KAZAKH BRITISH TECHNICAL UNIVERSITY

Assignment 3

Fragments, RecyclerView, and ViewModel in Android Development.

Prepared by Azimkhanov Bauyrzhan

Almaty, 10.11.2024

Table of contents

Introduction

Modern mobile apps need to handle complex interactions, persist data, and have responsive interfaces. Fragments, RecyclerView, and ViewModel play an important role in creating these features. Fragments provide modularity in design.

RecyclerView efficiently manages large data sets, and ViewModel. LiveData, ensures data persistence across configuration changes.

The goal of this assignment was to introduce these components, reinforcing their importance in developing efficient and responsive Android apps.

Exercise descriptions

Exercise 1: Creating a Basic Fragment

Objective:

1. Create a new Fragment that displays a simple message ("Hello from Fragment!").
2. Implement the fragment lifecycle methods (onCreateView, onStart, onResume, onPause, onStop, onDestroyView) to log lifecycle events.

Description of the implementation steps:

1. Created a new Fragment class with a layout file to display “Hello from Fragment!” as a TextView.
2. Overrode lifecycle methods (onCreateView, onStart, onResume, onPause, onStop, onDestroyView) within the Fragment class.
3. Added log statements in each lifecycle method to track the Fragment’s state changes in the logcat.
4. Added the Fragment to an Activity, either programmatically or via XML, to display it in the app.

Expected outcome: Fragment shows the message, and lifecycle events are logged as the Fragment transitions through states.

Exercise 2: Fragment Communication

Objective:

1. Create two Fragments: one for input (EditText) and one for output (TextView).
2. Implement communication between these Fragments using a shared ViewModel to display the input text in the output Fragment.

Description of the implementation steps:

1. Created two Fragments, one containing an EditText for input and the other a TextView for displaying the input text.
2. Set up a shared ViewModel to manage data between the Fragments.
3. In the first Fragment, bound the EditText to an input field in the ViewModel, observing text changes to update the ViewModel.
4. In the second Fragment, observed the ViewModel data and updated the TextView whenever the input text changed.
5. Tested real-time communication by typing in the EditText and verifying the text appeared in the second Fragment’s TextView.

Expected outcome: Real-time updates in the output Fragment when the input in the first Fragment changes.

Exercise 3: Fragment Transactions

Objective:

1. Design an Activity that hosts two Fragments.
2. Implement buttons to switch between the two Fragments using Fragment transactions (add, replace, remove).

Description of the implementation steps:

1. Designed an Activity layout containing a container (FrameLayout) to host Fragments.
2. Implemented two Fragments with distinct content.
3. Created buttons in the Activity to add, replace, and remove Fragments.
4. Set up FragmentTransaction logic to switch between Fragments when buttons are clicked.
5. Verified that the Activity could dynamically switch between Fragments with smooth transitions.

Expected outcome: The Activity can dynamically switch between two Fragments with smooth transitions.

Exercise 4: Building a RecyclerView

Objective:

1. Create a RecyclerView that displays a list of items (e.g., a list of favorite movies).
2. Implement a basic Adapter to populate the RecyclerView with data.

Description of the implementation steps:

1. Defined an XML layout for the RecyclerView in the Activity or Fragment.
2. Created a custom layout file for individual list items (e.g., TextView to display movie titles).
3. Set up a RecyclerView Adapter to bind a list of items (e.g., favorite movies) to the list item layout.
4. Initialized the RecyclerView in the Activity or Fragment and set its layout manager (e.g., LinearLayoutManager).
5. Connected the Adapter to the RecyclerView to populate it with data.

Expected outcome: RecyclerView displays the list of items, scrolling smoothly.

Exercise 5: Item Click Handling

Objective:

1. Extend the RecyclerView from Exercise 4 to handle item clicks.
2. Display a Toast message showing the clicked item's name.

Description of the implementation steps:

1. Modified the Adapter to accept a click listener for each item.
2. Set up the click listener within the onBindViewHolder method, displaying a Toast message with the clicked item’s name.
3. Verified the click handling by clicking each item to see the Toast message.

Expected outcome: A Toast message displays the name of each clicked item in the RecyclerView.

Exercise 6: ViewHolder Pattern

Objective:

1. Implement a ViewHolder class within your Adapter for the RecyclerView.
2. Optimize your RecyclerView implementation by applying the ViewHolder pattern properly.

Description of the implementation steps:

1. Created an inner ViewHolder class within the Adapter, referencing views from the list item layout.
2. Initialized the views within the ViewHolder class to reduce repetitive findViewById calls.
3. Modified the Adapter’s onBindViewHolder to reuse the ViewHolder objects.
4. Tested the RecyclerView to ensure optimized performance with smoother scrolling.

Expected outcome: Optimized performance and smoother scrolling within the RecyclerView.

Exercise 7: Implementing ViewModel

Objective:

1. Create a ViewModel that stores a list of items (e.g., a list of users).
2. Observe LiveData from the ViewModel in your Activity or Fragment to update the UI when the data changes.

Description of the implementation steps:

1. Created a ViewModel class with a LiveData list of users or other data.
2. Used MutableLiveData within the ViewModel to allow changes to the data.
3. Observed the LiveData from an Activity or Fragment, updating the UI whenever the data changed.
4. Tested the setup by modifying the list in the ViewModel and observing the UI updates.

Expected outcome: UI updates automatically when data in the ViewModel changes.

Exercise 8: MutableLiveData for Input Handling

Objective:

1. Extend your ViewModel from Exercise 7 to handle user input via MutableLiveData.
2. Implement an input field in your UI that updates the ViewModel, and observe changes to reflect them in the UI.

Description of the implementation steps:

1. Extended the ViewModel from Exercise 7 to include a MutableLiveData variable for user input.
2. Set up an EditText in the Activity or Fragment to capture user input and update the ViewModel’s input data.
3. Observed the input MutableLiveData in the Activity or Fragment, updating UI components to display changes.
4. Tested by entering text in the EditText and verifying that the displayed text updated automatically.

Expected outcome: Real-time reflection of user input in the UI.

Exercise 9: Data Persistence
Objective:

1. Create a ViewModel that retrieves data from a local database (e.g., Room).
2. Use LiveData to observe changes in the database and update the UI accordingly.

Description of the implementation steps:

1. Created a Room database, defining an entity and DAO for storing items in a local SQLite database.
2. Extended the ViewModel to fetch data from the Room database, observing it with LiveData.
3. Set up the UI to observe the ViewModel, reflecting changes when the database data was updated.
4. Tested data persistence by modifying data in Room and verifying that the UI displayed the updated data.

Expected outcome: UI displays updated data as changes occur in the Room database.

Code snippets

In the previous sections, I covered the implementation details for each exercise. Here, I'd like to highlight the creation of a RecyclerView and the use of the ViewHolder pattern, which optimizes the performance of the RecyclerView by reducing the need to call findViewById repeatedly. This part of the exercise was critical to ensuring smooth scrolling performance and efficient data binding.

To start, I created an Adapter and a ViewHolder class to manage a list of items (like favorite movies). In the Adapter, the ViewHolder pattern allows us to reuse views and avoid redundant view lookups, especially when working with large data sets.

The code for the Adapter is shown below. Next the ViewHolder implementation is shown.

```kotlin
class MovieAdapter(private val movieList: List<String>, private val
clickListener: (String) -> Unit) :
    RecyclerView.Adapter<MovieAdapter.MovieViewHolder>() {

    // ViewHolder class to cache views for each list item
    class MovieViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val movieTitle: TextView = view.findViewById(R.id.movie_title)

        fun bind(movie: String, clickListener: (String) -> Unit) {
            movieTitle.text = movie
            itemView.setOnClickListener { clickListener(movie) }
        }
    }

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

    override fun onBindViewHolder(holder: MovieViewHolder, position: Int) {
        holder.bind(movieList[position], clickListener)
    }

    override fun getItemCount() = movieList.size
}
```

Figure 1. MovieAdapter class implementing the ViewHolder pattern.

The MovieViewHolder class caches a TextView to display each movie title, which improves performance by minimizing findViewById calls. The bind() method in the MovieViewHolder class sets the movie title text and attaches an onClick listener to each item, displaying a Toast message with the movie title when clicked.

```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="wrap_content"
    android:layout_marginBottom="0sp"> <!--50sp-->

    <TextView
        android:id="@+id/username_item_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="username"
        android:textStyle="bold|italic"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintBottom_toTopOf="@+id/image_item_image_view"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintVertical_bias="0.5"/>

    <TextView
        android:id="@+id/movie_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:textSize="18sp"
        android:textColor="@android:color/black"
        />
</androidx.constraintlayout.widget.ConstraintLayout>
```
Figure 2. Layout for individual RecyclerView items.

This setup for RecyclerView demonstrates using a custom adapter with the ViewHolder pattern to provide efficient scrolling performance. By binding data in the onBindViewHolder method and caching views in the ViewHolder, we reduce memory usage and improve performance, especially with large data sets..

Results

For each exercise, the expected results were achieved. The main challenges included ensuring real-time updates between fragments and efficiently handling clicks on RecyclerView items. Each problem was solved by carefully applying Android development principles and debugging techniques.

Conclusion

The assignment increased my understanding of Fragments, RecyclerView, and ViewModel with LiveData. These components improve modularity, performance, and data persistence, making them indispensable in Android development. Overall, the exercises provided hands-on experience in structuring complex Android applications.

References

1. https://www.geeksforgeeks.org/android-image-picker-from-gallery-using-activityresultcontracts-in-kotlin/
2. https://www.geeksforgeeks.org/bottom-navigation-bar-in-android/
3. https://medium.com/@everydayprogrammer/implement-android-photo-picker-in-android-studio-3562a85c85f1
4. https://medium.com/@prasanth968/kotlin-image-picker-from-gallery-using-activityresultcontracts-9b5aa32e42d0
5. https://developer.android.com/training/data-storage/shared/photopicker#kotlin
6. https://proandroiddev.com/implementing-photo-picker-on-android-kotlin-jetpack-compose-326e33e83b85
7. https://medium.com/geekculture/searchable-recyclerview-e316289edc25
8. https://www.geeksforgeeks.org/gridview-in-android-with-example/
9. https://www.tutorialspoint.com/how-to-create-gridview-layout-in-an-android-app-using-kotlin
10. https://www.geeksforgeeks.org/android-gridview-in-kotlin/
11. https://stackoverflow.com/questions/20191914/how-to-add-gridview-setonitemclicklistener
12. https://developer.android.com/reference/kotlin/androidx/gridlayout/widget/GridLayout
13. https://abhiandroid.com/ui/gridview#gsc.tab=0
14. https://www.geeksforgeeks.org/gridview-in-android-with-example/
15. https://www.tutorialspoint.com/how-to-create-gridview-layout-in-an-android-app-using-kotlin
16. https://www.geeksforgeeks.org/android-gridview-in-kotlin/
17. https://www.geeksforgeeks.org/how-to-implement-android-searchview-with-example/
18. https://developer.android.com/reference/
19. https://kotlinlang.org/docs/home.html

Link to my GitHub repository (screenshots included):

- **https://github.com/BauyrzhanAzimkhanov/Mobile-programming-MSc.git**