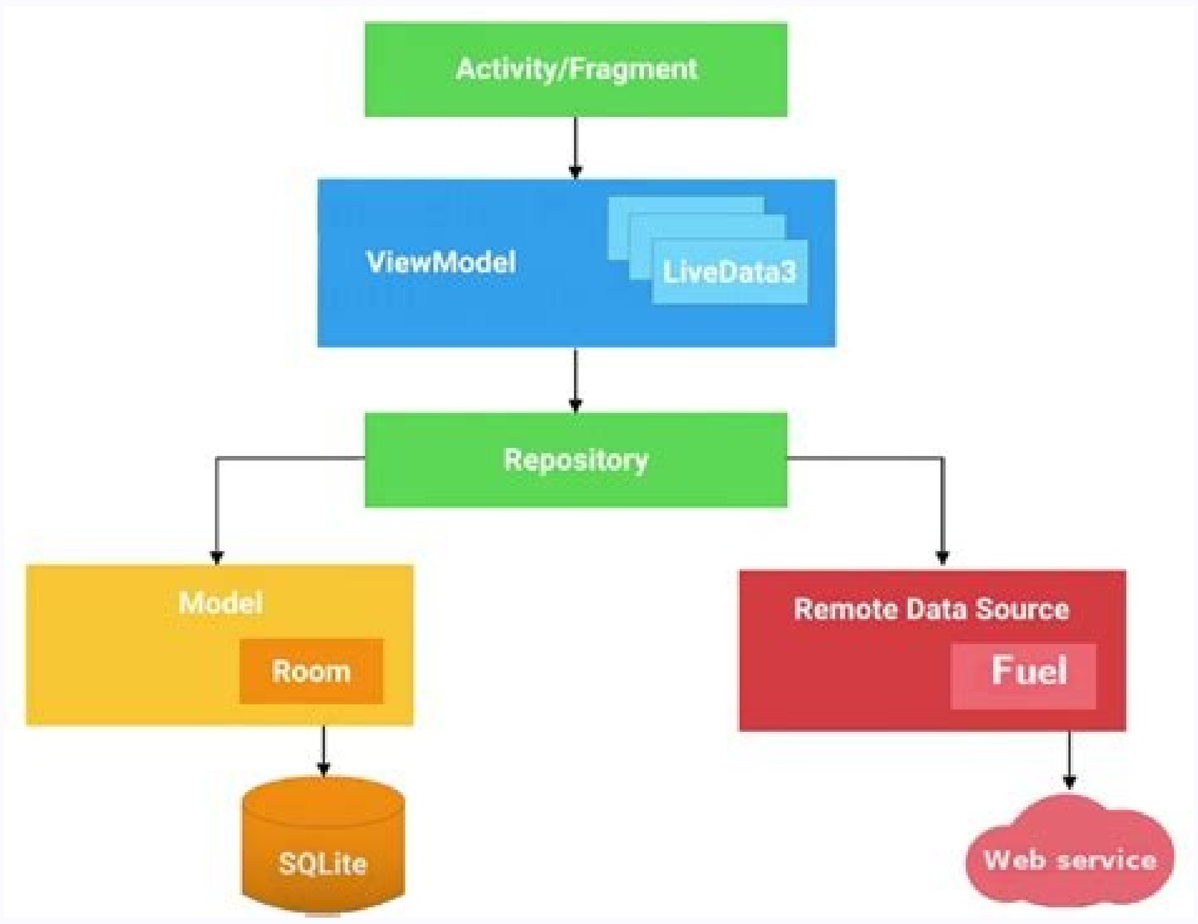
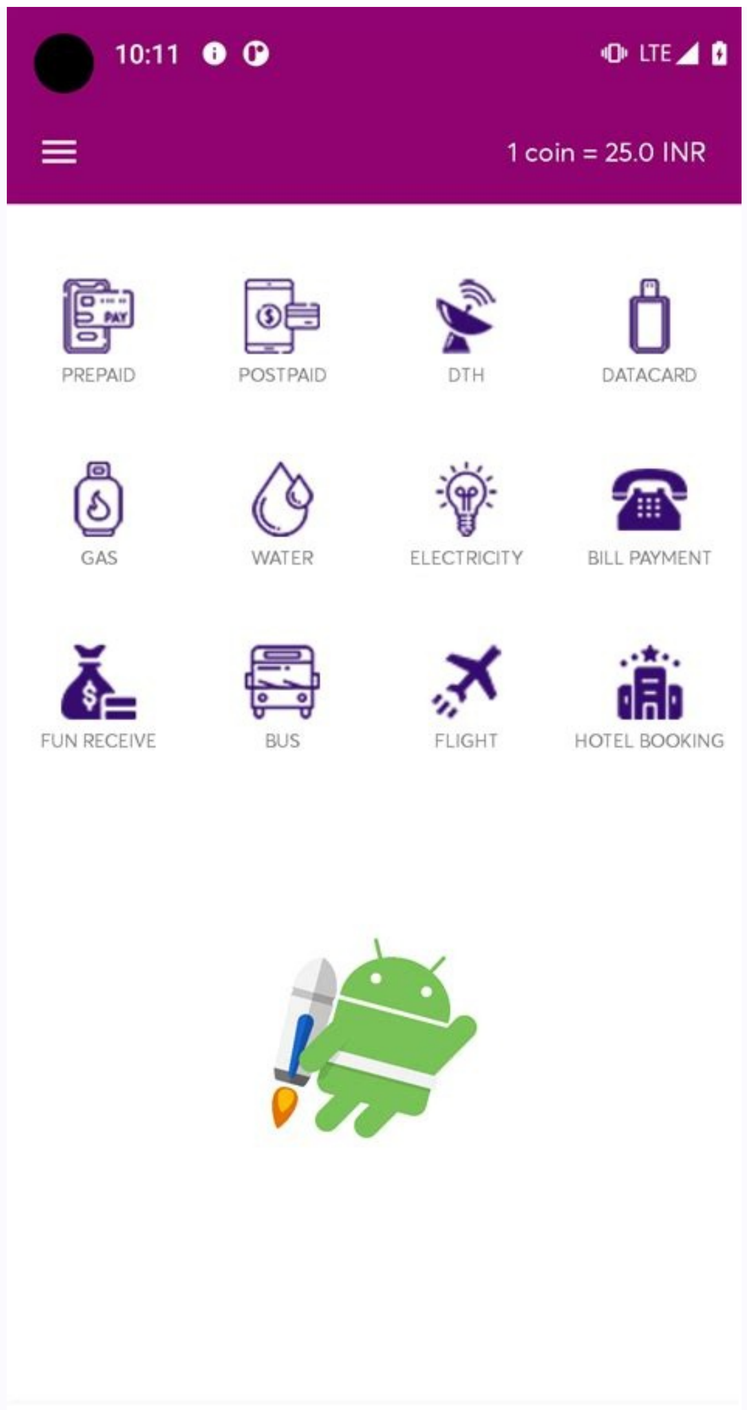
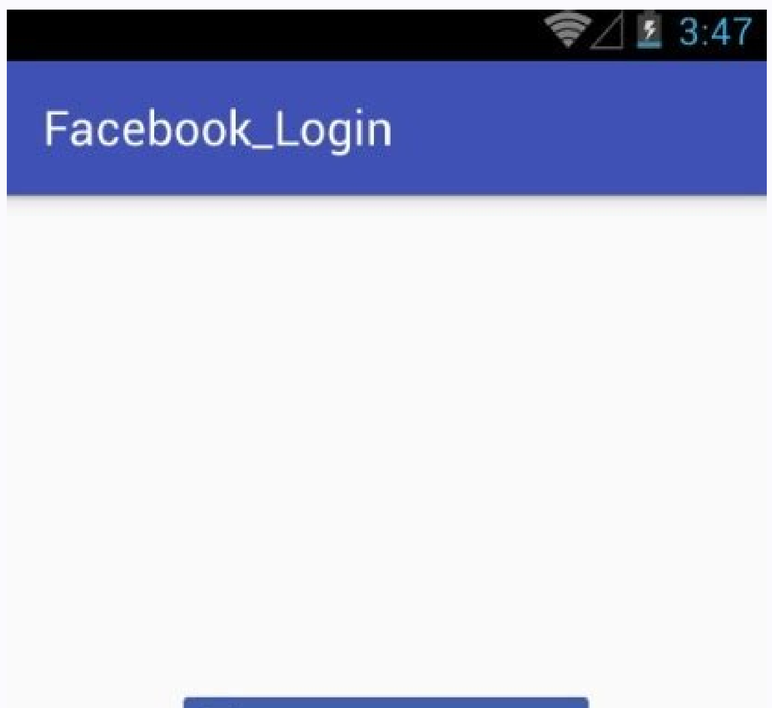
Kotlin mvvm retrofit login example


Activity/Fragment
ViewModel
LiveData3
Repository
Model
Room
SQLite
Remote Data Source
Fuel
Web service


2:55
Details Page
Kara Zor El
4 Follower(s)
FOLLOW
GENDER
FEMALE
COUNTRY
Bahamas
Registeration Date
2020-03-30
STATUS
ACTIVE
ROLE
ADMIN
About Me
Kara Zol El is a





Comrams are an interesting new language of Kotlin that allows us to write a more idiomatic asynchronous code. - It also means that you can write an asyncholy code in the same way that you usually write synchronous code in your project. I explained Mvvm in detail in another article. Please check this for a better understanding of MVVM. In this article, I will focus on Conquram and joint work. The basic safety block of the main flow ensures that any suspension function can be caused by the main flow. According to Kotlin's documentation, Conquram are considered a light alternative to flows. Before starting - with a cheaper and more controlled process - I would like to speak briefly about the concepts and functions usually used in the choutine. Grugrams are based on normal features, adding two new operations. Adds Conquram, suspension and summary. Where a "pause with suspension" is the usual Kotlin function with a further suspension modifier that indicates that the function can suspend the implementation. Conramgram. Remember the functions when we get an answer from our asynchronous task. He works together to change reminders and continue. To understand the functions of the suspension, we must also know the distributors provided by Kotlin. To indicate where the Conquuuram must be performed, Kotlin provides three distributors that can be used: disparars.amail. Use this distributor to perform a consequence of the main Android flow. This should be used only to interact with the user interface and quick work. The examples include the call of suspension functions, the execution of the operations of the Android user interface platform and the Liversa update.dyspatchers.io. The examples include the use of a component of the room, reading or writing files and the start of any networkCorutins are a great new Kotlin feature that allows us to write an asynchronous code in a lot of idiotic way. - It also means that you can write an asynchronous code just as you usually write the synchronous code in your project. I have already explained in detail MVVM in another entry. Check it out to better understand MVVM. In this record, I will focus on how the corrutations and modifications work together. MVVM with the refinement and processing of viewing in cutlin [example] Corutin with modifications on the main thread of the Viewmodel block in the main security ensures that any suspension function can be called from the main thread. According to Kotlin's documentation, the corrutor is a mild alternative to threads. Corutins are a way to avoid locking and switching the threads and making it cheaper and easier to manage ... Before we start, I want to briefly explain the concept of the program and the most common functions. Corutines are based on the usual features, adding two new activities. In addition to the call (or calling back) and returning the corrutation to the suspension and resume.SUSPEND - stops the current corrutation execution, retains all local variables only ordinary Kotlin function with an additional suspension modifier to indicate that the function cancel can stop execution. . Corutines when we get an answer from our asynchronous task. Stop and resume work together to replace the callbacks. To understand the stop functions, we also need to know about the dispatchers provided by Kotlin. To determine where the applications need to work, Kotlin provides three dispatchers you can use: Disatcher.Main - Use this dispatcher to run the application in the main Android thread. It should only be used to interact with the user interface and work quickly. Examples: Calling Stop Functions, Exercising Android UI System, and Updating Ledgate Objects. Dispatchers.io - This dispatcher is optimized for the disk or network I/O of the main thread. Examples include using the component of the room, reading or writing files and launching any networkThis expeditionary program is optimized in the work intensively loading the processor for the main fiber. Examples of usage are the list and parse json.lets, see it in the example -> our API is called "Koruuties". So we use Dispatchers.io. When we call the suspension metalmov() method, it pauses in our program. The collaboration between the main thread will continue with the result once the block with the context is complete. Note: Using the suspension function does not mean that the valley does not activate the function in the background thread. It is normal for the suspension function to work in the main thread. Laufen, Asnclaunch and Async are the most commonly used crown makers. Which program will be aborted if the task is canceled. Async creates a new overall program and returns its future result as a delayed implementation. Starting the CO program is aborted when the resulting object is canceled. See. This code snippet as an example. An example is the difference between initial and asynchronizing asynchronization that ASNC can return the future result where the delayed type can call the function() the deferred variable to get the CORS result and the flats have that result returned. Everyone is the creator of a program (e.g. start, assimilation, etc.) is an extension of the corouutuescope. When the binoculars depart, the Korats will also disappear from view. Fortunately, Android Lifecycle-ViewModel-KKTX is an easy way to get a Corouutin scope ViewModel. I'll show you how to do it later. Start the kroden in your android project and add this library to build. The Addiction of Gradle: NOTE: You must also be using Valley 1.3 or a newer version. Working with Retroft? Retrofitting is a safe type of Android and Java HTTP clients. Starting with the Retroft 2.6.0 you no longer need a link adapter as the retrofit now develops a modifier suspension in the features. You want to get started, leave ... add link between modernization to our assembly file. In this example I use API to get list of movies. Please note this fragment of our interface: you can notice that instead of now we get a function with a suspended modifierYour interface has a function. When this function is in the documentation, this function will behave as a normal call behind the scenes. Information such as the response code. There is no hope () because it is automatically processed! Like all Android networks, this is done in the background. And this is a very clean way to do it! The creation of the modernization of RemDrofit Serviceash will seem the next fragment of code: ViewModel with Corinescops Co -ures Monitor all the Kouutes created. Therefore, if the area is canceled, you will raise all the conquram it creates. This is particularly important if ViewModel Lancia Choroutine. If your viewmodel is destroyed, any asynchronous job you can do. Otherwise, you will spend resources and potential memory losses. If you think that after the viewmodel has been destroyed, you should maintain asynchronous work, it should be done at the lowest level of your architectural application. Add Choiroutinescope to your viewmodel, creating a new area with the superjorjob activity that you have deleted, ONCLEARED (). The gruggles created with this area will be used as long as ViewModel is used. Coroutines and Livetalvetata are a user interface supervisor and we predict that we access the value from the main flow. By releasing Lieata-17.1.0-Alpha1 Google, he guaranteed the compatibility between the collapse and chorourines. Exceptions of processing from Kotlin Chorouzesif. But when we work with the corutins, we can elaborate an exception using the global exceptions of choroutine called choroutineexception. To use it, first viewmodel we create an exceptional processor and then add the View Modelscope processor. KTNEXT action composed of a repository.mainrepostory.ktsetup viewmodel configuration, MyViewModeFactory.ktmainview.kt, finally, in our main display model and call it Getallmovies ()This example is on github. This is a very simple magazine example using MVVM and operational data and data in Android. It receives UI input via Data @=â, saves it to linedata and displays it in the UI. This example is for those who want to know the easiest way to get UI data. This is useful in many ways, such as saving time for development, code refactoring, verification, etc. It's no surprise that this is used throughout the Android community. So, let's start using these technologies together in one application: what is MVVM? What is a database? What is Live Data? Step-by-step implementation conclusion What is MVVM? Answer: MVVM is a design pattern for organizing applications with a GUI, which has become popular on Android. This concept will introduce you to the three main components of MVVM: View, Model, and ViewModel. The Presentation Pattern (MVVM) is an architectural pattern used in software development created by Microsoft, which specializes in design pattern design patterns. It is based on MVC (the model of the communication model) and focuses on modern UI development platforms (WPF and Silverlight), where the UX developer has other requirements than “more traditional” developers. ". MVVM is a way to build client applications that use core WPF functions, can easily test application functionality, and help developers and designers work with less technical difficulty. Overview: Performance is determined in XAML and should have no background logic. Model view lines only through the data transmission channel. Model: The model is responsible for providing data so that WPF can easily use it. If necessary, it is necessary to implement InotifyPropertyChanged and/or inotifyCollectionChanged. ViewModel: ViewModel is a model of the application inside the application or, could say the abstraction of the presentation. Provides presentation-related data and reveals the behavior of ideas, usually using commands. Model: Definition, role and responsibility. What should be in the model layer and what not. Benefits of model isolation and its impact on testing. Presentation: definition, role and responsibility. How it works with ViewModel. ViewModel: Definition, role and responsibility. Because it deals with display, providing actions and observed state. Interaction with the model. Isolation in the future. What is a database? Answer: data binding libraryAn assistance library with which you can connect the components of the user interface in your provisions to the origins specified in your application with a declaration format instead of at the code level and much more. What is experienced? Answer: Livedata is a great observable database. In contrast to a normal observable, it is aware of the life cycle, which means that it respects the life cycle of other components of application such as activities, fragments or services. This awareness guarantees that the updates only experienced the observers of the components of the application, which are in the state of the active life cycle. The use of LivedATA offers the following advantages: guarantees that the user interface corresponds to the data status: experienced the observer model. Livedata informs the objects of the observer when the condition of the life cycle changes. In these objects of observers, it is possible to consolidate the code and update the user interface. Instead of updating the user interface every time the application data changes, your observer can update the user interface every time a change is made. No memory loss: observers are connected to life cycle objects and their associated life cycle is cleaned during destruction. No abnormal arrest due to the interrupted activity: if the observer's life cycle is inactive, for example for activities at the bottom of the stack, he does not get a live event. No more manual life cycle management: The components of the user interface only observe the relevant data and interrupt or do not use observations. LivedATA automatically manages all of this because it is aware of the changes in the state of the relevant life cycle during observation. Updated data: If the life cycle becomes inactive, it receives the latest data after it is active again. For example, an activity that was located in the background receives the latest data immediately after returning. Proper configuration changes: If an activity or fragment is revised due to a change in the configuration, for example, the rotation of the device immediately receives the latest available data. Resource release: You can expand the animated object with the Singleton models to wrap the system services so that they can be used in your application. The lived object is connected to the system service, so that every observer who needs the resource can simply look at the object lived. You can find more information in the live expansion. Passo-Passo 1 Implementation: Add the Grace file an association of data and implementations: Android {... Data connection {Activated True}} Def Life_Versions = "1.1.1" // Implement the components for the life cycle $ LIFE_VERSIONS "AnotationonprocessorImprove the Android.UTİL.PATTERNS; Common Class Loginuser {Private String Stremailaddress; Private Zeičenfolge Strpassword; Public Loginuser (String E-Pastiaaddress, String Password) {Stremailaddress = E-Pastiaaddress; Strpassword = Password; } Public String GetStremailaddress () {return Stremailaddress; } public string getstrpassword () {return Strpassword; } General Boolen isemailvalid () {turn scheme.email_address.matcher (getstremailaddress ()). Matches (); } Public Boolean Returns () {Getstrassword (). length ()> 5; }} Import the Android.arch.lifecycle.mutablelivedata; Improve the Android.arch.lifecycle.ViewModel; Improve Android.View.View; Common Class Loginviewmodel Erweiter viewmodel {public mugablelivedata e-pastiaddress = new validate (); Public Mutableliivedata Password = new validate (); SPECIAL CONTRACTEDATA Public Mutablelivedata getuser () {if (usermutableliivedata == null) {usermutablelivedata = new validate (); } UsermutableliData zurückgeben; } Public Void OnClick (View) {Loginuser Loginuser = New Loginuser (email - email usermutablivedata.setvalue (Loginuser); Inwardly the insertion of android.os.bundle Importieren; android.support.annotation.nullable; loginviewmodel.Class) Observer () {@override Public void onchanged (@nulllable Loginuser Loginuser) {if (textutils.isempty (object.requirenonnull (Loginuser) .getstresiladdress ()). ein "); Binding.txtemailaddress.requestfocus ();{ Binding.txtPassword.setherror("PASSWORD"); Lising.txtPassword.requestfocus(); } else if (! loginUser.ispasswordlengthgreaterthan5()) {binding.txtassword.seterror("Geben si Mindestens 6 PasswortNummern ein"); Lising.txtPassword.requestfocus(); } son {binding.lblemailanswer.settext(loginuser.getstrailaddress()); Lising.lblasswordanswer.settext(loginuser.getStrapsword()); }}}); }} Zum Schluss: Die xml-Datei: (Wichtige

änderungen hier) LoginviewModel.onclick (V)}" Android: TextSize = "18SP" Application: Layout_Constainnd_tondof = "Parent" Application: Layout_ContainTart_Tostartof = "superior" application: layout_Constaintop_Tobottomof = "@+ID/TXTPASSWORD"/>