GitHub Username: kethieu

Song Database LoveLive! SIF [JPN]

Description

Convenience app for playable songs in Love Live! School Idol Festival [Japanese version]

Playable songs displayed and separated by Attribute[Smile, Pure, Cool]. Each song has information such as name, BPM, difficulty and note count, and more.

Intended User

Intended Users are, for now, players of the Japanese version of Love Live! School Idol Festival. As there is an English version on the App Store that runs on a different server and behind the Japanese version in terms of content, this database will reflect up-to-date songs playable on Japanese version. Note that I will not provide any links or information on how to obtain the Japanese version of the game as it is region locked on the Google Play Store.

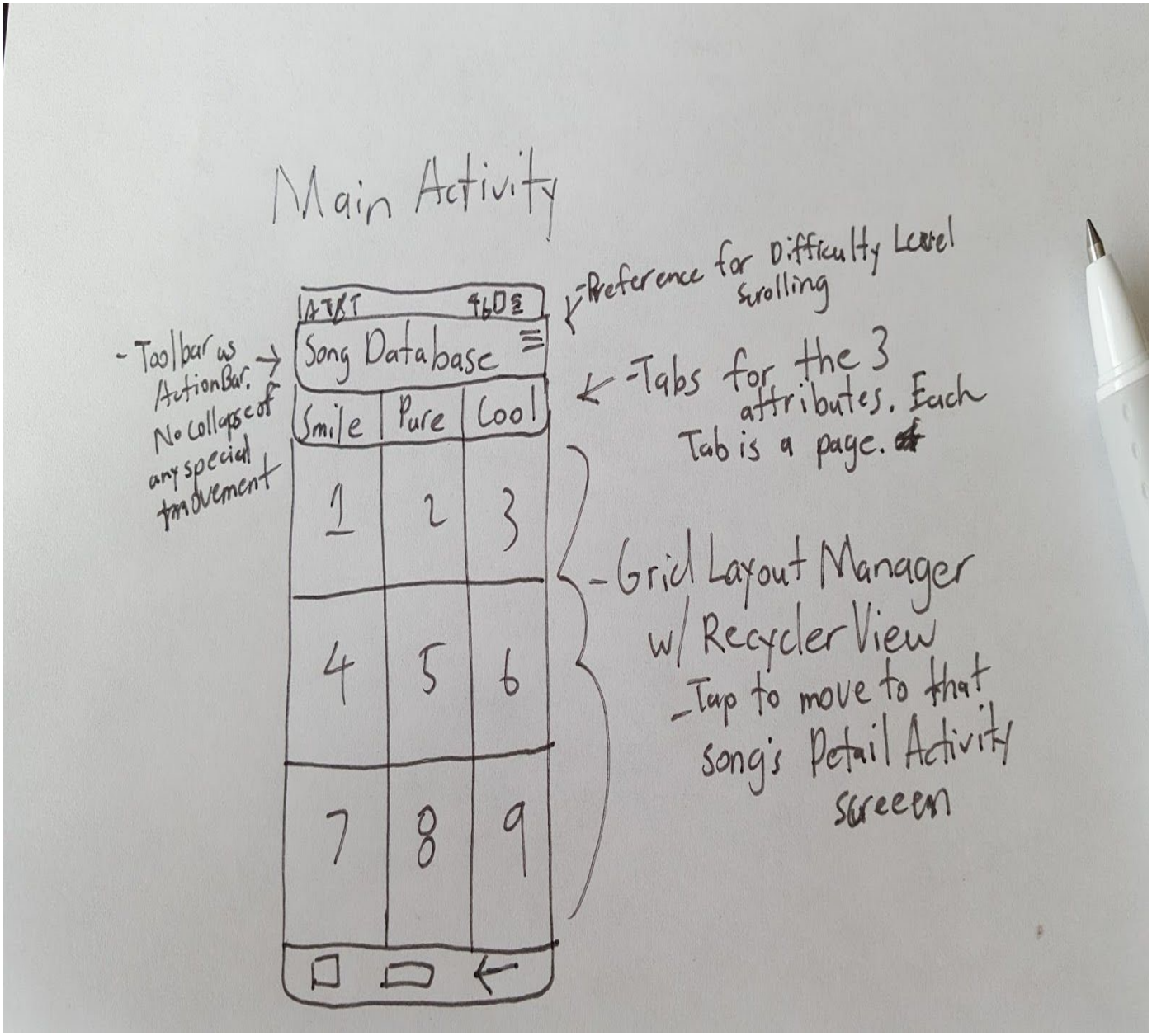
The English version is currently pushing content out to catch up to the Japanese version, but the servers remain distinct.

Features

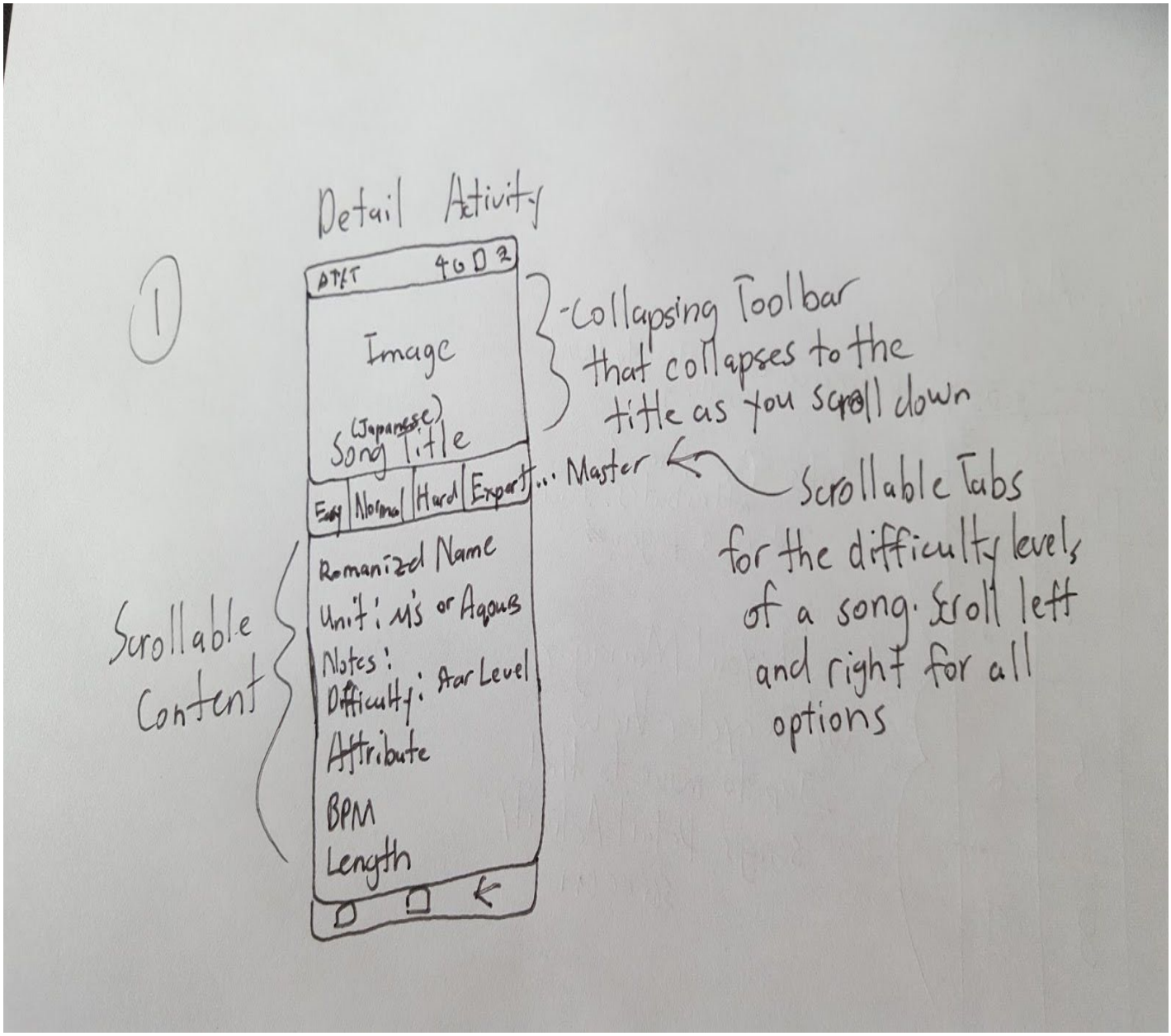
- Has necessary and relevant data for playable songs from the game.
- Details page has information such as translated name, song attribute(smile, pure, cool), each difficulty star level(easy, normal, hard, expert, master) and their respective note counts. Basically, extracted attributes from the Song Endpoint of the API.
- Details Page can swipe between difficulty levels of the same song within the page context; does not swipe into a “next” song’s details page.
- Widget that displays a random song of a random attribute that, when tapped, leads to the detail screen for that song.

User Interface Mocks

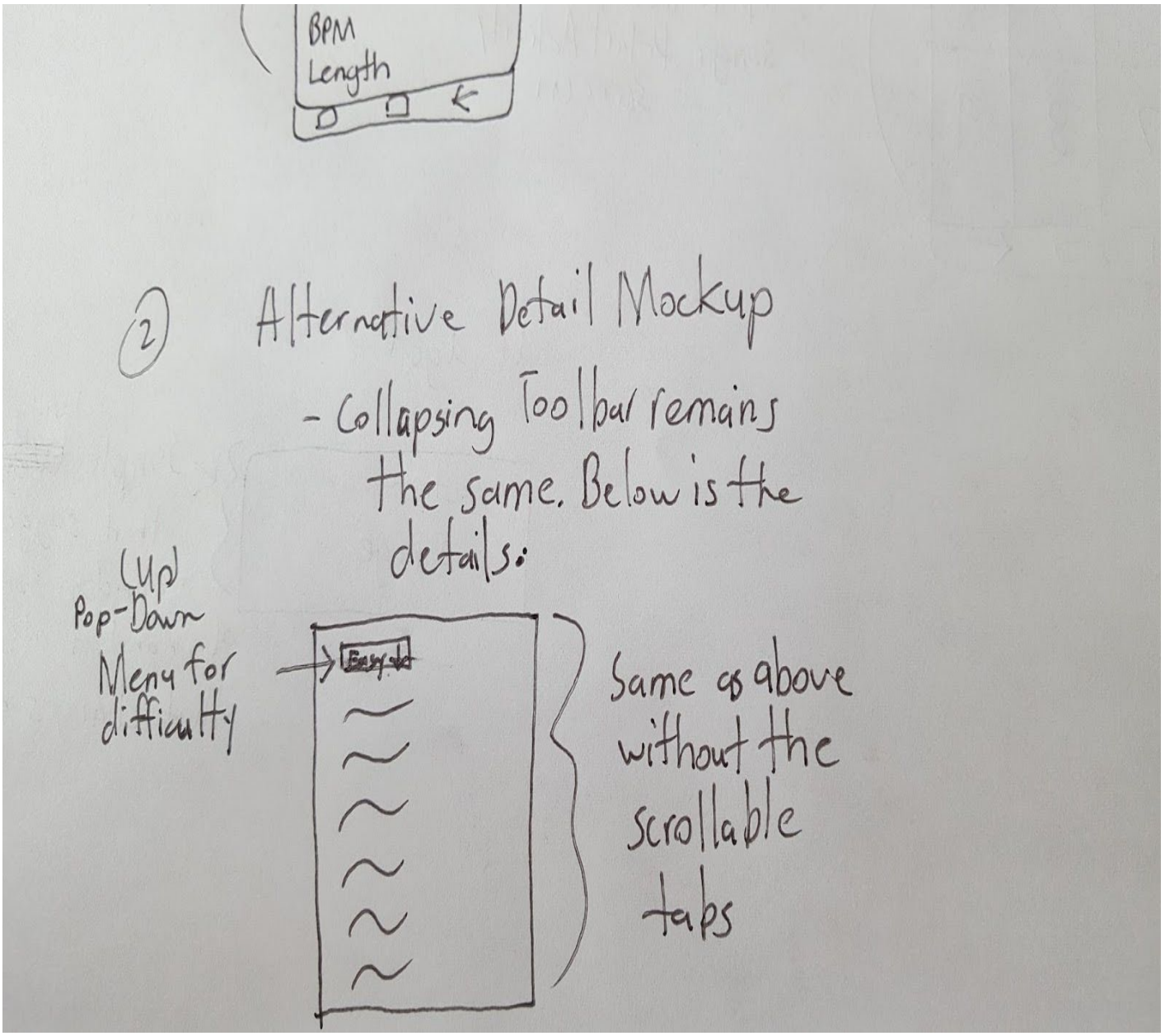
Main Activity



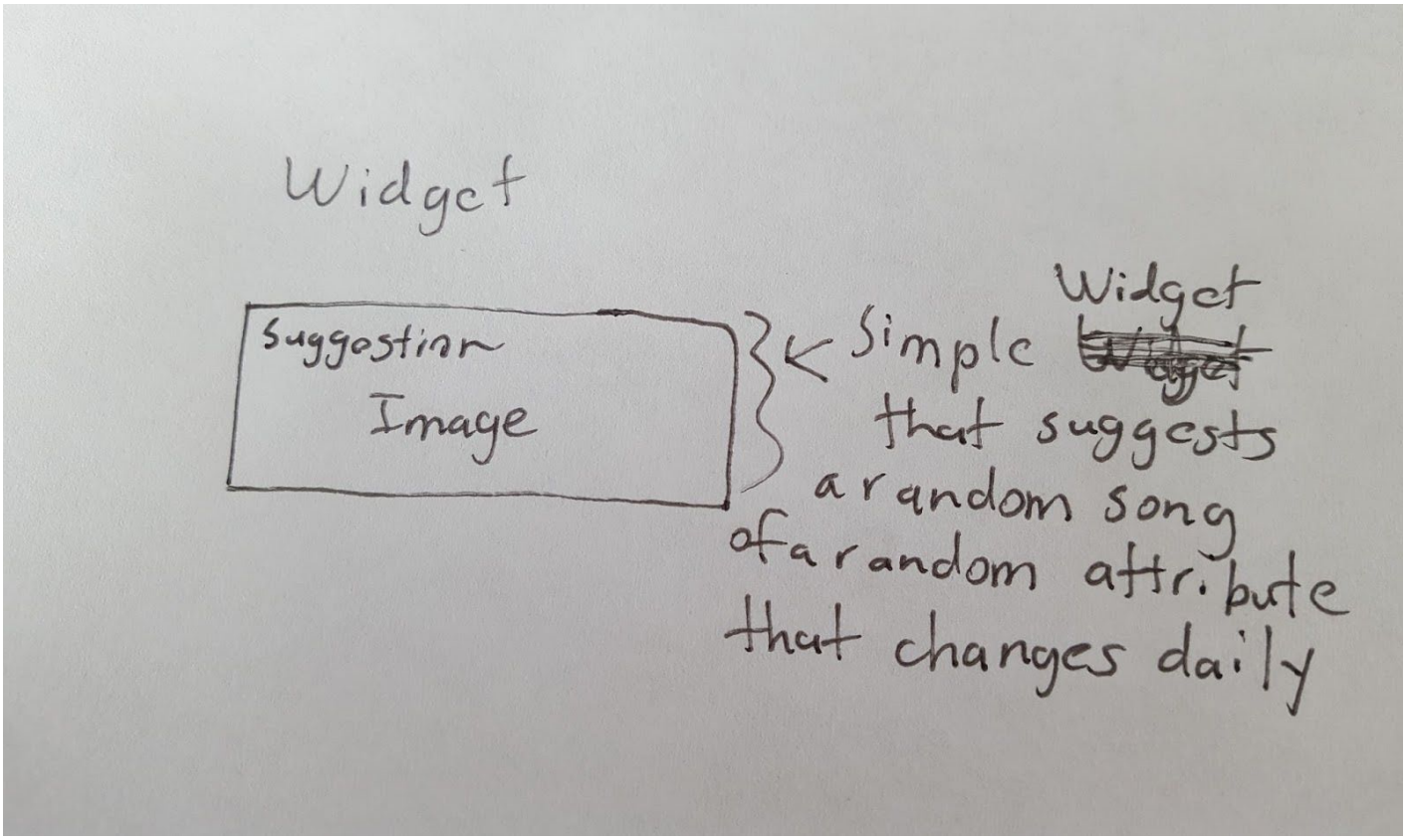
Detail Activity



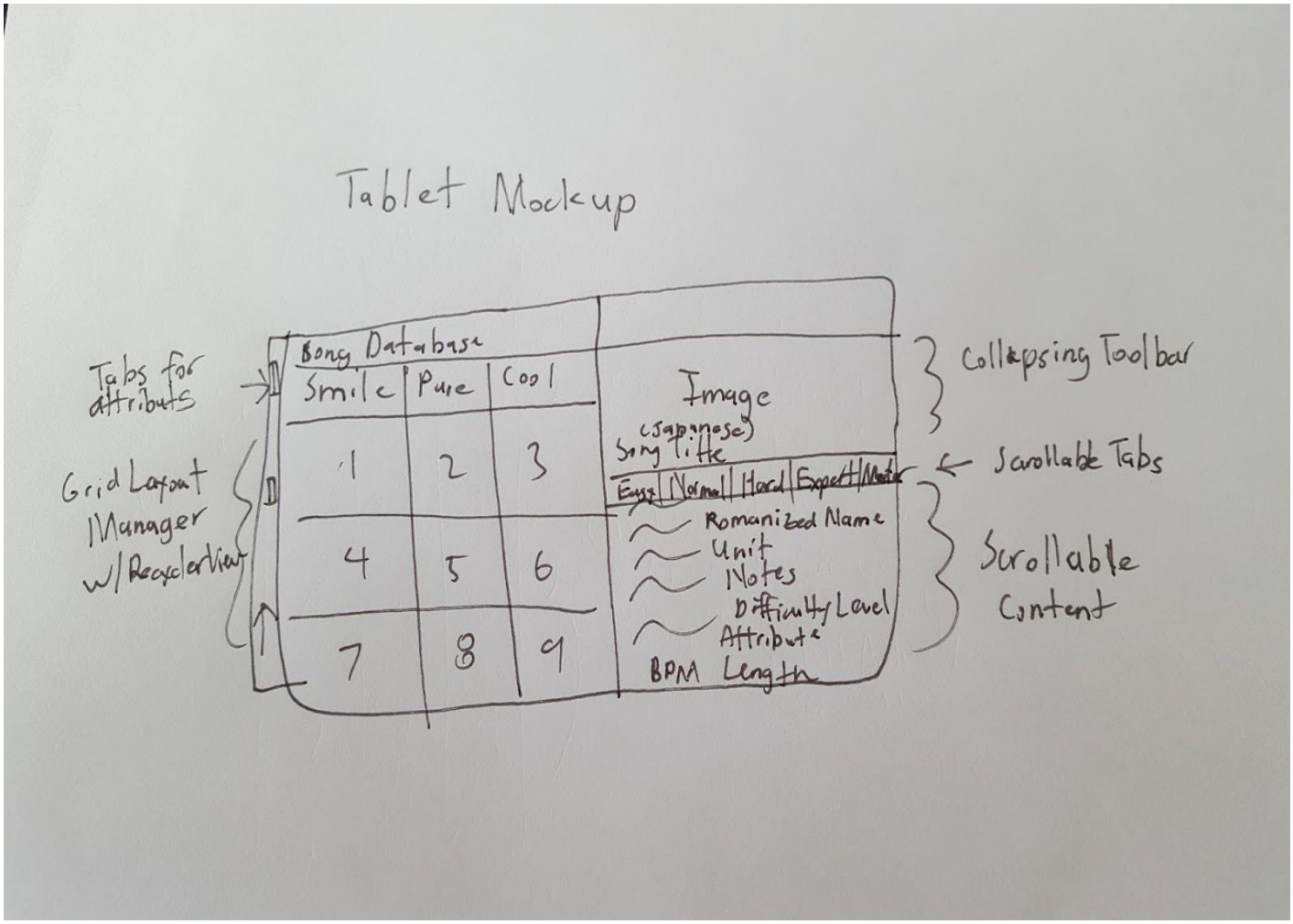
Detail Activity (Alternative)



Widget


Widget
Suggestion Image
Simple Widget that suggests a random song of a random attribute that changes daily

Tablet Mockup



Key Considerations

How will your app handle data persistence?

Describe how your app with handle data. (For example, will you build a Content Provider or connect to an existing one?)

- Songs come from a Third Party API [https://github.com/SchoolIdolTomodachi/SchoolIdolAPI/wiki/LoveLive%21-School-Idol-API] and stored within a Content Provider that I create myself instead of using a third party library.

- IntentService or AsyncTask that fetches and parses the json object array returned by the above endpoint; runs when user decides to, as songs and their respective information are updated whenever the game updates.

- A possible idea would be to set a preference for difficulty level(Easy, Normal, Hard, Expert) as a shared preference, so that on a details screen, the app would automatically switch to the tab or page of that setting. This assumes that I will settle on a ui like tab layout or viewpager.

Describe any corner cases in the UX.

For example, how does the user return to a Now Playing screen in a media player if they hit the back button?

- Back Button on the Android Device instead of a back button on the toolbar for up navigation.

- Each detail page is unique and not tied to each other. A possible idea would be for a viewpager within each detail fragment for each difficulty level, and swiping would swipe pages within that detail page, not swiping between other detail pages.

- The three tabs of the main activity that holds the grid layout can be swiped between each other like a viewpager.

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

- Using Glide library to display image thumbnails of each song in a Grid Layout through Recycler View, each image when tapped transitions into a details page for that song. Image Uris come from the song endpoint and stored within the content provider.

Describe how you will implement Google Play Services.

- Google Analytics for tracking users in terms of which screens they visit, who is actively using and from where. There is no goals or ecommerce functionality that I plan to implement as part of Analytics capability, just a simple tracker that sends screen hits.

- Google AdMob for test ads using adviews on a details page or interstitial ads when transitioning between the grid screen and the details screen, more than likely the former as I wish to have smooth content transitions via Material Design practices.

Next Steps: Required Tasks

Task 1: Project Setup

- This is a progressive tasklist, but as I begin to implement Glide, Google Play Services, etc, add the necessary dependencies to the project's gradle build script and Android Manifest.

- Create the SQLDatabase as a Content Provider to locally stored retrieved song data. Not using an external library for this but rather create one using a contract and provider necessary implementations such as query, uri matcher, and more.

Task 2: Getting Data, testing and inserting into database

- Create an AsyncTask that fetches data from the Song Endpoint of the API I'm using, parses the objects and their respective values of difficulty, notes, bpm, and more. Create a Song Object or a temporary container to hold these values as I test out insertion and querying of the Provider.

- Insert values into the local database, and implement a Cursor Loader that queries the data. This loader will be used to populate each attribute page of the main activity.

Task 3: Design Main Activity

- Main Activity, from UI Mock above, will have three tabs, one for each attribute(Smile, Pure, Cool) songs. These are pages that can be swiped to each other as well as tapping on the tab representing it.

- Each tab has a Recycler View using a Grid Layout Manager, the image content loaded into by a cursor loader. Querying by attribute should be simple, same with loading into the grid. Tapping onto an image leads to the details page for the song that the image represents.

Task 4: Design Detail Activity

- Details Page, from the UI Mock above, will have tabs to represent the difficulty levels. Tabs may or may not be swipeable as I implement this stage of the app.

- Each details page for each song is independent of each other. Other information like English and romanized names, BPM, Attribute, Unit Name and more are included into the page.

- Having a shared preference, accessible via a Settings menu option on the toolbar to store difficulty preference and have the app “scroll” to the preferred difficulty tab of the song.

- Collapsing Toolbar that collapses as you scroll down and re-enters as you scroll to the end at the top. An ImageView for the song’s thumbnail will be in this toolbar and will scroll with the toolbar.

- There is an alternative mockup I provided above in the event that Tab Layout proves difficult or unfitting to implement, where the difficulty levels as tabs are replaced with a pop-up menu to select the correct page. I shall prioritize the tab layout implementation.

Task 5: Implement Analytics and AdMob

- Create an application that uses a single global instance of a tracker to send screen hits of activities and fragments of my choosing to the Analytics console. No auto-tracking is gonna be used.

- Admob for a testAd within the detail page. No build variants for ad and no-ad versions are planned.

Task 6: Widget Design and Implementation

- Create a non-sizeable widget that displays a random song of a random attribute. According to the UI Mock above, tapping on it will lead to the details page.

- Tapping back within that page should return to the main activity of the app. Transitions may vary, will look into that as Sunshine accessed via widget ignores the return transition from details to list screens.

Task 7: Tablet Implementation

- Implement the tablet mockup I provided for the app. Using a Master-Detail flow implementation, main activity on the left with the detail activity on the right. I'm unsure how transitions will work with this, so I marked this task as less priority over the mobile/phone versions.
