CSCU9YH APP REPORT

2721301

Contents

1 – Structure 2
2 - System Overview 3
2.1 – Page1 (Date Picker) 3
2.2 – Page2 (Add Note) 4
2.3 – Page3 (View Notes) 4
3 - Key Features 6
3.1 – Basic Features 6
3.2 – Advanced Features 7
4 - UI Design 10
5 - Reflection 11
5.1 - Reflection on Solution 11
5.2 - Boundaries of Solution 12
6 - Test Cases 13
7 - Code 17
References 42

1 – Structure

The structure of an app can make a huge difference on user experience. If not implemented in a satisfactory way the user may have a terrible experience and are likely not to use the app again. I heavily considered the user experience when designing my app, and due to this I have made my app structure very linear and easy to understand, clearly showing the options the user has and steps they may take to complete an action.

The structure contains three pages that the users will encounter, each using fragments. Starting from the first, the date selection page, the user has the options to either select a date or move to the view note screen. If view notes is clicked the view will change to the view notes screen where the users can see any previous notes. The user also can click the “new note” button that will navigate them back to the date selection page. I use buttons to allow users to navigate to the screens they desire as without them the user may feel trapped on a screen. Navigation buttons show the options users have when using the app. If the user selects and confirms a date, the view will change to the new note page where a new note can be input and submitted. From this screen the user also can go back to date selection. On submission of the new note, the screen will automatically change to the view notes page for the users to see what they have added. Figure 1 below shows the screens and how they are linked.

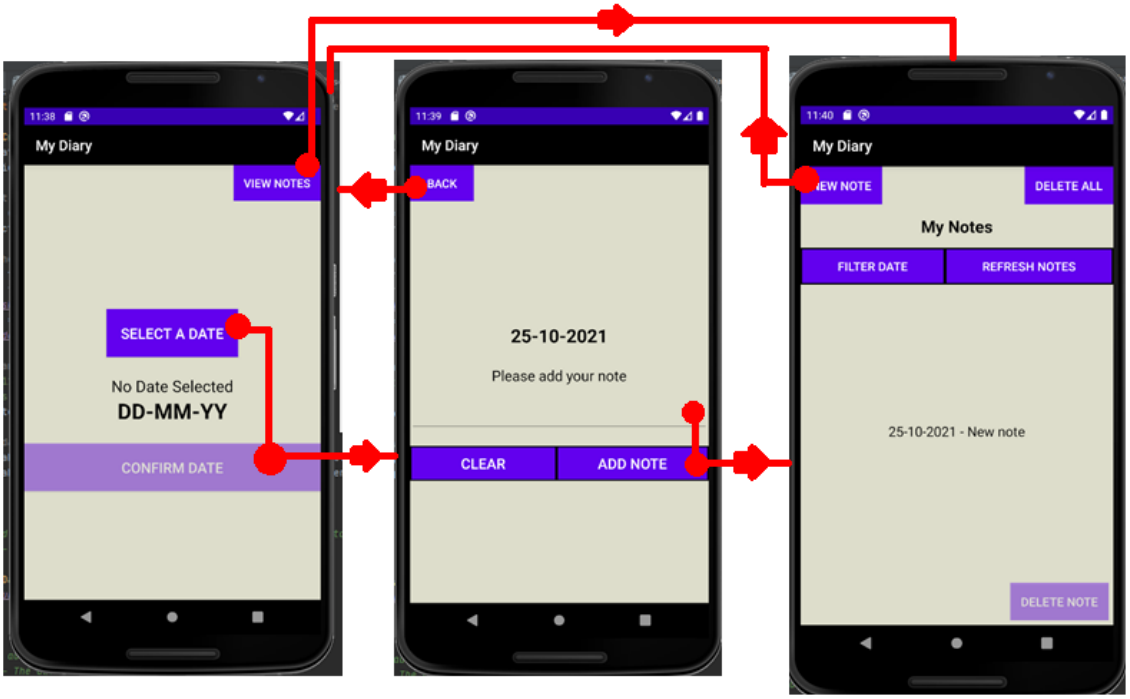

Figure 1: Page1 (date select), Page2 (add note), Page3 (view notes)

2 - System Overview

Fragments can be described as a sub-section of an activity and are useful for splitting up a task into sections, each displayed on each screen. My app contains three fragments, page1 which is the date selection screen, page2 which is the note submission screen, and page3 which is the note viewer screen. I have created a page adapter for setting up these fragments when the app is run. I have also implemented a viewPager that allows the fragment currently being viewed to change in response to some action.

Within fragments the view cannot be accessed like in the main activity class, so view binding is required to access the UI elements of each fragment screen. Data cannot also be passed like a regular class and other methods need to be implemented. The methods I decided on were view models and a room database. View models are a useful tool to pass data between user interface elements as it allows observation of live data, and the live data can exist after a screen configuration change such as changing to horizontal mode. The room database is set up with a “Note” entity class, a “NoteDao” interface class, and an abstract “NoteRoomDatabase” class used to instantiate and return an instance of the database. “NoteDao” provides database manipulation methods to be called from main, and can be used to return, delete, update, and insert notes. A “Note” entity class in instantiated for each note and contains a date primary key and a note body string. A note entity is instantiated and are added to the room database. Each fragment uses a DataPasser interface to pass data from the fragments back to main, to manipulate the database or inform that the screen view needs to change. These database manipulation methods are the same as the ones contained within “NoteDao”. These methods are overridden in main to allow fragments to call them through the interface. I have provided a UML diagram showing how the classes link in figure 3.

I use toast messages throughout my application to inform users on their actions. Toast displays a little temporary message at the bottom of the page and will tell the users things such as their note has been added successfully.

2.1 – Page1 (Date Picker)

This is the first fragment and will be the first screen that the user will meet, for the purpose of selecting a date. This page contains some buttons, “VIEW NOTES” that allow users to move to Page3, “SELECT DATE” which will open a date picker, and “CONFIRM DATE” which will initially be greyed out and unclickable until the user selects a date. My method of date selection is using a datePickerDialogue which provides a handy calendar that the users can navigate and select any date they please. The initial date selected will be the current date as this is likely what users will be adding to most, making the process faster. On clicking the “CONFIRM DATE” button, the date will be saved to the view model class called “SharedViewModel”, to a MutableLiveData variable that will allow Page2 to observe the date selected. Clicking the “CONFIRM DATE” button will also change the view to page2 to add a note, by use of the dataPasser that communicates with the main activity.

2.2 – Page2 (Add Note)

This page is where the user will input some text to add a note. The page containing a TextView that displays the selected date and a message informing users what to do, there is a “BACK” button, a text box, a “CLEAR” button, and a “ADD NOTE” button. The date is displayed by observing the “LiveData” variable in the “SharedViewModel” class. The user can input text into the text box which will assist users by auto capitalization and auto correct. Users can clear what they have entered, and they can also submit using buttons. If the user tried to submit a blank message, a toast message will inform them they need to enter text, and nothing will happen. If text is entered and added the page view will move to “Page3” and a toast message will confirm the addition, and lets users know what to do next.

2.3 – Page3 (View Notes)

“Page3” is where the user may access their notes and they have some actions they can perform here. The buttons contained on this page are “NEW NOTE” that allows users to navigate back to “Page1”, “DELETE ALL”, “FILTER DATE”, “REFRESH NOTES”, “DELETE NOTE”. After adding a note, they are informed they need to click the “REFRESH NOTE” button, which will retrieve all the notes stored in the room database and they will be displayed in the text box that is centre in the page. When too many notes are displayed on this page, a scroll wheel will be available to scroll through the notes. “DELETE ALL” will completely wipe the database of any notes to start fresh, this is done on start-up currently to avoid confusion when testing the app. “FILTER DATE” will display a DatePickerDialogue much like “Page1” and the users select a date they want to see a note for. If the note is found, a confirmation toast message will display, and the note will display in the centre of the page. The “DELETE NOTE” button will then become clickable and fully coloured and allows users to delete the note that they have selected.

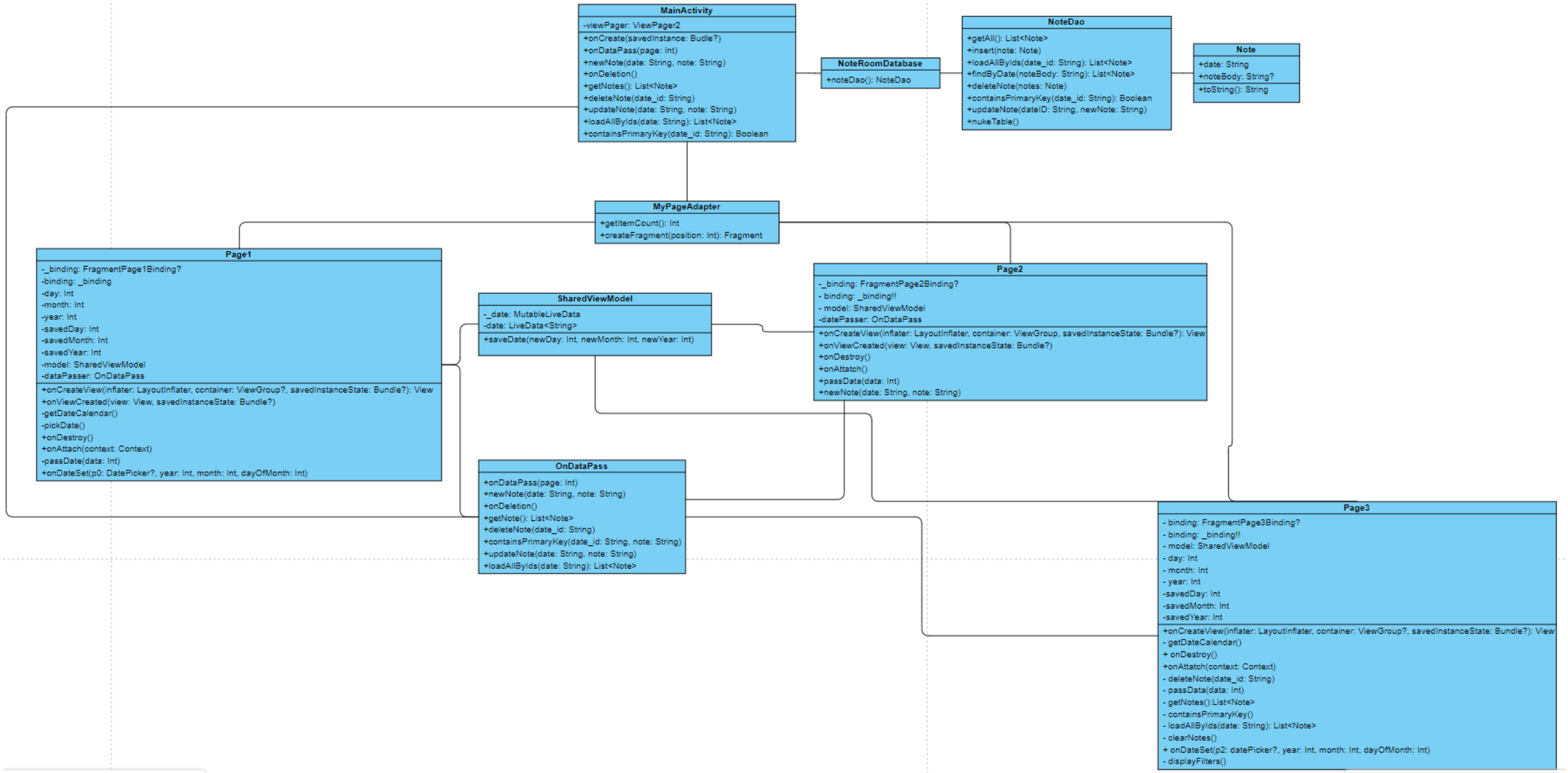

Figure 2: UML Class Diagram

3 - Key Features

Several features are suggested in the project brief, some being basic features that the app must contain, and others being ideas for advanced features. My application implements all the basic features listed and has been further enhanced with some advanced features.

3.1 – Basic Features

3.1.1 – Date Picker

As described in the brief, the first page must allow users to select a date in a user-friendly way. The method I have used to implement this is using a date picker from the DatePickerDialog. This method displays a calendar that is currently set to the current date, but users may select any date they please. This date can be confirmed when the user has decided.

3.1.2 – Adding a note for selected date

The second screen allows users to input notes to the diary. View models are implemented within these fragments to allow screen 2 to observe the date selected in screen 1. This screen displays the date previously selected and allows the users to input text.

3.1.3 – Navigation

Each page of the app contains buttons that allow users to navigate to another page if they wish to do so. From the first date selection screen the user can click the view notes button to move to the view notes page. On the add note page after selecting the date, the user can navigate back to the date selection screen by clicking back. On the view note screen the user can navigate back to the date selection screen by clicking the new note button.

3.1.4 – View all notes

On the view notes page, the user can view their notes by first clicking the refresh button. All notes previously stored will be displayed in the centre of the page beside the date they have been added to. If no notes have been stored, then a message will display that no notes have been found. This window also contains a scroll bar if the list of notes is too large to be displayed on one window.

3.2 – Advanced Features

3.2.1 – Delete all notes

From the view notes screen, the user can delete all notes they have previously stored. After clicking the delete all button, a message will appear confirming that the user has deleted all their notes. On clicking the refresh button, the page will display a message that no notes have been found.

3.1.2 – Filtering note by a date

On the view note screen the users can view any notes they stored for a specific date. On clicking the filter date button, a DatePickerDialogue box will appear allowing users to select a date, much like the first screen. On date selection, a message will display informing users if a note has been found for that date, or not. If a note is found for that date, then the note will be displayed in the centre of the page.

3.1.3 – Update note

Users can update their notes that they have previously stored. If the user selects a date that already contains a note and inserts some text on the new note screen as normal, a message will display that a note already exists and will be updated. This note will be added as a new sentence to the end of the existing note. The updated note will be formatted with a full stop and space to separate the notes, so it looks presentable and correct.

3.1.4 – Deleting a note by a date

After finding a note for a certain date, the delete note button will then become clickable and no longer greyed out. The user can then delete the note for the selected date, a message will inform them of the deletion and the message in the centre of the page will display no notes have been found.

3.1.6 – Screen scaling

When creating the layout, I made sure to make everything included in the screen relative. No element in my app is in a fixed position, so that when the screen size is changed the elements in the page will also resize the match the display. The app can be used on both small devices and large tablets as shown on figure 4 at the end of this section.

3.1.5 – Screen rotation

Due to the relativity of my UI elements, the view can handle being rotated to an extent. The elements will fit the horizontal view and will still look presentable to the user. The rotation is only a cosmetic change currently however, and the data stored will be wiped on every rotation. The view is deleted and remade on each orientation change. The data is lost due to the database being deleted on start-up, and the orientation would work correctly if this was removed. I wanted to keep this in

however to avoid any confusion when testing my app. The horizontal layout can be viewed in figure 5.

3.1.7 – Database

I used an advanced method for storing the notes by use of a RoomDataBase. This method allows for better functionality for retrieving, updating, and deleting notes. I have a class within my code called NoteDao used to access the database and runs SQL commands on the at the call of the required function. The functions have methods for inserting, getting a note by a date ID, deleting a note by a date ID, checking if a date ID exists on the database, updating a note stored in the database, and deleting all notes stored in the database. This is a far better method for note storage than my original method of storing it in a list and provides more options for manipulating the stored notes.

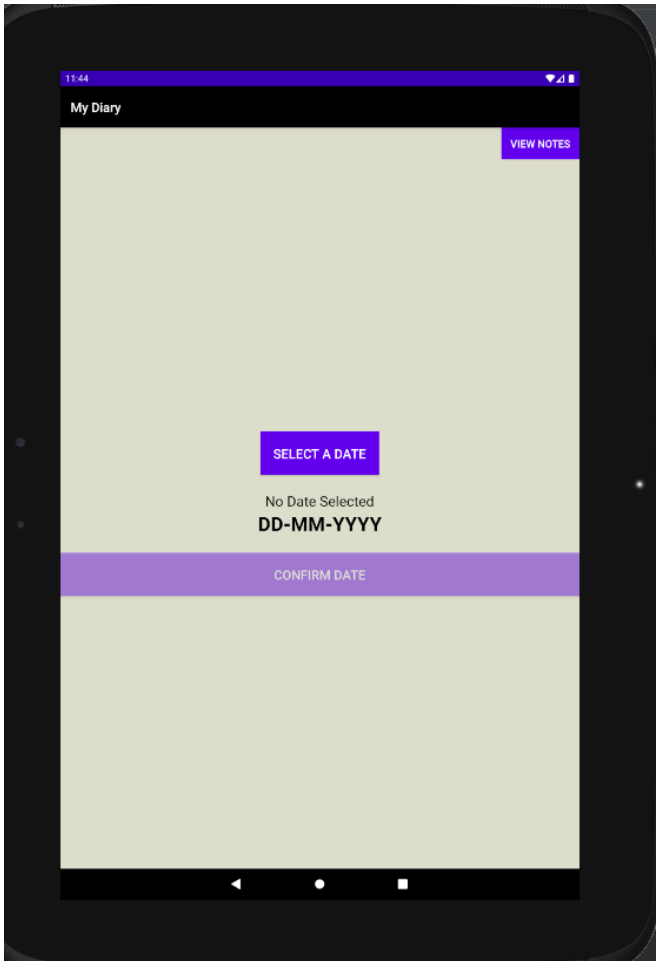

Figure 3: Large Resolution Tablet View

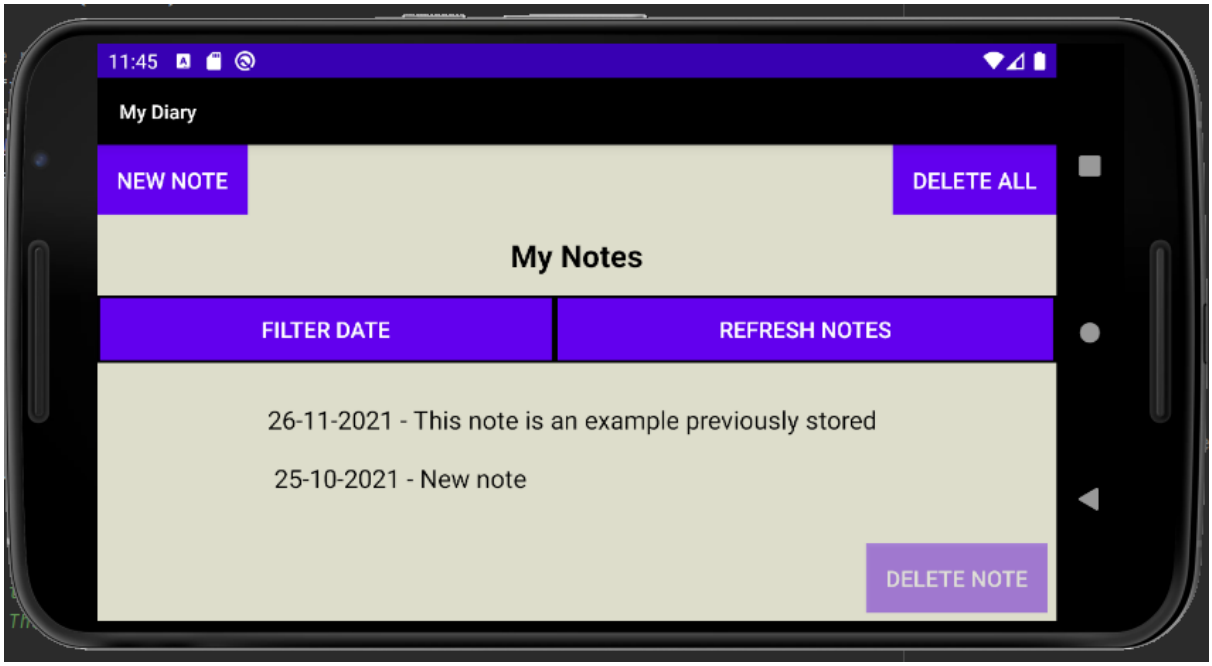

Figure 4: Horizontal Screen View

4 - UI Design

My main priority when creating my UI was to keep it simple and prevent stressing out my users by overloading them with information. The only elements contained within my pages are just what is absolutely required to allow full functionality of the app. It is very apparent to the users the steps they may take. I am very direct with the text contained in the buttons and they inform users exactly what will happen if they click on it. To enhance confirming user actions I added toast messages that will inform users what each button click has done. I have messages that tells users if their actions are successful, what the action has done, if the action is unsuccessful, and why the action was unsuccessful. This really lets the users understand what is going on when using my application. Each page follows a consistent pattern, once the user knows how to use the first page then they will know how to use them all. Every page contains the navigation buttons at the top and the main content requiring input in the centre of the page as to draw attention. Navigation stops users feeling stuck on screen and give the freedom to navigate to any screen they desire. Without them user may be claustrophobic or frustrated at being stuck on a screen. Some buttons are greyed out until they become useful, and the user will only be concerned with them when they are relevant as to avoid giving them too many options at once.

I chose to use dark purple elements containing light white text, on a white background. This helps to inform users of all the available options on the page as the buttons really stand out, and the text can be read after identifying the button. The choice of colour passes colour contrast test for colour blindness [1], meaning that visually impaired users will have no problem using the application. To also aid visually I created a border xml class to wrap around buttons that are next to each other, this can help users distinguish between the buttons. My headers are in bold and are larger than standard text, informing users there is a hierarchy within these texts and what content will be displayed on this page. They will read this header first to understand the page, their content can be viewed in smaller text below the header.

Users begin at the date selection page, that is automatically assigned to the current date as to allow faster note submission. I did not assume the goals the users have when accessing my app, they may not want to add a new note, so I allow them fast access of the view notes page from the first screen. On the second screen I print in bold the date they have selected, and if they realise it is the wrong date they can navigate back to start again.

5 - Reflection

5.1 - Reflection on Solution

On reflection I am very proud of my app and pleased with the result. The process of building this application was quite challenging and really tested my troubleshooting skills, as every little change resulted in hours of solving errors. I believe my app works exceptionally and is fully complete in compliance with the project brief, from the basic features to several advanced features. I found adding the database functionality challenging but was very rewarding after it was fully implemented as it resulted in much better data manipulation. As a result of this I was able to add some extra features such as filtering by date and deleting by date, resulting in better application. After rigorous testing I hope to have solved most of the bugs I could find and at present only one can be found which is discussed in the solution boundaries.

Data passing ended up being quite a challenge and I had a hard time trying to implement a method that worked. Trying multiple different ways ended up with me using two different ways of passing data, when I should have just been consistent with one. I chose two methods as observing live data allowed for instance update of the date on the second screen, whereas using the data passer method users must manually refresh to update the page. I am not very pleased that data needs to be manually refreshed on “viewNotes” however I did not take time to investigate how to improve it due to time constraints, as instant note printing on viewing the page would be better. I also should have changed the names of my fragments from page1,2,3 to something that made more sense, but I left this to the end and worry that changing it will produce errors. I also should have changed the project name but have note due to the same reason. I am happy with my advanced features but the delete buttons can be pressed to easily and would benefit a confirmation window. This is where the user could confirm their action to avoid accidental deletions.

I found it useful to reference the android design principles when developing my app to ensure it would be successful. My layout follows the “keep it brief” principle and does not display any large amount of text that forces the user to read to understand my app. Instead, everything is linear, basic, and as obvious as possible as what the user can do without overloading them with text. Each screen contains short messages, obviously labelled buttons, and small toast messages also confirm user actions. I followed “Only show what I need when I need it” principle with some of the buttons as they will not always be available. Buttons such as “CONFIRM DATE” and “DELETE NOTE” are greyed out and unclickable until a user completes an action that made them available, where they will then be fully coloured and clickable. I am happy with my decision with these as it means the users will not focus on them until the action becomes relevant, also preventing overloading them. The principle “Make important things fast” was considered when creating the screens, so I provided buttons for users to quickly get where they need to go. If they user wants to view their notes on Page3 they may instantly access this page with the “VIEW NOTES” button and can quickly go back to add a note. “Do the heavy lifting for me” was relevant to hide the long process of accessing the database behind every database manipulating button, allowing users to be unaware of how painstaking accessing the room database is.

5.2 - Boundaries of Solution

After much testing I have eliminated a lot of the situations where my app doesn't function correctly, the only one found being that the screen is not fully rotational. When the screen rotates the data is wiped and created again from scratch. I was initially using view models to pass data, and this would have provided methods to make the screen fully rotational, however I changed my data passing method to allow myself to create a database as I assumed this would grant more marks. Due to the time constraints I had to decide my priorities and have not managed to find the time to fully implement screen rotation, but the layout will scale properly to a rotated screen and will look presentable to users.

6 - Test Cases

Test ID	Description	Test steps	Pre-requisites	Expected Results	Pass/Fail	Remarks
SelecteDate_1	Select a date on the first screen	1. Click the select date button 2. User selects date from the calendar pop-up 3. Click ok on calendar	1. User is on the date select screen	The date selected will be displayed on the screen	Pass	Allows user to successfully select a date
ConfirmDate_1	Confirm the date that has been selected	1. Click the confirm date button	1. User is on the date selection screen 2. A date has been selected	Change view to the add note screen, showing the date selected	Pass	User can confirm a date to then add a note to
ConfirmDate_2	Confirm the date when no date has been selected	1. Click the confirm date button	1. User is on the date selection screen 2. No date has been selected	Nothing will happen as the button is only clickable once a date is selected	Pass	To avoid any errors, the confirm button will only be clickable if a date has been selected
Navigation_1	Navigate from date selection screen to view note screen	1. Click the view notes button	1. User is on the date selection screen	Change view to the view notes screen	Pass	User can easily navigate between pages; the date will be held if user decides to navigate back
AddNote_1	Add a note for the selected date	1. Input note text into the text box 2. Click the add note button	1. User is on the add note screen 2. A date has been selected previously	Change view to the view notes screen, with pop-up message informing users that a note for that date has been added	Pass	User can successfully add a note for a selected date

AddNote_2	Add a note for the selected date, where note content is blank	1. Click the add note button	1. User is on the add note screen 2. No text has been entered into the text box	Screen view will not change, pop-up message informs users that text must be entered to add a note	Pass	User will not be able to insert a blank note, as to avoid errors
AddNote_3	Add note for a date that already contains a note stored previously	1. Input note text into text box 2. Click the add note button	1. User is on the add note screen 2. A note has been stored previously against the date currently selected	Change view to the view notes screen, pop-up message informs users that the note for this date has been updated. New note is appended to previous note	Pass	Note gets updated on the database. When the notes are displayed, the new note is a new sentence added to the end of a previous note
ClearNote_1	Clear any text entered in the textbox	1. Click the clear button	1. User is on the add note screen	If there is text in the text box, it will be cleared, and a pop-up message informs users that the note has been cleared	Pass	Text in the text box is cleared
Navigation_2	Navigate from add note screen back to select date screen	1. Click the back button	1. User is on the add note screen	Change view back to the add date screen, previously selected date will be cleared and will ask user to select a date	Pass	User navigates back to the date selection page, date selected is wiped so users can select a new date

ViewNotes_1	View the notes stored in the database	1. Click the refresh notes button	1. User is on the view note screen 2. Notes exist in the database	Pop-up message tells user the notes are refreshing, all the stored notes will then be displayed in the page, showing what date they were stored	Pass	User can successfully display all the notes
ViewNotes_2	Try to view notes when the database is empty	1. Click the refresh notes button	1. User is on the view note screen 2. No notes exist in the database	Message in the centre of the page will inform user that no notes have been found	Pass	The note text box displays a suitable message
DeleteAll_1	Delete all the notes stored in the database	1. Click the delete all button	1. User is on the view note screen	Centre text will display that all notes have been deleted, pop-up message will also confirm user action. Database will be empty.	Pass	Empty database is confirmed when clicking the refresh button, centre message will display that no notes have been found
Navigation_3	Navigate from view note screen back to select date screen	1. Click the new note button	1. User is on the view note screen	Change view back to the add date screen, with no date selected	Pass	Navigates user back to date selection to add a new note
FilterDate_1	Find a note for a specified date where the note exists	1. Click the filter date button 2. Select a date to display a note from	1. User is on the view note screen 2. A note exists against the date selected	Centre text will display the note stored for that date; pop-up message will confirm that note for this date has been found	Pass	Users can successfully search for a note from a selected date

FilterDate_2	Finding a note for a specified date where the note does not exist	1. Click the filter date button 2. Select a date	1. User is on the view note screen 2. A note does not exist against the date selected	Centre text will display that no notes are found, pop-up message confirms that no note for the selected date has been found	Pass	No note will be found, and suitable messages are displayed
DeleteNote_1	Delete a note that has been found by date	1. Click the delete note button	1. User is on the view note screen 2. User has found a note stored for a selected date, as in FilterDate_1	Centre text will display that no notes have been found, pop-up text will confirm that note for selected date has been deleted	Pass	User can successfully delete a note, which will not affect any other note in the database
DeleteNote_2	Delete a note when no notes are selected	1. Click the delete note button	1. User is on the view note screen 2. No note has been found for a selected date	Nothing will happen as the delete button is only clickable after a note has been found	Pass	Delete button is only clickable when a note is found, so the program knows what to delete
ChangeRotation_1	Change the orientation of the screen to view on horizontal mode	1. Change the orientation of the phone when using the app.	1. Some notes have been previously stored	The screen will rotate the user can use the application on horizontal mode, while keeping all their stored notes	Fail	The data is wiped on screen rotation and all notes stored previously are lost

7 - Code

MainActivity

```
package uk.ac.stir.cs.fragments

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.widget.Toolbar
import androidx.viewpager2.widget.ViewPager2
import java.util.*

class MainActivity : AppCompatActivity(), OnDataPass{
    private lateinit var viewPager : ViewPager2  //ViewPager for changing
views

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //Toolbar at top of page
        val toolbar = findViewById<Toolbar>(R.id.toolbar)
        setSupportActionBar(toolbar)

        //Setting the number of tabs and viewpager adapter
        viewPager = findViewById<ViewPager2>(R.id.pager)
        //Stop user from swiping to other fragments
        viewPager.isUserInputEnabled = false
        val adapter = MyPageAdapter(this,3)
        viewPager.adapter = adapter

        //REMOVE THE REST OF THIS METHOD TO MAKE ORIENTATION CHANGE WORK
CORRECTLY
       //Creating an example note
        val d1="26-11-2021"
        val n1="This note is an example previously stored"
        val n2 = Note(d1, n1)

        //Clearing database then adding the example note

NoteRoomDatabase.getInstance(this@MainActivity).noteDao().nukeTable()
//Wipe the database on start up, used for debugging

NoteRoomDatabase.getInstance(this@MainActivity).noteDao().insert(n2)
//Example note added
    }

    /**
     * Function used to change the fragment currently being viewed, to move
to next fragment
     * @param page - The page number to be switched too
     */
    override fun onDataPass(page: Int) {
        viewPager.currentItem = page
    }

    /**
```

```
     * Function for adding a new note to the database
     * @param date - The date selected to add note to
     * @param note - The note body being added
     */
    override fun newNote(date: String, note: String) {
        val newN = Note(date, note) //New note object
        //Check if a note for date selected already exists
        if(!containsPrimaryKey(date)){
            //If note doesn't exist, create new note and insert it to
database, then display toast message

NoteRoomDatabase.getInstance(this@MainActivity).noteDao().insert(newN)
            Toast.makeText(this@MainActivity, "New note for $date added,
please refresh note list", Toast.LENGTH_LONG).show()
        }
        else {
            //If note for this date already exists, then add to existing
note
            val noteFound =
NoteRoomDatabase.getInstance(this@MainActivity).noteDao().loadAllByIds(date
)
            //Remove the date from the not found, or else note toString
would contain 2 dates
            val notesString =
noteFound.toString().drop(date.length+3).replace("[", "").replace("]",
"").replace(",","").trim()
            val updatedNote = ("$notesString. $note")

NoteRoomDatabase.getInstance(this@MainActivity).noteDao().updateNote(date,
updatedNote)
            Toast.makeText(this@MainActivity, "Note for $date already
exists, adding to your note ...", Toast.LENGTH_LONG).show()
        }
    }

    /**
     * Function to delete all entries in database, just helpful for
debugging
     */
    override fun onDeletion() {

NoteRoomDatabase.getInstance(this@MainActivity).noteDao().nukeTable()
    }

    /**
     * Function to get all entries in the database
     * @return List<Note> - List of notes stored
     */
    override fun getNotes(): List<Note> {
        return
NoteRoomDatabase.getInstance(this@MainActivity).noteDao().getAll()
    }

    override fun deleteNote(date_id: String) {

NoteRoomDatabase.getInstance(this@MainActivity).noteDao().deleteNote(date_i
d)
    }

    /**
     * Function to get all entries in the database
```

```
     * @return List<Note> - List of notes stored
     */
    override fun updateNote(date:String, note:String) {
        return
NoteRoomDatabase.getInstance(this@MainActivity).noteDao().updateNote(date,
note)
    }

    /**
     * Function to delete a note by selected ID
     */
    override fun loadAllByIds(date: String): List<Note> {
        return
NoteRoomDatabase.getInstance(this@MainActivity).noteDao().loadAllByIds(date
)
    }

    /**
     * Function to check if an entry for selected date already exists in
database
     * @return pKey - True if key is found, false if no key found
     */
    override fun containsPrimaryKey(date_id:String): Boolean {
        return
NoteRoomDatabase.getInstance(this@MainActivity).noteDao().containsPrimaryKe
y(date_id)
    }


}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
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"
    tools:context=".MainActivity">
    <!-- Vertical Linear Layout-->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#DDDDCB"
        android:orientation="vertical">

        <!--Toolbar at top of the page displaying title of the app-->
        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="?attr/colorPrimary"
            android:elevation="6dp"
            app:layout_constraintBottom_toTopOf="@+id/tab_layout"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <!--For transitioning between fragments-->
        <androidx.viewpager2.widget.ViewPager2
            android:id="@+id/pager"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="@+id/tab_layout"
            app:layout_constraintTop_toBottomOf="@+id/tab_layout" />

    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

MyPageAdapter

```
package uk.ac.stir.cs.fragments

import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentActivity
import androidx.viewpager2.adapter.FragmentStateAdapter
class MyPageAdapter(fa: FragmentActivity, private val mNumOfTabs: Int) :
    FragmentStateAdapter(fa) {
    /**
     * Getting the number of tabs
     */
    override fun getItemCount(): Int {
        return mNumOfTabs
    }
    /**
     * Creating fragments with positions
     */
    override fun createFragment(position: Int): Fragment {
        return when (position) {
            0 -> Page1()
            1 -> page2()
            2 -> page3()
            else -> Page1()
        }
    }
}
```

Page1

```
package uk.ac.stir.cs.fragments

import android.app.DatePickerDialog
import android.content.Context
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.DatePicker
import android.widget.Toast
import androidx.lifecycle.ViewModelProvider
import uk.ac.stir.cs.fragments.databinding.FragmentPage1Binding
import java.util.*


class Page1 : Fragment(R.layout.fragment_page1),
DatePickerDialog.OnDateSetListener{

    //Binding for fragment, to allow access to elements
    private var _binding : FragmentPage1Binding? = null
    private val binding get() = _binding!!

    //Variables for the dates
    var day = 0
    var month = 0
    var year = 0

    //Saved values assigned when date is selected on pop up
    var savedDay = 0
    var savedMonth = 0
    var savedYear = 0

    //ViewModel to insert date
    lateinit var model: SharedViewModel
    //dataPasser to pass data to main
    lateinit var dataPasser: OnDataPass

    override fun onCreateView(inflater: LayoutInflater, container:
ViewGroup?, savedInstanceState: Bundle?): View {
        //Inflate the layout for this fragment
        _binding = FragmentPage1Binding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        //Instantiate the ViewModel
        model =
ViewModelProvider(requireActivity())[SharedViewModel::class.java]

        //Confirm date cannot be clicked until date is selected
        binding.btnConfirm.alpha = .4f
        binding.btnConfirm.isClickable = false

        //Get calendar and allow user to select a date
        binding.btnDatePicker.setOnClickListener {
            //get current date
```

```
            getDateCalendar()
            //Display calender to pick date, date already selected is
current date
            DatePickerDialog(requireContext(), this, year, month,
day).show()
            //To handle confirm button
            pickDate()
        }
        //View notes button changes view to page 3 where the notes are
displayed
        binding.btnViewNotes.setOnClickListener {
            passData(2)
        }
    }

    /**
     *Function to assign global date variables to current date
     */
    private fun getDateCalendar(){
        val cal = Calendar.getInstance()
        day = cal.get(Calendar.DAY_OF_MONTH)
        month = cal.get(Calendar.MONTH)
        year = cal.get(Calendar.YEAR)
    }

    /**
     *Function to confirm date selection
     */
    private fun pickDate(){
            binding.btnConfirm.setOnClickListener {
                //Save date to ViewModel, to then be displayed on page 2
                model.saveDate(savedDay, savedMonth, savedYear)

                //Set confirm button to unable to edit again
                binding.btnConfirm.alpha = .4f
                binding.btnConfirm.isClickable = false

                //Clear the save date
                savedDay=0
                savedMonth=0
                savedYear=0

                //Back to default values and move to page 2
                binding.txtDate.text="DD-MM-YY"
                binding.txtReturn.text="No Date Selected"
                passData(1)
            }
    }

    /**
     * Function to destroy binding when activity ends
     */
    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }

    /**
     * Function to attach dataPasser to this context
     */
    override fun onAttach(context: Context) {
```

```
        super.onAttach(context)
        dataPasser = context as OnDataPass
    }

    /**
     * Passes data (screen to change to) to the dataPasser, within override
functions in main
     */
    private fun passData(data: Int){
        dataPasser.onDataPass(data)
    }

    /**
     * Function that runs when the date is selected
     *
     * @param p0 - The datepicker
     * @param year - Year selected
     * @param month - Year selected
     * @param dayOfMonth - Year selected
     */
    override fun onDateSet(p0: DatePicker?, year: Int, month: Int,
dayOfMonth: Int) {
        //Set the dates selected to the saved date variables
        savedDay = dayOfMonth
        savedMonth = month
        savedYear = year

        //Makes confirm button unclickable again
        binding.btnConfirm.alpha = 1f;
        binding.btnConfirm.isClickable = true;

        //Display the date that has been entered
        binding.txtReturn.text="Date Selected"
        binding.txtDate.text = "$savedDay-$savedMonth-$savedYear"
    }
}
```

fragment_page1.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:gravity="center"
    android:orientation="vertical"
    tools:context=".Page1">

    <!--Horizontal Linear Layout at top of the page, to display the
viewNotes button on top right-->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="right"
        android:orientation="horizontal">

        <!--Button to change view to page3-->
        <Button
            android:id="@+id/btn_viewNotes"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="14dp"
            android:textSize="16sp"
            android:textColor="@color/white"
            android:background="@color/purple_500"
            android:text="View Notes" />
    </LinearLayout>

    <!--Vertical Linear Layout at middle of the page, to display the date
buttons and text-->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:orientation="vertical">

        <!--Date picker button-->
        <Button
            android:id="@+id/btn_datePicker"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="@color/purple_500"
            android:padding="20dp"
            android:text="Select a date"
            android:textColor="@color/white"
            android:textSize="20dp" />

        <!--Display returned date-->
        <TextView
            android:id="@+id/txt_return"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:paddingTop="25dp"
            android:text="No Date Selected"
            android:textAlignment="center"
            android:textColor="@color/black"
```

```
            android:textSize="22sp" />

        <!--TextView intro and conformation text-->
        <TextView
            android:id="@+id/txt_date"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:paddingBottom="25dp"
            android:text="DD-MM-YYYY"
            android:textAlignment="center"
            android:textColor="@color/black"
            android:textSize="30sp"
            android:textStyle="bold" />

        <!--Button to confirm selected date-->
        <Button
            android:id="@+id/btn_confirm"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="@color/purple_500"
            android:padding="20dp"
            android:text="Confirm Date"
            android:textColor="@color/white"
            android:textSize="20dp" />
    </LinearLayout>

</LinearLayout>
```

Page2

```
package uk.ac.stir.cs.fragments

import android.content.Context
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProvider
import uk.ac.stir.cs.fragments.databinding.FragmentPage2Binding


class page2 : Fragment() {

    //Binding, viewModel and dataPasser variables
    private var _binding : FragmentPage2Binding? = null
    private val binding get() = _binding!!
    lateinit var model: SharedViewModel
    lateinit var dataPasser: OnDataPass

    override fun onCreateView(inflater: LayoutInflater, container:
ViewGroup?, savedInstanceState: Bundle?): View {
        //Inflate the layout for this fragment
        _binding = FragmentPage2Binding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        model =
ViewModelProvider(requireActivity())[SharedViewModel::class.java]

        //Observe the LiveData variable date from the viewModel
        model.date.observe(viewLifecycleOwner, Observer {
            //Display observed date
            binding.txtDate.text= it.toString()
        })

        binding.btnAddNote.setOnClickListener {//Add note button
            //Check if any text has been input to the textInput
            if (binding.txtNote.text.isEmpty()) { //If no input
                //Toast popup message to advise users
                Toast.makeText(activity, "Your note cannot be empty!",
Toast.LENGTH_LONG).show()

            }else{ //If there is an input
                //Pass date and note input to newNote class
                newNote(binding.txtDate.text.toString(),
binding.txtNote.text.toString())
                binding.txtNote.setText("")
                //Change view to page 3 where notes are displayed
                passData(2)
            }
        }
        binding.btnBack.setOnClickListener {//Back button
            //Clear the text inputted and change view to page 1
```

```
            binding.txtNote.setText("")
            passData(0)
        }
        binding.btnClearNote.setOnClickListener {//Clear button
            //Clear the text inputted and display toast message informing
            binding.txtNote.setText("")
            Toast.makeText(activity, "Note Cleared",
Toast.LENGTH_LONG).show()
        }
    }

    /**
     * Function to destroy binding when activity ends
     */
    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }

    /**
     * Function to attach dataPasser to this context
     */
    override fun onAttach(context: Context) {
        super.onAttach(context)
        dataPasser = context as OnDataPass
    }

    /**
     * Passes data (screen to change to) to the dataPasser, within override
functions in main
     */
    private fun passData(data: Int){
        dataPasser.onDataPass(data)
    }

    /**
     * Function to add the note, passes it to main activity and is then
added to database
     * @param date - Date selected
     * @param note - Note entered
     */
    private fun newNote(date:String, note:String){
        dataPasser.newNote(date, note)
    }

}
```

fragment_page2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".page2">

    <!--Horizontal Linear Layout at top of the page, to display the back
button on top left-->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <!--Back to page1-->
        <Button
            android:id="@+id/btn_back"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="@color/purple_500"
            android:textColor="@color/white"
            android:textSize="16sp"
            android:padding="14dp"
            android:text="Back" />
    </LinearLayout>

    <!--Vertical Linear Layout at center of page to display date and input
textbox-->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:orientation="vertical">

        <!--Text to display date-->
        <TextView
            android:id="@+id/txt_date"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:paddingBottom="25dp"
            android:text="DD-MM-YYYY"
            android:textAlignment="center"
            android:textColor="@color/black"
            android:textSize="25sp"
            android:textStyle="bold" />

        <!--Text tells users to add a note-->
        <TextView
            android:id="@+id/txt_info"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:paddingBottom="20dp"
            android:text="Please add your note"
            android:textAlignment="center"
            android:textColor="@color/black"
            android:textSize="20sp" />
```

```
        <!--Text to input the note, implements autocorrect and auto
capitalization-->
        <EditText
            android:id="@+id/txt_note"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:gravity="start|top"

android:inputType="textMultiLine|textCapSentences|textAutoCorrect" />

        <!--Just to create a space-->
        <TextView
            android:id="@+id/txt_noteConf"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="" />


        <!--Horizontal Linear Layout for the buttons to display next to
each other-->
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal">

            <!--Clear text box button-->
            <Button
                android:id="@+id/btn_clearNote"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:background="@drawable/my_border"
                android:text="Clear"
                android:textColor="@color/white"
                android:textSize="20dp"
                android:layout_weight="1"/>

            <!--Add note button-->
            <Button
                android:id="@+id/btn_addNote"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:background="@drawable/my_border"
                android:text="Add Note"
                android:textColor="@color/white"
                android:textSize="20dp"
                android:layout_weight="1"/>
        </LinearLayout>

    </LinearLayout>

</LinearLayout>
```

Page3

```
package uk.ac.stir.cs.fragments

import android.content.Context
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProvider
import uk.ac.stir.cs.fragments.databinding.FragmentPage2Binding


class page2 : Fragment() {

    //Binding, viewModel and dataPasser variables
    private var _binding : FragmentPage2Binding? = null
    private val binding get() = _binding!!
    lateinit var model: SharedViewModel
    lateinit var dataPasser: OnDataPass

    override fun onCreateView(inflater: LayoutInflater, container:
ViewGroup?, savedInstanceState: Bundle?): View {
        //Inflate the layout for this fragment
        _binding = FragmentPage2Binding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        model =
ViewModelProvider(requireActivity())[SharedViewModel::class.java]

        //Observe the LiveData variable date from the viewModel
        model.date.observe(viewLifecycleOwner, Observer {
            //Display observed date
            binding.txtDate.text= it.toString()
        })

        binding.btnAddNote.setOnClickListener {//Add note button
            //Check if any text has been input to the textInput
            if (binding.txtNote.text.isEmpty()) { //If no input
                //Toast popup message to advise users
                Toast.makeText(activity, "Your note cannot be empty!",
Toast.LENGTH_LONG).show()

            }else{ //If there is an input
                //Pass date and note input to newNote class
                newNote(binding.txtDate.text.toString(),
binding.txtNote.text.toString())
                binding.txtNote.setText("")
                //Change view to page 3 where notes are displayed
                passData(2)
            }
        }
        binding.btnBack.setOnClickListener {//Back button
            //Clear the text inputted and change view to page 1
```

```
            binding.txtNote.setText("")
            passData(0)
        }
        binding.btnClearNote.setOnClickListener {//Clear button
            //Clear the text inputted and display toast message informing
            binding.txtNote.setText("")
            Toast.makeText(activity, "Note Cleared",
Toast.LENGTH_LONG).show()
        }
    }

    /**
     * Function to destroy binding when activity ends
     */
    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }

    /**
     * Function to attach dataPasser to this context
     */
    override fun onAttach(context: Context) {
        super.onAttach(context)
        dataPasser = context as OnDataPass
    }

    /**
     * Passes data (screen to change to) to the dataPasser, within override
functions in main
     */
    private fun passData(data: Int){
        dataPasser.onDataPass(data)
    }

    /**
     * Function to add the note, passes it to main activity and is then
added to database
     * @param date - Date selected
     * @param note - Note entered
     */
    private fun newNote(date:String, note:String){
        dataPasser.newNote(date, note)
    }

}
```

fragment_page3.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    tools:context=".page3">

    <!--Vertical Linear Layout for top of the page to hold buttons-->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <!--Relative Layout to display buttons on either side of page-->
        <RelativeLayout
            android:id="@+id/relativeLayout1"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1">

            <!--Button to go back to page1-->
            <Button
                android:id="@+id/btn_newNote"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:padding="14dp"
                android:layout_alignParentStart="true"
                android:background="@color/purple_500"
                android:text="New Note"
                android:textColor="@color/white"
                android:textSize="16dp" />

            <!--Button to delete all stored notes-->
            <Button
                android:id="@+id/btn_clearNotes"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentEnd="true"
                android:background="@color/purple_500"
                android:padding="14dp"
                android:text="Delete All"
                android:textColor="@color/white"
                android:textSize="16dp" />

        </RelativeLayout>

        <!--TextView to display heading on page-->
        <TextView
            android:id="@+id/txt_myNotes"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="14dp"
            android:text="My Notes"
            android:textStyle="bold"
            android:textAlignment="center"
            android:textColor="@color/black"
            android:textSize="22sp" />
    </LinearLayout>
```

```
    <!--Horizontal Linear Layout to hold filter and refresh button side by
side-->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <!--Filter note by date button, both buttons use my border xml
class to make seperation of buttons clear-->
        <Button
            android:id="@+id/btn_filter"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="@drawable/my_border"
            android:text="Filter date"
            android:textColor="@color/white"
            android:textSize="16dp"
            android:layout_weight="1"/>

        <!--Refresh notes button to display the stored notes-->
        <Button
            android:id="@+id/btn_refresh"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="@drawable/my_border"
            android:text="Refresh Notes"
            android:textColor="@color/white"
            android:textSize="16dp"
            android:layout_weight="1"/>
    </LinearLayout>

    <!--Linear Layout to hold TextView that displays notes in centre of
page-->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:layout_gravity="bottom"
        android:layout_weight="1">

        <!--TextView to display notes-->
        <TextView
            android:id="@+id/txt_notes"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="left"
            android:padding="20dp"
            android:scrollbars="vertical"
            android:text="No Notes Found"
            android:textColor="@color/black"
            android:textSize="18dp" />
    </LinearLayout>

    <!--Vertical Linear Layout to hold delete button at bottom right of
page-->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="6dp"
```

```
        android:layout_gravity="bottom"
        android:layout_weight="0">

        <!--Button for deleting filtered notes-->
        <Button
            android:id="@+id/btn_deleteDate"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="@color/purple_500"
            android:padding="14dp"
            android:layout_gravity="right"
            android:text="Delete Note"
            android:textColor="@color/white"
            android:textSize="16dp" />
    </LinearLayout>

</LinearLayout>
```

my_border.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- For buttons that are next to each other, add a border around them-->
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <solid
        android:color="#6200EE">
    </solid>

    <!-- Border-->
    <stroke
        android:width="2dp"
        android:color="@color/black">
    </stroke>
</shape>
```

fragmentViewModel

```
package uk.ac.stir.cs.fragments

import androidx.lifecycle.LiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.MutableLiveData

/**
 * ViewModel class used for storing and observing the date from page 1
fragment to page 2 using LiveData
 */
class SharedViewModel: ViewModel() {

    //LiveData varaible of the date, used to be observed by page 2 fragment
and display the date
    private var _date = MutableLiveData("")
    val date: LiveData<String> = _date

    /**
     * Function for saving the data in a suitable format to the liveData
variable
     * @param newDay - Day being added
     * @param newMonth - Month being added
     * @param newYear - Year being added
     */
    fun saveDate(newDay: Int, newMonth:Int, newYear:Int){
        _date.value = ("$newDay-$newMonth-$newYear")
    }
}
```

Note

```
package uk.ac.stir.cs.fragments

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

/**
 * An entity class for the note, used to instantiate a note and then store
it
 * @param date - The date for the note, used as a primary key
 * @param noteBody - The note body stored
 */
@Entity
data class Note(
    @PrimaryKey val date: String,
    @ColumnInfo(name = "note_body") val noteBody: String?
)

/**
 * toString method used for when printing the dates and notes
 * @return - String suitable for printing
 */
{
    override fun toString(): String = "$date - $noteBody\n\n"

}
```

NoteDao

```
package uk.ac.stir.cs.fragments

import androidx.room.*

/**
 * Database Access Object for database interactions
 */
@Dao
interface NoteDao{

    //Query to get list of all notes
    @Query("SELECT * FROM note")
    fun getAll(): List<Note>

    //For inserting a new note
    @Insert
    fun insert(note:Note)

    //Getting note from a selected date
    @Query("SELECT * FROM note WHERE date IN (:date_id)")
    fun loadAllByIds(date_id: String): List<Note>

    //Used for finding note in database by the note body. Not useful might
remove?
    @Query("SELECT * FROM note WHERE note_body LIKE :noteBody LIMIT 1")
    fun findByDate(noteBody:String): List<Note>

    //Delete note from database
    @Query("DELETE FROM note WHERE date = :date_id")
    fun deleteNote(date_id: String);

    //Query to find a note under a specified date
    @Query("SELECT count(*)!=0 FROM note WHERE date IN (:date_id)")
    fun containsPrimaryKey(date_id: String): Boolean

    @Query("UPDATE note SET note_Body=:newNote WHERE date=:dateID")
    fun updateNote(dateID: String, newNote: String)

    //Delete all values stored in database
    @Query("DELETE FROM note")
    fun nukeTable()
}
```

NoteRoomDatabase

```
package uk.ac.stir.cs.fragments

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

/**
 * Database used for storing notes
 */
@Database(entities = [Note::class], version = 1)
abstract class NoteRoomDatabase : RoomDatabase() {
    abstract fun noteDao(): NoteDao //Database Access Object for database
interactions
    companion object{
        var INSTANCE: NoteRoomDatabase?=null
        fun getInstance(context: Context):NoteRoomDatabase
        {
            if(INSTANCE==null){ //If an instance of this class doesn't
already exist
                INSTANCE = Room.databaseBuilder(
                    context.applicationContext,
                    NoteRoomDatabase::class.java, "note_database"   //Name
of database
                ).allowMainThreadQueries().build()
            }
            return INSTANCE!!   //Return this instance
        }
    }
}
```

OnDataPass

```
package uk.ac.stir.cs.fragments

/**
 * Inteface for passing data, these methods are overwritten in main
 */
interface OnDataPass {
        fun onDataPass(page: Int)

        fun newNote(date: String, note: String)

        fun onDeletion()

        fun getNotes():List<Note>

        fun deleteNote(date_id:String)

        fun containsPrimaryKey(date_id:String):Boolean

        fun updateNote(date:String, note:String)

        fun loadAllByIds(date:String): List<Note>

}
```

References

[1] Dequeuniversity.com. 2021. *Check Text and Background for Sufficient Color Contrast | Accessibility Tips*. [online] Available at: https://dequeuniversity.com/tips/color-contrast [Accessed 15 November 2021].

stuff.mit.edu. (n.d.). *Application Structure | Android Developers*. [online] Available at: https://stuff.mit.edu/afs/sipb/project/android/docs/design/patterns/app-structure.html [Accessed 18 Nov. 2021].

Android Developers. (n.d.). *Slide between fragments using ViewPager*. [online] Available at: https://developer.android.com/training/animation/screen-slide [Accessed 20 Nov. 2021].

raywenderlich.com. (n.d.). *Android Fragments Tutorial: An Introduction with Kotlin*. [online] Available at: https://www.raywenderlich.com/1364094-android-fragments-tutorial-an-introduction-with-kotlin Accessed 20 Nov. 2021].

Sanson, J. (2020). *Handling Lifecycle with View Binding in Fragments*. [online] Default to Open. Available at: https://medium.com/default-to-open/handling-lifecycle-with-view-binding-in-fragments-a7f237c56832 [Accessed 24 Nov. 2021].

www.freelancer.com. (2015). *5 Key Android Design Principles | Freelancer Blog*. [online] Available at: https://www.freelancer.com/community/articles/5-key-android-design-principles [Accessed 24 Nov. 2021].