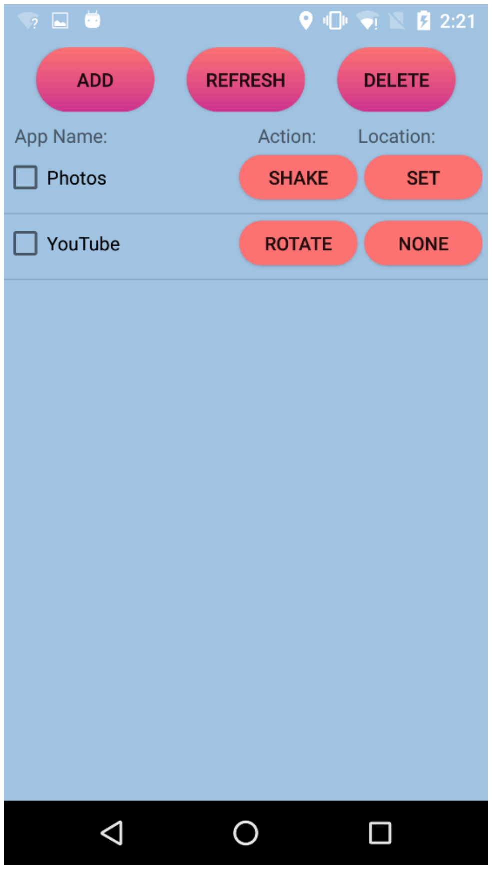
Key Components of our Project:

- Shaking to open the app.
- Rotating the phone to open the app.
- The app runs as a permanent service in the background
- Location based app opening for the user.
- Database that handles all the information entered by the user and used for the app.

Describe the user interface of your system and how it makes the most common tasks easy and quick to accomplish:

- Our app consists of the following buttons, better displayed in the screenshot below. ○ ADD
    - ▪ This allows the user to add the application they desire to add to the app.
  - ○ REFRESH
    - ▪ Refreshes the list of events after an app is added.
  - ○ DELETE
    - ▪ Allows the user to delete an application with the action and location related to it.
- Once a user adds an application to the list there is a list view for the apps that are added and this list view has a check box and two buttons next to each app ○ CHECKBOX
    - ▪ Select and delete the app ○ ACTION

  (button listed under action column)
    - ▪ Allows the user to select from the three following options
      - ● Shake
      - ● Rotate
      - ● None
  - ○ LOCATION (button listed under location column)
    - ▪ When pressed takes the use to a google maps API to select the location they desire on the map.
      - ● The user could also search the location using the search bar that is provide. Example – Name of the store such as Starbucks or BestBuy.



All the buttons are showed in action in this screenshot.

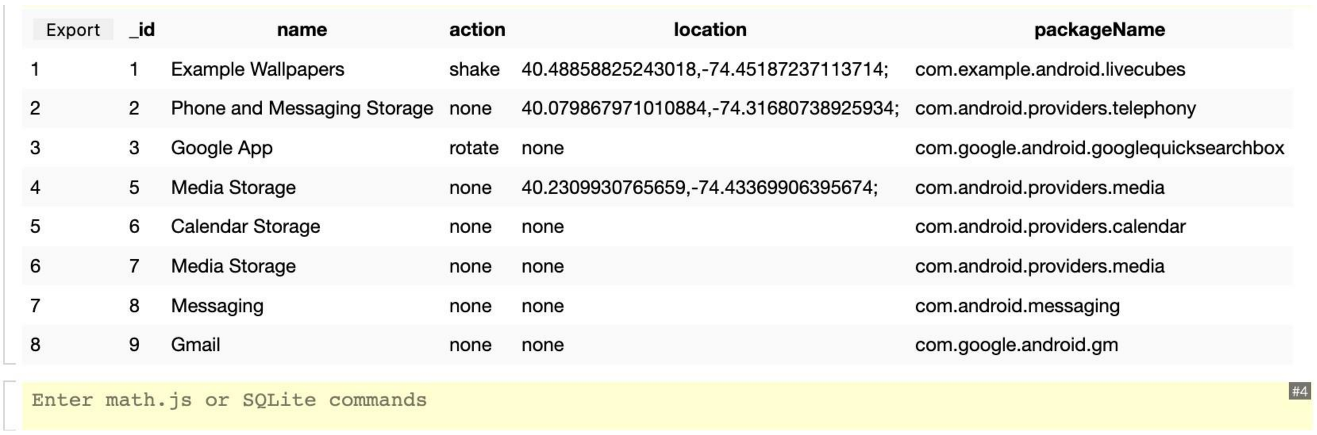
Describe the key components of your systems (e.g., activities, services, remote database) and how they interact:

<u>Key Components of our Project:</u>

- Shaking to open the app.
  - This function is implemented using the hardware libraries for the sensors. Our app uses Accelerometer to measure the shake by the user. Detecting the shake is completed by ShakeDetector.java this is then connected to ShakeService to link it with the interface and the database for the app.
- Rotating the phone to open the app.

- Rotation to open the app is implemented in ShakeService.java this is also connected to the database eventDBHelper.java

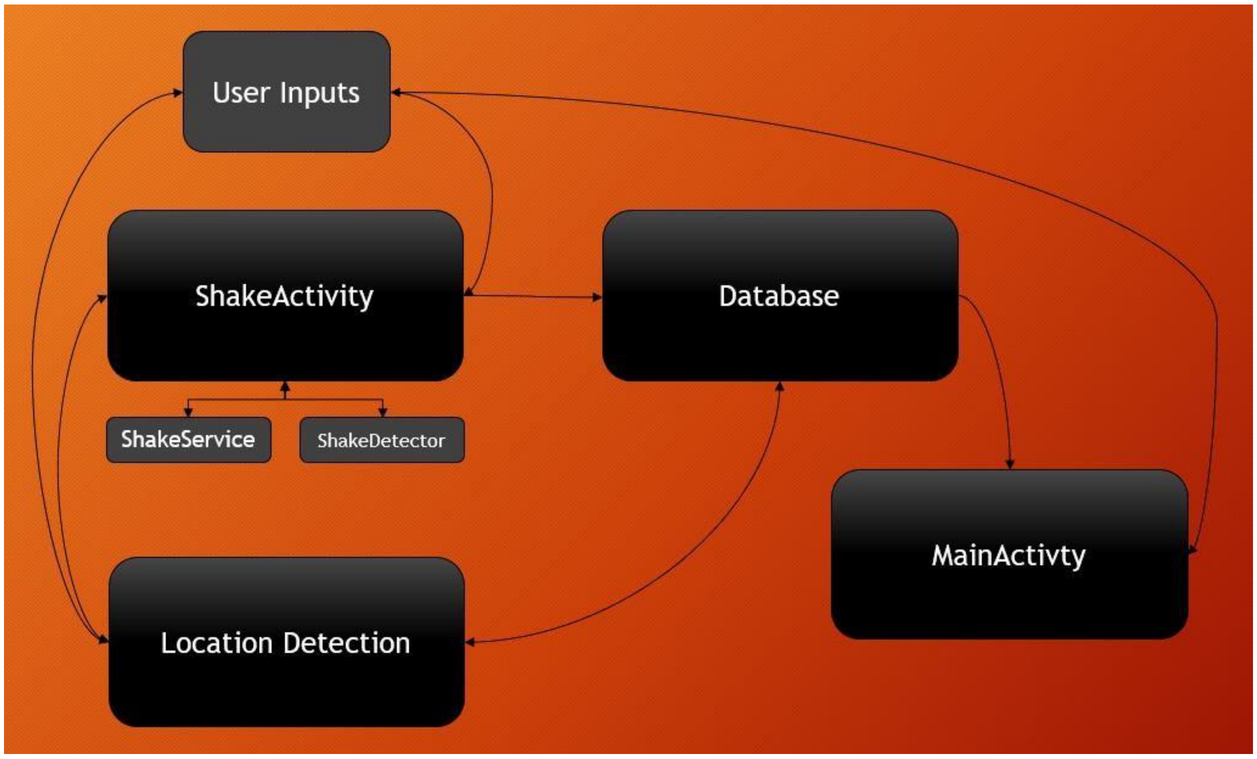
- Location based app opening for the user.
  - For getting the location we have used Google Maps API. When user selects location to be set it takes the user to a map fragment and a search bar that allows the user to select a location to add to the database.
  - The database takes locations from the user and then we check that location to the current location. Once the user is close the app opens.
- Database set up:
  - Our database is set up in the following way as shown below.



Export	_id	name	action	location	packageName
1	1	Example Wallpapers	shake	40.48858825243018,-74.45187237113714;	com.example.android.livecubes
2	2	Phone and Messaging Storage	none	40.079867971010884,-74.31680738925934;	com.android.providers.telephony
3	3	Google App	rotate	none	com.google.android.googlequicksearchbox
4	5	Media Storage	none	40.2309930765659,-74.43369906395674;	com.android.providers.media
5	6	Calendar Storage	none	none	com.android.providers.calendar
6	7	Media Storage	none	none	com.android.providers.media
7	8	Messaging	none	none	com.android.messaging
8	9	Gmail	none	none	com.google.android.gm

  - It consists of the following columns:
    - ID – Index for the apps
    - Name – Name of the app
    - Action – Has three options as mentioned before

- Location – Location user inputs for the app to open.
- packageName – To extract the app packages.



Describe the implementation of selected key components so that a knowledgeable app engineer could create a similar app with the provided information:

- Shaking to open the app.
  - The shaking was implemented mainly through the use of two classes, a detector class and a service class. The shake detector class defined the minimum shake value needed to trigger the shake, how long to wait between shakes to avoid overloading the user with too many responses to the action, and a variable to reset the amount of shakes recorded if no more shakes are done for a certain amount of time. So, we override

onSensorChanged to make use of the accelerometer for this task. If the shaking done is over the minimum required to count as a shake(1.1f), we make sure shakes aren't too close together(we used a value of 500ms), and they're within a tight window frame to confirm this is an intentional shake(three seconds). Now that we have confirmed the legitimacy of the shake, we move to the second class, the shake service class. This class implements the shake detection as a permanent service running in the background of the device, to always listen for shakes on the device. After defining the sensors for the accelerometer and the helpers we need, within onStartCommand, we initialize the listeners for the accelerometer. Then, within onSensorChanged, we use the data from the accelerometer to calculate how heavy the shake was. If it is strong enough, the service is then triggered to do the selected action-namely, open the desired app. Importantly, within the main activity, we override onShake not because we have to do anything with it, but because the sensors and shake detection only work with this method appearing in the main activity. Note: this all worked in both the background and foreground through the use of a foreground service.

- Rotating the phone to open the app.
    - This was actually fairly simple in the end. Within the same class for the shake service detection we also accomplished the rotation detection. Since we already had a listener set up for the accelerometer, we used those same values to listen for when the screen was rotated horizontally, within onSensorChanged. The x and y values would return the necessary data for this detection. This was found to be when the absolute value of the x value was greater than 5, and the absolute value of the y value was less than 5. This service worked both within the app and the background of the device,

and once this was triggered, the desired action would take place. Note: this all worked in both the background and foreground through the use of the foreground service.

- The app runs as a permanent service in the background. ○ This was done using a foreground service, with a simple notification set up to inform the user of the service. This was all done in the shake service class, within the onStartCommand method. A toast was also added to the main activity, to inform the user that the service starts up when the app starts up. This was always done to remind the user of all the app's activities .

- Location based app opening for the user.
    - Location based app-opening was accomplished through the use of a specific series of steps. After the user selects the apps that they want to use and those apps get added to the main screen, the user can choose to add locations to those specific apps by clicking the button on the main screen. The user is then sent to a different screen where there is a map, a search bar (so the user can enter in the name of a specific place near them, and then select one of those places) and a submission button. This page is used by the user to set the locations that they want the app to open by and then, after selecting the locations, the entered information is constantly checked by the service to see if the user is within the range of any of the selected locations for the app. The portion works by: -sending the app ID, for the app that the user wants to add locations too, to this new map activity.
      -Opening a map using the Google Maps SDK
      -allowing searching for locations by using Google's geocoder, and using the getFromLocationName function (additional information: we decided to use the closest two locations). -using the location manager and onLocationChanged function

to find the user’s current location, set the map to zoom to that location automatically using the built in map functions, and check the searched location based on the closest locations to the user (geocoder can check by closest location by setting a latitude and longitude based rectangle).
-setting map click listeners so that users can select locations on the map and get a popup that allows them to confirm their selection on the map.
-opening the SQLite database connection and, on submission, sending the new selected locations to the database as a concatenated string of latitude and longitude values in the format (lat1,long1;lat2,long2;lat3,.......etc).
- using a service and opening another database connection in the service, creating another location manager, and using another onLocationChanged function to check the users current location at certain time intervals (we chose every one 1 minute and 40 seconds).
-using a “for loop” to access the stored location strings in the database, split them into all of their latitudes and longitudes, compare those latitudes and longitudes to the user’s current location, and then open the necessary app if one of the locations in the string was within 100m of the location of the user.

- Database that handles all the information entered by the user and used for the app.
  - The database is implemented using SQLite. We used a single table named ‘eventlist’; the table has 5 columns for each record. The 5 columns consisted of _ID, ‘name’, ‘action’, ‘location’, and ‘packageName’. _ID is used to give each record a unique identifier, ‘name’ is the name of the app, ‘action’ is the action that the user has selected to initiate the opening of an app, ‘location’ is the set of latitude and longitude

coordinates that will also initiate the opening of an app, and ‘packageName’ is the string saved from each application data which is then used to open an app. In the database, only the _ID is stored as an integer variable type, all other variables are stored as string type. The database helper class has custom made functions that will insert data, delete data, edit data, and get all data from the database. All of these functions utilized a string query in SQLite syntax and is then executed to perform actions on the database. When utilizing the database in other parts of the code, an instance of the database helper is created and then a readable or writable version of the database is required to interact with the database. Readable versions of the database were used for getting all data, and writable versions were used for all other actions.

Concepts applied from class:

- Setting up the database
- Using location and the Google Maps API
- Using listviews, services, fragments, and activities