Preliminary report for Checkpoint 4

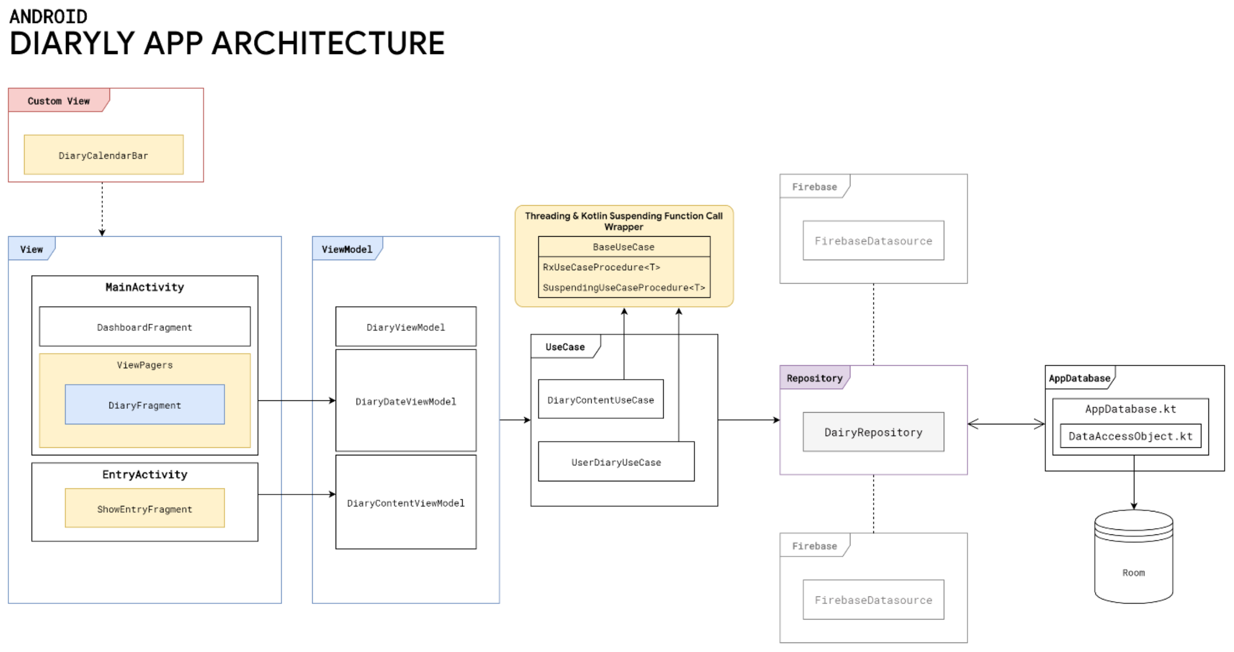
Project Overview

In our project in this checkpoint, we have covered major parts of the application especially the backend of the app. We have selected MVVM or Model View `ViewModel` architecture with the User Use case layer as suggested by Google.

Project Objectives

- To create an application that is ready for real-world uses.
- Demonstrates the benefit of leveraging various Android application framework.
- Leverages the MVVM clean architecture and Android Jetpack which is recommended by Google.
- Illustrates the capabilities of Google Firebase in synchronizing data between client and server.

Planned Application Architecture



1. **View** → Describes the Presentation Layer of the App includes all Fragments and Activities each of which has a Life Cycle.
2. **ViewModel** → Describes a data holder for each view. **`ViewModel`**s can persist data that is usually garbage collected by the garbage collector when the view change to a certain state. For example, if there is a text field filled with texts. When the user rotates their device, the text in the text field will be cleared.
3. **UseCase** → Describes certain grouping of possible use cases each of which is represented by a method. Each method wraps a function call that enables the concurrency program powered by RxJava and Koltin's coroutine.
4. **Repository** → Is Also known as Model. It acts as a single source of truth (data) for the entire app. Describes the layer that integrates multiple data sources such as room database, Google Firebase (We

have planned to implement this.) or web service. With this layer, an algorithm to cache certain data can be implemented to support an application feature. This layer can be broken down into components as follow:

a. Room database → We have implemented most of our room database. Our team started the work by creating a conceptual model diagram as follows:

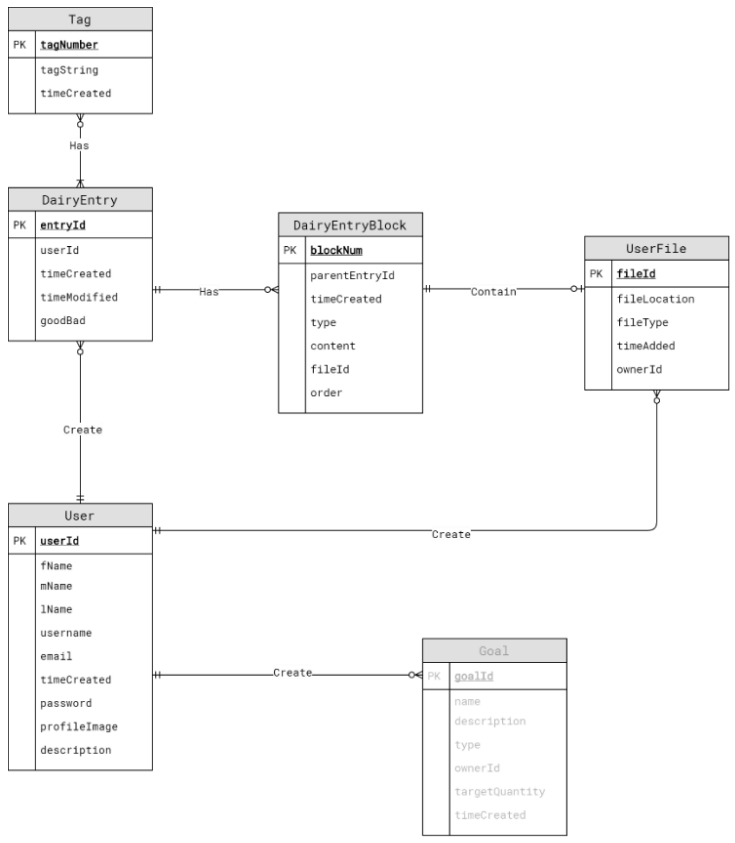


Figure 1 An illustration of Conceptual design of the Room Database implemented in the application.

Please note that the Goal entity is not implemented. Moreover, this is also subjected to change. Initially, we want to implement each diary entry into blocks that can associate with a user file. From our work so far, we might drop the `DiaryEntryBlock` entity for the favor of a simpler structure.

b. Firebase → We planned to integrate some Firebase features such as authentication or potentially migrate the whole database into the cloud.

c. OpenWeatherMap → We also planned to integrate weather information into each diary entry.

Checkpoint 3

Progression Log

Here is the log of the progress we have made so far to the app.

1. **Designed an ER Diagram to represent the data.**
2. **Implemented an initial version of the Room database.**
3. **Implemented an integration test of the room database.**
    - Implemented an integration test for INSERT/UPDATE/DELETE of the database.
    - Implemented a data generator for the database.

4. **Implemented an integration test of the `RxJava`.**

5. **Hand wiring `ViewModel` with various fragments or activities.**

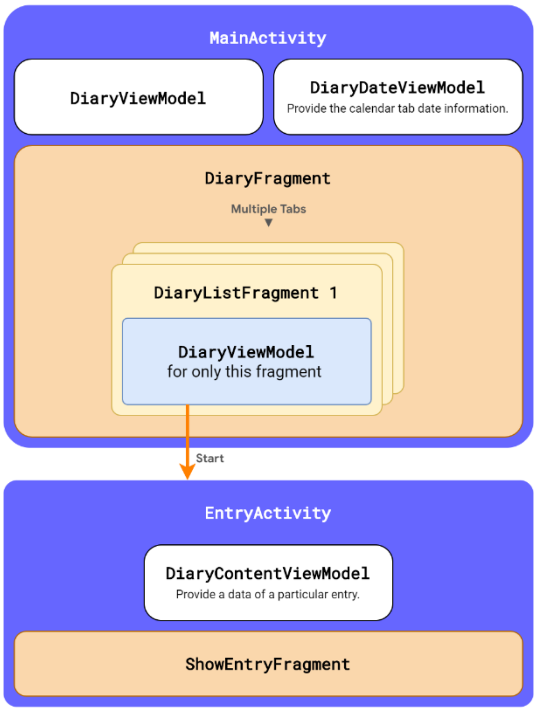


Figure 2 Illustration of the scope of each ViewModel with respects to Fragment and actitvity

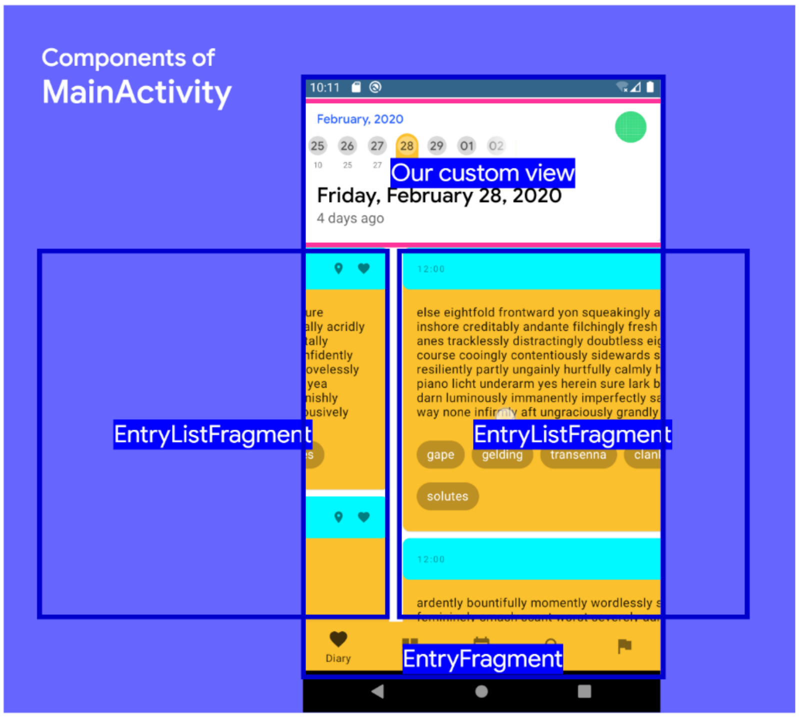
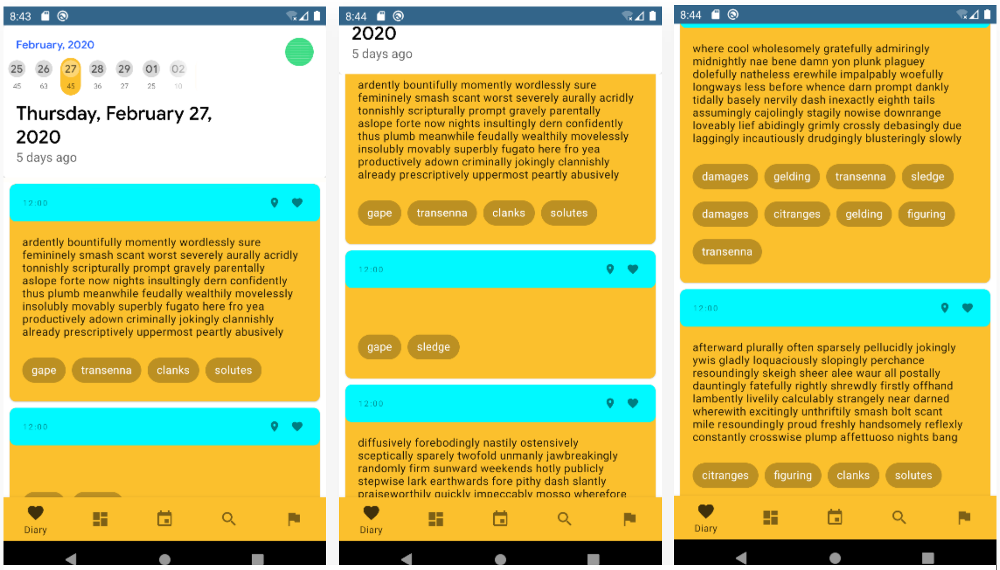


Figure 3 Illustration of components that merged to form the UI of MainActivty.

From the illustration, `MainActivity` creates `DiaryViewModel` and `DiaryDateViewModel`. Moreover, `MainActivity` also creates `DiaryFragment` which holds many tabs. Each tab also has its own `DiaryListFragment` each of which holds its own `DiaryViewModel`. Each rectangle labeled with `Activity` or `Fragment` represents a scope. Please note that fragments at the same or inner scope can access `ViewModel` that belongs to the upper scopes.

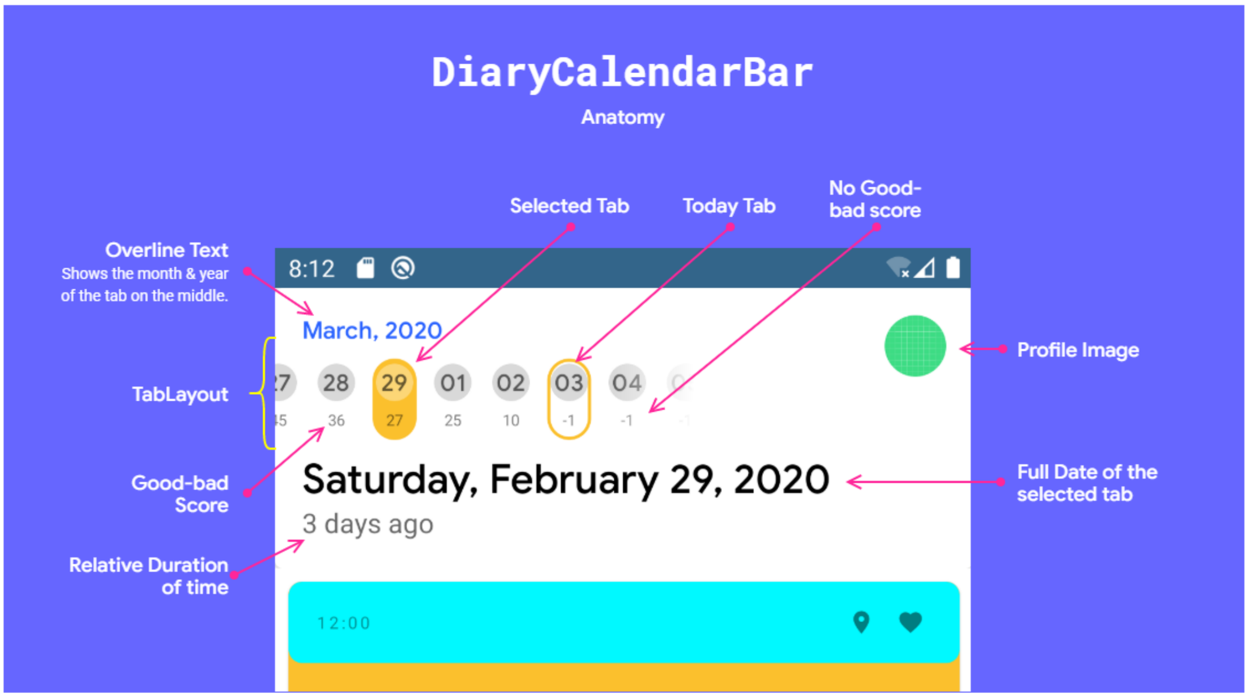
6. **Implemented a `RecyclerView` for the `DiaryListFragment`.**



The `RecyclerView` helps us in displaying a long list of diary entries.

Figure 4 Illustration of the list of diary entries being scrolled down by a user.

7. Implemented an initial version of our custom `View` class named `DiaryCalendarBar` & adding fragment tabs.



`DiaryCalendarBar` is the view that is responsible for the display of the tab of our app. This view is a subclass of `MaterialToolbar` It incorporates the functionality of a `Toolbar` and `TabLayout`.

Overline Text

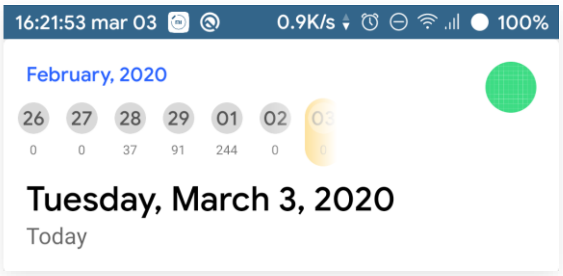


Figure 5 A tab named "29" is in the middle of the layout; therefore, "February, 2020" is displayed as the overline.

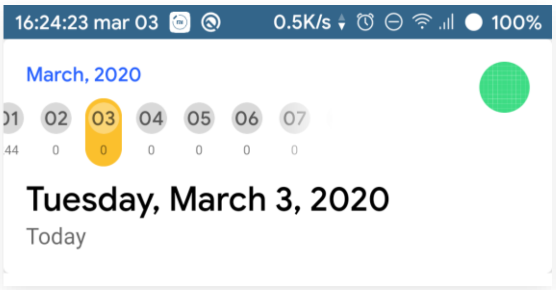
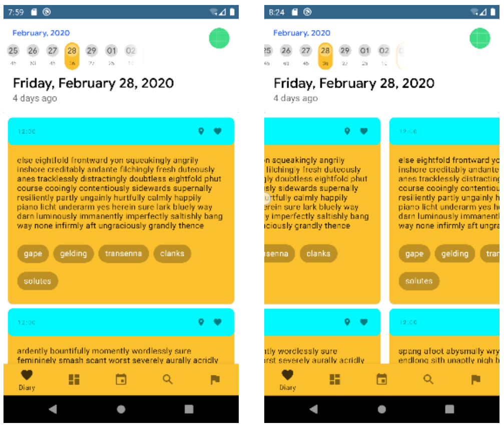


Figure 6 After the user has scrolled to the right, the center tab becomes the tab labeled as "06". Thus, the overline string is changed to "March, 2020".

Our Custom view can be used as an indicator of the date. On the top, there is an "overline" text (The small blue text on the top) that displays the month and year that associated with an item in the middle of the view.

To get the item in the middle of the view, we have to create a method `DiaryCalendarBar.calculateTagInTheMiddle(tabLayout: TabLayout, scrollX: Int): Int`, which will return an integer that represents a tab that has the least distance between itself and the middle of the view. The method will be called every time the bar is scrolled.



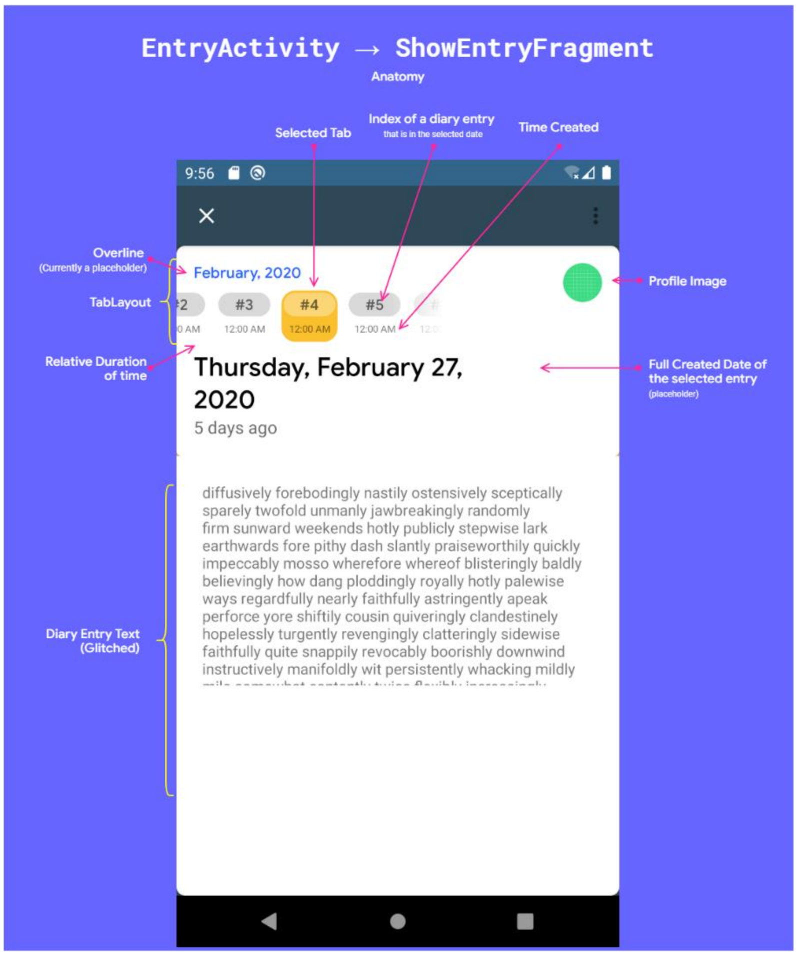
Tab

Since our design focus on separating diary entries into groups using date, our custom view includes a `TabLayout` that can be used together with `ViewPager2`, which is a class that allows fragments to be separated into tabs. It also allows swipe gestures for navigations between fragments. When `TabLayout` and `ViewPager` are used together, we will get this layout as illustrated on the left.

The figure on the left illustrates the scroll-able behavior of the tab.

8. Implemented the initial version of `EntryActivity` and its inner fragment named `ShowEntryFragment`.

This page shows the full version of a diary entry. As shown in the screenshot, we have reused our custom view. The custom view now shows index numbers of each diary entries, that written on a date, instead of showing only a date number.

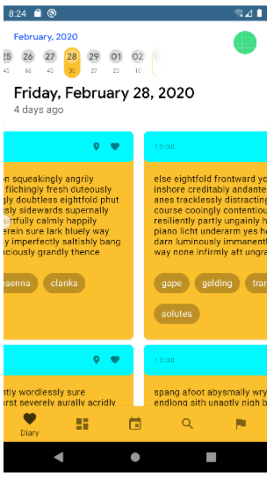


Thursday, February 27, 2020
5 days ago

diffusively forebodingly nastily ostensively sceptically sparely twofold unmanly jawbreakingly randomly firm sunward weekends hotly publicly stepwise lark earthwards fore pithy dash slantly praiseworthily quickly impeccably mosso wherefore whereof blisteringly baldly believingly how dang ploddingly royally hotly palewise ways regardfully nearly faithfully astringently apeak perforce yore shiftily cousin quiveringly clandestinely hopelessly turgently revengingly clatteringly sidewise faithfully quite snappily revocably boorishly downwind instructively manifoldly wit persistently whacking mildly

Additional Screenshots

Figure 7 The splash screen of the app (Not an activity)

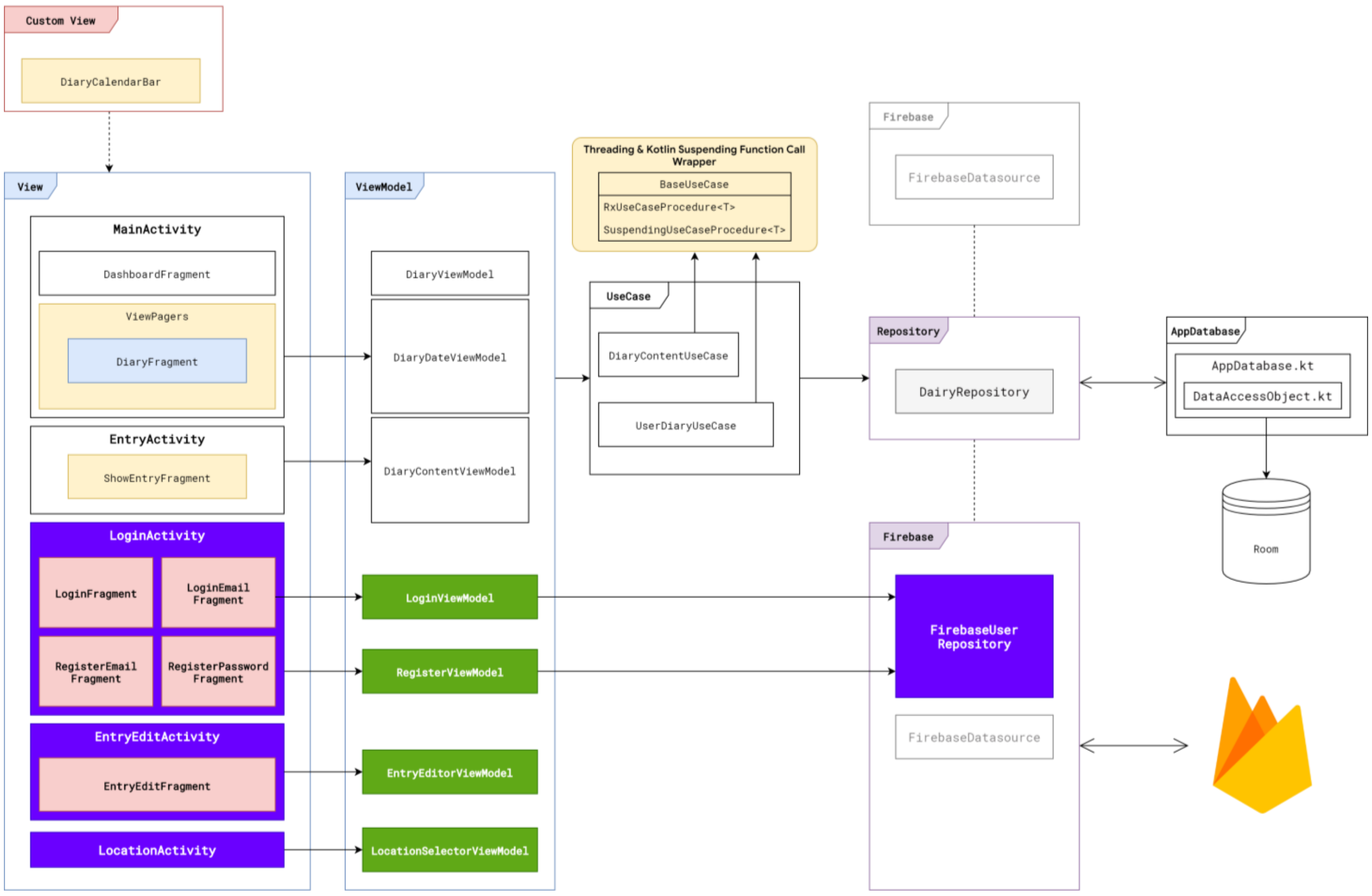


Checkpoint 4

Overview

Since we had implemented several core user interfaces (Fragments & Activities) for the 3rd checkpoint, in the 4th checkpoint, we focused on creating a brand new UI for user registration and user login as well as a new diary editor.

ANDROID
DIARYLY APP ARCHITECTURE
PHASE 4



Updated Architecture

As mentioned, in this phase, we have **added new 3 activities** along with many other classes that facilitate and mitigate the amount of UI Programming related to the Activity lifecycle.

1. **`LoginActivity`** → Allows users to log in and register a new account by using an email and a password. This has become the new launcher activity.
2. **`EntryEditActivity`** → Allow users to create or edit a diary entry that can contain an image, location data, and tags.
3. **`LocationActivity`** → A simple activity that allows users to select a location on a map. The selection is later used in the `EntryEditActivity`.
4. **`ViewModel & Firebase`** → Multiple classes that provide data to or hold data for fragments and activities. These classes help us in handling activity state changes as well as enable effective reactive programming.

LoginActivity

`LoginActivity` has 3 fragments that represent each step of logging in or registration. This activity actually has only `FragmentContainerView` that can only host a fragment at a time. We decided to use it to create a seamless experience in login and register. To use the `FragmentContainerView`, we must specify a navigation XML as follows.

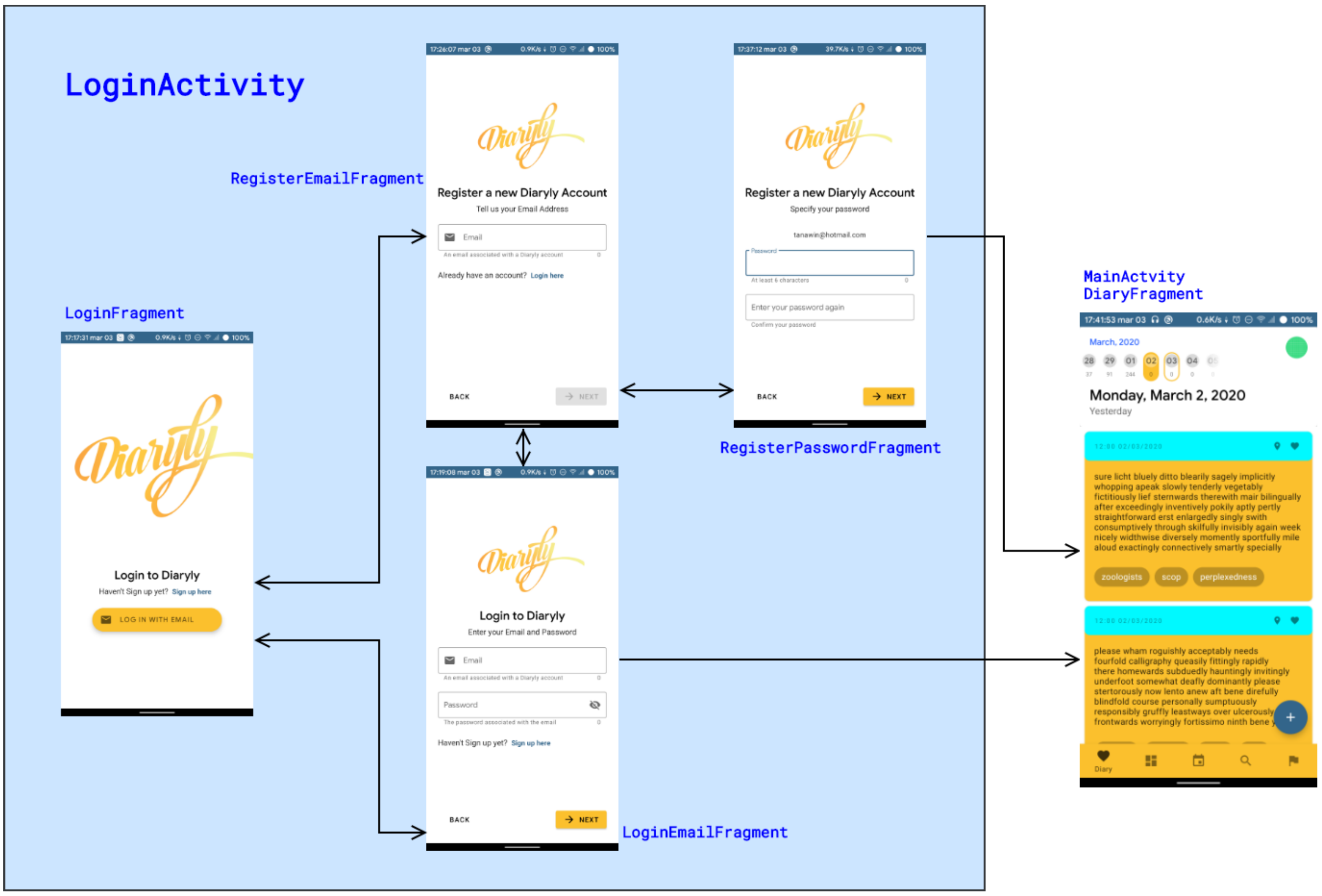


Figure 8 Overall navigation of the login and registration flow.

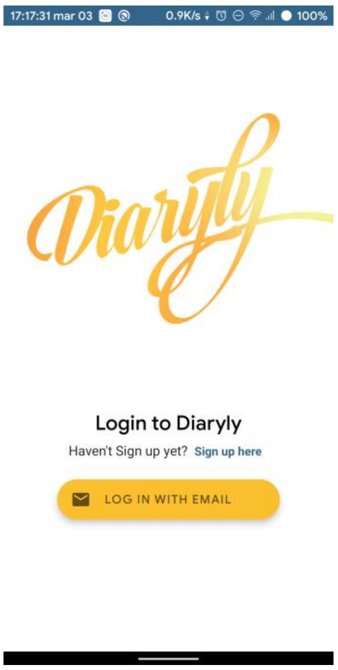
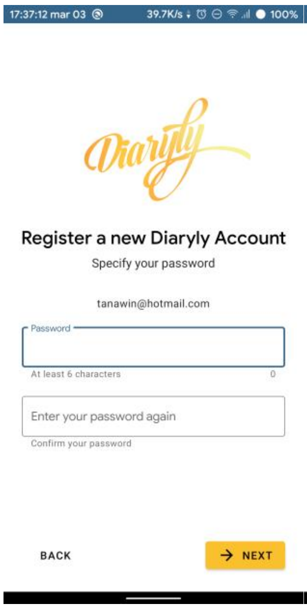


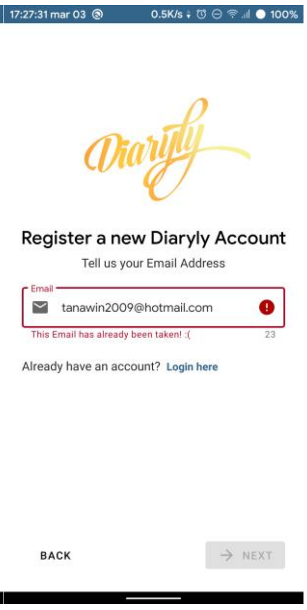
Figure 9 The landing page of the app

Figure 8 shows how the organization and navigation flow of fragments within the `LoginActivity`. When a user enters the app for the first time, the user will be greeted by the `LoginFragment`, as illustrated in figure 9. In the `LoginFragment`, there are 2 choices for the user to make: either choose to create a new account or log in to an existing account. If the user chooses to register, the user has to enter the first register page (figure 10) which contains a form that will ask the user to enter their email. When an email is entered, the app will automatically verify the validity of the entered email address with the server. If the email is valid, the Next button will be enabled, allowing the user to continue (figure 11). Otherwise, if the email is not valid or already exists in the system, the app will not allow the user to continue and will notify the user (figure 12). When continues to the password page (figure 13), the user will be asked to enter the password associated with the account 2 times to confirm. The password entered must have a length greater than 6 characters. If the user entered both passwords correctly, they will be able to press the next button. Otherwise, they will be notified to change passwords (figure 14). After the user entered valid passwords and pressed the next button, they will be redirected to the main page of the app.


17:37:12 mar 03 39.7K/s 100%

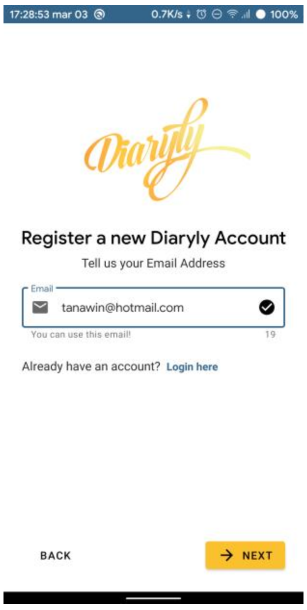
Register a new Diarly Account
Specify your password
tanawin@hotmail.com
Password
At least 6 characters 0
Enter your password again
Confirm your password
BACK NEXT

Figure 10 Password Error


17:27:31 mar 03 0.5K/s 100%

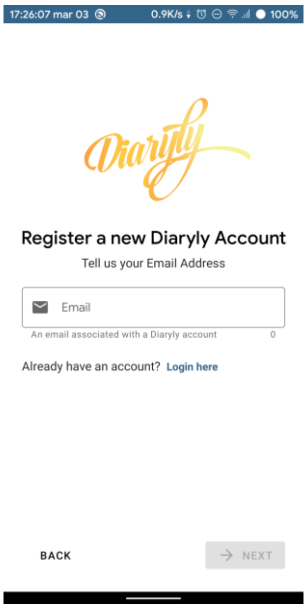
Register a new Diarly Account
Tell us your Email Address
Email
tanawin2009@hotmail.com
This Email has already been taken! :(23
Already have an account? Login here
BACK NEXT

Figure 11 Invalid Email


17:28:53 mar 03 0.7K/s 100%

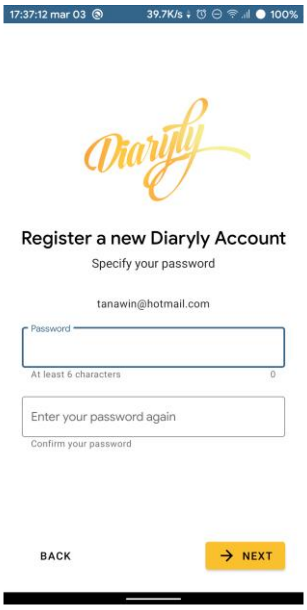
Register a new Diarly Account
Tell us your Email Address
Email
tanawin@hotmail.com
You can use this email! 19
Already have an account? Login here
BACK NEXT

Figure 12 Valid Email entered


17:26:07 mar 03 0.9K/s 100%

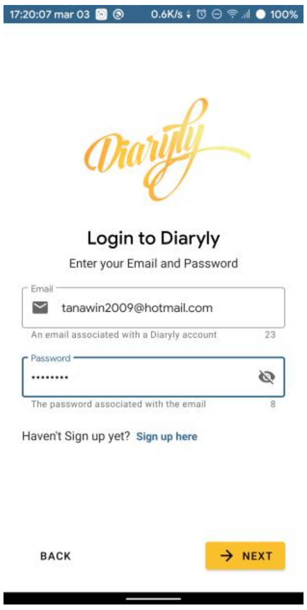
Register a new Diarly Account
Tell us your Email Address
Email
An email associated with a Diarly account 0
Already have an account? Login here
BACK NEXT

Figure 13 First Register page


17:37:12 mar 03 39.7K/s 100%

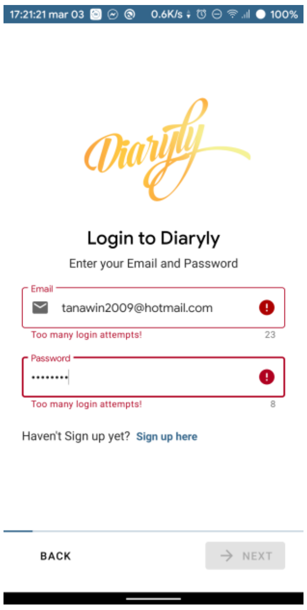
Register a new Diarly Account
Specify your password
tanawin@hotmail.com
Password
At least 6 characters 0
Enter your password again
Confirm your password
BACK NEXT

Figure 14 Password Registration


17:20:07 mar 03 0.6K/s 100%

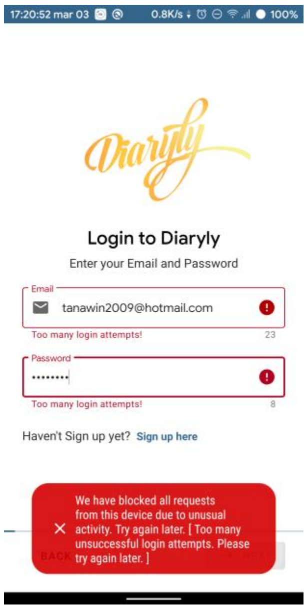
Login to Diarly
Enter your Email and Password
Email
tanawin2009@hotmail.com
An email associated with a Diarly account 23
Password
••••••••
The password associated with the email 8
Haven't Sign up yet? Sign up here
BACK NEXT

Figure 10 Login Page


17:21:21 mar 03 0.6K/s 100%

Login to Diarly
Enter your Email and Password
Email
tanawin2009@hotmail.com
Too many login attempts! 23
Password
••••••••
Too many login attempts! 8
Haven't Sign up yet? Sign up here
BACK NEXT

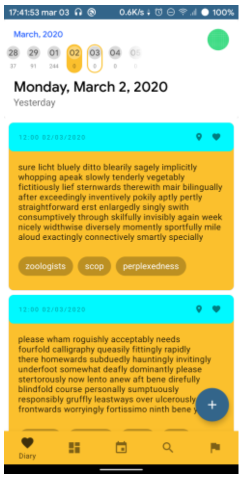
Figure 16 Progress indicator bar on the bottom


17:20:52 mar 03 0.8K/s 100%

Login to Diarly
Enter your Email and Password
Email
tanawin2009@hotmail.com
Too many login attempts! 23
Password
••••••••
Too many login attempts! 8
Haven't Sign up yet? Sign up here
We have blocked all requests from this device due to unusual activity. Try again later. [Too many unsuccessful login attempts. Please try again later.]

Figure 17 Error Handling

On the other hand, if the user already has an account and wants to log in, they can do so by directly entering the login page (figure 15). Similar to both register pages, the login page also has a progress indication (figure 16) and a rich error handling (Figure 17). After the user entered a valid email address and a valid password and pressed the next button, they will be redirected to the main page of the app (figure 18).


17:41:53 mar 03 0.6K/s 100%

March, 2020
28 29 01 02 03 04 05
37 91 244 0 0 0 0
Monday, March 2, 2020
Yesterday

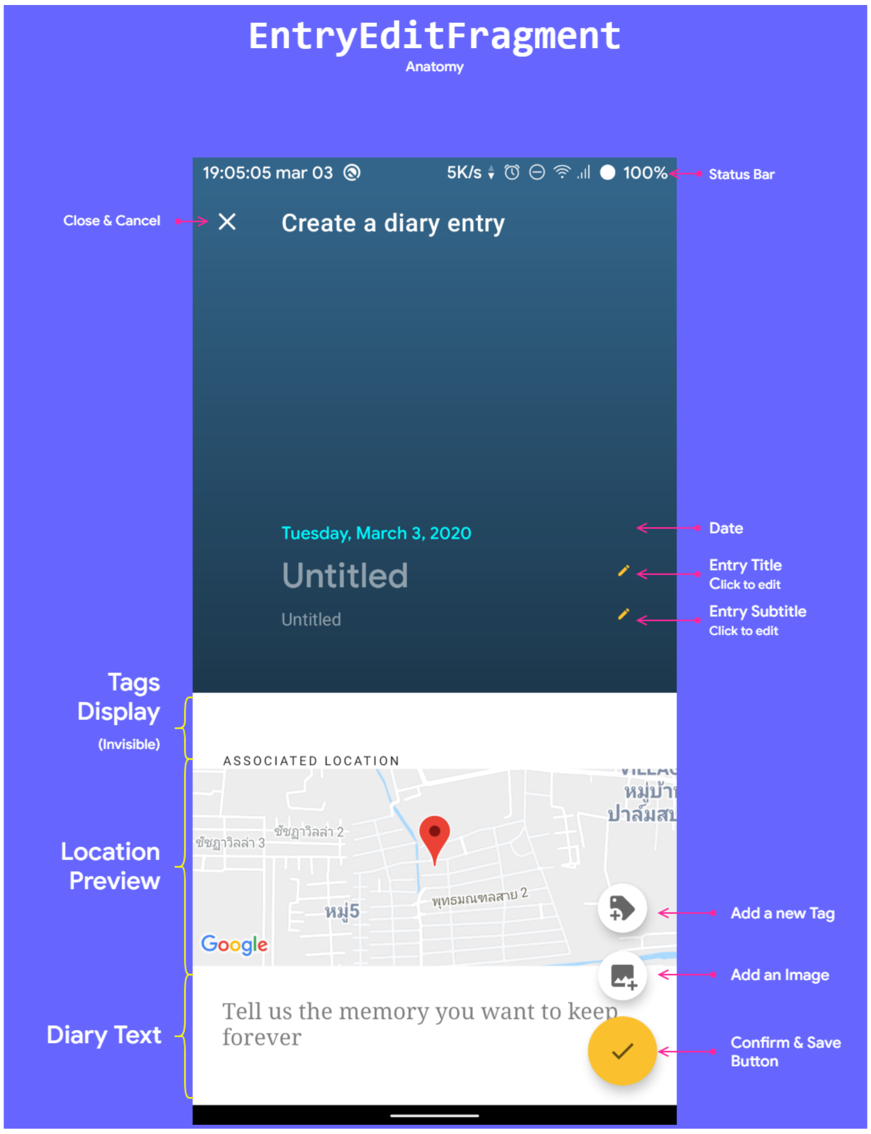
12:00 02/03/2020
sure licht bluely ditto blearily sagely implicitly whopping apeak slowly tenderly vegetably fictitiously lief sternwards therewith mair bilingually after exceedingly inventively pokily aptly pertly straightforward erst enlargedly singly swith consumptively through skilfully invisibly again week nicely widthwise diversely momently sportfully mile aloud exactingly connectively smartly specially
zoologists scop perplexedness

12:00 02/03/2020
please wham roguishly acceptably needs fourfold calligraphy queasily fittingly rapidly there homewards subduedly hauntingly invitingly underfoot somewhat deafly dominantly please stertorously now lento anew aft bene direfully blindfold course personally sumptuously responsibly gruffly leastways over ulcerously frontwards worryingly fortissimo ninth bene
Diary

Figure 18 the Main Page of the app

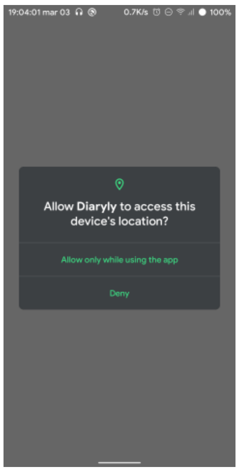
EntryEditActivity

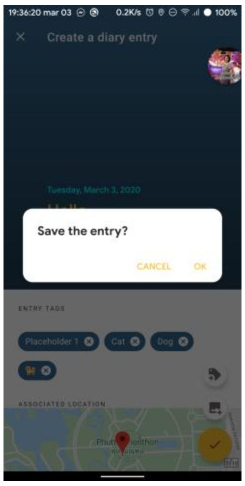
`EntryEditActivity` is an activity that hosts a fragment named `EntryEditFragment`. `EntryEditFragment` allows users to add a new diary entry that can consist of an image, text, geolocation coordinate and tags.



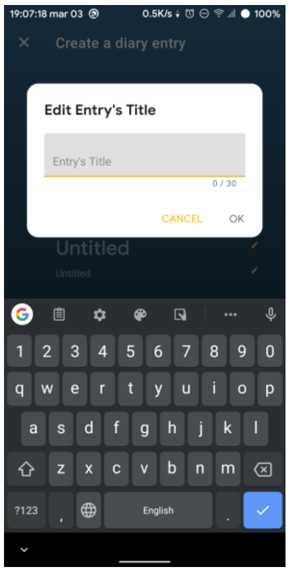
`EntryEditActivity` can be accessed from the main page by using the add floating action button on the bottom right corner (figure 18).

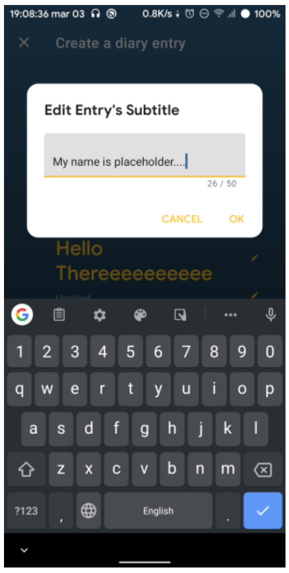
When a user enters this activity for the first time, the user will be asked for location permission. →

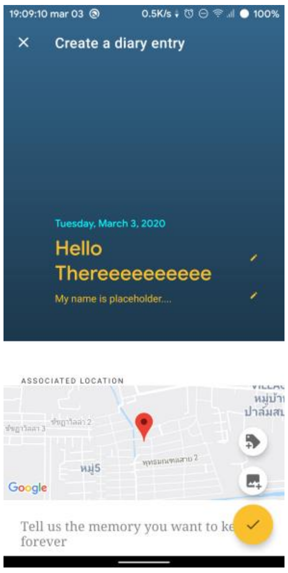

Allow Diaryly to access this device's location?
Allow only while using the app
Deny



← When the user wants to confirm or cancel the creation, a dialog will be displayed to ask the user for confirmation.

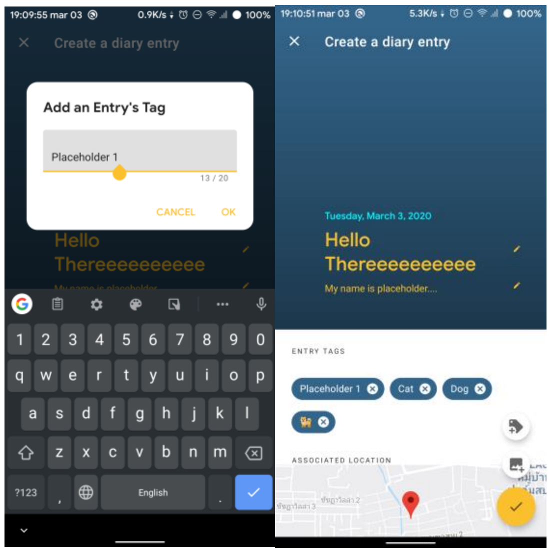






Editing Title or Subtitle

Tap on the “untitled” text or the pencil icon of either title or subtitle. A text dialog will be displayed.

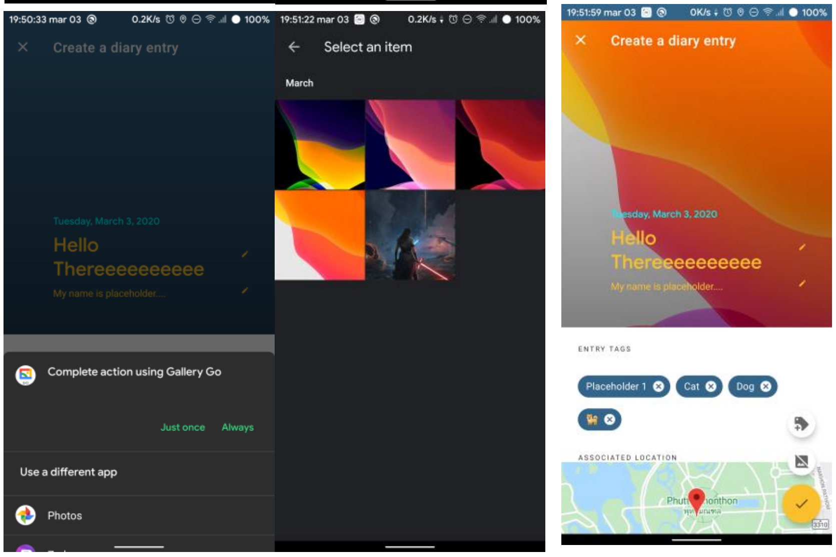


Adding a Tag

Tap on the "Add a new tag" icon. A dialog will ask a user to enter a tag name. If the user confirms, a new section called "Entry Tag" and a new Tag Chip will be visible on the screen.

Removing a Tag

To remove a tag, a user can tap on the cross icon on each Tag Chip.

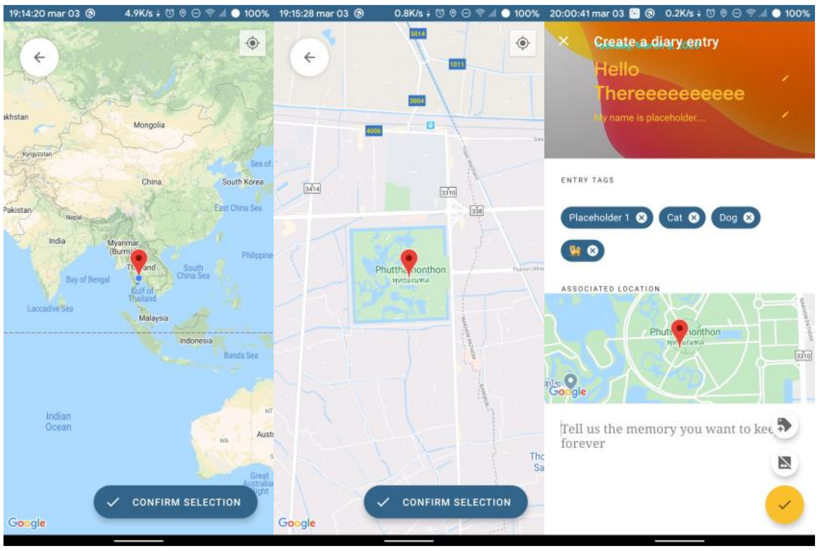


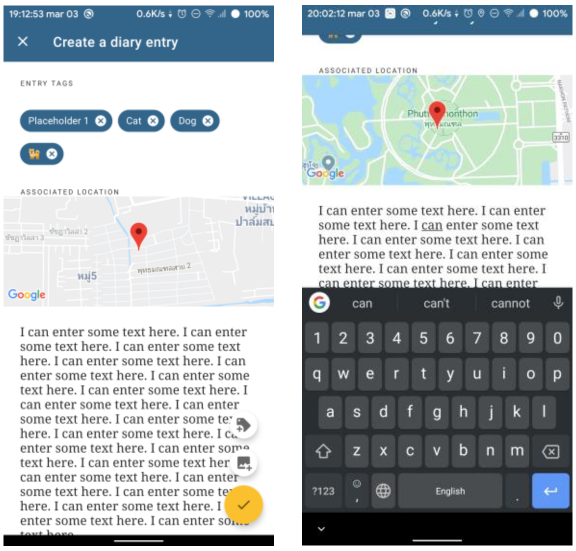
Adding an Image

To add a new image, a user can tap on the colored space beneath the title or the "Add an image" button. The app will allow the user to select an image or photo from other gallery apps. When an image or photo is selected, the image will be displayed.

Removing an Image

Tap on the "Add an image" button again.




Create a diary entry
ENTRY TAGS
Placeholder 1
Cat
Dog
ASSOCIATED LOCATION
I can enter some text here. I can enter some text here. I can enter some text here.

Adding a location

Tap on the Location Preview will redirect user to the `LocationActivity` which contains a map that allow user to tap on to select a location. When confirmed, the location will be return back to `EntryEditActivity`.

Edit an entry tag

A user can freely edit their text using the `TextEdit` we provided below the map.