Student ID: 292 53 99

Module: CSCU9YH
Year: 2023

Diary app report

Generative AI Statement

I acknowledge that I used ChatGPT (https://chat.openai.com/chat) on 22/11/2023 to generate materials that are included within my submission. Content generated by Generative AI has been used to help with this assignment by helping make the UML class diagram by generating PlantUML text. This was passed to an online PlantUML parser named Planttext which draws UML diagrams using PlantUML text. It also helped get started with parts of the writing by providing examples of how to say things I wanted to get across. And on an earlier date it helped come up with colours to use in my theme which would look nice and contrast with each other.

Table of Contents

Table of Figures

System overview

This system is a diary app. The Diary app is structured into three main screens: Date Selection, Diary Entry, and Diary History. Users can write new diary entries and read past diary entries.

The Android studio development environment and Kotlin programming language were used for the development and implementation of this system because they offer several advantages over alternatives like Visual Studio and Java. Using them meant that I could use related learning resources and help provided by the university because the course this app was created for was taught using Android Studio and Kotlin. Additionally, I found Android Studio easy to use because it is based off IntelliJ IDE which I have experience with. I chose to use Kotlin instead of Java because it is the official language for Android app development which means it has support from Google, and because it has better support for null safety features which come in handy when processing data.

The app can be executed on a device running the Android operating system. The app can be executed using the Android smartphone emulator built into Android Studio, or by installing it on an Android smartphone connected to Android studio.

The system architecture of the app is simple, with three fragments within an activity. An activity is like a window that displays the user interface of your app. It's the screen that you see when you open an app. An activity can contain one or more fragments. Fragments are like smaller windows that can be combined to create a larger window. Each fragment has its own user interface and can be updated independently. The first fragment is used for selecting the date, the second fragment is used for entering the diary entry text, and the third fragment is used for displaying stored diary entries. The user interface uses tabs to allow users to navigate between fragments. Tabs are like the tabs in a notebook or binder. Each tab represents a different section of the app. Using three fragments within an activity allows me to separate the different parts of the app into smaller, more manageable pieces. This makes it easier to develop and maintain the app over time. Crucially, each fragment has its own user interface which can be updated independently. This makes for a more flexible user interface because the fragments which make up the activity can be updated while the activity is running.

The fragments interact with each other through the activity, which acts as a container for the fragments. To exchange data between fragments, I used a view model. A view model is a class that stores data and provides it to the fragments. When a fragment needs to access the data, it can get it from the view model. Using a view model to exchange data between fragments has several benefits over other methods. A view model is aware of the lifecycle of the activity or fragment that it is associated with. This means that it can survive configuration changes, such as screen rotations

or tab changes, without losing its data. Also, by using a view model, the data from the UI logic are separate. This makes it easier to maintain code and test for bugs because different parts of the program are in different classes so there are less lines of codes to go through when something is wrong with the UI of a fragment or the data it displays.

Structure of the app

This Android application is structured around Fragment components managed within an Activity. This modular approach comprises three screens, each responsible for a distinct aspect of the diary management process: selecting the date, entering the diary entry, and viewing stored diary entries. The interactivity between these components is facilitated by Android's Fragment and ViewModel classes. The Fragments serve as individual pages within the app, each with a specific function, and they communicate through a shared ViewModel, which acts as a messenger holding the selected date and other shared data. The persistent storage is handled by a dedicated class using shared preferences, ensuring that user data is saved and retrievable across app sessions.

Class diagram

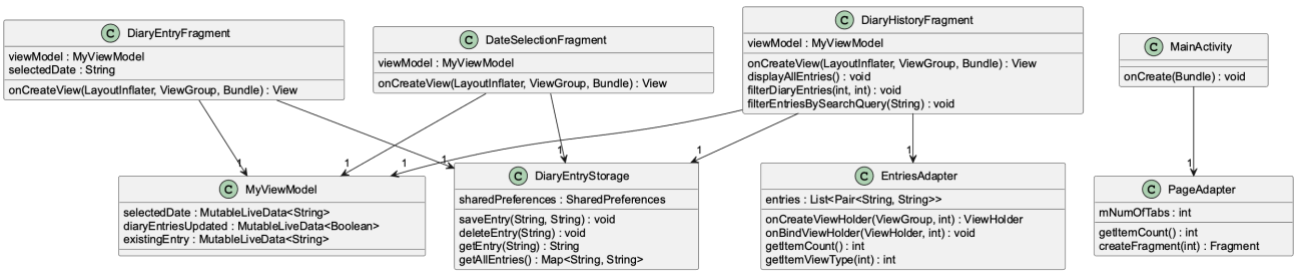


Figure 1: UML class diagram

Components

MainActivity (Activity)

This is the container for the entire app. The MainActivity is the central Activity that hosts the fragments using a ViewPager2 and mediates navigation between the screens with the help of a TabLayout and TabLayoutMediator which synchronizes tab selection with the ViewPager2.

PageAdapter (ViewPager2 adapter)

It's a bridge between the ViewPager2 in the MainActivity and the fragments, managing the instantiation and lifecycle of the fragments.

Fragments

DateSelectionFragment

This fragment allows users to select a date. This class is responsible for presenting a user interface for date selection. It interacts with the MyViewModel class to store the selected date and navigates to the DiaryEntryFragment upon selecting a date. It employs a DatePicker to allow users to pick a date and interfaces with MyViewModel to store the selected date across different components of the application.

DiaryEntryFragment

This fragment enables users to write, save, clear, or delete their diary entries. It observes the selected date from MyViewModel and displays or updates entries accordingly. It uses DiaryEntryStorage to persist entries, ensuring data is retained across sessions.

DiaryHistoryFragment

This fragment is tasked with displaying all diary entries stored in the app. It retrieves entries from DiaryEntryStorage component and orders them chronologically for user viewing. It utilizes DiaryEntryStorage to fetch entries and displays them in a RecyclerView. It also provides functionality to filter entries based on date or search criteria.

EntriesAdapter (RecyclerView Adapter)

Used in DiaryHistoryFragment to display diary entries in a list format. It binds data from DiaryEntryStorage to each RecyclerView item.

DiaryEntryStorage (Data storage class)

A module which stores data across sessions using shared preferences. It provides methods to save, delete, and retrieve diary entries.

MyViewModel (ViewModel)

Serves as a communication hub between the Fragments. Acting as a shared data holder, it maintains the selected date and tracks whether diary entries have been updated. This ViewModel ensures that data is kept during configuration changes and is shared across the fragments.

Interaction Between Components

Date Selection to Entry Input

Upon selecting a date in DateSelectionFragment, MyViewModel updates with the chosen date, which then triggers the DiaryEntryFragment to load the corresponding diary entry if it exists or provide a blank canvas for a new entry.

Entry Input to Data Storage

When a diary entry is saved or deleted in DiaryEntryFragment, the DiaryEntryStorage class is invoked to store or delete the entry. The ViewModel is also updated to reflect that entries have been modified, which prompts the DiaryHistoryFragment to refresh its display.

Entry Viewing

The DiaryHistoryFragment fetches and displays all entries. It provides functionality to filter and search entries, interfacing with DiaryEntryStorage for data manipulation.

MainActivity and Fragments

MainActivity initializes the Fragments using PageAdapter. Fragments are swapped in and out based on user interaction with TabLayout.

Fragments and MyViewModel

DateSelectionFragment updates MyViewModel with the selected date.

DiaryEntryFragment reads and writes diary entries to MyViewModel.

DiaryHistoryFragment listens to changes in MyViewModel to refresh the entries list.

Fragments and DiaryEntryStorage

DiaryEntryFragment interacts with DiaryEntryStorage to perform create, update, and delete operations on diary entries.

DiaryHistoryFragment interacts with DiaryEntryStorage to perform read operations on diary entries to display them to the user.

Key features

Search entries in real time

Figure 2: Real time entries search demo.

The real-time search functionality is provided through a SearchView widget within the DiaryHistoryFragment. As the user types into the search box, the setOnQueryTextListener method

triggers a live update of the search results by calling the filterEntriesBySearchQuery method.

```
157         /**
158          * Filter the entries by the search query when the user submits a search.
159          */
160         val searchView: SearchView = view.findViewById(R.id.searchView)
161         searchView.setOnQueryTextListener(object : SearchView.OnQueryTextListener {
162             override fun onQueryTextSubmit(query: String): Boolean {
163                 filterEntriesBySearchQuery(query)
164                 return false
165             }
166
167             override fun onQueryTextChange(newText: String): Boolean {
168                 // update the search results in real-time while the user is typing
169                 filterEntriesBySearchQuery(newText)
170                 return false
171             }
172         })
```

Figure 3: adding an event listener to the search bar and calling the filterEntriesBySearchQuery method.

This method filters the diary entries that contain the search query string and updates the RecyclerView adapter with the results. The filterEntriesBySearchQuery performs a case-insensitive check on the diary entries, ensuring that all relevant entries are displayed as the user types, providing an immediate search experience, i.e. search results are displayed real time as the user types.

```
132     /**
133      * Get all diary entries,
134      * filter the entries by the search query,
135      * and update the RecyclerView adapter with the filtered entries.
136      */
137     fun filterEntriesBySearchQuery(query: String) {
138         val allEntries = diaryEntryStorage.getAllEntries()
139         val filteredEntries = allEntries.filterValues { it.contains(query, ignoreCase = true) }
140
141         val sortedEntries = filteredEntries.entries.sortedByDescending { it: Map.Entry<String, String>
142             LocalDate.parse(it.key, DateTimeFormatter.ofPattern( pattern: "d/M/yyyy"))
143         }
144
145         val entriesWithDateHeaders = mutableListOf<Pair<String, String>>()
146         for (entry in sortedEntries) {
147             val entryDate = LocalDate.parse(entry.key, DateTimeFormatter.ofPattern( pattern: "d/M/yyyy"))
148             val formattedDate = entryDate.format(DateTimeFormatter.ofPattern( pattern: "dd/MM/yyyy"))
149             val entryText = entry.value
150             entriesWithDateHeaders.add(Pair(formattedDate, entryText))
151         }
152
153         // creates a new instance of EntriesAdapter with the sorted entries, i.e., displays the sorted entries
154         entriesRecyclerView.adapter = EntriesAdapter(entriesWithDateHeaders)
155     }
```

Figure 4: The filter entries by search query method inside the diary history fragment.

Filter entries by month & year

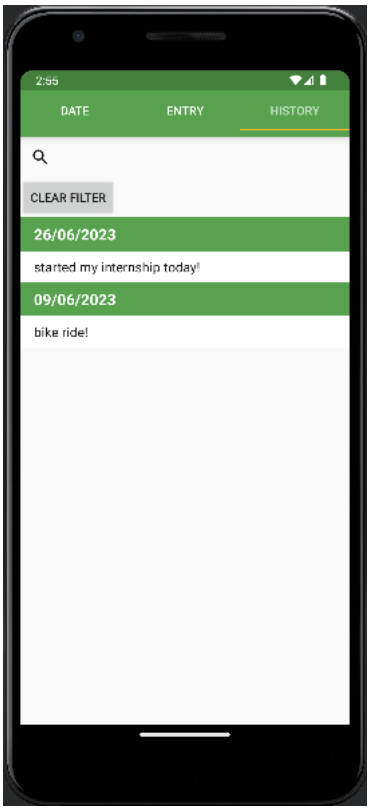


Figure 5: Filtering entries by June 2023 demo.

The filtering feature is implemented using a DatePickerDialog and a filter button in the DiaryHistoryFragment. When the user selects a date and confirms the selection, the

filterDiaryEntries method is called, filtering the entries by the chosen month and year.

```
110         // add a click event listener to the filter button
111         val filterButton: Button = view.findViewById(R.id.filterButton)
112         filterButton.setOnClickListener { it: View!
113             if (filterButton.text == "Filter by month & year") {
114                 val datePickerDialog = DatePickerDialog(
115                     requireContext(),
116                     { _, year, month, _ ->
117                         filterDiaryEntries(year, month: month + 1)
118                         filterButton.text = "Clear filter"
119                     },
120                     Calendar.getInstance().get(Calendar.YEAR),
121                     Calendar.getInstance().get(Calendar.MONTH),
122                     Calendar.getInstance().get(Calendar.DAY_OF_MONTH)
123                 )
124                 // Show the DatePickerDialog when the filter button is pressed
125                 datePickerDialog.show()
126             } else { // if button text is clear filter
127                 displayAllEntries()
128                 filterButton.text = "Filter by month & year"
129             }
130         }
```

Figure 6: The filter by month button and click event listener inside the diary history fragment.

This method sorts the entries and updates the RecyclerView to only display the entries from the selected date.

```
/**
 * The filterDiaryEntries method gets all diary entries,
 * filters the entries by the selected date,
 * and updates the RecyclerView adapter with the filtered entries.
 */
fun filterDiaryEntries(year: Int, month: Int) {
    val allEntries = diaryEntryStorage.getAllEntries()
    val filteredEntries = allEntries.filterKeys { it: String
        val entryDate = LocalDate.parse(it, DateTimeFormatter.ofPattern( pattern: "d/M/yyyy"))
        entryDate.year == year && entryDate.monthValue == month ^filterKeys
    }

    val sortedEntries = filteredEntries.entries.sortedByDescending { it: Map.Entry<String, String>
        LocalDate.parse(it.key, DateTimeFormatter.ofPattern( pattern: "d/M/yyyy"))
    }

    val entriesWithDateHeaders = mutableListOf<Pair<String, String>>()
    for (entry in sortedEntries) {
        val entryDate = LocalDate.parse(entry.key, DateTimeFormatter.ofPattern( pattern: "d/M/yyyy"))
        val formattedDate = entryDate.format(DateTimeFormatter.ofPattern( pattern: "dd/MM/yyyy"))
        val entryText = entry.value
        entriesWithDateHeaders.add(Pair(formattedDate, entryText))
    }

    // creates a new instance of EntriesAdapter with the sorted entries, i.e., displays the sorted entries
    entriesRecyclerView.adapter = EntriesAdapter(entriesWithDateHeaders)
}
```

Figure 7: Filter diary entries method inside the diary history fragment.

Enhanced history viewing

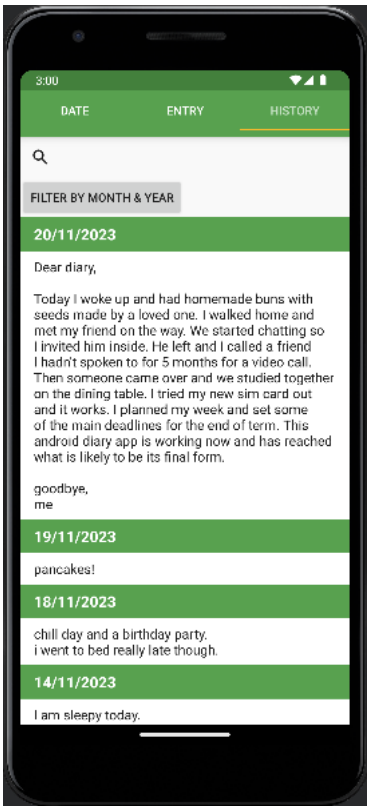


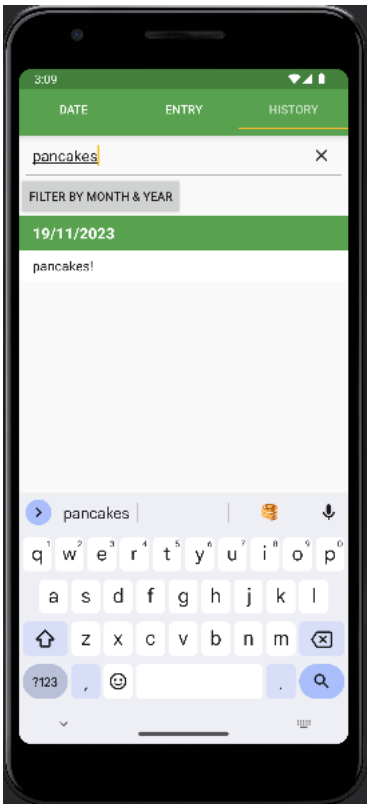
Figure 8: Diary history fragment interface demo.

The history of entries is presented using a RecyclerView, which is populated by an EntriesAdapter. This adapter takes a list of entries, each with a date and text, and binds them to the view, highlighting the date in a bold typeface to serve as a header for the entry text.

```
/**
 * Display all entries in entry storage in chronological order starting with newest
 */
fun displayAllEntries() {
    // The saved entries are available in a list called savedEntries
    val savedEntries = diaryEntryStorage.getAllEntries() // update list of saved entries with potential new/removed entries
    // Sort the entries by date recency before displaying them
    val sortedEntries = savedEntries.entries.sortedByDescending { it: Map.Entry<String, String> // Sort the entries by date recency before displaying them
        LocalDate.parse(it.key, DateTimeFormatter.ofPattern( pattern: "d/M/yyyy")) // Convert the date strings to LocalDate objects and sort them
    }

    val entriesWithDateHeaders = mutableListOf<Pair<String, String>>()
    for (entry in sortedEntries) {
        val entryDate = LocalDate.parse(entry.key, DateTimeFormatter.ofPattern( pattern: "d/M/yyyy"))
        val formattedDate = entryDate.format(DateTimeFormatter.ofPattern( pattern: "dd/MM/yyyy"))
        val entryText = entry.value
        entriesWithDateHeaders.add(Pair(formattedDate, entryText))
    }

    // creates a new instance of EntriesAdapter with the sorted entries, i.e., displays the sorted entries
    entriesRecyclerView.adapter = EntriesAdapter(entriesWithDateHeaders)
}

displayAllEntries() // display all diary entries when app starts
```

Figure 9: The display all entries method of the Diary history fragment.

One entry per date



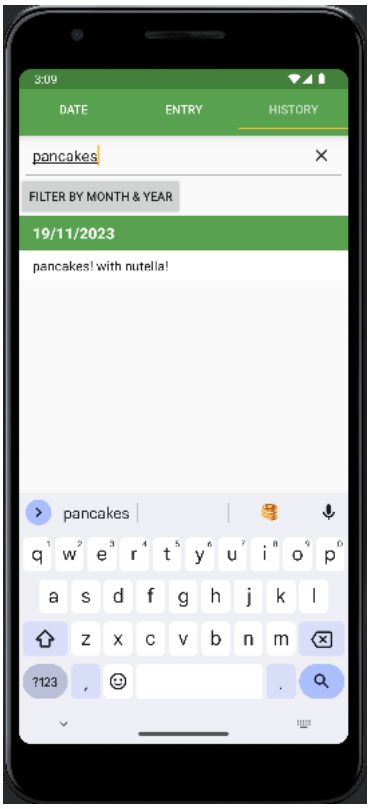


Figure 10: One entry for 19/11/2023

Figure 11: Still one entry for 19/11/2023 after overwriting previous entry.

The app enforces a one entry per date rule through the DiaryEntryStorage class. It retrieves the entry for the selected date and displays it for editing. If the user tries to save a new entry for the same date, the existing entry is overwritten, ensuring there's always just one entry per date.

```
val saveButton = view.findViewById<Button>(R.id.saveButton)
saveButton.setOnClickListener { it: View!
    val diaryEntry = diaryText.text.toString()
    val diaryEntryStorage = DiaryEntryStorage(requireContext())
    // If an empty entry is saved the entry for that date should be deleted
    if (diaryEntry.isBlank()) {
        diaryEntryStorage.deleteEntry(selectedDate)
    } else {
        // only save the entry if the user actually wrote an entry, i.e., text !blank
        diaryEntryStorage.saveEntry(selectedDate, diaryEntry)
    }
    // update the MyViewModel instance shared between fragments when a new entry is saved,
    // so that DiaryHistoryFragment can updates its list of displayed entries
    viewModel.diaryEntriesUpdated.value = true

    // automatically navigate to the diary history page when save button is clicked
    (activity as MainActivity).findViewById<ViewPager2>(R.id.pager).currentItem = 2
}
```

Figure 12: Save button on click listener in Diary entry fragment.

Edit entries

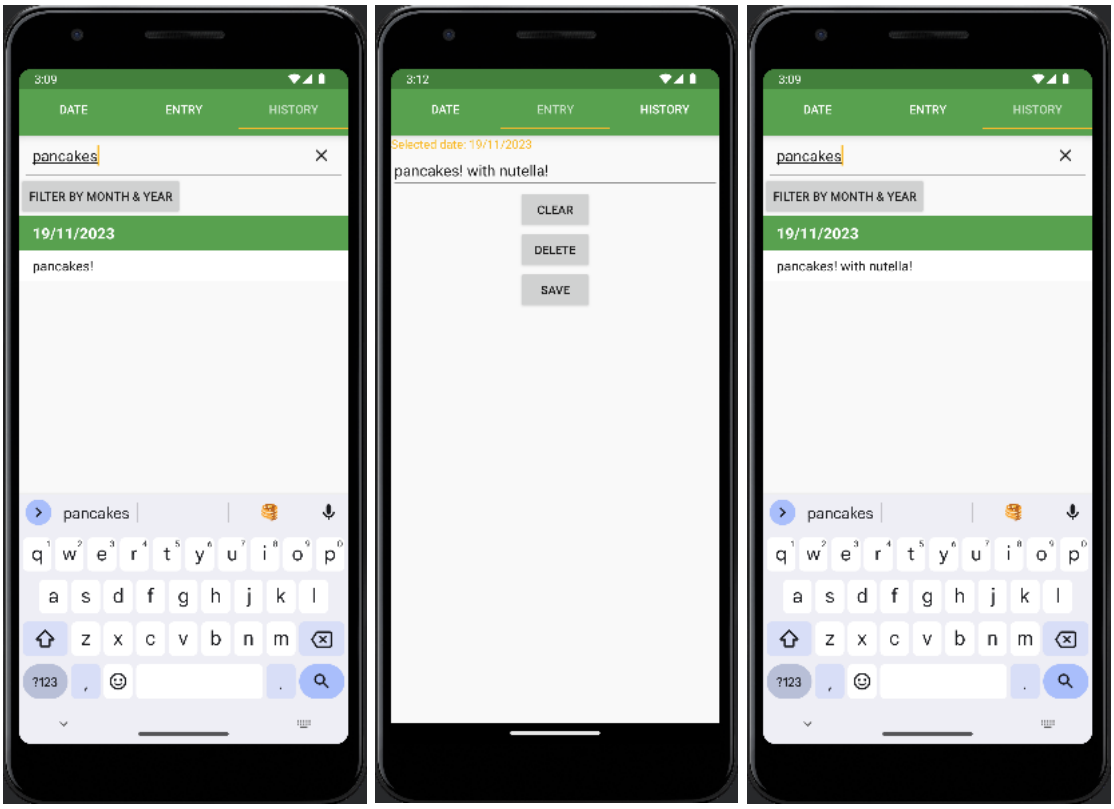


Figure 13: Entry for 19/11/2023 before edit.
Figure 14: Editing entry for 19/11/2023.
Figure 15: Entry for 19/11/2023 after edit.

The DiaryEntryFragment facilitates editing of diary entries. It uses a ViewModel to observe changes in the selected date and loads the corresponding entry text into an EditText.

```
class MyViewModel : ViewModel() {
    val selectedDate = MutableLiveData<String>()
    // The selectedDate is a MutableLiveData object, so you can't directly assign a String value to it.
    // Instead, the String value is assigned to the value property of the MutableLiveData
    boostaboosh
    val value: MutableLiveData<String>
        get() = selectedDate
    boostaboosh
    init {
        // set the default (starting value of) selected date to be the current date
        val calendar = Calendar.getInstance()
        val day = calendar.get(Calendar.DAY_OF_MONTH)
        val month = calendar.get(Calendar.MONTH) + 1 // +1 because months indexed starting at zero
        val year = calendar.get(Calendar.YEAR)
        selectedDate.value = "$day/$month/$year"
    }
```

Figure 16: selected date value stored in the ViewModel.

```
val selectedDateView = view.findViewById<TextView>(R.id.selectedDate)
// make a variable which stores the selectedDate
val selectedDateValueObserver = Observer<String> { newValue ->
    selectedDate = newValue // Update selectedDate here, which is defined at a class level, here
    selectedDateView.text = "Selected date: $selectedDate"
}
viewModel.value.observe(viewLifecycleOwner, selectedDateValueObserver)
// observes changes to the value of the selected date in the view model
// and updates the selected date observer variable value here when it changes
// selectedDateValueObserver is set before saveButton click listener because this updates selectedDate,
// and I want to ensure this happens before I try to save the selectedDate when the saveButton is clicked.

val diaryText = view.findViewById<EditText>(R.id.diaryText)
// observe existingEntry and set the text of diaryText when it changes
viewModel.existingEntry.observe(viewLifecycleOwner, Observer { entryText ->
    diaryText.setText(entryText)
})
// set the diary text to that of the current date entry if it exists when the app launches
val currentDate = SimpleDateFormat( pattern: "d/M/yyyy", Locale.getDefault()).format(Date())
val diaryEntryStorage = DiaryEntryStorage(requireContext())
val entryText = diaryEntryStorage.getEntry(currentDate)
diaryText.setText(entryText)
```

Figure 17: The diary entry fragment observes, i.e. listens, to changes to the value of the selected date and the existing entry in the ViewModel and updates its UI accordingly

When the save button is clicked, the saveEntry method of DiaryEntryStorage is called to save the changes by adding the entry to the entries stored in the shared preferences of the app.

```
fun saveEntry(date: String, entry: String) {
    sharedPreferences.edit().putString(date, entry).apply()
}
```

Figure 18: The saveEntry method of the DiaryEntryStorage fragment.

Delete entries

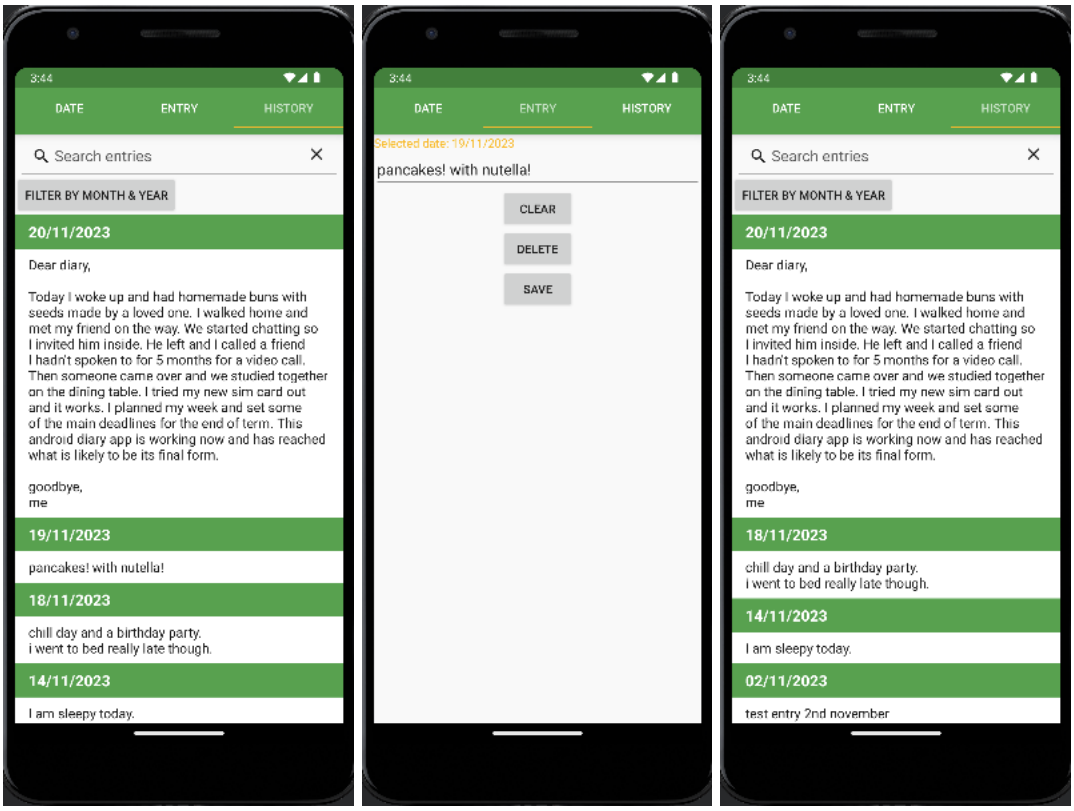


Figure 19: 19/11/2023 entry visible before deletion.

Figure 20: Deleting the 19/11/2023 entry in the diary entry fragment.

Figure 21: The 19/11/2023 entry no longer visible in the history fragment after being deleted.

Deletion of entries is managed by the deleteButton's OnClickListener within the DiaryEntryFragment.

```
val deleteButton = view.findViewById<Button>(R.id.deleteButton)
deleteButton.setOnClickListener { it: View!
    val diaryEntryStorage = DiaryEntryStorage(requireContext())
    diaryEntryStorage.deleteEntry(selectedDate)
    viewModel.diaryEntriesUpdated.value = true
}
```

Figure 22: deleteButton click listener in entry fragment.

When triggered, it calls deleteEntry from the DiaryEntryStorage, which removes the entry from the storage.

```
fun deleteEntry(date: String) {
    sharedPreferences.edit().remove(date).apply()
}
```

Figure 23: deleteEntry method in diary entry storage.

Next & save button presses jump to the next page

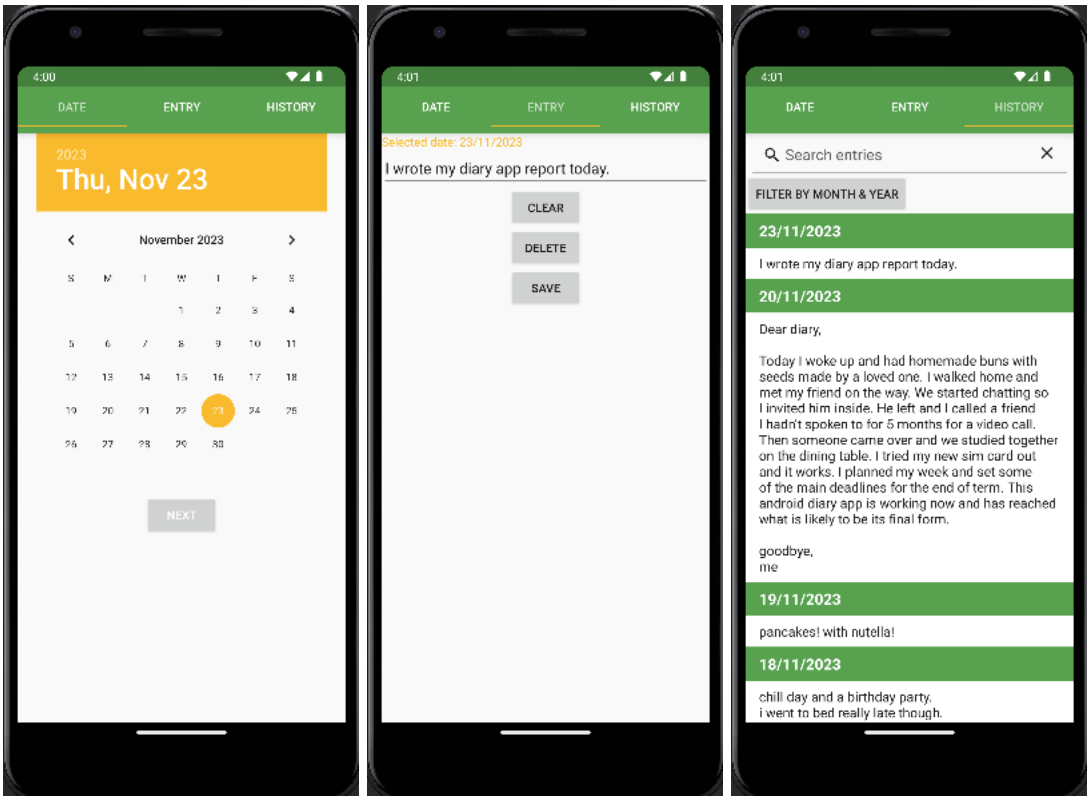


Figure 24: Next button before jumping to entry page.
Figure 25: Save button before jumping to history page.
Figure 26: History page having been jumped to from entry page.

The nextButton in the DateSelectionFragment and the saveButton in the DiaryEntryFragment both use the ViewPager2 widget to navigate between the fragments. The currentItem property of ViewPager2 is set to the index of the next fragment to display. This makes it so that when the next button or save button are pressed the user is automatically taken to the next relevant tab in the diary. The entry tab and the history tab respectively.

```
// automatically navigate to the diary entry page when next button is clicked
(activity as MainActivity).findViewById<ViewPager2>(R.id.pager).currentItem = 1
```

Figure 27: Setting the index of the fragment to display, the diary entry fragment, in the ViewPager2 when the next button is clicked in the next button click event listener in the date selection fragment.

```
// automatically navigate to the diary history page when save button is clicked
(activity as MainActivity).findViewById<ViewPager2>(R.id.pager).currentItem = 2
```

Figure 28: Setting the index of the fragment to display, the diary history fragment, in the ViewPager2 when the save button is clicked in the save button click event listener in the entry fragment.

Save button press jumps and scrolls to saved or edited entry

Figure 29: Saving or editing an entry from a past date.
Figure 30: Being scrolled down to the position of the saved or edited entry in the history tab after saving said entry in the entry tab.

After saving an entry, the ViewPager2 is updated to show the DiaryHistoryFragment, where the updated list of entries is displayed. The RecyclerView is automatically scrolled to the position of the newly saved or edited entry, providing instant feedback to the user.

```
// update the entries display whenever a new entry is saved by observing the diaryEntriesUpdates variable
viewModel.diaryEntriesUpdated.observe(viewLifecycleOwner, Observer { updated ->
    if (updated) {
        val savedEntries = diaryEntryStorage.getAllEntries() // update list of saved entries with potential new/removed entries
        val sortedEntries = savedEntries.entries.sortedByDescending { it: Map.Entry<String, String> // Sort the entries by date recency before displaying them
            // Convert the date strings to LocalDate objects and sort them
            LocalDate.parse(it.key, DateTimeFormatter.ofPattern( pattern: "d/M/yyyy"))
        }
        displayAllEntries()

        // scroll to the edited entry in DiaryHistoryFragment
        val position = sortedEntries.indexOfFirst { it.key == viewModel.selectedDate.value } // finds the position of the edited entry in the entries list
        entriesRecyclerView.scrollToPosition(position) // scrollToPosition(position) then scrolls the RecyclerView to that position

        viewModel.diaryEntriesUpdated.value = false // reset the value of diaryEntriesUpdated
                                                    // to false after updating the diary entries
                                                    // display to avoid unnecessary updates
    }
})
```

Figure 31: The history tab observes, i.e. listens, to the value of the diaryEntriesUpdated value in the ViewModel and uses the scrollToPosition method of the RecyclerView class scroll the Recycler view to the position of the last saved or edited entry after updating the list of entries to display.

Landscape orientation using a scrollable view

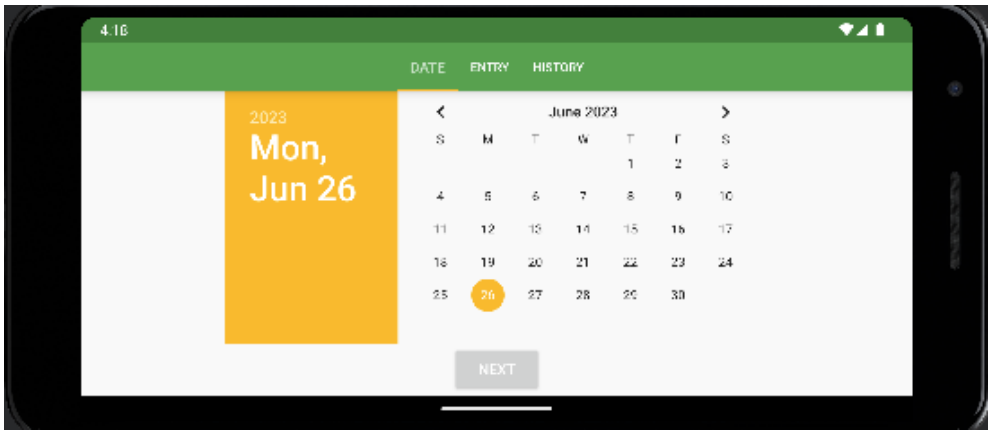


Figure 32: The date selection fragment in landscape orientation.

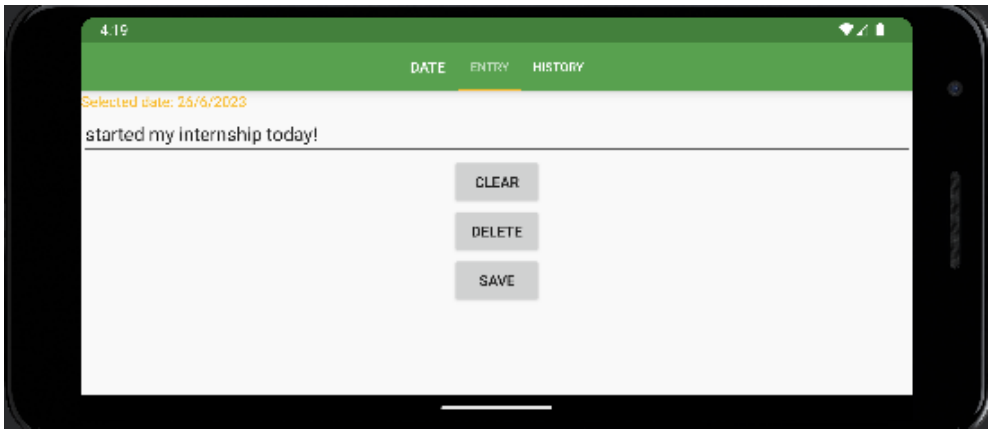


Figure 33: The diary entry fragment in landscape orientation.

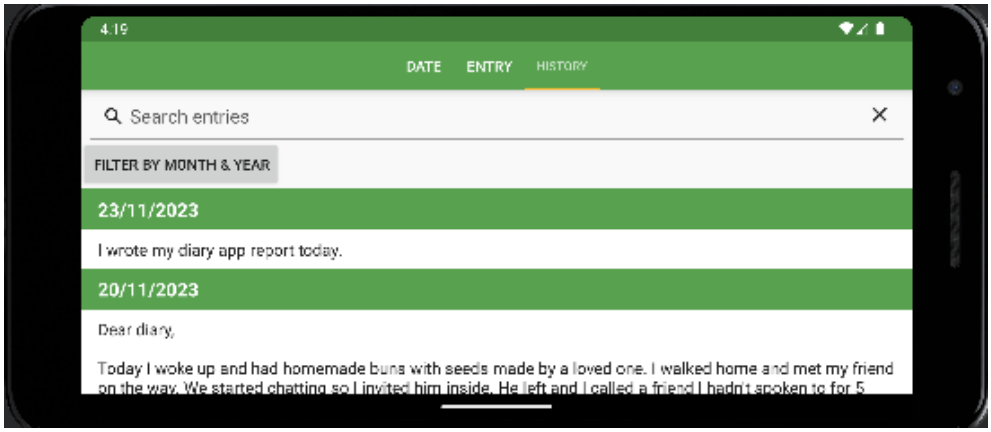


Figure 34: The history fragment in landscape orientation.

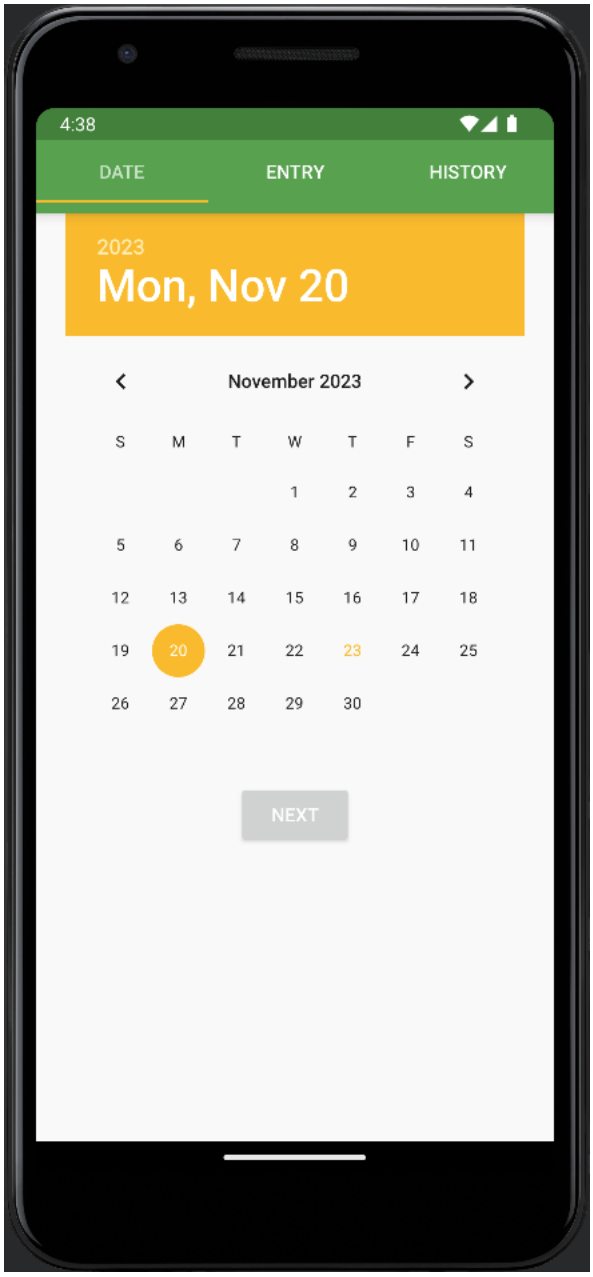
The NestedScrollView in the main activity's XML layout wraps the ViewPager2 widget, allowing the content to be scrolled vertically. This is particularly useful when the content within a fragment extends beyond the visible screen area, ensuring that all content can be accessed through scrolling. For example, when the screen is rotated into landscape orientation all of the content can be accessed through scrolling so that the app is still usable, and functionality does not break.

```
<androidx.core.widget.NestedScrollView
    android:id="@+id/nestedScrollView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fillViewport="true"> <!--  android:fillViewport="true" makes the ViewPager2 fill the
                                      NestedScrollView vertically, allowing you to scroll down
                                      when the content is taller than the screen -->

    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/pager"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</androidx.core.widget.NestedScrollView>
```

Figure 35: The nested scroll view in the main activity's layout XML file which wraps, is a parent of, the ViewPager2 UI element.

UI design

Date selection tab



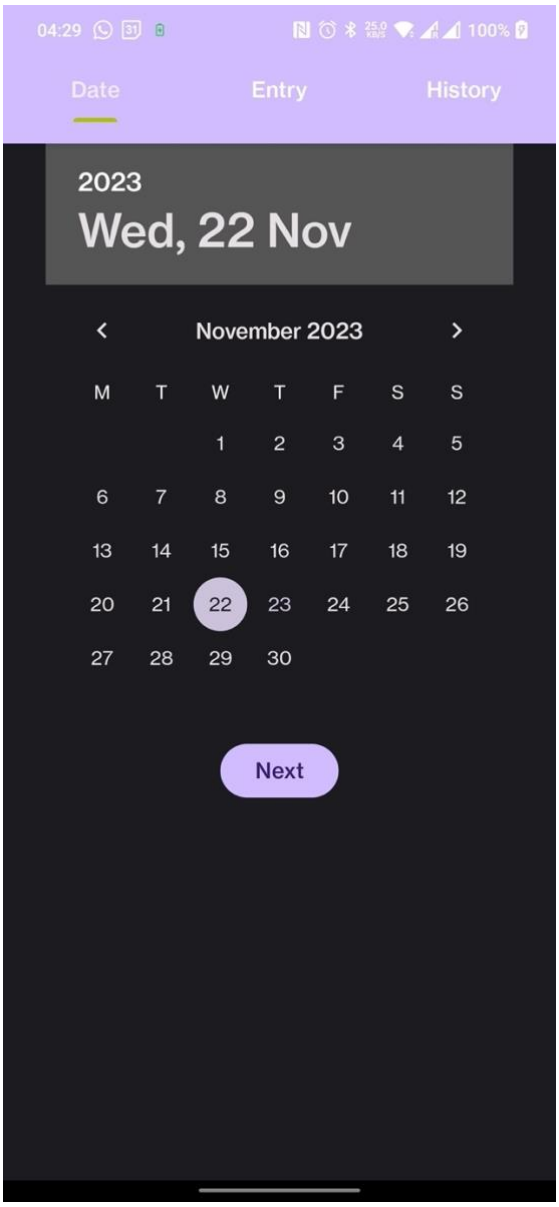


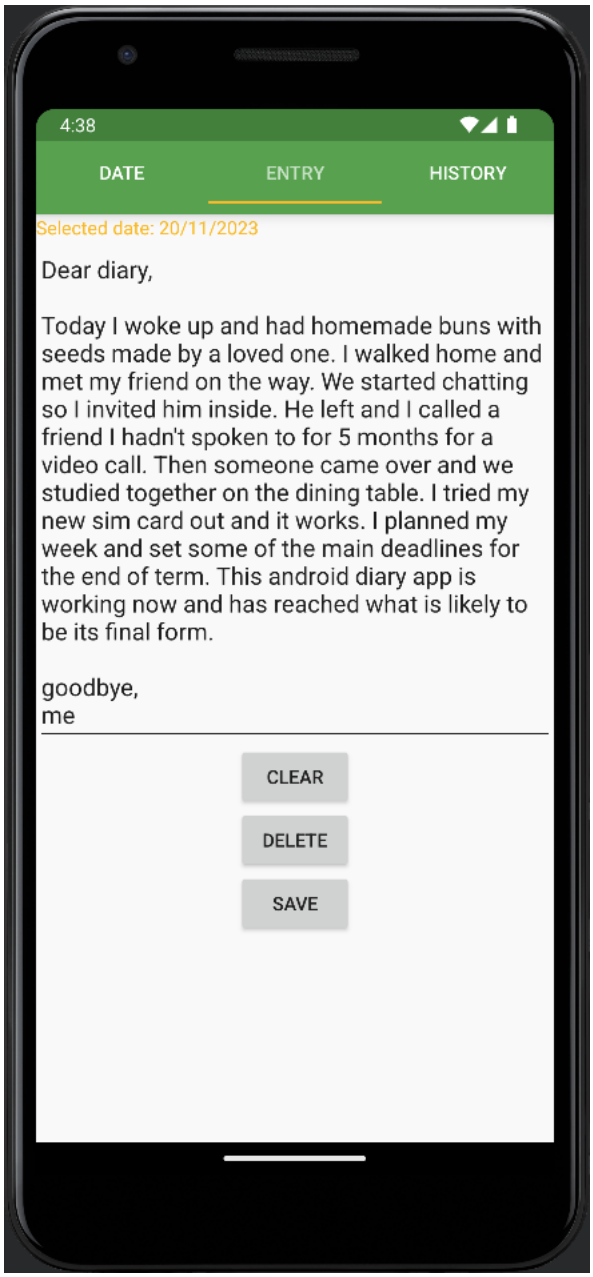
Figure 36: Date selection tab in light mode.

Figure 37: Date selection tab in dark mode. (running on an external Android device)

The Date Selection tab features a user-friendly layout that allows users to pick a date for their diary entry. The DatePicker widget is centrally placed for immediate interaction, with its traditional

calendar view providing a familiar interface. The 'Next' button, positioned below the date picker, guides the user naturally from selecting a date to the next step of entering their diary content. The layout is minimalistic, avoiding unnecessary distractions and focusing the user's attention on the date selection process.

Entry tab



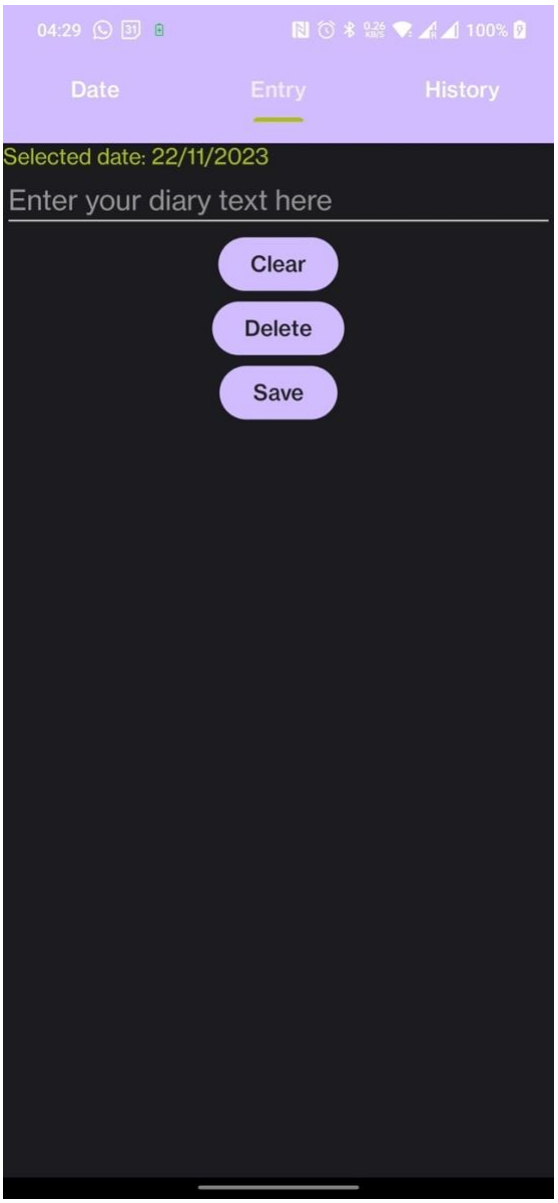
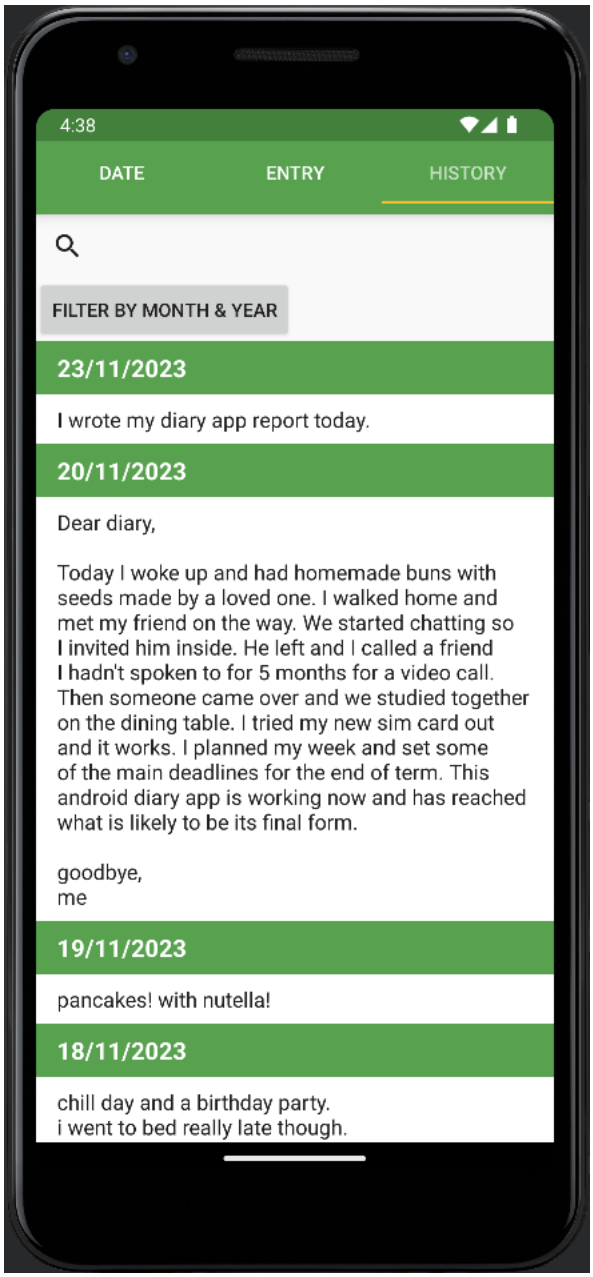


Figure 38: Entry tab in light mode.

Figure 39: Entry tab in dark mode. (running on an external Android device)

The Entry tab is designed for users to write their diary entries. It includes a non-intrusive text box where the selected date is displayed at the top in an accent colour, immediately informing users of the date of their entry. The text box allows for multiline input, catering to extensive writing. For convenience, 'Clear', 'Delete', and 'Save' buttons are positioned below the text box, offering clear options for managing the entry. The simple and intuitive layout ensures a straightforward and distraction-free writing experience.

History tab



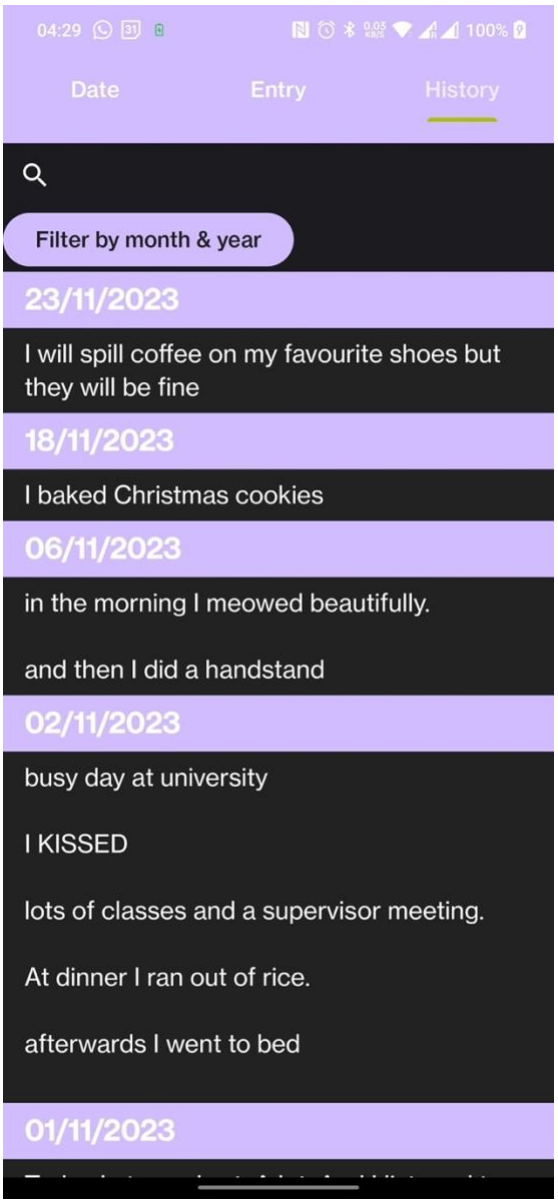


Figure 40: History tab in light mode.
Figure 41: History tab in dark mode. (running on an external Android device)

The History tab is organized to showcase past diary entries. Each entry is presented with a bold date header, distinguishing individual entries by date. The layout employs a linear, scrollable list, making it easy to browse through time. The use of a RecyclerView optimizes performance, allowing for smooth scrolling even with a large number of entries. A search feature at the top and a filter button provide users with tools to quickly find specific entries, enhancing the usability of the history view.

Light and dark mode

The application supports both light and dark modes, catering to user preference and situational lighting conditions. Light mode utilizes a bright colour scheme with dark text for readability, while dark mode inverts this for a darker background with light text. This not only aids in user comfort but also extends to battery saving on devices with OLED screens. The transition between modes is seamless, ensuring a consistent user experience.

Colour scheme

The colour scheme of the application is designed to be visually appealing and functional. The primary colour is a vibrant green, symbolizing freshness and creativity, which is used for headers and important UI elements. Or purple symbolizing mysticism and creativity in dark mode. The secondary colour is white or dark grey, depending on the mode, providing a neutral background that ensures content readability. Accent colours are used sparingly, drawing attention to interactive elements like tabs, buttons, and the current date. Text is rendered in colours that contrast well with their backgrounds, adhering to accessibility standards for readability.

Reflection on boundaries of implemented solution

How complete is my solution?

The solution is very complete with regards to the assignment's criteria. It fulfils the fundamental requirement of a diary app, allowing users to create, edit, and view diary entries by date. It meets the basic assignment criteria and implements many advanced features. It is straight forward to use and is ready for personal use on an Android phone, providing a simple medium for a diary.

Challenges faced

During the development, several challenges were encountered. In particular:

1. **Using a RecyclerView and the EntriesAdapter Class**:

The use of EntriesAdapter to display entries in the RecyclerView within the DiaryHistoryFragment was challenging, particularly managing the data structure using Pair and List. This complexity arose from ensuring that the date headers and entry texts were read correctly, assigned to the correct variables, and had the appropriate layout applied to them when added to the RecyclerView. The solution involved many iterations of the EntriesAdapter and DiaryHistoryFragment classes to ensure the method calls between the two were working correctly and behaving as expected.

2. **Themes and Colours**:
   Integrating the themes and colours specified in the XML files with the layout was initially troublesome. Consistency across different parts of the app was key. The challenge was addressed by meticulously referencing the correct theme and colour resources within the layout XML files, ensuring that the UI remained cohesive and adhered to the design specifications.

Future improvements

Looking beyond the initial scope, the following functionalities could improve the app further:

1. Improved support for landscape orientation and larger tablet screens would make the app more versatile.
2. A toggle for dark and light mode would be a beneficial addition. This would allow users to override the default system theme they have set their Android environment and choose their preferred theme within the app. Implementing this feature would involve adding a switch to the user interface and saving the user's preference, which the app would then apply upon launch.
3. The app currently does not support syncing entries across multiple devices or backing up data to a cloud service. This feature was not implemented due to the increased complexity and the need for secure authentication and data transmission mechanisms which could not be accommodated within the project's time constraints.

Test cases

The diary application underwent a series of manual test cases to ensure each area of functionality performed as intended. These manual tests were sufficient for this application, as they covered all functional aspects of the system and confirmed the app works correctly in standard usage scenarios. The manual testing performed ensures that the app is reliable for everyday use.

Below is an outline of the conducted tests:

Date Selection Tab

Test Case 1: Date Selection Accuracy

Procedure: Open the date selection tab and choose a variety of dates, the current date on launch, and boundary dates (e.g., January 1st, December 31st).

Expected Result: The app should accurately capture and display the selected date in the format "dd/MM/yyyy", and display said date in the entry tab.

Entry Tab

Test Case 2: Entry Creation and Editing

Procedure: Navigate to the entry tab, create a new entry, edit the text, and use the 'Clear', 'Delete', and 'Save' functions.

Expected Result: New entries should be saved and retrievable, and edits should be accurately reflected. The 'Clear' and 'Delete' functions should empty or remove the entry, respectively.

History Tab

Test Case 3: Viewing and Searching Entries

Procedure: Use the history tab to view all entries and utilize the search function to locate specific entries.

Expected Result: All entries should be listed chronologically, and the search function should filter entries in real time, displaying matches as the user types.

Light and Dark Mode

Test Case 4: Theme Consistency

Procedure: Switch between light and dark mode settings of the device and observe the app's theme response.

Expected Result: The app should consistently apply the appropriate theme across all tabs without any visual glitches.

General App Functionality

Test Case 5: Data Storage

Procedure: Create and save entries, close the app completely, and then reopen it.

Expected Result: Previously saved entries should be stored and be displayed correctly upon app restart.

Test Case 6: Navigation

Procedure: Navigate between the 'Date Selection', 'Entry', and 'History' tabs to ensure smooth transitions and state persistence.

Expected Result: Navigation should be seamless, and previously entered data should remain intact when switching between tabs.

Appendix of full Kotlin and UI XML program code

DateSelectionFragment

```
package com.example.diaryentryandroidapp

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.DatePicker
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModelProvider
import androidx.viewpager2.widget.ViewPager2

class DateSelectionFragment : Fragment() {
    lateinit var viewModel: MyViewModel
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        viewModel = activity?.run {
            ViewModelProvider(this)[MyViewModel::class.java]
        } ?: throw Exception("Invalid Activity")
        val view = inflater.inflate(R.layout.date_selection_fragment, container,
false)
        val datePicker = view.findViewById<DatePicker>(R.id.datePicker)
        val nextButton = view.findViewById<Button>(R.id.nextButton)

        nextButton.setOnClickListener {
            val day = datePicker.dayOfMonth
            val month = datePicker.month + 1 // +1 because months indexed
```

```
starting at zero
            val year = datePicker.year
            val selectedDate = "$day/$month/$year"
            viewModel.value.setValue(selectedDate)

            val diaryEntryStorage = DiaryEntryStorage(requireContext())
            val existingEntryText = diaryEntryStorage.getEntry(selectedDate)
            viewModel.existingEntry.setValue(existingEntryText) // set the
existing entry text
            viewModel.value.setValue(selectedDate)

            // automatically navigate to the diary entry page when next button is
clicked
            (activity as
MainActivity).findViewById<ViewPager2>(R.id.pager).currentItem = 1
        }
        return view
    }
}
```

DiaryEntryFragment

```
package com.example.diaryentryandroidapp

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import androidx.fragment.app.Fragment
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProvider
import androidx.viewpager2.widget.ViewPager2
import java.text.SimpleDateFormat
import java.util.Date
import java.util.Locale

class DiaryEntryFragment : Fragment() {
    lateinit var viewModel: MyViewModel
    private var selectedDate: String = "" // selectedDate is defined at the class
level
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        viewModel = activity?.run {
            ViewModelProvider(this)[MyViewModel::class.java]
        } ?: throw Exception("Invalid Activity")
        val view = inflater.inflate(R.layout.diary_entry_fragment, container,
false)

        val selectedDateView = view.findViewById<TextView>(R.id.selectedDate)
        // make a variable which stores the selectedDate
        val selectedDateValueObserver = Observer<String> { newValue ->
            selectedDate = newValue // Update selectedDate here, which is defined
at a class level, here
            selectedDateView.text = "Selected date: $selectedDate"
```

```
        }
        viewModel.value.observe(viewLifecycleOwner, selectedDateValueObserver) //
observes changes to the value of the selected date in the view model
                                                                                //
and updates the selected date observer variable value here when it changes
        // selectedDateValueObserver is set before saveButton click listener
because this updates selectedDate,
        // and I want to ensure this happens before I try to save the
selectedDate when the saveButton is clicked.

        val diaryText = view.findViewById<EditText>(R.id.diaryText)
        // observe existingEntry and set the text of diaryText when it changes
        viewModel.existingEntry.observe(viewLifecycleOwner, Observer { entryText
->
            diaryText.setText(entryText)
        })
        // set the diary text to that of the current date entry if it exists when
the app launches
        val currentDate = SimpleDateFormat("d/M/yyyy",
Locale.getDefault()).format(Date())
        val diaryEntryStorage = DiaryEntryStorage(requireContext())
        val entryText = diaryEntryStorage.getEntry(currentDate)
        diaryText.setText(entryText)

        val saveButton = view.findViewById<Button>(R.id.saveButton)
        saveButton.setOnClickListener {
            val diaryEntry = diaryText.text.toString()
            val diaryEntryStorage = DiaryEntryStorage(requireContext())
            // If an empty entry is saved the entry for that date should be
deleted
            if (diaryEntry.isBlank()) {
                diaryEntryStorage.deleteEntry(selectedDate)
            } else {
                // only save the entry if the user actually wrote an entry, i.e.,
text !blank
                diaryEntryStorage.saveEntry(selectedDate, diaryEntry)
            }
            // update the MyViewModel instance shared between fragments when a
new entry is saved,
            // so that DiaryHistoryFragment can updates its list of displayed
entries
            viewModel.diaryEntriesUpdated.value = true

            // automatically navigate to the diary history page when save button
is clicked
            (activity as
MainActivity).findViewById<ViewPager2>(R.id.pager).currentItem = 2
        }

        val clearButton = view.findViewById<Button>(R.id.clearButton)
        clearButton.setOnClickListener {
            diaryText.text.clear()
        }

        val deleteButton = view.findViewById<Button>(R.id.deleteButton)
        deleteButton.setOnClickListener {
            val diaryEntryStorage = DiaryEntryStorage(requireContext())
            diaryEntryStorage.deleteEntry(selectedDate)
            viewModel.diaryEntriesUpdated.value = true
```

```
        }

        return view
    }
}
```

DiaryEntryStorage

```
package com.example.diaryentryandroidapp

import android.content.Context

class DiaryEntryStorage(context: Context) {
    /* When a new instance of a class is created, the data is not lost every time
the app runs
     * because the data is stored in memory.
     * When the app is closed, the data is saved to disk so that it can be
retrieved
     * when the app is opened again.
     * This is done using storage mechanisms such as shared preferences.
     */
    private val sharedPreferences = context.getSharedPreferences("DiaryEntries",
Context.MODE_PRIVATE)

    fun saveEntry(date: String, entry: String) {
        sharedPreferences.edit().putString(date, entry).apply()
    }

    fun deleteEntry(date: String) {
        sharedPreferences.edit().remove(date).apply()
    }

    fun getEntry(date: String): String {
        return sharedPreferences.getString(date, "") ?: ""
    }

    fun getAllEntries(): Map<String, String> {
        return sharedPreferences.all.filterValues { it is String } as Map<String,
String>
    }
}
```

DiaryHistoryFragment

```
package com.example.diaryentryandroidapp

import android.app.DatePickerDialog
import android.os.Build
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.SearchView
```

```
import androidx.annotation.RequiresApi
import androidx.fragment.app.Fragment
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProvider
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import java.time.LocalDate
import java.time.format.DateTimeFormatter
import java.util.Calendar

class DiaryHistoryFragment : Fragment() {
    private lateinit var viewModel: MyViewModel
    @RequiresApi(Build.VERSION_CODES.O) // indicate that methods in this class
require API level 26 or higher (i.e., Android Oreo) to function properly
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        viewModel = activity?.run {
ViewModelProvider(this)[MyViewModel::class.java]
        } ?: throw Exception("Invalid Activity")
        val view = inflater.inflate(R.layout.diary_history_fragment, container,
false)
        val entriesRecyclerView: RecyclerView =
view.findViewById(R.id.entriesRecyclerView)
        entriesRecyclerView.layoutManager = LinearLayoutManager(requireContext())

        // Create a new instance of DiaryEntryStorage class and assign it to
diaryEntryStorage variable
        val diaryEntryStorage = DiaryEntryStorage(requireContext())

        /**
         * Display all entries in entry storage in chronological order starting
with newest
         */
        fun displayAllEntries() {
            // The saved entries are available in a list called savedEntries
            val savedEntries = diaryEntryStorage.getAllEntries() // update list
of saved entries with potential new/removed entries
            // Sort the entries by date recency before displaying them
            val sortedEntries = savedEntries.entries.sortedByDescending {// Sort
the entries by date recency before displaying them
                LocalDate.parse(it.key, DateTimeFormatter.ofPattern("d/M/yyyy"))
// Convert the date strings to LocalDate objects and sort them
            }

            val entriesWithDateHeaders = mutableListOf<Pair<String, String>>()
            for (entry in sortedEntries) {
                val entryDate = LocalDate.parse(entry.key,
DateTimeFormatter.ofPattern("d/M/yyyy"))
                val formattedDate =
entryDate.format(DateTimeFormatter.ofPattern("dd/MM/yyyy"))
                val entryText = entry.value
                entriesWithDateHeaders.add(Pair(formattedDate, entryText))
            }

            // creates a new instance of EntriesAdapter with the sorted entries,
i.e., displays the sorted entries
            entriesRecyclerView.adapter = EntriesAdapter(entriesWithDateHeaders)
        }
```

```
        displayAllEntries() // display all diary entries when app starts


        // update the entries display whenever a new entry is saved by observing
the diaryEntriesUpdates variable
        viewModel.diaryEntriesUpdated.observe(viewLifecycleOwner, Observer {
updated ->
            if (updated) {
                val savedEntries = diaryEntryStorage.getAllEntries() // update
list of saved entries with potential new/removed entries
                val sortedEntries = savedEntries.entries.sortedByDescending { //
Sort the entries by date recency before displaying them
                    // Convert the date strings to LocalDate objects and sort
them
                    LocalDate.parse(it.key,
DateTimeFormatter.ofPattern("d/M/yyyy"))
                }
                displayAllEntries()

                // scroll to the edited entry in DiaryHistoryFragment
                val position = sortedEntries.indexOfFirst { it.key ==
viewModel.selectedDate.value } // finds the position of the edited entry in the
entries list
                entriesRecyclerView.scrollToPosition(position) //
scrollToPosition(position) then scrolls the RecyclerView to that position

                viewModel.diaryEntriesUpdated.value = false // reset the value of
diaryEntriesUpdated
                                                            // to false after
updating the diary entries
                                                            // display to avoid
unnecessary updates
            }
        })

        /**
         * The filterDiaryEntries method gets all diary entries,
         * filters the entries by the selected date,
         * and updates the RecyclerView adapter with the filtered entries.
         */
        fun filterDiaryEntries(year: Int, month: Int) {
            val allEntries = diaryEntryStorage.getAllEntries()
            val filteredEntries = allEntries.filterKeys {
                val entryDate = LocalDate.parse(it,
DateTimeFormatter.ofPattern("d/M/yyyy"))
                entryDate.year == year && entryDate.monthValue == month
            }

            val sortedEntries = filteredEntries.entries.sortedByDescending {
                LocalDate.parse(it.key, DateTimeFormatter.ofPattern("d/M/yyyy"))
            }

            val entriesWithDateHeaders = mutableListOf<Pair<String, String>>()
            for (entry in sortedEntries) {
                val entryDate = LocalDate.parse(entry.key,
DateTimeFormatter.ofPattern("d/M/yyyy"))
                val formattedDate =
entryDate.format(DateTimeFormatter.ofPattern("dd/MM/yyyy"))
                val entryText = entry.value
```

```
                entriesWithDateHeaders.add(Pair(formattedDate, entryText))
            }

            // creates a new instance of EntriesAdapter with the sorted entries,
i.e., displays the sorted entries
            entriesRecyclerView.adapter = EntriesAdapter(entriesWithDateHeaders)
        }


        // add a click event listener to the filter button
        val filterButton: Button = view.findViewById(R.id.filterButton)
        filterButton.setOnClickListener {
            if (filterButton.text == "Filter by month & year") {
                val datePickerDialog = DatePickerDialog(
                    requireContext(),
                    { _, year, month, _ ->
                        filterDiaryEntries(year, month + 1)
                        filterButton.text = "Clear filter"
                    },
                    Calendar.getInstance().get(Calendar.YEAR),
                    Calendar.getInstance().get(Calendar.MONTH),
                    Calendar.getInstance().get(Calendar.DAY_OF_MONTH)
                )
                // Show the DatePickerDialog when the filter button is pressed
                datePickerDialog.show()
            } else { // if button text is clear filter
                displayAllEntries()
                filterButton.text = "Filter by month & year"
            }
        }

        /**
         * Get all diary entries,
         * filter the entries by the search query,
         * and update the RecyclerView adapter with the filtered entries.
         */
        fun filterEntriesBySearchQuery(query: String) {
            val allEntries = diaryEntryStorage.getAllEntries()
            val filteredEntries = allEntries.filterValues { it.contains(query,
ignoreCase = true) }

            val sortedEntries = filteredEntries.entries.sortedByDescending {
                LocalDate.parse(it.key, DateTimeFormatter.ofPattern("d/M/yyyy"))
            }

            val entriesWithDateHeaders = mutableListOf<Pair<String, String>>()
            for (entry in sortedEntries) {
                val entryDate = LocalDate.parse(entry.key,
DateTimeFormatter.ofPattern("d/M/yyyy"))
                val formattedDate =
entryDate.format(DateTimeFormatter.ofPattern("dd/MM/yyyy"))
                val entryText = entry.value
                entriesWithDateHeaders.add(Pair(formattedDate, entryText))
            }

            // creates a new instance of EntriesAdapter with the sorted entries,
i.e., displays the sorted entries
            entriesRecyclerView.adapter = EntriesAdapter(entriesWithDateHeaders)
        }
```

```
        /**
         * Filter the entries by the search query when the user submits a search.
         */
        val searchView: SearchView = view.findViewById(R.id.searchView)
        searchView.setOnQueryTextListener(object : SearchView.OnQueryTextListener
{
            override fun onQueryTextSubmit(query: String): Boolean {
                filterEntriesBySearchQuery(query)
                return false
            }

            override fun onQueryTextChange(newText: String): Boolean {
                // update the search results in real-time while the user is
typing
                filterEntriesBySearchQuery(newText)
                return false
            }
        })

        return view;
    }
}
```

EntriesAdapter

```
package com.example.diaryentryandroidapp;

import android.graphics.Typeface;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.recyclerview.widget.RecyclerView;

class EntriesAdapter(private val entries: List<Pair<String, String>>) :
    RecyclerView.Adapter<EntriesAdapter.ViewHolder>() {

    // One ViewHolder class that can handle both types of views
    class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val dateHeader: TextView = itemView.findViewById(R.id.dateHeader);
        val entryText: TextView = itemView.findViewById(R.id.entryText);
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder
{
        val view =
LayoutInflater.from(parent.context).inflate(R.layout.entry_item, parent, false);
        return ViewHolder(view);
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val (date, text) = entries[position];
        holder.dateHeader.text = date;
        holder.entryText.text = text;
    }
```

```
    override fun getItemCount() = entries.size;

    override fun getItemViewType(position: Int): Int {
        // Assuming that if the entry text is empty, it's a date header
        return if (entries[position].second.isEmpty()) 1 else 2;
    }
}
```

MainActivity

```
package com.example.diaryentryandroidapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.appcompat.widget.Toolbar
import androidx.viewpager2.widget.ViewPager2
import com.google.android.material.tabs.TabLayout
import com.google.android.material.tabs.TabLayoutMediator

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main) // setting the view
//        val toolbar = findViewById<Toolbar>(R.id.toolbar) // initialise the
toolbar by loading its
                                                          // resource and passing
it into a Toolbar
                                                          // object
//        setSupportActionBar(toolbar)

        // set up the tabs we need for each of the fragments
        val tabLayout = findViewById<TabLayout>(R.id.tab_layout)

tabLayout.addTab(tabLayout.newTab().setText(R.string.date_selection_tab_page1))

tabLayout.addTab(tabLayout.newTab().setText(R.string.diary_entry_tab_page2))

tabLayout.addTab(tabLayout.newTab().setText(R.string.diary_history_tab_page3))
        tabLayout.tabGravity = TabLayout.GRAVITY_FILL

        // link the ViewPager2 area in the layout with an instance of the adaptor
class defined above
        val viewPager = findViewById<ViewPager2>(R.id.pager)
        val adapter = PageAdapter(this, 3)
        viewPager.setAdapter(adapter)

        // synchronize tab selection with the ViewPager2
        TabLayoutMediator(tabLayout, viewPager) { tab, position ->
            when (position) {
                0 -> tab.text =
resources.getString(R.string.date_selection_tab_page1)
                1 -> tab.text =
resources.getString(R.string.diary_entry_tab_page2)
                2 -> tab.text =
resources.getString(R.string.diary_history_tab_page3)
            }
        }.attach() // attach() method is called to attach the mediator to the
```

```
TabLayout and ViewPager2
    }
}
```

MyViewModel

```
package com.example.diaryentryandroidapp

import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import java.util.Calendar

/**
 * MyViewModel is used to share data between different fragments.
 * It holds the selected date and whether the diary entries have been updated
(diaryEntriesUpdated).
 * It also holds the existing entry (existingEntry),
 * which is used to populate the diary entry text when a date that already has an
entry is selected.
 */
class MyViewModel : ViewModel() {
    val selectedDate = MutableLiveData<String>()
    // The selectedDate is a MutableLiveData object, so you can't directly assign
a String value to it.
    // Instead, the String value is assigned to the value property of the
MutableLiveData
    val value: MutableLiveData<String>
        get() = selectedDate
    init {
        // set the default (starting value of) selected date to be the current
date
        val calendar = Calendar.getInstance()
        val day = calendar.get(Calendar.DAY_OF_MONTH)
        val month = calendar.get(Calendar.MONTH) + 1 // +1 because months indexed
starting at zero
        val year = calendar.get(Calendar.YEAR)
        selectedDate.value = "$day/$month/$year"
    }

    val diaryEntriesUpdated = MutableLiveData<Boolean>()

    val existingEntry = MutableLiveData<String>()
}
```

PageAdapter

```
package com.example.diaryentryandroidapp

import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentActivity
import androidx.viewpager2.adapter.FragmentStateAdapter

class PageAdapter(fa: FragmentActivity, private val mNumOfTabs: Int) :
    FragmentStateAdapter(fa) {
```

```
    override fun getItemCount(): Int {
        return mNumOfTabs
    }
    override fun createFragment(position: Int): Fragment {
        return when (position) {
            0 -> DateSelectionFragment()
            1 -> DiaryEntryFragment()
            2 -> DiaryHistoryFragment()
            else -> DateSelectionFragment()
        }
    }
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <com.google.android.material.tabs.TabLayout
        android:id="@+id/tab_layout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"

        android:background="?attr/colorPrimary"
        app:tabIndicatorColor="@color/colorAccent"
        app:tabTextColor="?android:attr/textColor"
        app:tabSelectedTextColor="?android:attr/textColorSecondary"

        android:elevation="6dp"
        android:minHeight="?attr/actionBarSize"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" />

    <androidx.core.widget.NestedScrollView
        android:id="@+id/nestedScrollView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:fillViewport="true"> <!--  android:fillViewport="true" makes the
ViewPager2 fill the
                                          NestedScrollView vertically, allowing
you to scroll down
                                          when the content is taller than the
screen -->

        <androidx.viewpager2.widget.ViewPager2
            android:id="@+id/pager"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

    </androidx.core.widget.NestedScrollView>

</LinearLayout>
```

date_selection_fragment.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <!-- android:datePickerMode="spinner" instead of cal view shown for spinner
cal -->
    <DatePicker
        android:id="@+id/datePicker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:calendarViewShown="true" />

    <Button
        android:id="@+id/nextButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Next"
        android:layout_gravity="center_horizontal" />

</LinearLayout>
```

diary_entry_fragment.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/selectedDate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="@color/colorAccent" />

    <EditText
        android:id="@+id/diaryText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textMultiLine|textImeMultiLine"
        android:gravity="start"
        android:hint="Enter your diary text here" />

    <Button
        android:id="@+id/clearButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Clear"
```

```
        android:layout_gravity="center_horizontal"
        android:textColor="@color/selectedTextColorOnPrimaryBackground" />

    <Button
        android:id="@+id/deleteButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Delete"
        android:layout_gravity="center_horizontal"
        android:textColor="@color/selectedTextColorOnPrimaryBackground" />

    <Button
        android:id="@+id/saveButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Save"
        android:layout_gravity="center_horizontal"
        android:textColor="@color/selectedTextColorOnPrimaryBackground" />

</LinearLayout>
```

diary_history_fragment.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <SearchView
        android:id="@+id/searchView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:queryHint="Search entries" />

    <Button
        android:id="@+id/filterButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Filter by month &amp;amp; year"
        android:textColor="@color/selectedTextColorOnPrimaryBackground" />

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/entriesRecyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

entry_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <TextView
        android:id="@+id/dateHeader"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingStart="16dp"
        android:paddingTop="8dp"
        android:paddingBottom="8dp"
        android:textSize="18sp"
        android:textStyle="bold"
        android:textColor="?android:textColor"
        android:background="?attr/colorPrimary" /> <!-- Date Header Background
Color -->

    <TextView
        android:id="@+id/entryText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingStart="16dp"
        android:paddingEnd="16dp"
        android:paddingTop="8dp"
        android:paddingBottom="8dp"
        android:textSize="16sp"
        android:textColor="?android:textColorSecondary"
        android:background="@color/colorSecondaryBackground" /> <!-- Entry
Background Color -->
</LinearLayout>
```

colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <!-- Base colors for app -->
    <color name="colorPrimaryBackground">#4CAF50</color>
    <color name="colorPrimaryDarkBackground">#388E3C</color>
    <color name="colorAccent">#FFC107</color>
    <color name="textColorOnPrimaryBackground">#FFFFFF</color>
    <color name="selectedTextColorOnPrimaryBackground">#212121</color>
    <!-- entry_item colors -->
    <color name="colorSecondaryBackground">#FFFFFF</color>
</resources>
```

colors.xml (night)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <!-- Base colors for app -->
    <color name="colorPrimaryBackground">#444444</color>
```

```
    <color name="colorPrimaryDarkBackground">#333333</color>
    <color name="colorAccent">#AAC107</color>
    <color name="textColorOnPrimaryBackground">#FFFFFF</color>
    <color name="selectedTextColorOnPrimaryBackground">#212121</color>
    <!-- entry_item colors -->
    <color name="colorSecondaryBackground">#222222</color>
</resources>
```

themes.xml

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Base.Theme.DiaryEntryAndroidApp"
parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your light theme here. -->
        <!-- <item name="colorPrimary">@color/my_light_primary</item> -->
        <!-- Primary color. -->
        <item name="colorPrimary">@color/colorPrimaryBackground</item>
        <!-- Primary dark color for the status bar and deeper colored areas. -->
        <item name="colorPrimaryDark">@color/colorPrimaryDarkBackground</item>
        <!-- Accent color (used as the default value for colorControlActivated,
which is used to tint widgets). -->
        <item name="colorAccent">@color/colorAccent</item>
        <!-- Default text colors to ensure readability -->
        <item name="android:textColor">@color/textColorOnPrimaryBackground</item>
        <item
name="android:textColorSecondary">@color/selectedTextColorOnPrimaryBackground</it
em>
    </style>

    <style name="Theme.DiaryEntryAndroidApp"
parent="Base.Theme.DiaryEntryAndroidApp" />
</resources>
```

themes.xml (night)

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Base.Theme.DiaryEntryAndroidApp"
parent="Theme.Material3.DayNight.NoActionBar">
        <!-- Customize your dark theme here. -->
        <!-- <item name="colorPrimary">@color/my_dark_primary</item> -->
        <item name="android:textColor">@color/white</item>
        <item name="android:textColorSecondary">@color/white</item>
    </style>

    <style name="Theme.DiaryEntryAndroidApp"
parent="Base.Theme.DiaryEntryAndroidApp" />
</resources>
```