GitHub Username: ethan627hsu

Wake

Description

Staying awake throughout the day isn't easy, that is why Wake is here to help! Wake uses the screen, vibration, sound and light capabilities of your smartphone to keep you going. By staying awake and alert, you can achieve more in your day.

Intended User

My app is meant for users who have very busy schedules. People who are working full-time, students, and frequent travelers are all users with most days being long. Long work days can make people very tired, so my app tries to combat that. Users should hopefully be more effective and productive after using Wake.

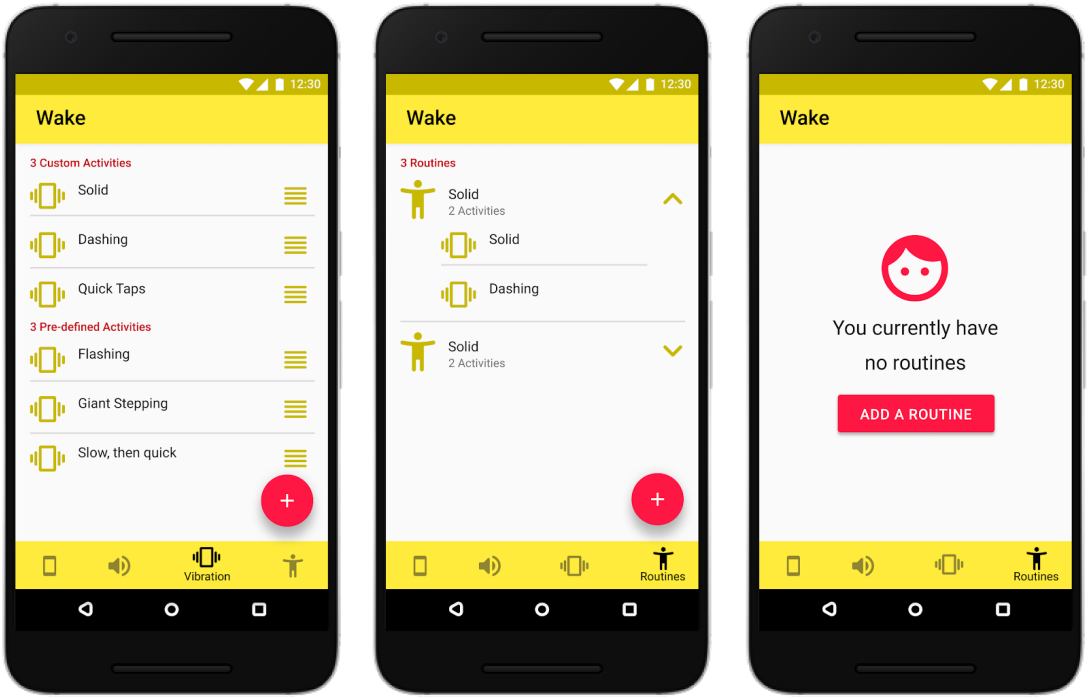
Features

- Wakes up the user by screen

- Stimulates the user by vibration
- Alerts the user by sound
- Provides activities (meaning the actions, varying animations, sounds and vibrations) by default
- User is able to group activities into routines
- Stores customization and activity/routine information on the device
- User can customize and add their own activities

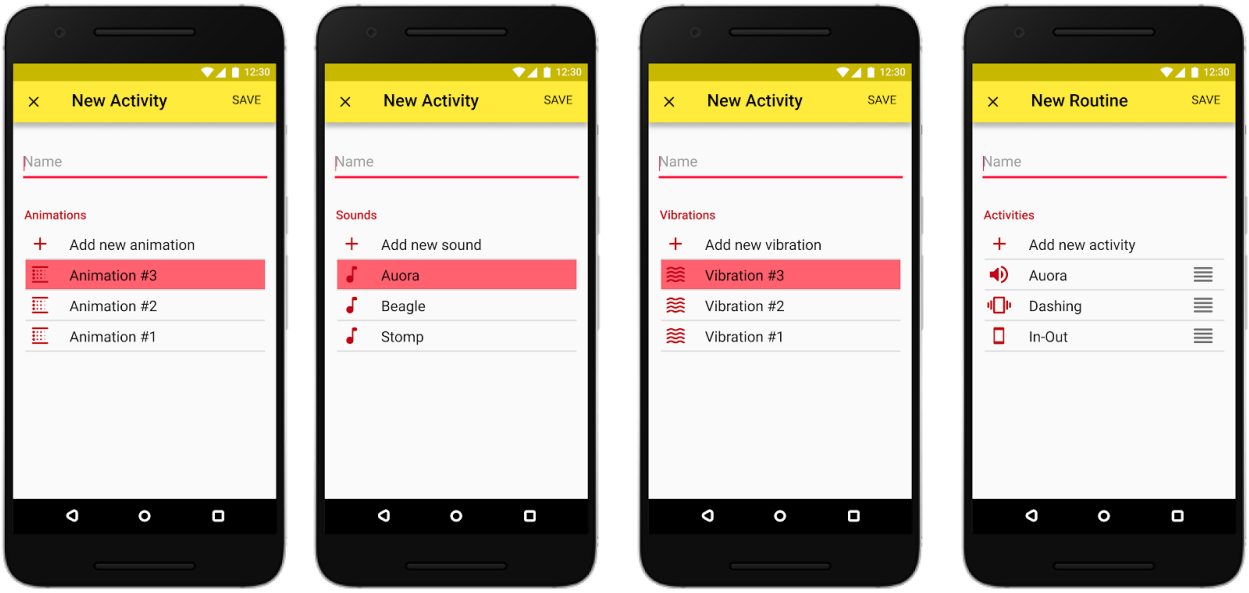
User Interface Mocks

Activity/Routine Viewer (Main UI)

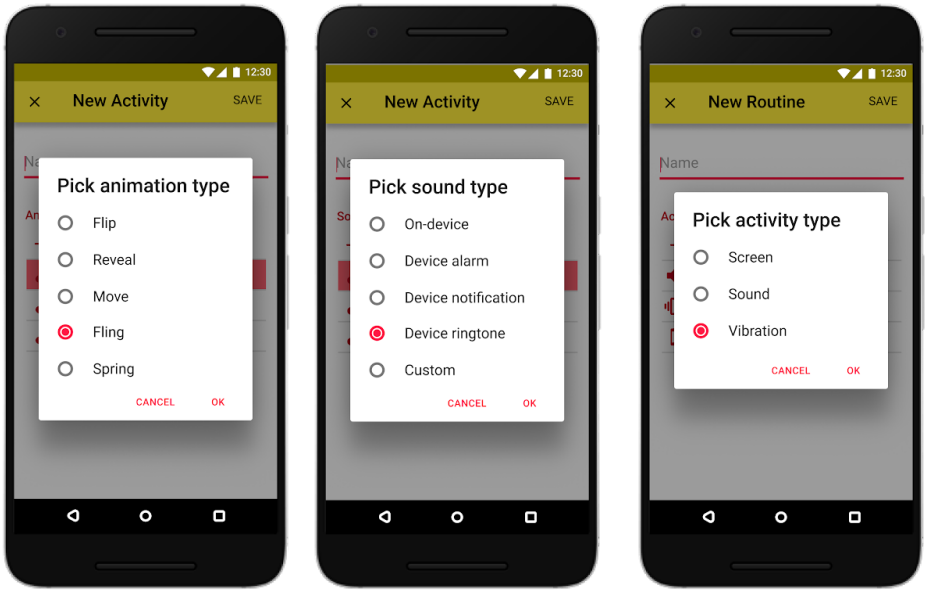


The activity/routine viewer provides the user with a way to efficiently look at their activities/routines. The viewer has bottom navigation to switch between types of activities and the routines. When an item on the bottom navigation is clicked, an animation will play to make the UI more smooth. Activities are displayed in a classic list, making the learning curve almost none. List items have an icon on the left to signify their type (screen, sound or vibration) and their name is displayed next to the icon. Additionally, holding the item, or the reorder icon on the right will allow the list item to be moved across the list. Finally, swiping left on a activity item will delete it, as seen with a red background and delete icon. To start adding activities/routines, the user can click the red floating action button with the common icon “add”. When clicked, the button will start the process of adding the type of activity/routine they were looking at in the viewer. Note, the button will disappear when viewing activities in free mode, since adding custom activities is a premium feature.

New Activity #1



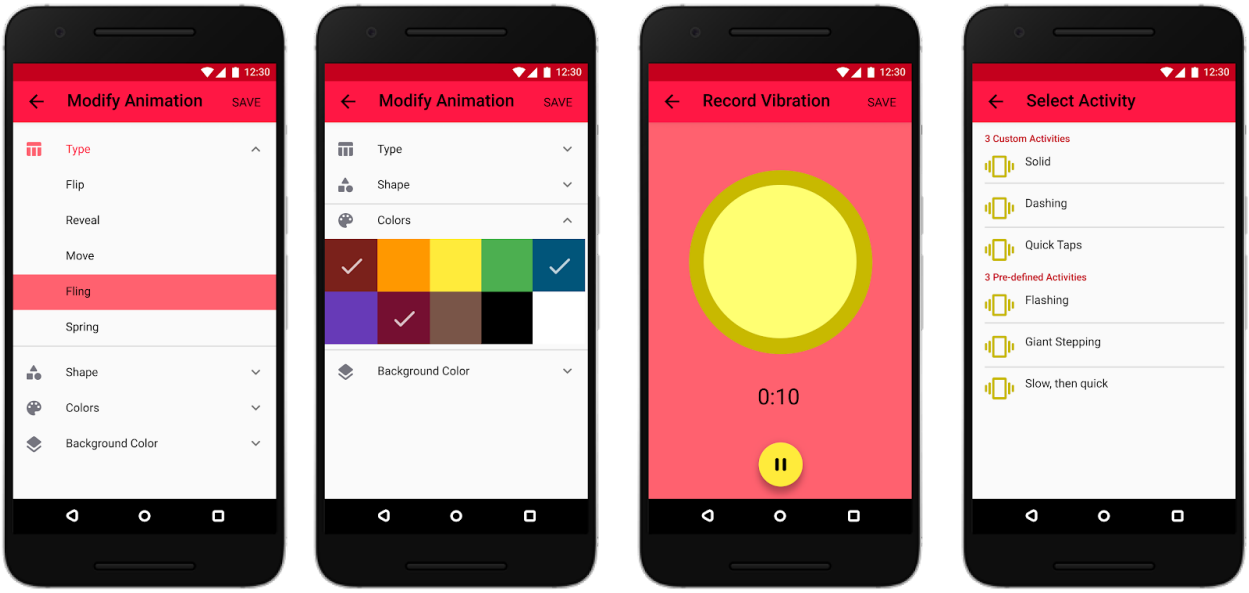
When clicking on the add button in the main UI, this is the first page they are land at to add an activity/routine. The page simply has two components, a text box, and a list. The text serves the place to type in the name of the desired activity. The list, provides easy-to-view access to animations/sounds/vibrations/activities the user has added during the current creation process. For activities, the content stored in the list serves as the action when the activity is played. To choose the piece of content to play during the activity, you can simply click on the list item. For routines, a reorder option is provided to change order in which activities are played. All list items can deleted with a swipe to the left, the same as the main UI. Finally, at the top is an “add new” list item that when clicked, helps the user add a piece of content to choose from. On the top toolbar, a cancel option is provided to very left of the toolbar and the SAVE button helps save the activity when the user is done.



New Activity #2

This screen is the midiary step between screen 1 and screen 3. It is displayed for some activity adds only (no vibration), when the “add new” list item is pressed. The some cases, the purpose of the pop-up dialog is to split up the process of adding content. In other cases, the information gathered within the dialog can completely change the UI that is seen after. The bottom of the dialog holds “CANCEL” and “OK” buttons to close the dialog or continue with the process.

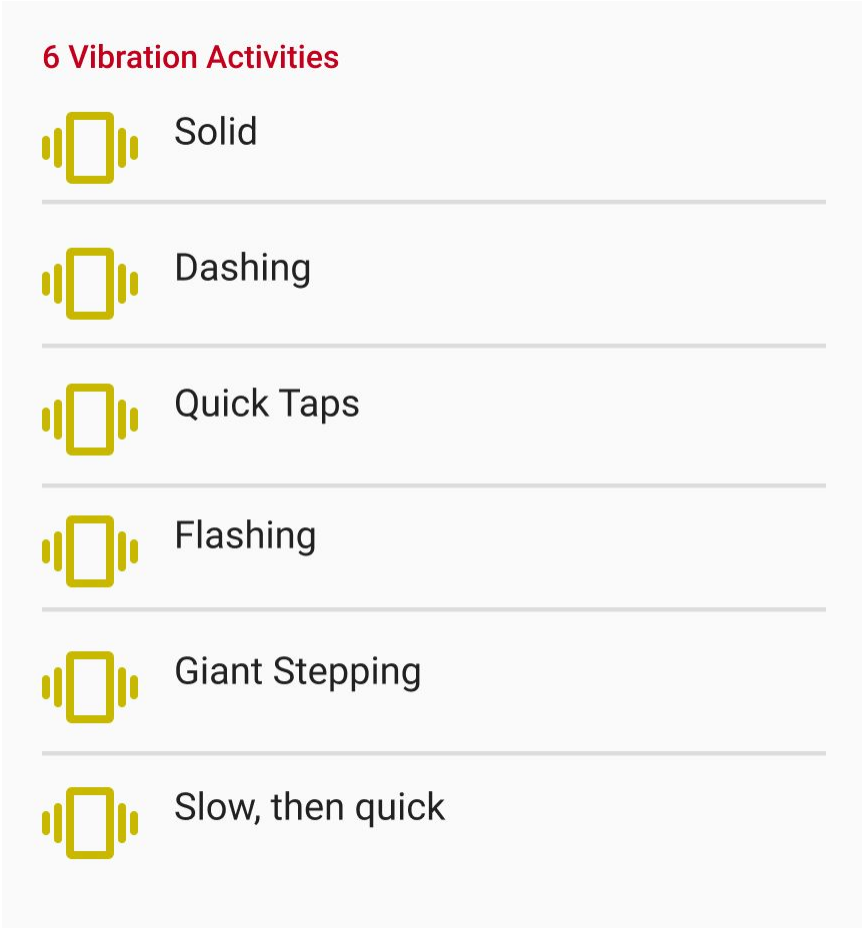
New Activity #3



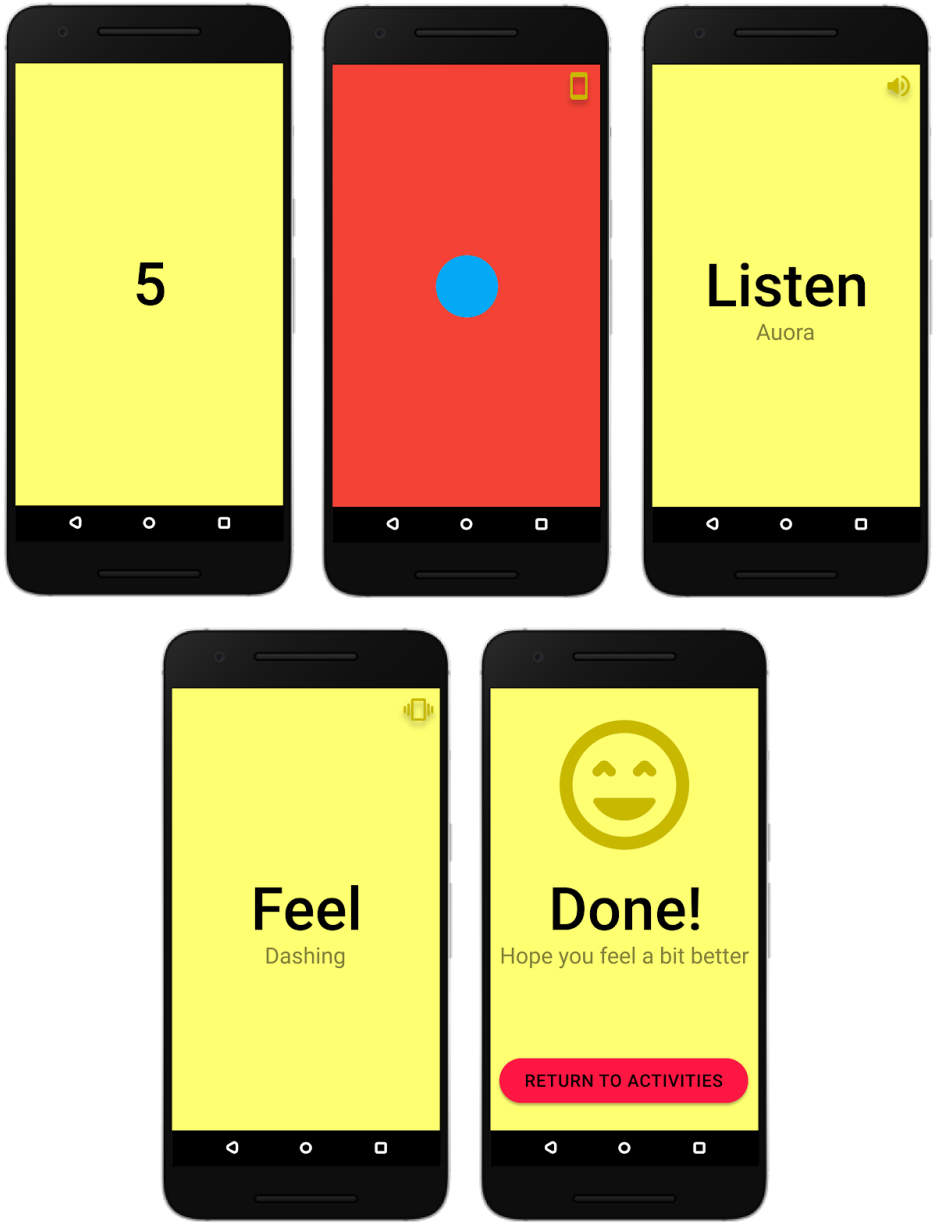
The third and final screen seen in the activity editor is the configuration screen. The screen helps the user specify their content the way they want. Each different activity/routine editor is provided with a different configuration screen. This is because the screens are designed with ease of use in mind. Different amounts of editing should lead to different screen layouts. The sound editor has no screen because the selection of a sound is handled through the device system. The top toolbar see the same “SAVE” button but instead of canceling, the back arrow brings the user back to the first screen with no added content.

App Widget

The app widget gives the user easier access to their activities/routines. With the addition of a widget close to the user, waking up is much easier. The interaction with Wake is one-click, making the UI very intuitive. The app widget is able to display one list of activities/routines. List type is specified by the user when the app widget is initialized. The behavior of the app widget is similar to the activity viewer in the main UI. The player still opens when an item is selected and both viewers will stay updated. But, the app widget list items aren’t deletable or reorderable.



Player



The player is the component of the app that does the action of waking up the user. When an activity or routine is pressed, the playing process starts using the player. FIrst, the player gives a countdown to user so they can ready themself for the waking up process. After the countdown, the screen will perform a slide animation to the first (or only) activity. When an activity is playing, you can identify its type by the visible icon in the top right corner. When a screen activity is playing, the user will see its animation. For other types of activities, a verb-instruction and title are provided. When all activities have been played, the user sees a “done” screen to tell them that they are done. Additionally, the “RETURN TO ACTIVITIES” button provides the user with the option of easily returning back to the main UI.

Material Design Specification

Colors

Icons

Key Considerations

Describe how your app handle data persistence?

Firebase Realtime Database will keep user information in the cloud, so they can access it on multiple devices. The Room library will simplify my process of creating a local database that will store user activities/routines as a cache (for offline).

Describe any edge or corner cases in the UX.

A common corner case across many users will be lack of powerful electronics (a dim screen, quiet speakers, or weak vibration motor). My app counteracts this case by providing multiple ways to wake up.

A common edge case that some users might not want to go through the process of adding customized activities. For this, my app will provide ample pre-defined activities to provide a experience close to the paid edition.

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

I will use the Firebase Authentication Android client library to help setup authentication, with a sign-in screen. Firebase Realtime database will store the activities/routines of the users. Firebase Cloud storage will store the user-generated sound-clips for the sound activities.

Android Architecture Components will help me handle events relating or happening in relation to the lifecycle. Room will help create a SQLite database with less code and more organization. ViewModel will maintain the state of a view on a configuration change, and LiveData will make views update properly.

Butterknife will be integrated to provide more organization and less findViewById() calls

Describe how you will implement Google Play Services or other external services.

I will implement the Firebase services by applying the Google Services plugin and adding the needed gradle dependencies. I will also generate the need SHA-1 key. I will use all libraries through their Gradle dependencies.

Next Steps: Required Tasks

Task 0: App Configuration

- App will be written in the Java programming language
- Android Studio 3.1 will be the IDE to help integrate and execute complex tasks
- Use Firebase version 16.1 for all Firebase client libraries
- Gradle version 4.9 will be used for project-related tasks
- Android Design Support Library 28 will provide a better design to the app
- Android Architechure Components (Version 1.1.1) will handle actions related to the lifecycle
- Butterknife version 8.8.1 will help write more efficient UI code
- Strings will be accessible in the strings.xml resource file

Task 1: Project Setup

Android Studio Application

- Create template Android Studio project
- Include appropriate app name, gradle domain and app icon
- Set minimum device api to API 21 (Lollipop)
- Set compile SDK to be most recent version
- Configure app themes (yellow, red themes)
- Add the project to Git and Github (with an initial commit and an updated .gitignore)

Firebase Application

- Initialize the project in the Firebase console
- Input the correct package name, nickname and SHA-1 key from the Android project to a new Firebase Android application
- Download the correct google-services file into the correct spot (add to .gitignore)
- Add the Firebase core dependency to Gradle
- Modify the public-facing name

Task 2: Create Activities and Fragments

Page with Horizontally Scrollable Content (Main UI, Player)

Activities

- Scrollable content container, and other views

Fragments

- Scrollable content item, displays based on data

Editor

Activities

- First page that holds the subactivities and is the portal to cancel and save
- Activity holding additional configuration for each subactivity

Fragments

- The list of subactivities
- Additional configuration for a screen activity
- The list items for the configuration of a screen activity
- The picker for the list items

- Color picker for the list items
- Recorder for adding vibration activities
- Activity (in-app concept) selector for routines

Task 3: Implement UI for Each Activity

Player

- Apply correct theme with status bar and background
- Add viewpager to scroll through items
- Include scroll dots to track the position
- Have a large rounded button below the viewpager

Main UI

- Apply correct theme with status bar and toolbar
- Include correct title in toolbar
- Add viewpager to scroll through items
- Include bottom navigation for further input and tracking
- Put a red FAB in the bottom right

Editor

- Apply correct theme with status bar and toolbar
- Include cancel and save options as well as a title in the toolbar
- Add an EditText for the name of the activity
- Add a header TextView to describe the contents of the list
- Add a recyclerview to contain subactivity items
- Apply correct theme and same toolbar for the configuration activity
- Replace cancel with back button

Player

- Make background yellow by default
- Display icon in top right corner with elevation shadow
- Create a text layout containing appropriate TextViews
- Have round button in the bottom of the layout

Task 4: Data Structures

- Create a base class representing an activity
- Subclass the base class to get classes for screen, sound, vibration activities and routines
- Create a class representing a basic subactivity
- Do the same subclassing, add a subactivity implementation of activity

Task 5: Implement UI for each Fragment and List Item

Main UI

- Create a list item with the correct icon, text, reorder icon and divider
- Add an optional heading to the list item
- Create a routine list item, replacing the reorder icon with an expand icon
- Design a layout to display when there are no items

Editor

- Create a list item with just a smaller icon, text, reorder icon and divider
- Create list item to represent each expandable card in the screen subactivity configuration

- Create a layout representing the color picker shown inline of some items
- Design a layout for the vibration recorder, a large button, with a FAB and rewind/redo buttons to the sides of the FAB

Task 6: Making Fragments Interactive

Main UI

- Display data in list when requested
- Include accurate drop down feature for routines
- Add accurate reordering functionality
- Add option to delete list items
- Inflate the placeholder layout when there is no data

Editor

- Display items that represent subactivities in a list
- Make sure each wrapper fragment (for sub-activity configuration) displays properly in full-screen
- Add optional reordering/deletion functionality
- Making expanding work for the screen configuration list items
- Inflate items properly for the sub-picker
- Add appropriate selecting logic for all pickers in screen configuration page
- Make record button start recording a vibration
- Make time TextView display correct time-length
- Give button visual feedback when pressed and unpressed
- Record the lengths of each press in a long (data type) array
- Stop recording when the FAB is pressed again
- Make FAB change icons accordingly
- Give options to play and redo vibrations
- Make play option play vibration
- When redo button is pressed reset the recorder to the original sate
- Implement the activity list from the main UI, and return data when item is pressed

Player

- Make animation animate the screen for the correct time
- Play sound when supposed to in UI
- Vibrate device based on long (data type) array
- Return just text (and return button) for the last card

Task 7: Making Activities Interactive

Main UI

- Connect display fragment to viewpager, display four fragments
- Display correct content in each fragment
- Add swipe functionality to the viewpager
- Record the position of the viewpager in the bottom navigation
- Make pressing the bottom navigation move the viewpager position
- Animate the bottom navigation items appropriately

Editor

- Inflate the list fragment inside of the activity
- Open the configuration activity with the correct fragment

- Validate data coming in when user wants to save activity

Player

- Inflate fragments into the viewpager
- Switch between them when requested
- Make button appear on last item

Task 8: Activity Open/Close Logic

- Set main UI as the launcher activity, and open it after auth UI
- Open editor from the add routines buttons (FAB and “no routine” prompt)
- Open subactivity configuration from editor
- Go back to main UI when the user is done adding a new subactivity
- Open player from main UI when activity is clicked, return to main UI when user is done

Task 9: Android Architecture Components

- Add needed setup for each component when adding

ViewModel

- Create ViewModel classes covering UI element data
- Implement the ViewModel itself

Room

- Annonte classes with Room database annotations
- Create type converters for non-primitive types
- Initialize the database with the needed type converters and entities
- Write DAOs that implement CRUD

LiveData

- Add LiveData to the Room DAO’s
- Create, and connect LiveData objects to the UI

Task 10: Widget

- Create a widget provider, configuration and layout to match the style
- Create a list item, and list adapter to match the recyclerview implementation in the main UI
- Grab data from appropriate data source

Task 11: Firebase Develop Suite

Authentication

- Implement FirebaseUI with Google Sign-In (with Smart Lock)
- Open authentication sign-in when not signed-in
- Make main UI
- Add Facebook to whole authentication flow
- Add Twitter to whole authentication flow
- Add Twitter to whole authentication flow
- Add Email to whole authentication flow
- Add Phone to whole authentication flow

Realtime Database

- Set database rules to open public
- Add child node to store task data
- Add function to fetch user data
- Validate data in fetch (or else crash)
- Read a copy of this information when when a refresh happens
- Make a refresh happen on user request in the main UI and when the app opens (onCreate)
- Refresh function loads data into the local database, LiveData will automatically update the views
- Run the push information when user edits information (CRUD)

Cloud Storage

- Store audio clips from activities in cloud storage
- Create references to files in the realtime database
- Fetch references in player

Task 12: Accessibility and Optimization

Accessibility

- Add content descriptions to appropriate views
- Test and handle the Wake experience on and D Pad
- Make sure RTL is supported throughout the entire app