

Assignment 4, mobile programming

Murat Tinal

23MD0442

01.12.24

Working with Databases in Kotlin Android

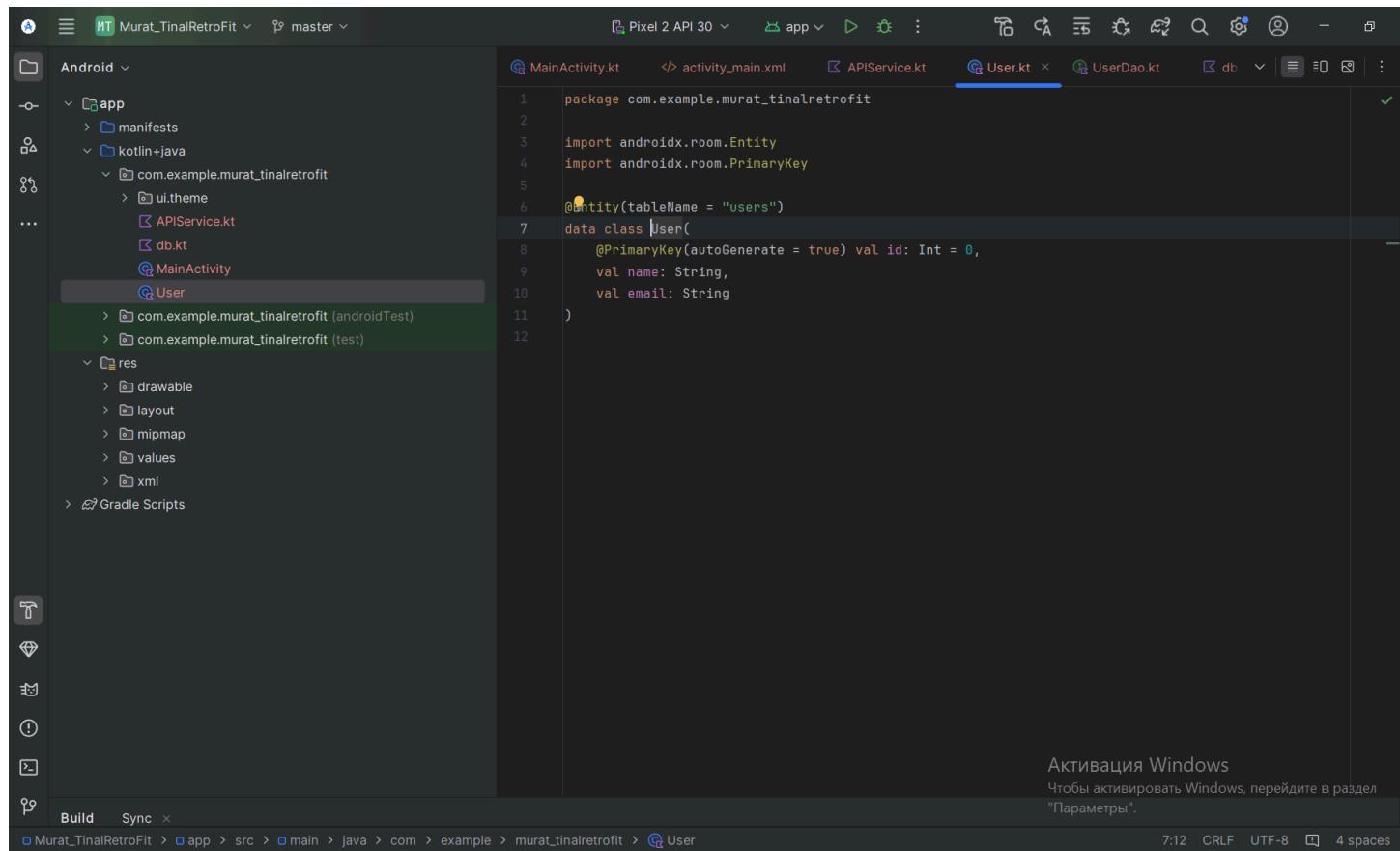


Fig. 1

This *Fig.1* code creates a class that works like a storage table for an app. The `@entity` tag sets up a place to save information. Each entry has three parts: a unique number, a person's title, and their contact details. This setup helps keep and find information easily.

The table includes three fields: an automatically generated number, a user description, and a contact address. This simplifies information management. Each record receives its own unique identifier, description, and method of communication. The table helps to organize the data for further use. The record has three elements: an individual number, a text signature, and an email address. This is necessary so that data can be conveniently added and found.

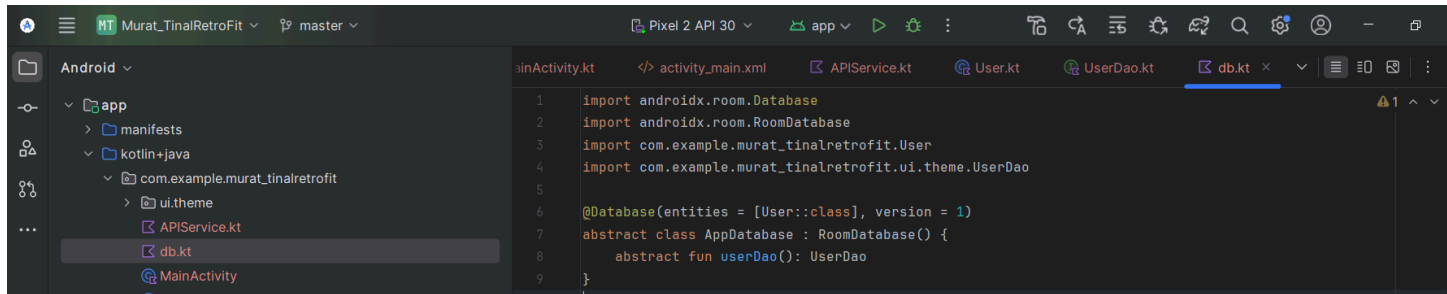


Fig. 2

This code defines a database class for an Android app using Room. The `@Database` annotation tells the app that this is a database setup, with a single table based on the `User` class. The version is set to `1`, meaning it's the first version of the database. The `userDao()` function allows the app to access the table and perform operations like adding, updating, or deleting data.

```
interface ApiInterface {

    @GET("volley_array.json")

    fun getUsers() : Call<List<User>>

    companion object {

        var BASE_URL = "http://velmm.com/apis/"

        fun create() : ApiInterface {

            val retrofit = Retrofit.Builder()

                .addConverterFactory(GsonConverterFactory.create())

                .baseUrl(BASE_URL)

                .build()

            return retrofit.create(ApiInterface::class.java)

        }

    }

}
```

Fig. 3

We need to create the Retrofit instance to send the network requests In Fig. 3. We need to use the Retrofit Builder class and specify the base URL for the service. Retrofit provides the list of annotations for each HTTP method: `@GET`, `@POST`, `@DELETE`, `@PUT`

The endpoints are defined inside an interface using retrofit annotations to encode details about the parameters and request method. T return value is always a parameterized `Call <T>`.

Because the POJO classes are wrapped into a typed Retrofit Call class.

```
val apiInterface = ApiInterface.create().getUsers()
```

Fig. 4

we call the `getMovies()` interface and implement the `Callbacks`. As part of the implementation, we need to override the `onResponse()` and `onFailure()`.

If the request succeeds the callback will come into `onResponse()`. If any error in the request the callback will go into the `onFailure()` method. We can `onResponse()` get our response from the response body

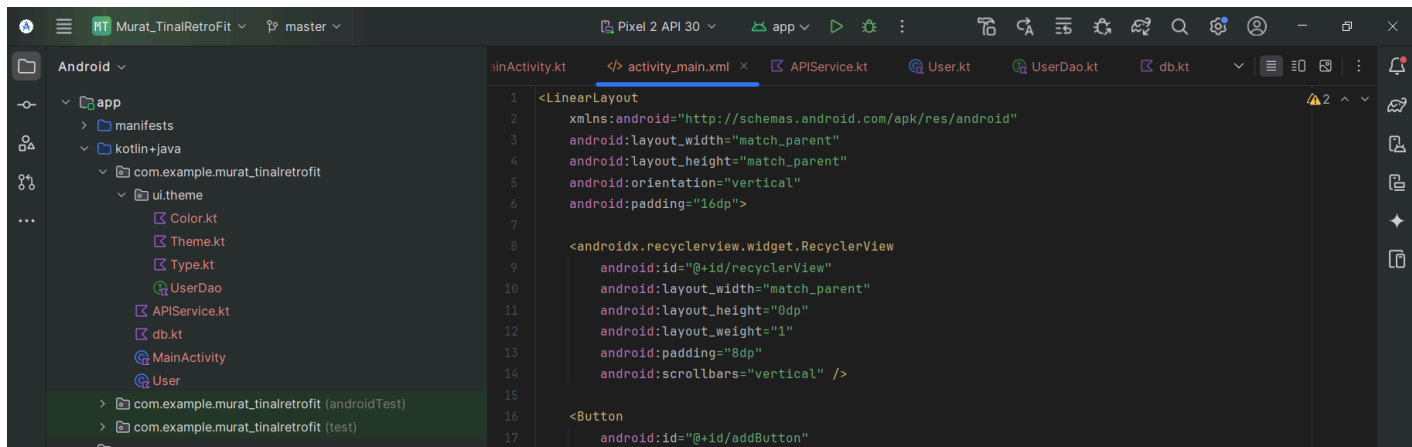
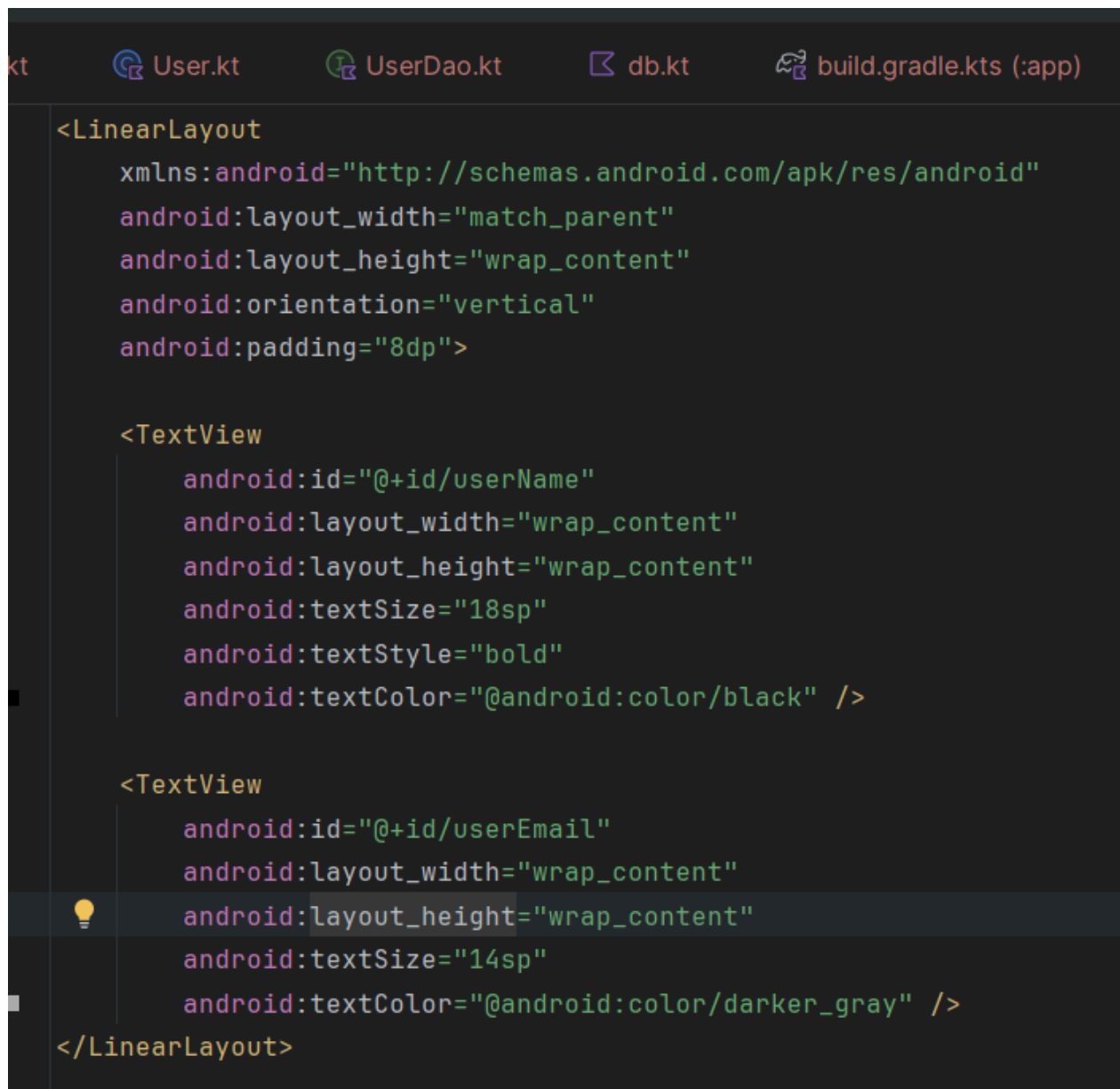


Fig. 5

The RecyclerView is set to take up most of the screen space, using the layout to adjust its size. The button at the bottom ensures users can add new entries easily. Both components are wrapped in a LinearLayout, which arranges them vertically. The layout user match_parent for width to fill the screen. Additionally, padding is added around both and the button for better spacing and appearance.

This layout organizes the screen vertically. At the top, there is a RecyclerView to show a list of items that can scroll. Below it, there's a button labeled "Add User" for adding new data. The list adjusts its size dynamically to fit the screen. Padding and spacing make the layout look clean and user-friendly.



```
kt      User.kt      UserDao.kt      db.kt      build.gradle.kts (:app)

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="8dp">

    <TextView
        android:id="@+id/userName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:textStyle="bold"
        android:textColor="@android:color/black" />

    <TextView
        android:id="@+id/userEmail"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="14sp"
        android:textColor="@android:color/darker_gray" />
</LinearLayout>
```

Fig. 6

This layout uses a vertical orientation to display two TextView elements. The first TextView shows the user's name in bold and larger text, making it more prominent. The second TextView displays the user's email in smaller, gray text. Both TextView elements are set to adjust their width and height based on the content. Padding is applied around the layout to create space and improve readability.

References

- <https://www.howtodoandroid.com/retrofit-android-example-kotlin/>
- <https://proandroiddev.com/demystifying-retrofit-network-call-with-kotlin-livedata-27437439137a>
- <https://medium.com/@pritam.karmahapatra/retrofit-in-android-with-kotlin-9af9f66a54a8>
- <https://medium.com/@imkuldeepsinghrai/api-calls-with-retrofit-in-android-kotlin-a-comprehensive-guide-e049e19deba9>
- <https://www.geeksforgeeks.org/retrofit-with-kotlin-coroutine-in-android/>

GitHub : https://github.com/kiraakz/Murat_Tinal_MobileApp4