**3.d. Application Outline**

**3.d.i. Registration and Login**

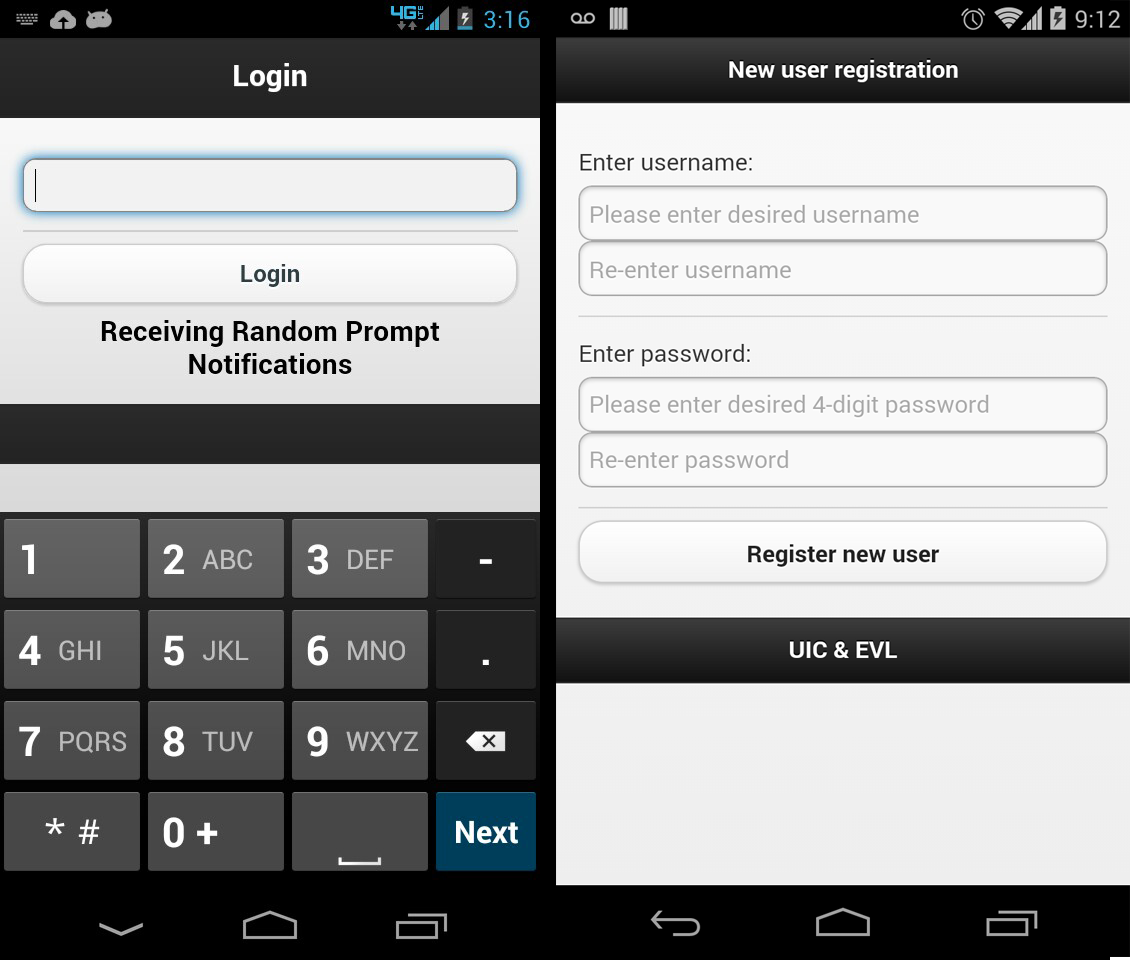


Figure 1: Screenshot composition of login pages within the running e-cigarette survey application. From left to right: the initial login screen for the application; the screen for new user registration.

The login and registration pages of the e-cigarette survey prompt application (Figure 5) had the following features:

* The login screen changed depending on whether or not the user has already registered and had a username and password saved to the device.
* If the user was new and did not have login credentials, a new user registration screen appeared asking the user to register their information (Figure 5).
  + A user must enter a username, with no restrictions to the name itself.
  + A user must also enter a 4-digit pin number used during subsequent logins. If a pin was entered incorrectly the process started again and the application asked the user to re-enter their credentials.
  + A user could not access the main portion of the application without having registered as a new user.
* Once a user registered their login credentials, a different login screen allowed the user to enter the application (Figure 5).
* The only time that a user could revisit the new user registration screen was if a member of the research group deletes their credential data.
* User credential data were saved to a ‘log.log’ file which held a single line of text in the form of ‘userID, xxxx’; where xxxx was the user’s 4-digit pin. A comma was used as the active delimiter.
* The presence of the ‘log.log’ file determined whether the login screen defaulted to the new user registration screen or the user login screen.
* If a user entered the incorrect pin number at the user login screen, they were notified of the error and asked to try again.
* The only way for a user to access the main portion of the application was to correctly enter the registered pin number.

**3.d.ii. Interviews**

During the preliminary study of this project, eight young adults were given a Motorola Razr M Android smartphone pre-loaded with the e-cigarette survey application. These users were of an ideal set of students as they were involved in the previous PDA study mentioned above. They were to have the smartphone available to answer survey questions during all waking hours. Data on e-cigarette usage were collected using two types of surveys. One survey (the Tobacco Use Interview) was to be completed after each usage of a tobacco product. For the second survey (the Random Prompt Interview) the user was randomly alerted by the smartphone through a loud notification that the user should log onto the application to take a survey.

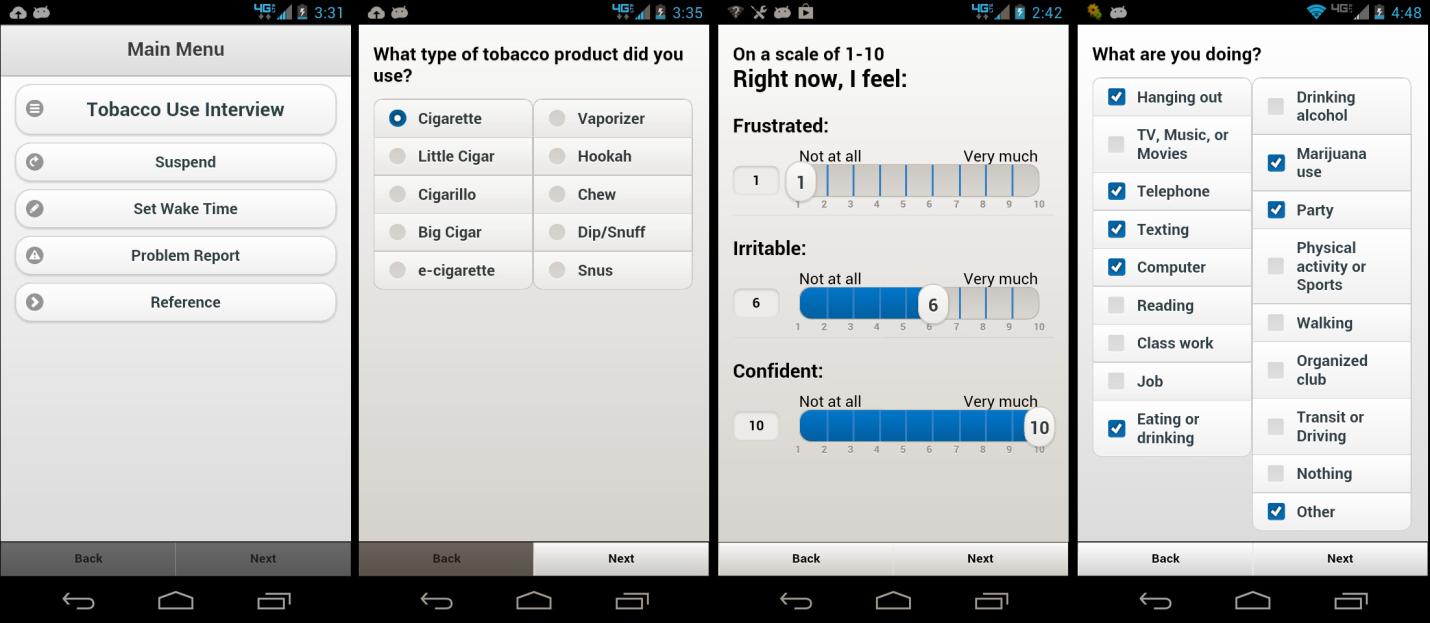


Figure 2: Screenshot composition of the different widget types within the running e-cigarette survey application. From left to right: the main menu of the e-cigarette survey prompt application; radio widget group; customized slider widget group; and checkbox widget group.

**3.d.iii Tobacco Use Interviews**

* Tobacco Use Interviews were surveys to be completed by the user on the smartphone after finishing a tobacco product.
* This set of questionnaires was different than the Random Prompt Interview survey (Appendix B).
* The survey was initiated by logging into the application and pressing the button labelled ‘Tobacco Use Interview’ from the main menu (Figure 6).
* The user was guided through a series of questions which utilized checkboxes, radio boxes, and sliders to record their responses. If a user did not answer all of the questions within a given page of the survey, they were not allowed to continue to the next page.

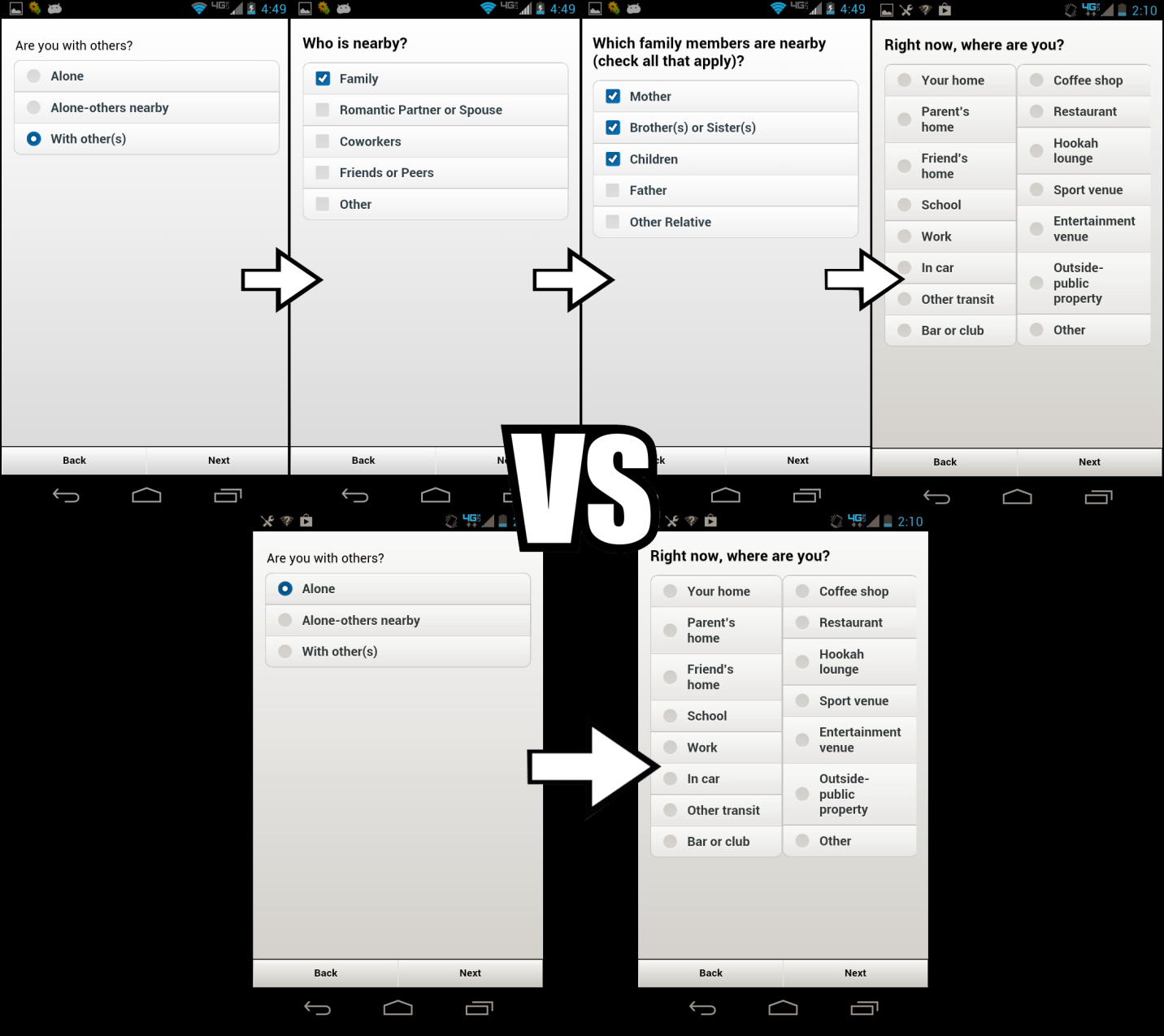


Figure 3: Screenshot composition of the running e-cigarette survey application. Two possible interview questionnaire routes; the top results in additional pages of survey questions, while the bottom does not.

* Depending on their answers, users might have to fill-out one or two additional questions within the survey (Figure 7).
* The survey input was only recorded to the .tob file (the file which keeps a record of all of the user’s Tobacco Use Interview answers) once they answer ALL questions within the survey.
* If a user leaves a Tobacco Use Interview survey, the application will NOT record the survey as having been abandoned or missed (Discussed below). This mechanic only applies to Random Prompt Interview surveys.
* If a user logs in to the application within three minutes of a notification firing they will be taken to the starting questionnaire of a Random Prompt Interview, and not be allowed to access the Tobacco Use Interview from the main menu.

**3.d.iv Random Prompt Interviews**

* Random Prompt Interviews were surveys to be completed by the user within three minutes of a Random Prompt Notification firing.
* This set of questionnaires was different than the Tobacco Use Interview survey (Appendix C).
* This survey could be initiated by logging into the application within three minutes of a notification firing.
* If there were multiple notifications within Android’s notification area, the user would only be able to initiate the Random Prompt Interview based off of the most recently fired notification. All other notifications were recorded as having been missed (Discussed below).
* The user was brought through a series of questions which utilized checkboxes, radio boxes, and sliders to record their responses.
  + If a user did not answer all of the questions within a given page of the survey, he/she was not allowed to continue to the next page.
* Depending on the answers, the user might have been requested to fill out one or two additional questions within the Random Prompt Interview survey (Figure 7).
* The survey input was only recorded to the ‘.rpt’ file (the file which kept record of all of the user’s Random Prompt Interview answers) once they answer ALL questions within the survey.
* If a user wanted access to the main menu of the application within three minutes of the last notification firing, he/she first had to take a Random Prompt Interview survey.
  + If a user began, but did not finish a Random Prompt Interview survey by exiting out of the application in the middle of the Random Prompt Interview survey, the survey was recorded as having been abandoned along with a timestamp at the point of abandonment (Discussed below).
  + The user did NOT receive an alert that the survey has been recorded as abandoned.
* If a user began a Random Prompt Interview Survey but did not finish it within five minutes, that survey was recorded as having been abandoned (Discussed below).
  + This could only occur if the display setting of the Android device was set to remain open for at least five minutes.
  + The user DID receive an alert that they had taken too long to complete the survey and had been recorded as abandoned.

**3d.v Time Functions**

Two functions within the application were developed for the convenience of the user. These functions allowed the user to cater local notifications around their own schedule. One such function suspended future notifications for a selected short period of time. The other function allowed a user to put the application to sleep for a longer period of time, during which time they did not receive any notifications. An alarm signaled the user that they had begun to receive notifications again.

**3.d.vi Suspending Notifications**

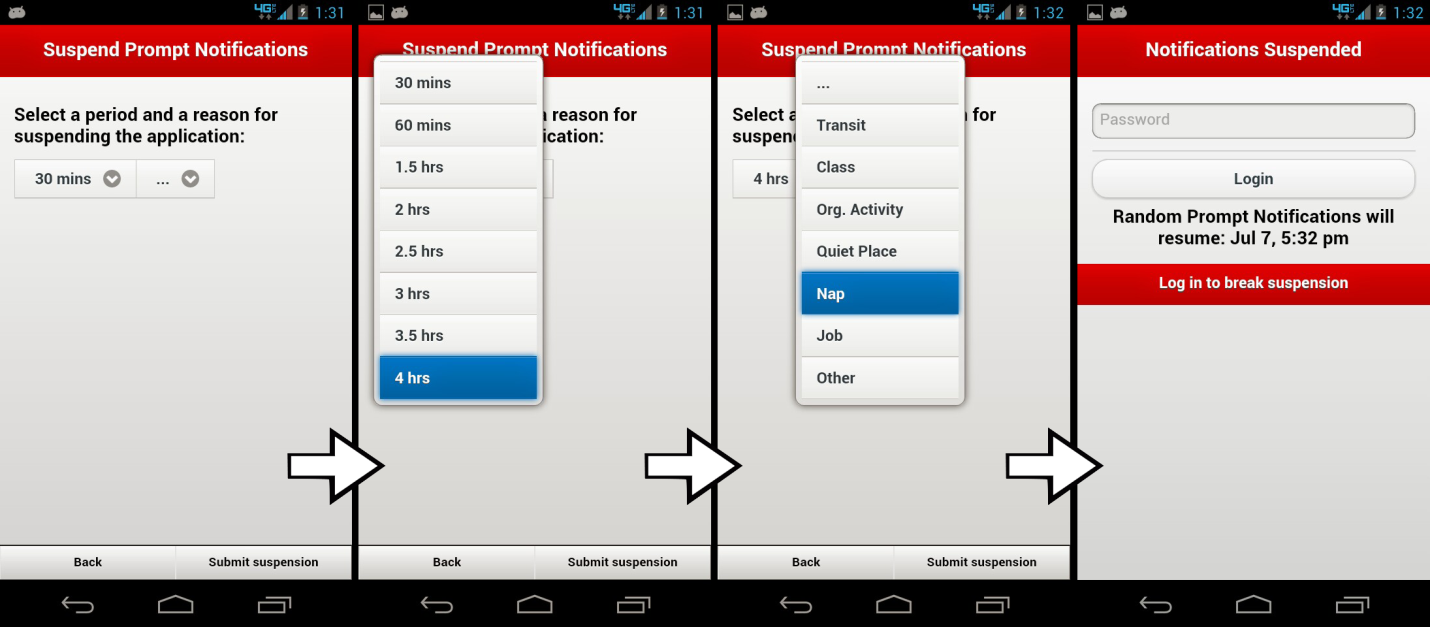


Figure 4: Screenshot composition of the the running e-cigarette survey application’s suspension of prompt notifications. From left to right: initial Suspension Prompt Notification page; time options drop down widget; reason for suspension drop down widget; resultant login screen after suspending notifications.

* A user could access the suspension function from the main menu of the application by tapping on the ‘Suspend’ button.
* The user was taken to the ‘Suspend Prompt Notifications’ page where he/she could change options related to the suspension (Figure 8).
  + A user had to choose an amount of time for which they wanted to suspend notifications.
  + A user must also choose a reason for the suspension.
    - If a user did not enter both an amount of time for suspension and a reason for suspension, he/she was alerted and not allowed to suspend any notifications.
    - This information was not recorded because the research group did not regard it as important.
* Once a user completed the steps to suspend notifications, they were brought back to the login page of the application (Figure 8).
  + The login page now conveyed the suspended status of the application with a red themed border profile.
  + There also was text on the page to indicate the suspension.
    - ‘Notification Suspended’ was displayed at the top of the page.
    - A date and time were displayed below the login form to tell the user when to expect Random Prompt Notifications to resume (Figure 8).
    - The time displayed did not reflect when the next notification would occur. Instead, it showed the beginning of the next randomized notification period.

**3.d.vii Putting Application to Sleep**

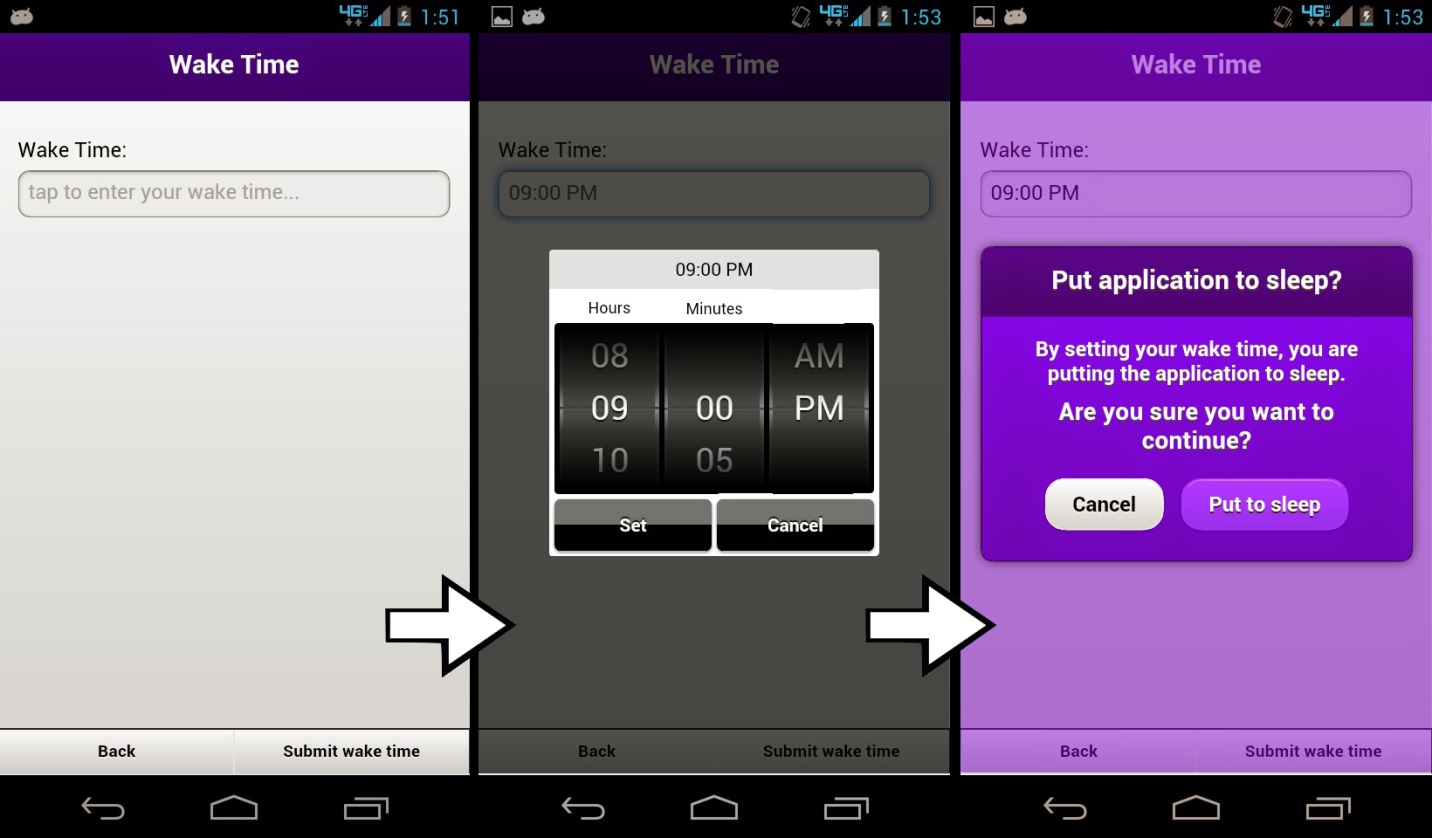
****

Figure 5: Screenshot compositionof the Wake Time alarm configuration page of the e-cigarette survey application; from left to right: initial Wake Time page; setting the wake time with the MobiScroll time scroller widget; waketime confirmation popup.

* A user could put the application to sleep to avoid receiving notifications for an extended period of time. This feature could be accessed from the main menu of the application by tapping ‘Set Wake Time’.
* Once a user was within the ‘Wake Time’ page, he/she could tap on an input field to bring up a three-columned time picker widget (Figure 9).
  + This widget comes from a polished Javascript widget library, *MobiScroll* [14].
  + The time range was in non-military format and the minutes changed in five-minute increments.
  + Once a time was chosen the user had to tap the ‘Set’ button to continue with the current wake time setting, or ‘Cancel’ to reset the widget’s time and clear the input field.
* Before the application could be put to sleep, the user had to select ‘Submit wake time’ in the lower right hand corner. This resulted in a pop-up asking the user whether or not they truly wanted to put the application to sleep (Figure 9).
  + If a user chose ‘Put to sleep’, no further notifications would be sent until the registered wake time was met.
  + If a user chose ‘Cancel’, then the user would simply be brought back to the ‘Wake Time’ page.

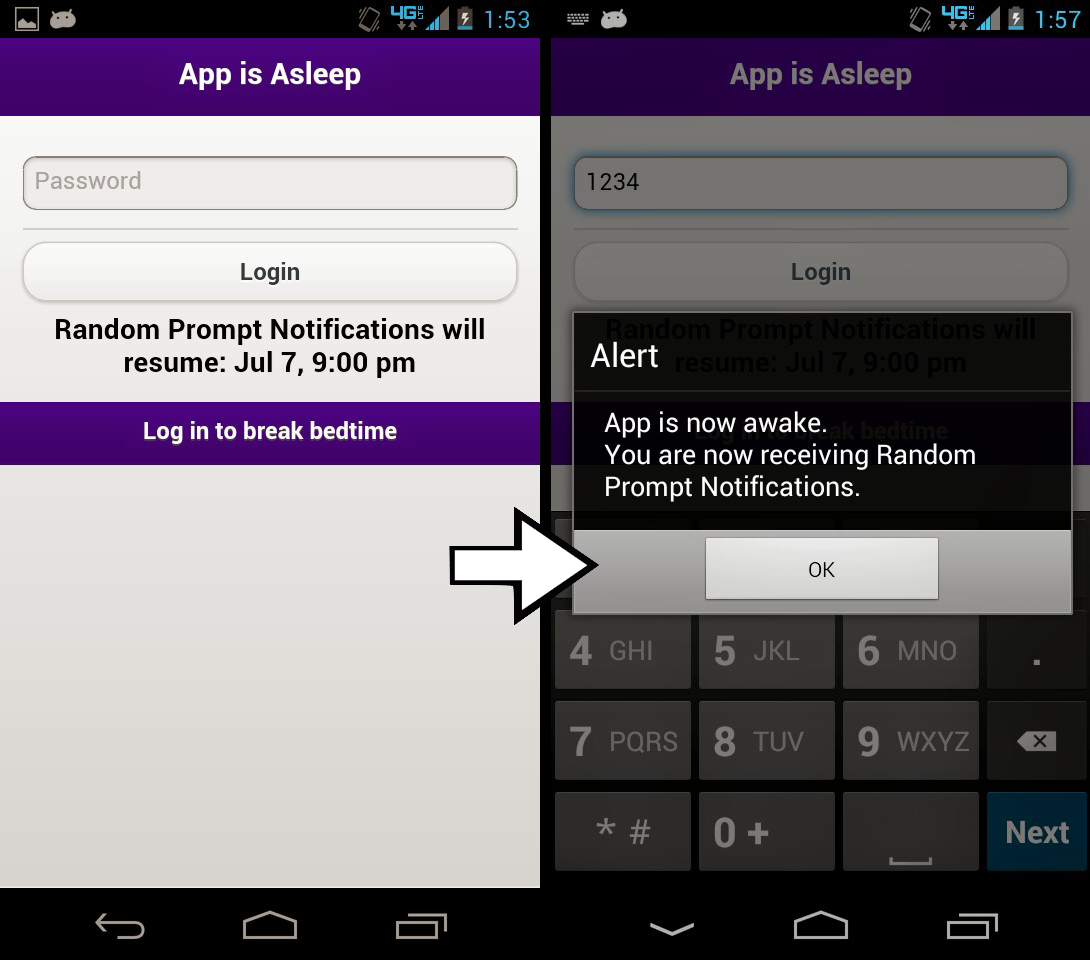
****

Figure 6: Screenshot of the Wake Time function of the e-cigarette survey application. From left to right: resultant login screen after setting a wake time alarm; alert after logging in while application is still asleep.

* Once a user put the application to sleep, the user would be brought to a purple colored login screen (Figure 10).
  + The header, body, and footer text of the login page would change to convey the new status of the login page.
    - ‘App is Asleep’ appeared on the header and ‘Log in to break bedtime’ appeared on the footer of the page.
    - Most importantly, a date and time that the user could expect to receive notifications again appeared below the login button.
* When the Wake Time set by the user had been reached, a unique three-minute long wake time ringtone accompanied the wake time notification, alerting the user that again local notifications could be received.
* The user could interact with the accompanying notification; however, this interaction would not result in the user being able to take a random prompt interview like after other notifications. Instead, the user would be taken to the main menu upon logging in.
* If a user decided to take the app out of sleep at a time earlier than the set Wake Time, the user only had to enter their login information at the login screen.
  + An alert would appear informing the user that the application was now awake and that the user was receiving notifications again (Figure 10).
  + At this time, all previous notifications were cleared and two days of randomized notifications were added.
  + The time span for these new notifications ranged from the time of logging in, until 48 hours later.

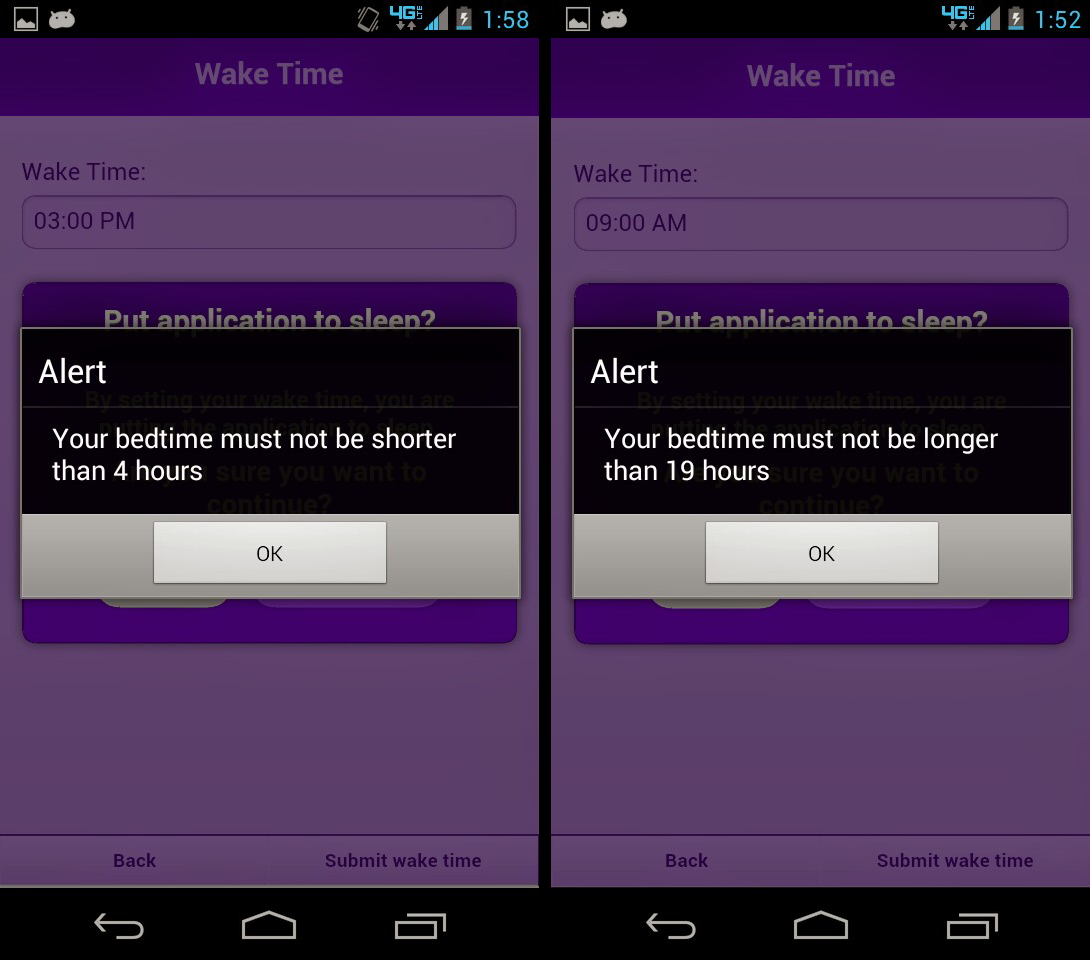
****

Figure 7: From left to right: alert after trying to enter a wake time within 4 fours of the current time; alert after tring to enter a wake time outside of 19 hours from current wake time.

* There were two limitations enforced by the wake time function (Figure 11).
  + The user could not enter a wake time that was less than 4 hours from the time that the user had put the application to sleep.
  + Also, the user could not enter a wake time that was more than 19 hours from the time the user put the app to sleep.
  + Both of these limitations were enforced by alerting the user to the issue.

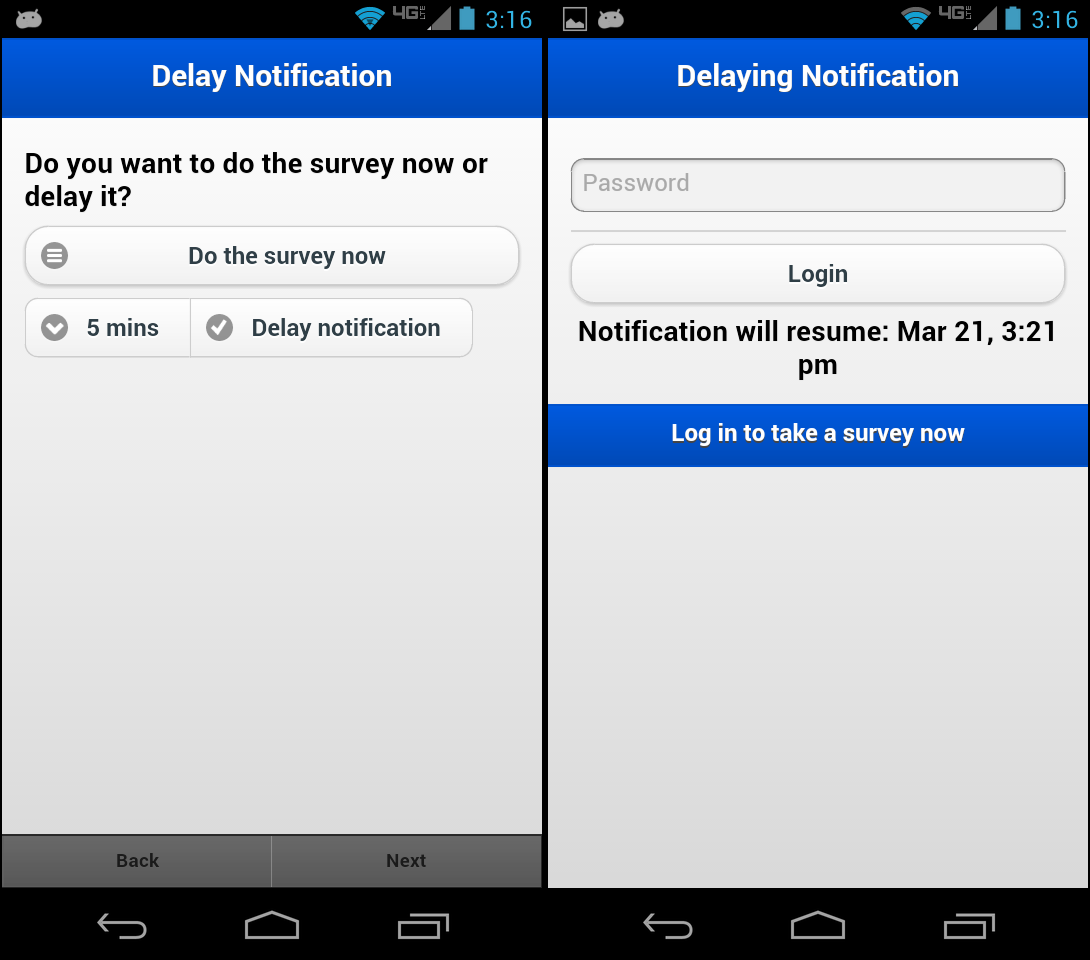


Figure 8: Screenshot composition of the defunct Delay Notifications capability within the e-cigarette survey application. From left to right: initial Delay Notification prompt; resultant login screen after delaying a notification.

An additional time function was added to the application which allowed a user to delay any notification as it fired (Figure 12). By tapping on a notification, a user would be brought to a screen—prior to the login screen—which would allow them to choose whether or not they wanted to delay the current notification. Not delaying the notification would result in the user having to take a random prompt interview immediately. The user could delay the notification anywhere from 5-30 minutes. Once a user delayed a notification, they would be brought to the login screen where a blue color profile and text would inform the user of the notification being delayed. Due to a necessary rework in notification mechanics within the application, this delay functionality had to be stripped from the final product.

**3.d. vii Notifications**

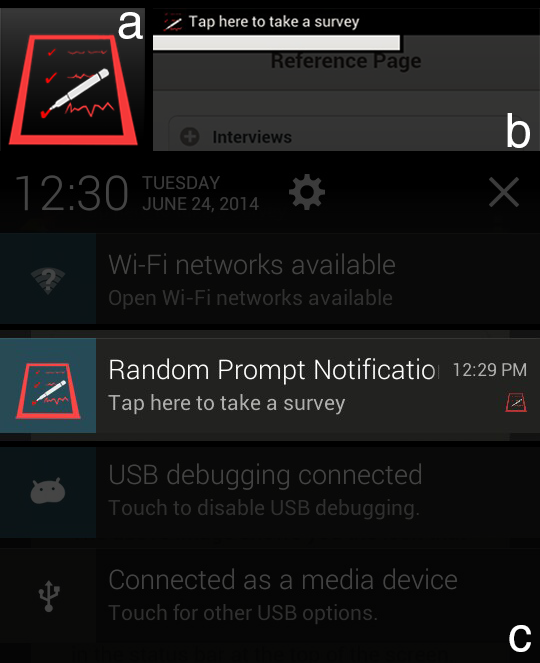


Figure 9: Screenshot composition of local notifications within the e-cigarette survey application.(a) Enlarged application & notification icon; (b) Highlighted status bar pop up upon receiving a notification; (c) Highlighted notification message within Android’s notificaiton drawer.

**3.d.viii.1 Random Prompt Notifications**

* A user received up to ten Random Prompt Notifications every 24 hours.
* Each Random Prompt Notification appeared in Android’s notification area and featured this icon: 
* A three minute long ringtone that increased in pitch every minute accompanied each notification.
* The text contained within the Random Prompt Notification requested that the user tapped on a notification to begin a Random Prompt Interview (Figure 13).
* The time and id of the Random Prompt Notification would also be displayed within the notification.
* Tapping on a Random Prompt Notification took the user to the login page of the application.
* All notifications which populated the Android notification area were cleared only when the user logged into the application.
* If any of the Random Prompt Notifications were older than three minutes, they were recorded as having been missed.
* If a user logged in to the application within three minutes of the latest Random Prompt Notification having fired, they were taken to the beginning of the Random Prompt Interview survey.

**3.d.viii.2 Wake Time Alarm Notifications**

* This notification only appeared if a user set a Wake Time Alarm for the application from the main menu.
* Each Wake Time Alarm Notification appeared in Android’s notification area and featured this icon: 
* A three minute long alarm ringtone accompanied each Wake Time Alarm Notification.
* This ringtone was different than the ringtone that accompanied Random Prompt Notifications with a sound that mirrored a beeping alarm clock.
* The text contained in the Wake Time Alarm Notification prompted the user that the application was now awake.
* The time of the Wake Time Alarm Notification was also displayed within the notification.
* Tapping on a Wake Time Alarm Notification brought the user to the login screen of the application.
* Wake Time Alarm Notifications differed from Random Prompt Notifications in the way they were cancelled.
* Simply tapping on a Wake Time Alarm Notification would cancel the notification.
* There was no connection between Wake Time Alarm Notifications and Random Prompt Notifications.
  + Missing a Wake Time Alarm Notification would not be recorded as the user having missed a notification.
  + Interacting with a Wake Time Alarm Notification would not result in the user having initiated a Random Prompt Interview.

**3.d.viii.3 Notification Mechanics**

* When a user started the application for the first time, the app populated two days-worth of Random Prompt Notifications for a maximum of 20 notifications.
* If the application was started later in the day, earlier Random Prompt Notifications which would have already occurred were not scheduled.
* This meant that the user usually received less than ten Random Prompt Notifications for that first day.

**3.d.ix. Reference Page**

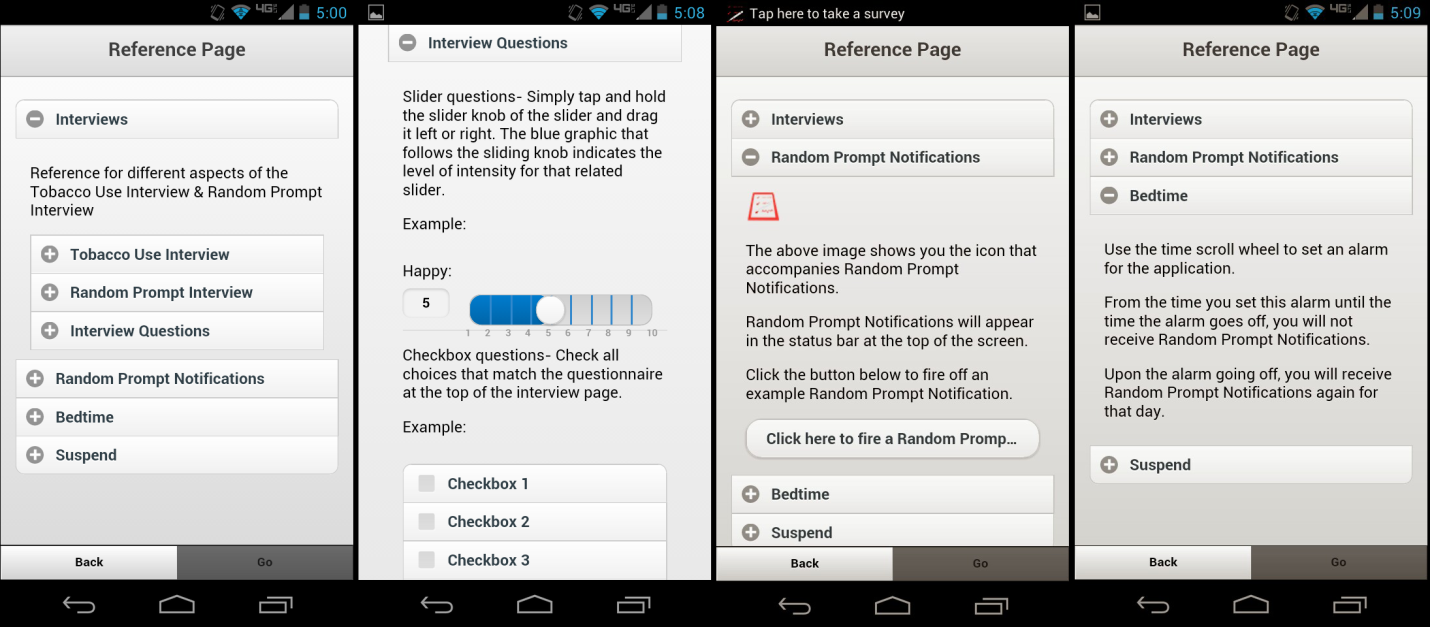


Figure 10: Screenshot composition of the Reference page within the running e-cigarette survey application; from left to right: dropdown segment showcasing the interviews within the application; dropdown further detailing the question & widget types which can be found within the interviews; dropwdown allowing a user to experience what a notification looks and sounds like; dropdown explaining the bedtime—or Wake Time—function within the application.

To help assist users in using the application, a page was dedicated to displaying different aspects of its performance. The reference section covered the general differences between the two interviews, the intended use of the Wake Time alarm, and the operation of the suspension function. This page also allowed a user to fire off an example notification in order to familiarize the user with future notifications.

**3.d.x File Structure**

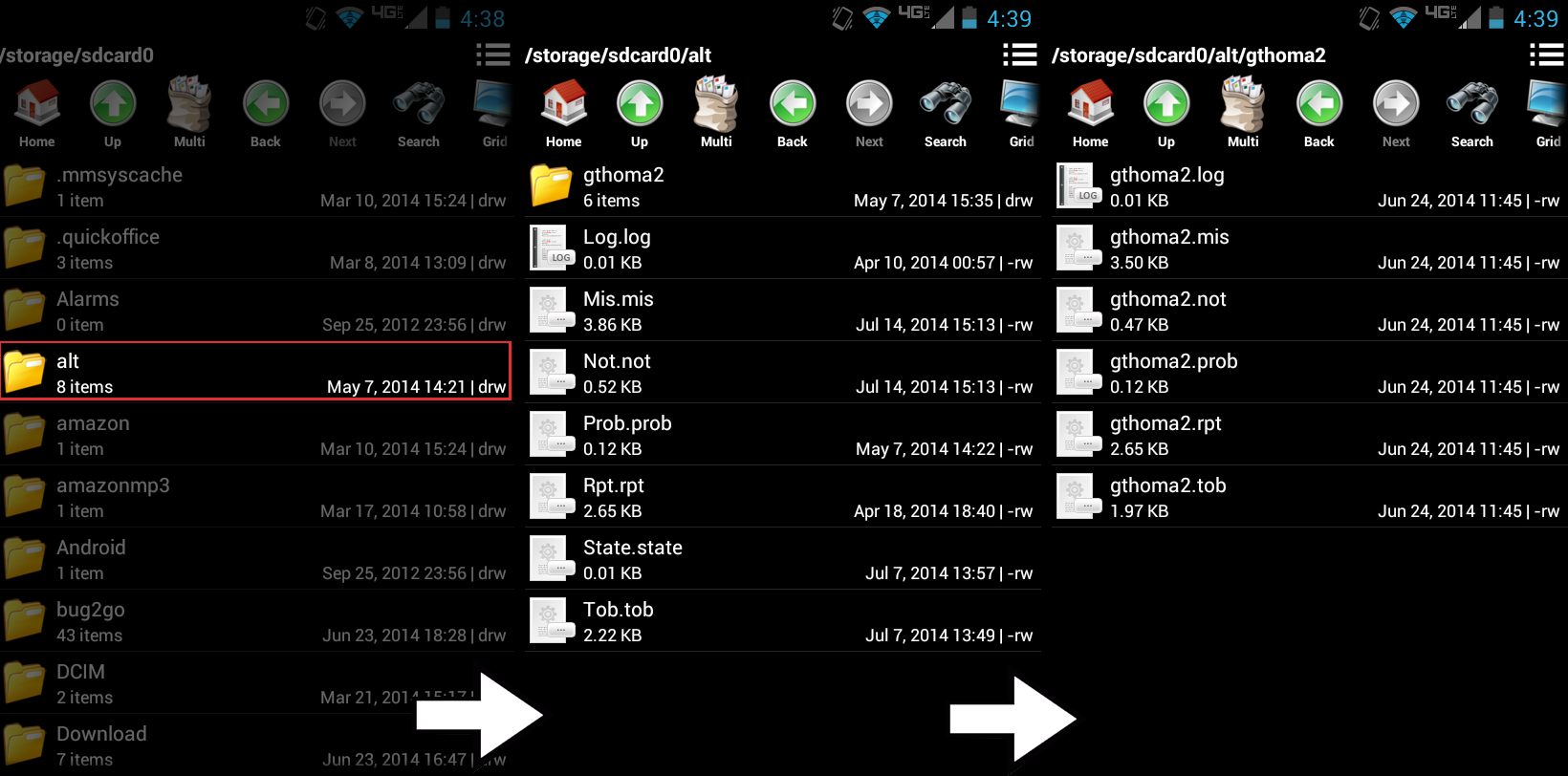


Figure 11: From left to right: From the root directory, a highlighted ‘alt’ subdirectory containing all user data for the application; inside the ‘alt’ directory, containing all encrypted data files; within the user directory which contains all relevant decrypted data.

The file structure which contained the data from the application was a simple one. From the root of the Android device, a folder named ‘alt’ was created on the initial startup of the application. All of the encrypted user data files were stored within ‘alt’. All of the names and extensions within this directory reflected the nature of their content. For example: ‘log.log’ was the file that contained the encrypted contents of the current users login credentials, while ‘tob.tob’ held the accumulated encrypted contents of that user’s tobacco use interviews.

To help distinguish between the active encrypted user data and the prepared decrypted user data, a subdirectory was created upon a researcher tapping ‘Prepare Files’ within the administrative profile menu (Discussed below). The name of the directory was the current user’s user id. All relevant data files from ‘alt/…’ were copied over to this directory, decrypted, and renamed to the user’s id (Figure 15). For example: tob.tob became userID.tob and log.log became userID.log (Appendices D and E).

Due to an odd error in Android’s storage system interfacing with Windows Explorer, an extra step was necessary for the research group to grab the processed data off of the phone. If a research member tried to grab the processed data from the newly created, decrypted user directory, the data would not appear from their PC. The solution to this problem was to archive the decrypted user directory. After archiving the user directory, the processed data would be visible within ‘alt/…’.

**3.d.xi Encryption and Decryption**

Because users of the device had free reign to visit the directories where the user data was stored, encrypting & hiding the data from users was essential for the research group. Initially, we discussed nesting the data within sub-directories of the Android system and hiding all files and folders. Unfortunately, this did not perform well as the folders and files were still visible from the Android File Manager app and the method of retrieving the user data proved too complicated for the research group. Instead, the previously mentioned file structure was formed and no folders or files were hidden (Figure 15).

*CryptoJS* [15] was a Javascript library that contained a plethora of cryptographic algorithms. 128-bit AES encryption & decryption was implemented within the application using *CryptoJS*. Unfortunately, the newline character was used as the active delimiter when decrypting documents with *CryptoJS* and the decryption would only cover the first line of the document. The research group desired decrypted user data with each entry on its own line, so an alternate method of encryption & decryption had to be used.

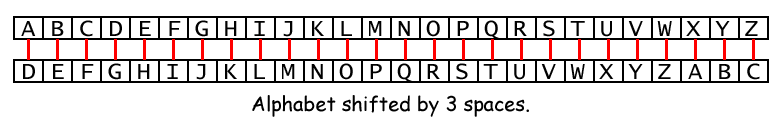


Figure 12: Diagram exaplaining the Caesar cipher.

Caesar cipher was a simple method of encryption achieved by shifting the place of letters by a certain number of spaces (Figure 16). This was accomplished in the application by increasing the ASCII value of each character by 13. Decryption occurred by decreasing the ASCII value of those characters by the same number. The original Caesar cipher Javascript library used in the application [16] did not incorporate the shift for numbers or punctuation marks, so the open source library was altered to support these additions (Appendices D and E).

**3.d.xii. Administrative Functions**

In order for the research group to easily grab all of the user data from each phone, an administrative pin number was hard-coded into the application. Logging into the application with ‘45459’ would navigate the user to an administrative profile page where researchers would be able to ‘Prepare Files’. This function was applied to all of the data to be collected by the administrator of the group. An administrator could also delete all of the data currently collected by the user.

**3.d.xii.1 Prepare Files**

* This function gathered all text files that have had data recorded to them, renamed the files and migrated them according to the file structure outlined in an earlier section of this report.
* As the data within the files were encrypted upon being recorded, the data needed to be decrypted before being read by researchers.
  + A reverse Caesar crypt was applied to the contents of each file.
* Due to a compatibility issue between Android storage devices and the Windows operating system, a user had to archive the created user data folder before being able to grab the data via Windows Explorer.
  + The updated folder of data simply wouldn’t appear otherwise.
* The only way to cease all future scheduled notifications from the application was to cancel all of them, so a final function call was made to cancel all future notifications.
  + In the case that this was not the final use of the application, and future notifications were desired, more notifications would be added upon the user logging into the app.

**3.d.xii. 2 Delete Files**

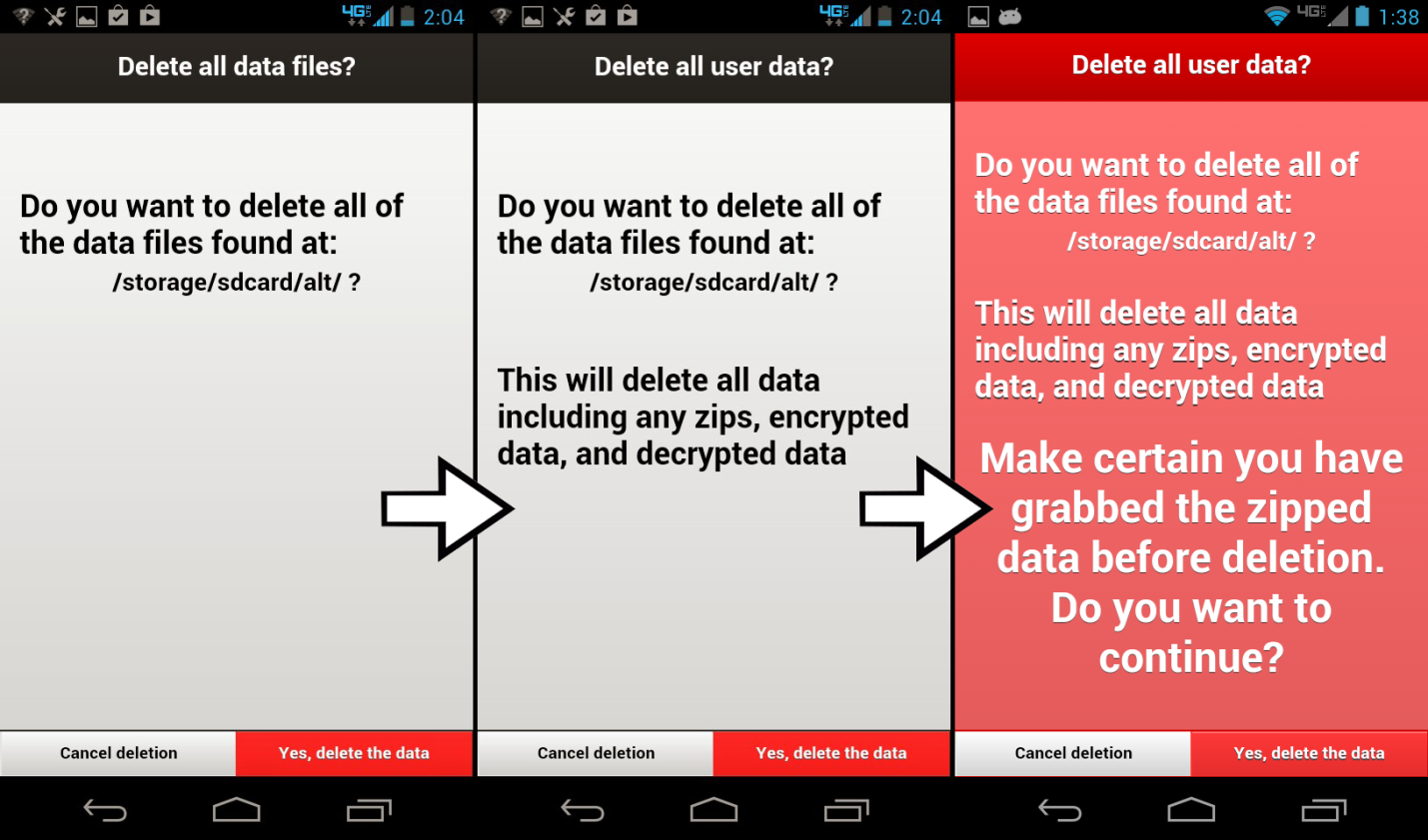
****

Figure 13: Screenshot composition showing the progression of redundant screens that an administrator must traverse before finally being able to delete all user data.

* This function simply deleted all of the data within the root folder of the application.
* This effectively reset the app so that it would be ready for a new user.
* Deleting the data files through this function call was final, so a warning screen was developed in triplicate. This required a user to click through before actually deleting any data (Figure 17).
* Similar to the ‘Prepare Files’ function, calling this method also deleted all future scheduled notifications.

**3.e Changes and Concessions**

At the beginning of the project, the research team outlined their desires for the final product. A responsive and intuitive cross-platform (Android and iOS) application was wanted. Because of this, the group looked into *PhoneGap* as a development platform. *PhoneGap* boasts the ability to more easily cross over mobile applications for Android, iOS, Windows Phone, Blackberry, etc. during development [10].

Unfortunately, the *PhoneGap* package proved to be problematic throughout development. Certain basic needs for the application were difficult to obtain and required outside *plug-ins* in order for them to be realized. The *plug-ins* themselves proved problematic as well, as they were often below version 1.0. Furthermore, each *plug-in* was dependent on the version of Android and *PhoneGap* used during development which introduced a multitude of compatibility issues to the project.

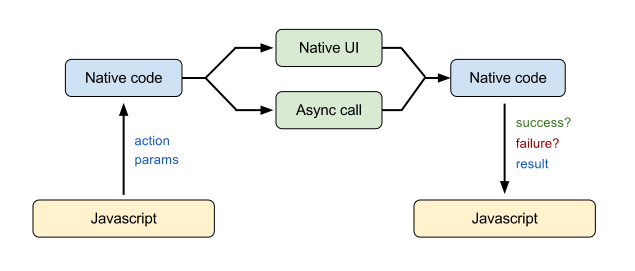


Figure 14: Flow chart of *PhoneGap’s* Javascript interface with mobile OS native code.

One reoccurring issue during development was coping with the interface between *PhoneGap,* *PhoneGap* plug-ins, and the asynchronous nature of Javascript code execution (Figure 18). Linear code operation was paramount to many aspects of the application running reliably. The local notification plug-in proved problematic in this regard as many functions within the application required clearing of prior notifications followed by the addition of new notifications. Asynchronous execution sometimes resulted in the addition of notifications in tandem with clearing notifications, which resulted in some or all of the new notifications having been cancelled. To address this particular issue a timer was instated before the addition of notifications so that no notifications would be added until one second had passed since the call to clear prior notifications.

Due to the rigid permission and priority system of the Android framework, some of the features available in the earlier Palm Pilot application were not viable for this project. Short of rooting each phone, the research group could not have absolute control over the volume of the phone nor the ability to stop a user from turning it off. This was remedied with instructions to the user group; they should not diminish the volume of the phone, nor have it turned off.

Each running process within an Android device was given a priority level. The priority level of a process changed over time depending on the amount of memory it was using, how often the process was visited by the user, as well as the nature of the process itself. WebView processes had the lowest priority level of any active process. This meant that when our application was pushed to the background of the Android device, it would inevitably relinquish its space in memory. Because of this, the local notification capability was particularly difficult to get in working order.

The initial method of handling local notifications relied on event listener functions, written in Javascript, to cause desired events upon a notification being added, cancelled, or triggered. The call to cancel a notification after three minutes would be added to the onAdd() event listener, for example. Similarly, the onCancel() event listener housed code that recorded whether or not the notification should be labeled as ‘missed’ by comparing the cancel time of the notification to the trigger time of the notification. Due to the above mentioned Android priority issue, these event listener functions would behave unreliably as the code within would not execute once the application memory was released. The result was that notifications would not faithfully cancel after three minutes, nor would notifications have been recorded as missed.

A major change was made to the application which made local notifications more reliable, more flexible, and still allowed the research group to keep much of the application’s desired performance. Due to the need for this change, however, one facet of the application was obviated; both the UI elements and mechanics behind delayed notifications were omitted. The reason for this omission was again due to the low priority level of the WebView application within the Android system once the application was pushed to the background. The delayed notification mechanic relied on the same event listener functions of the local notifications. Because of this, the delayed notification capability suffered from the same erratic behavior as the initial local notification facility. For example, in order to distinguish a missed notification from one that was delayed, an addendum was made to the body of the onCancel() event function which made a comparison between the time the notification was delayed and the time at which the delay should end, plus three minutes. The result was notifications wrongfully being regarded as not having been missed once the delay period had come and gone.

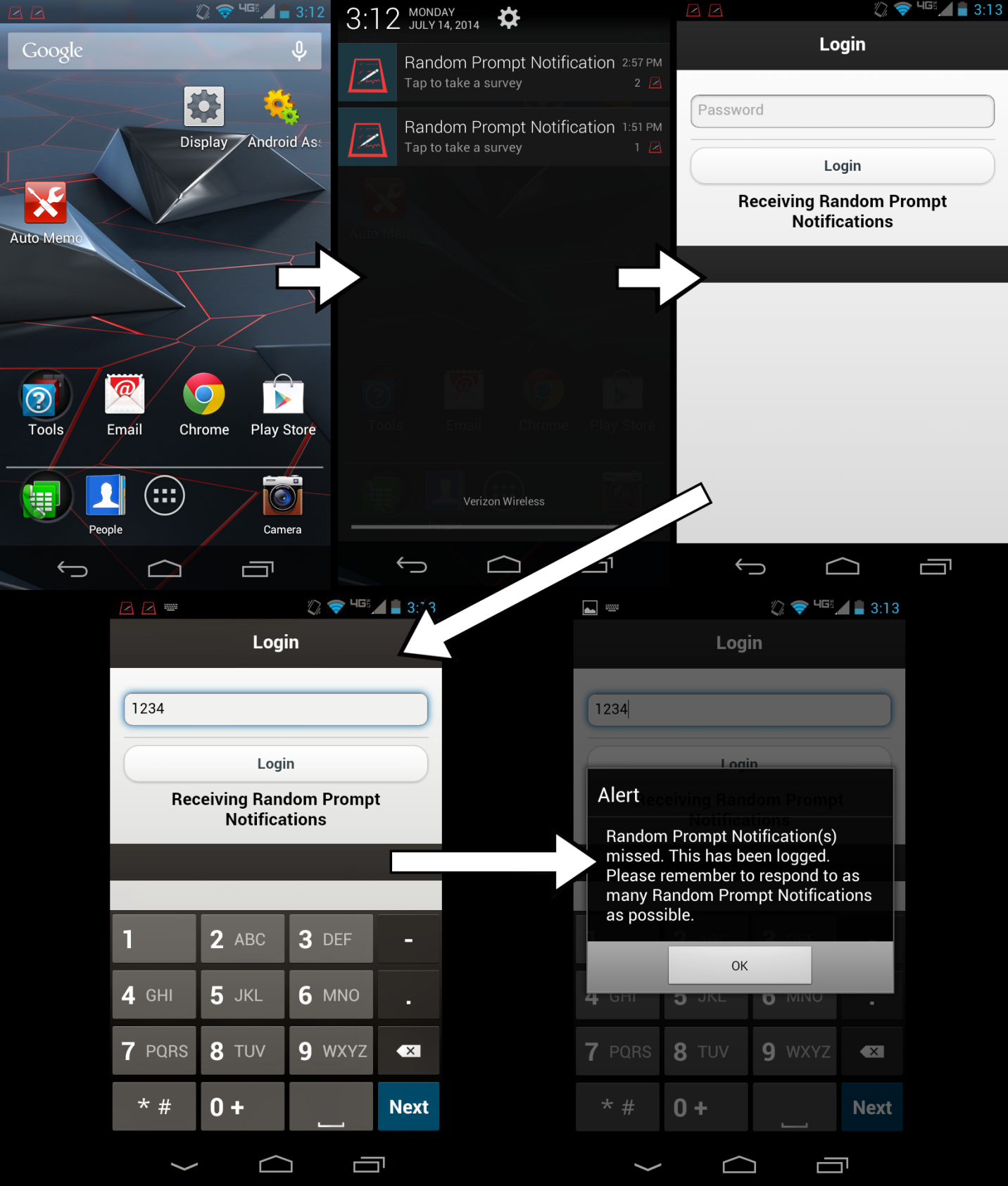
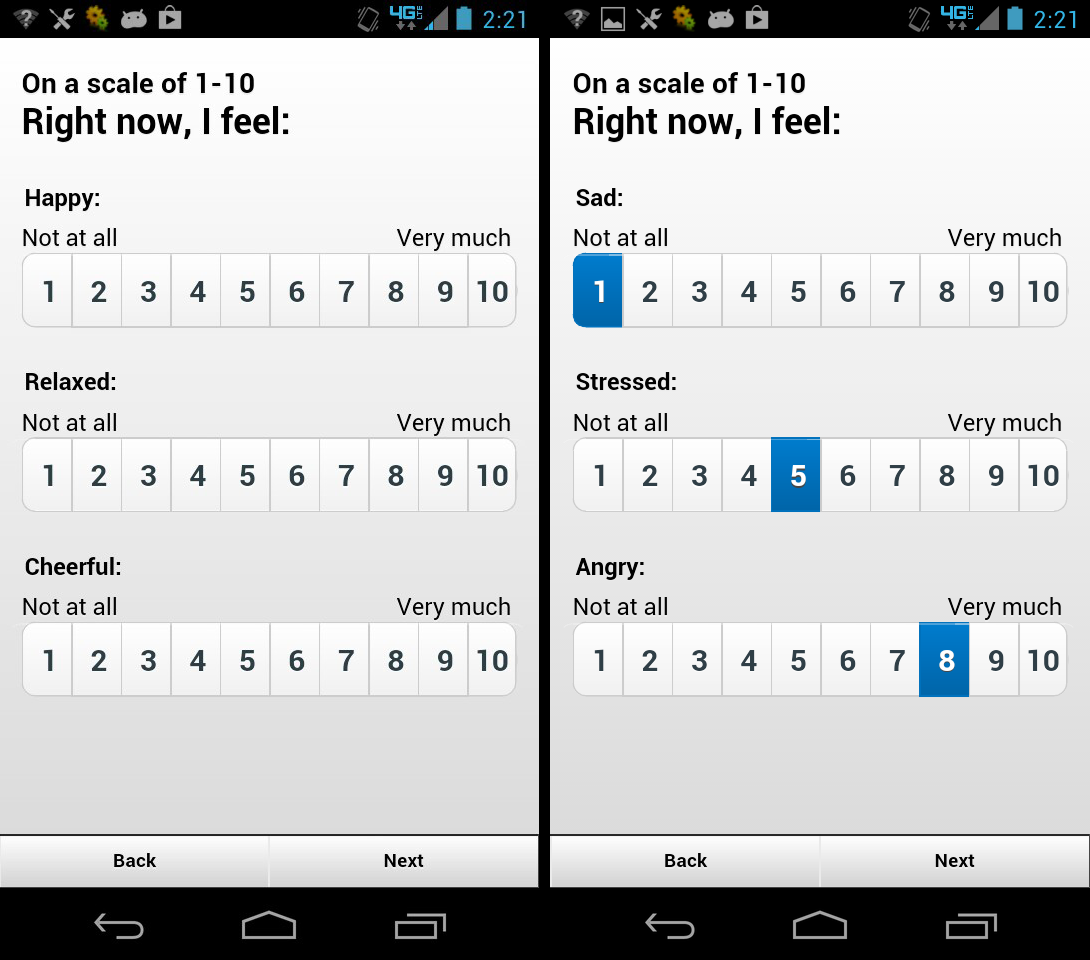


Figure 15: Screenshot composition outlining the flow of local notification interaction with regard to the revised notification mechanic. From Left to right & top to bottom: User has received two notificaitons which they have missed; user pulls down notification drawer to tap on a notification; the application is accessed with the login screen appearing first; user enters login pin; user is alerted to having missed notifications and the times at which they were missed is recorded, the missed notifications are cleared. At this time, the user is taken to the main menu of the application.

Due to the above mentioned change, the method for maintaining notifications within Android’s notification drawer was altered as well. Luckily these changes resulted in a more intuitive method for the user to handle any missed notifications. Before, notifications and their accompanying ringtone would only last for three minutes, after which they were cancelled and recorded as having been missed if a user did not interact with the notification. The new method of handling notifications resulted in the notifications remaining static within the drawer—they would not disappear after three minutes. This produced a reminder to the user that they had missed a notification. In the case that a user missed multiple notifications, they would be able to observe such, as the notifications would stack within Android’s notification drawer, as well as the status bar at the top of the phone. In order to record notifications as having been missed—with regard to this new management of notifications—all of the times that notifications were scheduled to fire were saved to an array of Moment objects—provided by a high-level Javascript Datetime library, *MomentJS* [17]. If—at the time of login— one or more of those times was greater than three minutes ago, the application would record all of those times to file as having been missed and the user would be alerted.

Only one error persisted to the final release of the application; multiple *jQuery Mobile* sliders grouped on the same page would result in a rarely occurring control issue. The issue being that sometimes when a user changed control of one slider handle to another slider, the newly controlled slider would regulate the position and value of the old slider as well. A user could easily fix the issue by controlling the old slider to reselect their old value. Researching and seeking help to fix this problem proved unfruitful as the underlying issue lay with *jQuery* [12], and implementing a UI library which did not rely on *jQuery*—such as *Sencha Touch* 2 [18]— would have resulted in a complete UI overhaul. A fix for the final release of the app increased the performance and responsiveness of the sliders, however. This resulted in a far less frequent occurrence of the problem.



**Figure 16:** Screenshot composition showing the new horizontal radio button widget groups. The left displays the widget group before they have been interacted with, while the right displays the widget group after each radio button group has been altered.

This issue with the sliders was ultimately resolved by replacing the sliders entirely. Instead of sliders, a group of horizontal radio buttons were implemented. The performance of these horizontal radio buttons was functionally the same as the sliders, the only difference between the two widget types was that one allowed the user to drag a slider to attain desired values while the other only allowed a user to tap the desired values. Because of this replacement, the above mentioned error has disappeared from the program.

Early user feedback has led the group to believe that the survey submission buttons were located too close to the Android hardware buttons. Users reported unknowingly hitting the ‘Home’ button during a Random Prompt Interview, exiting the app, and being unable to rejoin the Random Prompt Interview in progress. This would result in the user being recorded as having missed a random prompt interview. Because of this, the submission buttons, ‘Next’ and ‘Back’ were raised half an inch from the hardware button.

An additional change was attempted which allowed a user to resume their old progress in a Random Prompt Interview in the case that they accidentally left the application in the middle of the interview. Implementing this change proved too problematic and undermined some of the more critical aspects of the application. Because of this it was removed from the app.

**Updated as of 9/8/2014:** Local notifications were not added, nor did they fire reliably. Some of them would be added, others would not even though the scheduled times passed to the plugin were correct… This did not occur prior to the most recent Android OS update (4.4.2). The local notification plugin used in development did not cater to this latest version of Android and might be the reason behind the notification’s erratic behavior. Another reason for this problem might be that most recent testing occurred on a Google Nexus 4, not the Razr M model used by the research group. Updating the local notification plugin should resolve the issue (at the time of writing this update, the local notification plugin has not been updated to work with Android 4.4.2)

-https://github.com/katzer/cordova-plugin-local-notifications