- Description
- Intended User
- Features
- User Interface Mocks
  - Home Page
  - Screen 2
- Key Considerations
    - How will your app handle data persistence?
    - Describe any corner cases in the UX.
    - Describe any libraries you'll be using and share your reasoning for including them.
- Next Steps: Required Tasks
  - Task 1: Project Setup
  - Task 2: Implement UI for Each Activity and Fragment
  - Task 3: Build Database and Provider
  - Task 4: Build Remind Activity
  - Task 5: Build Present Fragment
  - Task 6: Build Notification Fragment
  - Task 7: Build Past Fragment
  - Task 8: Build Widget
  - Task 9: Build Long Click Ability
  - Task 10: Build Tablet UI
  - Task 11: Integrate AdMob, Google Analytics, and Product Flavors

GitHub Username: ekeitho

Re-Mind

Description

Our phones distract most of us and that's why there is Re-Mind. Become more aware of your app usage with a beautiful design and set helpful reminders to not be distracted for too long. Long click items to tag them under Distraction/Productive Apps in order to sort them or share items to brag how well you did during the day on your phone.

Intended User

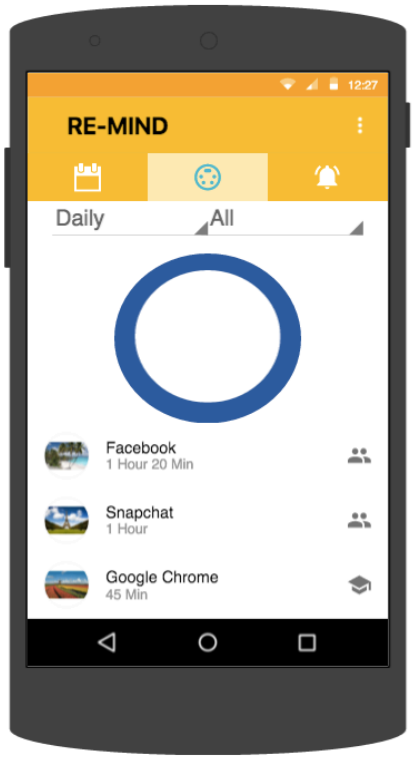
Users who are trying to become more productive or become more aware of their phone usage.

Features

- View app usage for a single day, week, month or a year
- Set goals and helpful time limitations on all or single apps

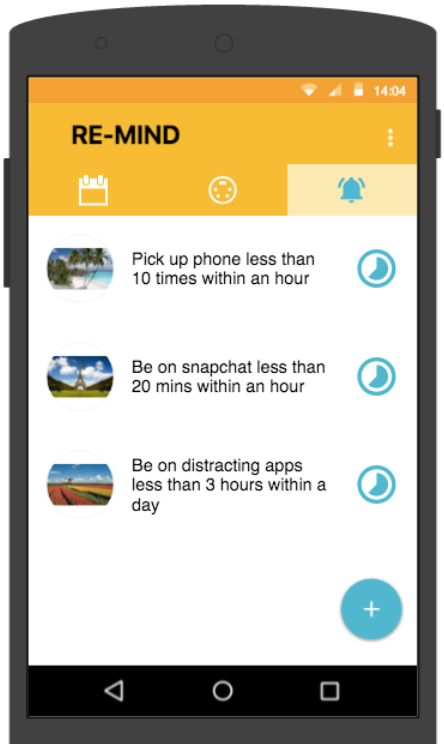
- Widget to easily view how much time left on goals
- Share and brag about your app usage results

User Interface Mocks



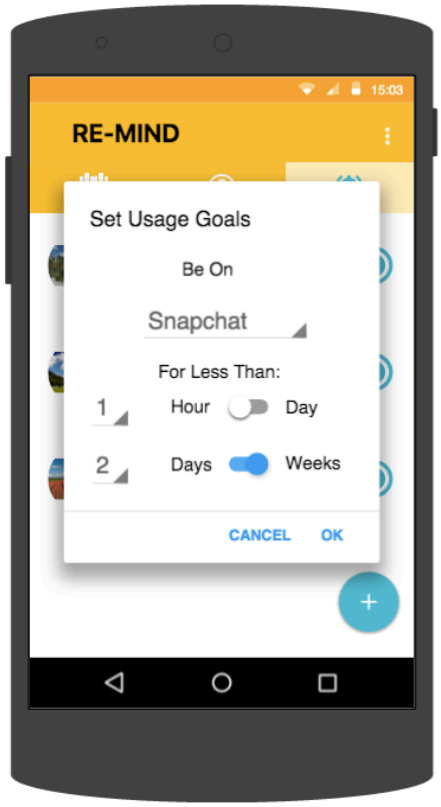
Home Page (present)

* On this screen the user can sort through their app usage based on the current day, and the beginning of the week, month, or year.
* The user will have the ability to share their app usage by long clicking the items. If the user chooses to share more than one item, the duration will aggregate.
* The user will be able to tag their items into productive/distractive by long clicking the items and choosing the category.
* The user will have the ability to sort through their items once they've chose a category for an app.
* There will be a pie chart to graphically describe their usage data



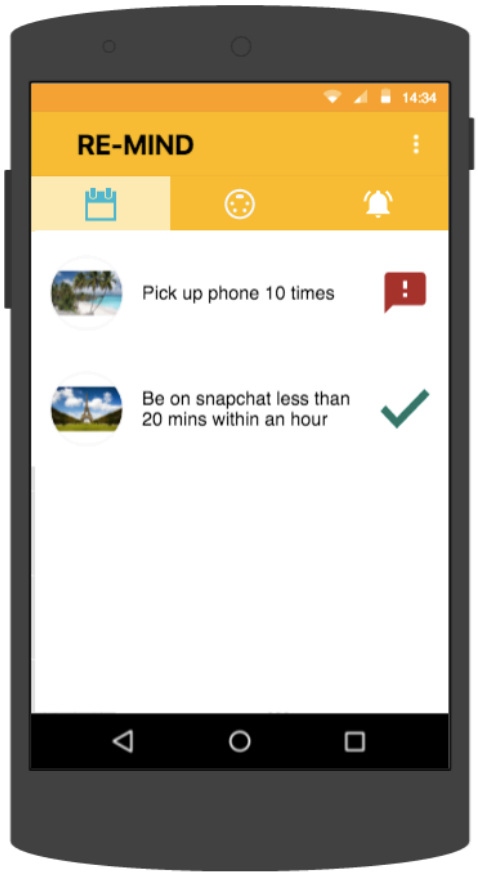
Notification Page (future)

* This is the screen where the user will add their goals to be on app for a certain amount of time.
* For each goal added, it will live update on the screen.
* The user will have the ability to long click the item and delete the goal.
* The data presented on this screen, will be the same data presented on the widget so that the user doesn't have to keep coming back to check on how much longer they have for the goal



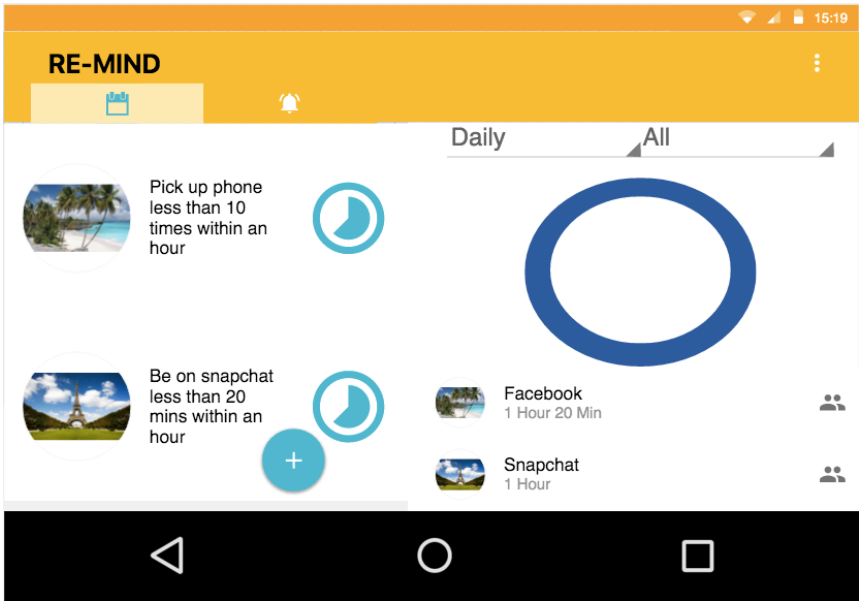
Dialogue (future)

* After the user clicks the fab button to add a goal, this will be the dialogue that the user will see
* The user can either set a time limit on all apps or a specific app. Based on this screen, the user will try to be on Snapchat for only 1 hour (choice length) in 2 weeks (duration length).
* If the user goes over their choice length, before their duration length, then they fail.
* If the user duration length is over before the choice length is, then the user passed the goal.



Failed/Passed Goals Page (past)

* This is the screen where all the completed goals will go (pass or failed)
* The user can use the information here to make better and more realistic goals the next time they decide to create another goal.
* For each failed goal, it'll have a red failed icon
* For each goal passed, it'll have a green check icon



Tablet UI

* present fragment is the main detail fragment
* notification and present fragment in view page

Key Considerations

How will your app handle data persistence?

For the home page, I will not need to store any data on the app usages, because the Usage Stats Manager generates this based on the request. However, I will store the goals that are added into the application using an SQLite table. Then create a content provider to grab the goal data from the widget service. I will also store the tags that the user adds into the Shared Preferences. The key being the app name and the value will be productive/distractive.

Describe any corner cases in the UX.

When the user long clicks a certain item and moves to another screen, the long clicks will not be cached.

Describe any libraries you'll be using and share your reasoning for including them.

I will be using and android chart library in order to get the pie graph set up on the home page. I'll specifically use PhilJay MPAndroid Chart library, since I've had past experience using this library and it is highly backed by many developers. I'll also integrate AdMob library on the free product version of my app, but use Google Analytics library on both product flavors. By using

the analytics library I can gain valuable insights on how often users are adding goals or how often they are checking their app usage.

Next Steps: Required Tasks

This is the section where you can take the main features of your app (declared above) and decompose them into tangible technical tasks that you can complete incrementally until you have a finished app.

Task 1: Project Setup

- Move every static string used in activity/fragment into strings.xml
- Create two product flavors (paid and free) using gradle
    - Make sure to use freeCompile on the adMob library gradle build
    - Configure dexOptions and set multiDexEnabled true
- Move all icons and pictures into drawable
- Add permissions (PackageUsageStats, WakeLock, Internet, AccessNetworkState) to manifest and under application tag enable RTL true
- Define colorPrimary, colorPrimaryDark, colorAccent, textColor, and secondaryTextColor in the colors.xml and create custom theme using those colors

Task 2: Implement UI for Each Activity and Fragment

- Build UI for RemindActivity
    - Coordinate Layout
        - Default AppBarLayout
            - Toolbar
            - TabLayout
        - LongClickAppBarLayout (when the user long clicks, this is shown)
            - Toolbar
            - TabLayout
        - ViewPager
        - FloatingActionButton
- Build UI for fragments inside of ViewPager
    - Build UI for PastFragment
        - RecyclerView
            - Detailed item layout
    - Build UI for PresentFragment
        - Two AppCompatSpinners
        - RecyclerView
            - Detailed item layout
    - Build UI for FutureFragment

        - RecyclerView
            - Detailed item layout

Task 3: Build Database and Provider

- Create a RemindNotificationItem class that will maintain all the data needed in one area
    - timeCreated, type, name, choice, choiceLength, notification, notificationLength, id, usageLeft
- Create a SQLiteOpenHelper with the variables used in the class
- Create a content provider
    - Authority: com.ekeitho.remind.provider
- Add content provider permission to manifest

Task 4: Build Remind Activity

- create a method to check if user has accepted the usage access permission
- setContentView
    - snackbar needs a context
- check permission
    - if no, show indefinite snackbar, and give them a quick action to get to the action settings
    - if yes or come back from the action settings, set up rest of the views
- set up views
    - get access to the toolbar, the fab, the view pager, the view pagers adapter fragmentStatePagerAdapter, and the tablayout
        - page one, PastFragment
        - page two, ChartFragment
        - page three, NotificationFragment
- add Icons and selection changes to the tablayout
- start a service
    - fire an async task that will aggregate app usage stats for the daily, weekly, monthly, and yearly all in one go.
    - fire an alarm manager to go off every minute, that grabs data on notification items if any and relays it to the Notification Fragment and Widget.
        - If notification items go over their time, send user a notification, and add to the past fragment

Task 5: Build Present Fragment

- hide button on this page
- the UsageStats has a lot more information than needed, so I'll create a RemindStat class that will contain only the necessary data
    - must be parceable to send with an intent
    - variables: packageName, durationTime, and appTitle
- build an adapter that will take two types
    - a chart type view holder (position 0)

    - a remind type view holder (position > 0)
    - in order to do this, I'll extend the class Recycler.ViewHolder called RemindHolder
        - then create PackageViewHolder and ChartHolder that extends RemindHolder, which in turn can check the types onBindView and set up the views properly
- set up the date Spinner that will allow the user to sort through the daily to yearly views and also the sort Spinner, which will filter out distraction/productive/all apps.
    - Add listeners to the spinners which will notify the adapter on change
- register a broadcast receiver that accepts an intent of 'data-available' which is sent from the service, when the async task is complete
- based on the data received from the service and which position the two spinners are, set up the recycler view with the pie data and all the remind stats
    - the pie data won't have a legend but a color code near each detailed item in the recycler view, which will minimize space and increase usability

Task 6: Build Future Fragment

- show add notification fab button on this page
    - on click, show dialogue to add a notification item
        - call content provider insert on dialogue 'ok'
- set up a loader manager
    - content provider query for items no longer in progress
    - refresh adapter and recycler view on update

Task 7: Build Past Fragment

- hide fab button on this page
- set up a loader manager
    - content provider query for items no longer in progress
    - refresh adapter and recycler view on update

Task 8: Build Widget

- create widget layout with a title bar and a list view under it
- create AppWidgetProvider that uses the layout on the remote view
    - have click pending intent that launches the activity
- create RemoteViewFactory that hosts the detailed view of each item in the list view
- create a service the launches the remove view factory, which is called when the service has gotten new data from the content provider (each minute)
- add widget service and widget provider permission to manifest

Task 9: Build Long Click Ability

- Build a wrapper long click class that will hide default app bar layout and expand the new app bar layout to show different functionalities
    - Each page will have it's own set, which contains what items are clicked
    - On first insertion, show toolbar
    - On empty set, hide toolbar
- On present fragment and notification fragment
    - On long clicks
        - If present fragment, show tag icon and share setting
        - If notification fragment, show trash icon

Task 10: Build TabletUI

- Build tablet UI
    - Past and Notification fragment still in a view pager
- Edit RemindActivity to test if dual pane is true
    - If so, use fragment manager to add a fragment to a layout which is the Present Fragment

Task 11: Integrate AdMob, Google Analytics, and Product Flavors

- Create paid and free products
- When user clicks fab button on free, show admob banner on both tablet and mobile layouts
- Integrate analytics on both flavors
    - Test what kind of apps users are setting notifications for
    - Test how often users add notifications