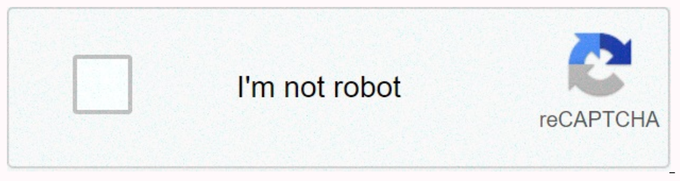

Continue

Android studio tutorial javatpoint

Google is determined to increase racial equality for black communities. How to look. The purpose of the architecture components is to provide guidance on the architecture of the app, with libraries for common tasks such as life management and data perseverance. The architecture components help you configure your app in a way that is strong, tested, and maintanabli with low boilerplate code. The architecture components are part of the Libraries Android Getpac. This is the Java programming language version of codelabel. The version can be found here in The Kotolon language. If you work through this codelab, you run into any problems (code worms, gramer errors, unclear words, etc.), please report the issue by which you have a link to a mistake by it. Terms you need to be aware of the basic principles of Java, objection-based design concepts, and Android development. Specifically: Recciclerview and Adapter suqlaity databaseand sulate query language threading and yakotorsaoraca This software helps to become familiar with the architecture printer sedate data from the user interface, such as atup or MVC. Apply the explained architecture in this codelab app guide for architecture. This codelab is focused on down-to-take Android architecture components. Off-topic concepts and codes are provided to you just copy and paste. This codelab provides all the code you need to build the full application. What you will do this codelab, you will learn how to design and build an application using the architecture components Room, Viewodal, and Lavedat, and create an application that does the following: Apply our recommended architecture using the components of android architecture. Works with databases to obtain and save data, and automatically before database with certain words. Shows all the words in Recccckalravi. When the user + buttons are not edited, the user will open another activity. When the user entered a word, the word is added to the database and list. The app has no frills, but is complex enough that you can use it as a template to build. Here's a preview: You'd need to know about how to use Android Studio 3.0 or later and after it. Make sure that the Android studio is updated, as well as your SDK and The Ghadi. Otherwise, you will have to wait until all updates are made. A free Android device or emulator. Note that the solution code is available on a zip and a gout. We encourage you to create the application from the rabbit and see this code if you get stuck. There are many steps to apply the architecture using the components of architecture and recommended. The most important thing is to create a mental model of what's going on, understanding how to fit along with pieces and that's the data that is running. As you work through this codelab, don't just copy and paste the code, but try to start building this inner understanding. Recommended Art What are the ingredients? Here is a brief introduction to the architecture components and how they work together. Note that The codelab focuses on the components, i.e. Lavedat, Viewodal and a subset of the room. When you use it, each component is defined. This diagram shows the basic form of architecture: institution: the mudon-reference class that defines the database table when working with the room. Security database: On device storage. The room continuously creates the library and maintains this database for you. David: Data Access Object. A definition of SQL questions for functions. When you use a david, you call his ways and the room cares for the rest. Room Database: Works as a point of access to the database work and basic suqlaity database (hides on scoutyopanaheil). The Room database uses david to question the security database. Storage: Used to manage more than one data

source. The Moviedial: Works as a communication center between storage (data) and UI. The UI no longer has to worry about the origin of the data. The activity/piece of the way of the events is alive. Lavedat: A dataholder class that can be observed. Always get a degree in the latest version of the data and notify its observers when the data changes. Lavedat is aware of life. UI components only observe relevant data and do not stop or resume observations. Lavedat manages it automatically with it because aware of the changes in the status of the life cycle concerned when observing. Looking for more? Check the full guide to app architecture. The Rumordasamaple Ractaia review shows all the pieces of the app in the following sketch. Each of the covered boxes (other than the silate database) represents a class you will create. Open Android Studio and start a new Android studio project. In the Create New Project window, select Empty Activity and click Next. On the next screen, name the app Roomoordsamplana, and click Finish. Next, you'll have to add component libraries to your ghadi files. In Android Studio, click the Projects tab and enhance the Ghadi Script folder. Open Construction. The module: application. To set target and medium compatibility for 1.8, add the following completely compatible block within the android block, which will allow us to use THE JDK 8 la bdas later: Compaliuptans {Swarkimpitabatabalaty = 1.8 Change dependency block with TargetComputer = 1.8} Appcompat: appcompat: $rootProject $rootProject. appCompatVersion//To work with the components of architecture/you will probably have to update the version numbers in the construction. Room: Room-Complexer: $rootProject. roomVersion androidTestImplation androidx. Room: Room-testing: $rootProject. roomVersion//Lifecycle components androidx. Lifecycle: Lifecycle-viewmodel: $rootProject. Implement androidx. Life Space: Lifecycle-livedata: $rootProject. LifecycleVersion Processing UI implementation indoradaan. Konstrintlyat: $rootProject. Konstraantlayoversion implementation com. Google. Android. Content: Material: $rootProject. materialVersion//testing test-amplymentaaaan This: junit $rootProject. The Version of June androidTestImplation androidx. Arc. Cover: Core Testing: $rootProject. Coretestversion androidTestImmentation (androidx. test. espresu: espresu-core: $rootProject. espressoVersion, {excluded group: 'com. android. support', module: 'support-explanations'}} androidTestImtimmentandroidx.. ext: This: $rootProject. androidxJunitVersion} in your build. The grdale (Project: RoomWordsSample) file, add the version number at the end of the file, as given in the code below: Get the most current version number from including the components on your project page. ext {Appcomparesaon = ' 1.2.0' ' //testing junitVersion = ' 4.13.1' espressoVersion = ' 3.1.0' androidxJunitVersion = ' 1.1.2 '} Is the data word for this app, and you will need a simple table to hold these values: The rheothectorcomponents allow you to create one through an institution. Let us do it now . Create a new file that is called Word. This class will define the institution (which represents the table of the silate) for your words. Each property in the class represents a column in the table. The room will finally use these features to create tables and ready-made items from rows in the database. Here is the code: Public Class Word {Private String mWord. The word @NonNull (the word } Public String getWord () {Get Back. MWord To make the word class meaningful for a one room database, you need to explain it. Identify the explanations of how each part of this class is related to entering the database. The room uses this information to create code. This code shows as you update your word class with explanations: @Entity (Tablanama = word_table) Public class word {@PrimaryKey @NonNull @ColumnInfo (name = word) Private string mWord. Public word (@NonNull string word) {it. mWord = Word;} Public String getWord () {Get Back. mWord;}} When you copy the paste code, you may have to manually import interpretation classes. You can move the code cursor for each error and enter the Project Quick-Fax Keyboard Shortcut (enter on Alt + Windows/Linux, enter option + mac) to import classes faster. Import Indoradaan..... Note that if you write yourself the descriptions (instead of pasting), android studio will import itself. Let's see what these descriptions do: @Entity (Tablanama = word_table) @Entity each class represents a silate table. To indicate that this is an entity to explain your class declaration. If you want it to be different from the name of the class, you can define the name of the table. The table of his name word_table. @PrimaryKey The institution needs a fundamental key. To keep things simple, every word works as its own basic key. @NonNull to check that the value of parameter, field, or return method cannot be null. @ColumnInfo the name of the column in the table (name = word) if you want it to be different from the name of the iPad variable. Every field stored in the database must be a public either getter method. This sample provides a getWord (method). You can find a complete list of descriptions in the room package summary reference. See the description of the data using the room institutions. Tip: You can automatically create unique keys from the basic key innotittang: @Entity (Tablanama = word_table) Public class word {' @PrimaryKey (self-generated = true) ' private int id; The word '@NonNull' 'private string'; '//.. Other areas, getters, satars'} associate with a David (Data Access Objection) time set up to your SQL and with a way to verify. In Your Room In David, you use simple descriptions, such as @Insert, to represent the most common database operation! The room uses David to create a clean API for your code. David must have an interface or summary class. As a default, all questions must be implemented on separate threads. Apply David Let's Write a David that provides questions: Deleting all words is ordered to insert a word that creates a new class file to delete all words. Copy and paste the following code into Vorado and correct imports necessary to set it up: @Dao allows the public interface to enter more than once of the same word by adopting the voreao {////conflict resolution strategy @Insert (onConflict = Conflictstrategy). Ignore) Undo (word word); @Query (delete from word_table) zero deleteAll (); @Query (select * to word_table from order <Word> Let's go through this: Voorado is an interface. Either interface or summary classes must be. @Dao The official identifies it as a David class for the room. Undo (word word); declares a way to insert a word : @Insert is a special David method where you don't have to provide any SQL! There are also @Delete and @Update to delete and update rows, but you are not using them in this application.) onConflict = Inconflctstrati.g. Ignore: Ignores a new word selected on dispute strategy if it is already the same in the same list. To learn more about the available conflict strategy, check the documents. deleteAll (): Declares a way to delete all words. There is no facility for deleting multiple entities, so it is @Query with general information. @Query (word_table delete from the server): @Query you need to provide a SQL query as a wire parameter. List of GetalpahbetaZidovardus (): A way to get all the words and it's a list of words back. @Query (word * Select from word_table order by: Return a list of words arranged in increasing order. Learn more about </Word> </Word> StainS When data changes, you usually have to take some action, such as displaying the latest data in the UI. This means that you have to observe the data so that when it changes, you can react. Depending on how the data is stored, it can be difficult. Observing a change in data in multiple components of your app can create clear, inflexible dependency routes between components. It makes it difficult to between other things, testing and fix. The Lavedat, a life cycle library class for data observation, solves this problem. Use the value of the type of a return of the lavedat in your method description, and when the database is updated, creates all the necessary code to update the lavedat. Note: If you use the lavedat freely from the room, you must arrange for the data to be updated. There are no publicly available methods to update the data stored in The Lavedas. If you want to update the data stored within the device, you must use Motblavedat instead of Lavedat. There are two public methods in the Motballavedat class that allow you to set a value of a lavedat objection, set price (T) and post price (T). Generally, motiblavedat is used inside the viewof the vaumudal, and then the videodal is only highlighted to the observers at the umt lavedat items. In Vorado, change the getalpahbetazidovardus () method signature so that the list of back is wrapped with the lavedat: @Query (select *from word_table order by word asec) Lavedat <> <Word>> GetalphebetaZidoverdus (); Later in this codelab, you track data changes by an reviewer. To learn more about other ways to use Lavedat, see the Lavedat documents, or see these architecture components: Lavedat and LifeCycle Video. The room has a database bed at the top of a silate database. The room takes care of multiple tasks that are used to handle you with a sclotiupanheilper. The room uses David about questions about his database. By default, to avoid poor UI performance, the room does not allow you to issue questions on the main thread. When room questions return to Lavedat, questions automatically run a minority on the subject of background. The room provides time checks for the compiled suo-mail statements. Your room database class must be summarized and expanded to Roumdatatabasi. Generally, you just need an example of the room database for the entire application. Let us make one now. Create a class file called remix Omdatabase and include code in: @Database (Institutions = {Word .class}, Version = 1, Exrovmdatatabasi Schema = false) Public Summary Class Spoiling Database expands {Public Summary Voradayo Voradayao (); Private static last int NUMBER_OF_THREADS = 4; static final yakutorsaravaka dataasivaratiaetaaaqatavar = yacotrus. novaaadtheredpole (NUMBER_OF_THREADS); static bad tehsil database getDatabase (final context) {if (e.g. = invalid) {(e.g. = = invalid)} = = invalid) {example = room. databaseBuilder (context getApplicationContext( ), Loadroco database .class, word_database). Construction ().}}} Example of return of ;}} .class </Word> Let's go</Word> By code: The database class for the room must be roumdatasiand extension that you use the class to become a one-room database with @Database to define and interpret parameters to declare the entities in the database and set the version number. Each organization is consistent with a table that will be created in the database. Database transfers are beyond the scope of this codelab, so we set up falsely duplicate schemas here to avoid a built-in warning. In a real app, you should consider setting up a directory to export the schema so that you can check the current schema in your version control system. Method for each time through a summary getter @Dao database. We have defined a whatever, corrupted, database to prevent more than one instance of open database at the same time. GetDatabase that backs up what you want. It will obtain the database for the first time, using the room database builder to create a roumdataxy objection in the context of its application to work and name the word_database. We have created a yakotorsaraoka with a fixed thread pool that you will use to run minority database operations on background subject. Note: When you modify the database, you must update the version number and define a transition strategy for a sample, a destroy and re-create strategy can be enough. But, for a real application, you have to apply a migration strategy. See understanding of understanding the matter with the room. In android studio, if you paste the code or get mistakes during the construction process, select > Make a clean plan. Select again > reconstruction plans, and then rebuild. If you use the provided code, there will be no mistake whenever you are directed to build the application. A storage class summarizes access to more than one data source. The components of storage architecture are not part of libraries, but the code is a recommended best practice for separation and architecture. A storage class provides a clean API for access ingesting data to the rest of the request. Why use storage? A storage management question and allows you to use for more than one backup. In the most common example, storage is the logic to obtain data from a network or use cash in a local database to decide it. Create a class file called Storage Voradorposatoreand paste the following code in: Class Voradaposatore {Private Voradayao Moverao; Private Lavedat <> <Word>> Malloverdus ;//Note That to test the voraderposatore for the unit, you must remove the request/dependency. It increases complexity and a lot of code, and is not about testing this pattern. The components that make android android are not stored in storage or voraderposatore (request request) {bad tehsil database = Wordroomdatabase. getDatabase base (request); Moverao = db. Vorao (); = موورداؤ. گیٹاپھبیٹازیڈوواردس ()</Word> ()</Word> On a separate thread. When data is changed, the observation will inform the Reviewer. Lavedat <> <Word>> getallwords () {Back To Malloverdus ;}//You must call it on non-UI thread or your app will throw an exception. The room ensures//you are not doing any long running operation on the main thread, blocking the UI. Undo (word word) {Daalu Omdata base. Data-serveraccess. Enforcement (()-> {Moverdao. Insert (word);}} Important steps: David is approved in the storage constructor and as opposed to the full database. This is because you only need access to David, because it contains all the reading/writing methods for the database. There is no need to expose the entire database to store. GetAllWords method backs the word's list from the room; we can do this because we explained the Getal-Phebetazidovardus method to return the lavedat in the Lavedat class stage. The room implements all questions on a separate thread. After that, the data has been changed when The Lavedat will notify the reviewer on the main thread. We need not enter on the main thread, so we use this yakotorsarawa that we created in it to enter a background topic. The repastremeans the medium between different data sources. In this simple example, you only have one data source, so storage is not much. See the box sample for more complex implementation. The role of The Oyumudal is to provide data in uI and maintain order changes. A wimwadal works as a communication center between storage and UI. You can also use the videotoal to share data between pieces. The Wayomudal life cycle is part of the library. For an introductory guide to this topic, see View View Oyomudal Review or Viewodales: Blog Post as a simple example. Why do you use the omeodayal? A video video contains a lifecycle of your app's UI data that is alive by changing the order. Dissofaring your app's UI data from your activity and piece classes allows you to follow the principle of individual responsibility: your activities and pieces are responsible for drawing data on screen, while your videodial can take care of all the data you need for ul and processing. In The Video, use the application to change the data that the UI will use or display. There are several benefits using the Lavedat: you can put an reviewer on the data (for this rather than polling changes) and only update the UI when the data actually changes. Storage and UI are completely separated by The Viewwadal. There are no database calls from The Oyomudal (all handled in storage), making the code more testable. Apply The Viewwadal Create a class file for The Wordodayudayal and include the code in it: Public Class Verduyumudal Extension Undervaduyumudal {Private Vorderposatore mRepository; Private Final Lavedat <> <Word>> Malloverdus; Public WordViewModel (Request Request) {Super (Request) ; mrepository = </Word> </Word> Malloverdas = mRepository. Getwords (); } Login<><Word>> getAllWords () {back To Malloverdus;} Insert Public Invalid (Word) {mRepository. (word);}} Here we have: A class called The Vorduyumudal which gets requested as a parameter and extends on the undervaduyumudal. Add a private iPad variable to hold a reference to the storage. To return a cache list of words, you have added a getAllWords () method. A konsterocta and apply that produces the worldrposatore. In Konsterocta and, using the Levidat store, the start of allWords. Insert a jam () method that call the (insert) method of storage. Thus, the insert imperandion () is protected from the UI. Warning: Don't keep a context reference that has a small life cycle than your video- omeodayal! Examples are: a reference-place can cause a memory leak, for example a reference to a damaged activity of the way way you do! All these items can be destroyed by operating systems and made when there is a configuration change, and it can happen several times over the course of a period of life. If you need the context of the application (which has a life life that is requested as long as it is requested), use the undervadudumoodal, as shown in this codelab. Important: The wayomydeals need more resources when not to live up to the action of the app that is killed in the background. For UI data that needs to survive the death process because of running beyond resources, you can use the Protected Status module for the wayodales are used. Find out more. To learn more about The Wayodayal Classes, see the architecture components: Video. Next, you need to add xML settings for lists and items. Assume that these are familiar with making the codelab layout in XML, so we are just providing you with the code. Make your request central idea, subject matter content Appappathimi parents on the main idea, topic.... Light. Daerkakataonber. Add a style to items listed in values/shelves <resservices> <!-- Base application theme. --> <style name=AppTheme parent=Theme.MaterialComponents.Light.DarkActionBar> <!-- Customize your theme here. --> <item name=colorPrimary>@color/colorPrimary</item> <item name=colorPrimaryDark>@color/colorPrimaryDark</item> <item name=colorAccent>@color/colorAccent</item> </style> <!-- The default font for RecyclerView items is too small. The margin is a simple delimiter between the words. --> <style name=word_title> <item name=android:layout_width>match_parent</item> <item name=android:layout_marginBottom>8dp</item> <item name=android:pandingLift>8dp</item> <item name=android:backroad>@android:color/holo_orange_light</item> <item name=android:textAppear >@android:style/TextApp address.Large</item> </style> </resservices> .xml: Add a setting/recyclerview_item .xml layout: <?xml version=1.0 xmlns:android= android:orientation=vertical android:layout_width=match_parent xmlns:android= android:orientation=vertical android:layout_width=match_parent ></LinearLayout xmlns:android= android:orientation=vertical android:layout_width=match_parent ></Word> ></Word> In <TextView android:id=@+id/textView style=@style/word_title android:layout_width=match_parent android:layout_height=wrap_content android:backaround=@android:color/holo_orange_light></TextView>Layout/activity_main .xml, replace the tattoyo with a recceclerevieww and add a smart action button (fiab). Now your setting looks like this <androidx.constrightlayout.wedget.ConstrintLayout xmlns:android= xmlns:app= xmlns:tools= android:layout_width=match_parent android:layout_height=match_parent tools:contact=. MainActivity > <androidx.recyclerview.wedget.recyclerView android:id=@+id/recyclerview android:layout_width=0dp android:layout_height=0dp tools:list item=@layout/recyclerview_item android:panding=@dimen/big_padding app:layout_constraintbottom_tobottomof=parents app:layout_constraintleft_toleftof=parents app:layout_constraintright_torightof layout_constrainttop_totopof=parents></androidx.recyclerview.wedget.recyclerView> <com.google.android.meterial.bloatingactionbutton.BloatingActionButton android:id=@+id/fab app:layout_constraintbottom_tobottomof=parents app:layout_constraintend_toendof=parents android:layout_width=wrap_content android:layout_height =wrap_content android:layout_margin=16dp android:contentdescription=@string/add_word></com.google.android.spiritual.fightingActionButton> </androidx.consistentlayout.wedget.ConsistentLayout> should: Your smart action button (FIB) should be in line with the action available in appearance, so we want to replace the icon with a + symbol. First, we need to add a new character asset: File > New > Select The V.C.R. Asset. Click on the Android Robot Icon to add: Search for add and select '+' asset. Click OK. Then, click Next. Confirm icon path as key > drawable and click Finish to add assets. Still update the faab to add layout/activity_main .xml, new drawables: <com.google.android.meterial.fightingactionbutton.FightingActionButton android:id=@+id/fab app:layout_constraintbottom_tobottomof=parents app:layout_constraintend_toendof=parents android:layout_width=wrap_content android:layout_height=wrap_content android:layout_margin=16dp and roid:contentdescription=@string/add_word android:sdc=@drawable/ic_add_black_24dp></com.google.android.spiritual.flotingactionbutton.FlotingActionButton> you're going to show data in a reccealview, which is a little better than just throwing the data into a tutorial. It's how you know recikalravi, recikalravi. The viewholder, and the work of the list adapter. Start by creating a new file that will place the viewholder, which displays a word. Create a bound method that sets text. Class-Over-the-holder extension recc.c.a.r.r. Viewholder {Private Final Tatyo Vvardatimev; Private Vordyoholder Attemeview) {super (itemView); wordItemView = itemView. findViewById (r id. textView);} Public False Bound (sterling text) {wordItemView. setText (text);} Create static WordViewHolder (ViewGroup Parent) {View = = . Air filled (R.layout.recyclerview_item, parents, false); return the new worldholder (see); }} Create a class list adapter that extends the list adapter. Dafofutal. Make Atimkalalbak as a static class. This code is: Expands the list adapter of the public class situation adapter<Word, wordviewfolder=> {Public Speech List Adapter (@NonNull Daffatal. Atimkalalbak<Word> Dafkallbak) {Super (Dafklalbak);} @Override Public Viewholder Int viewType {Create Return WordViewHolder (Parent);} @Override Public Invalid OnBindViewHolder (WordViewHolder Holder, int Position) {Word Current = getitam (Position); Holder. Bound (Current getword ()) ;} Static Class WordDiff Extension DiffUtil. ItemCallback<Word> {@Override Public Bolin AreItemsTheSame (@NonNull Word olditem, @NonNull word newshe) {Return Olditem = = Newshe ;} @Override Public Bolin AreContainsTheSame (@NonNull Word olditem, @NonNull Word Newshe) {Return Olditem. Getword () is equal (newed item. getword () in the reccecline () method. Reccechlravine reccechlreve = the method of onCreate after the pandaveoid (r id. reccecline). Final Word List Adapter = New AEA List Adapter (New WordOff List Adapter); Recceclyrev. Set Adapter ; Recceckalravi. Setlaevtmanaaaaagram (New Lanyalyovatmanaaaaaaaaaaaaaar (this); To ensure that everything works run your app. No items, because you haven't yet swayed the data. There is no data in the database. You will include data in two ways: add some data when the database is opened, and add an activity to include words. To delete all content and populate the database when the app is installed, you create a ruodatatabasi. Callback and creation (below). Here is the code to call back the moderock database class. Because you cannot perform database operations on the UI subject, onCreate () uses a pre-set database active to perform a labi-da on a background subject. La Mode deletes the contents of the database, then two words automatically with Hello and the world. Unbow to add more words! Private static Roumadatasi. Return Call Srumdatasikalalbak = New Ruomdatatabasi. Call back () {@Override Public Invalid onCreate (@NonNull Sportsclatedatatabasi db) {Super. onCreate (db);//if you want to keep data through application restaurants,//Comment the following block databases Actions (()-> {//want to populate the database in the background. Voradao Dawood = example. Voordao (); David (). Word = New word (Hello); Enter David (word); Word = New Word (World); Enter David (Word); }); }; }; Then, configure Call phone before you call. Build (on) room. Data-balander ():. Add instring resources to additional return call (Srumdatasikalalbak) values/string .xml: <string name=hint_word>Word... </string > <string name=button_save>Save </string> <string ></string > </Word> </Word,>Not safe because it is empty. Add resources of this color to value/colors .xml: <color name=buttonAble>#FFFFFF</color> Create a new dimension resource file: File > New > Select Android Resources File. From the available qualifying competition, select Dimensions. Set file name: Add these dimensions to values/dimensions .xml: <dimension name=small_padding>8dp</dimension> <diman name=big_padding>16dp</diman> Create a new free Android activity with empty activity template: Select File > New > Activity > Enter New Anuan activity for the name of the empty activity activity. Verify that the new activity has been added to Android. <activity android:name=. NewWordActivity></activity&gt activity_new_word .xml;Up <LinearLayout xmlns:android= android:layout_width=match_parent android:layout_height=match_parent> <EditText android:id=@+id/edit_word android:layout_width=match_parent android:layout_height=wrap_content android:minheat=@dimen/min_height android:fontfamily=sans-serif-light android:hint=@string /hint_word android:inputtype=textAutoComplete android:layout_margin=@dimen/big_padding android:textsize=18sp></EditText> <Button android:id=@+id/button_save android:layout_width=match_parent android:layout_height=wrap_content android:backaround=@color/colorPrimeandroid:text=@string/button_save android:layout_margin=@dimen/big_padding android :textcolor=@color/buttonAble></Button> </LinearLayout> Date: Update code for activity: Public class newkey activity Appcomputertavatty {public static last string EXTRA_REPLY = com. Extends, for example. Android. Literally. Answer; Private Adatitat Madattovardyo; @Override Public Invalid Oncreate (to save bundle) {Super. oncreate (Save). Madattoordyo = Fandaviewyad (R.id.edit_word); Last button button = Fandviobiad (R.id.button_save); button. setOnClickListen (see-> {intentions replyIndent = new intentions (); R.layout.activity_new_word if (TextUtils. isimple (mEditWordView. Gate Text ()) {Sitas result (RESULT_CANCELED, replyIndent);} and {sterling word = mEditWordView. Gate text (). Tostering (). ; replyIndent. putExtra (EXTRA_REPLY, Word); Result of set (RESULT_OK, replayyIntent); The last step is to attach the database to the UI by saving and displaying the current content of the word database in Recccultrev. To display the existing content of the database, add an reviewer who sees the contents in The Videodial Whenever the data is changed, onChanged () is a call back, which calls to the adapter's method of refreshing cached data and updating the list displayed. Create a member variable for the viewodayal: Private Vuordumoodal Moverduyumudal; Use the viewto link your viewwith your activity. When your First start, then Will make a wayofadal. When activity is destroyed, for example, by changing the order, the wayodayal continues. When activity regenerates, the wayodelprovidres take back the current time. For more information, see View View-O-Dayudayal. OnCreate () Under the Recckalravi code block, get a viewfrom the viewodelprovarader: Moverduyumudal = New ViewmodelProvidra (this). Get (.class). Also onCreate (add an reviewer) for the return edit by GetalpahbetaZidovardus(OnChanged () Method Fire when the observed data change and activity is in the preview: Moverduemudayal. getAllWords (). Observe (update the cached copy of words in this, words-> {//adapter. adapter. submetlast (words);}); Describe a request code as an iPad: Public Static Final int NEW_WORD_ACTIVITY_REQUEST_CODE = 1; In this activity, add the code to The Anictaoatiresolat () for ASA activity. If the activity is returned with RESULT_OK, enter the word returned to the database by using the varduemodial insertion () procedure: public false inletatorisolt (int recistcoedi, int resolitcode, intentions data) {super. anictoaterisault (recistcoadi, ResultCode, Data); if (requestCode = = NEW_WORD_ACTIVITY_REQUEST_CODE & & resultCode == RESULT_OK) {Word word = new Word(data.getStringExtra(NewWordActivity.EXTRA_REPLY);; enter mWordViewModel. (word);} and {test. data (getApplicationContact(), R.string.empty_not_saved, Toast.LENGTH_LONG). Show () ;}} In the fib, when the user is used with each other, the activity of the neumen begins. In this fib, find creation and add an onclockclassatner with this code: Flavangektoonbutaon fib = Fandviobiad (R id. fiab); fib. Setanaklastiner (see-> {intention intention = new intention (), it is a new activity .class); sutratatotavatiforresolit (intentions, NEW_WORD_ACTIVITY_REQUEST_CODE) ;}); Now, run your app! When you add a word to the database in the New SAA activity, the UI will automatically update. Now you have a working app, let's have what you built. Here is the structure of the application: The application has components: Your activity: Displays words in a list using a recceclarave and a EA adapter. In their activity, there is an reviewer who is able to get words from the database and is notified when it changes. Newkey Activity: Add a new word to the list. The WorldOdayudayal: Provides ways to access the data screen, and it comes back to Lavedat so that its activity can establish a reviewer's relationship. * Lavedat >: Auto updates in UI components In the activity of The Lavedat, there is an reviewer who sees words outside the database and is notified when it changes. Storage: Manages one or more data sources. Ways to highlight storage for the videodayal to interact with the primary data provider. In this app, this is a room database. Room: There is a jam around and a silate implements the database The room works a lot for you that you did Do it to ye. David: The map method calls for database questions, so that when storage can select a method such as GetalpahbetaZidovardus (), the room * from the word_table order by word asec*. * Word: The institution class which contains a single word. The scenes and activities (and pieces) interact with the data only via The Viewwadal. Thus, it doesn't matter where the data comes from. If you're not already there, you may have a look at the solution code for codelab. You can view the gout store or download the code here: Open the downloaded zip file to the source code pack. It will pack a root folder, a one-point-of-view with android-room-master, which includes the full application. The solution includes unit tests for code room database. Testing is beyond this codelab scope. If you're interested, look at the code. If you need to move an application, see 7 steps in the room after you successfully complete this codelab. Note that it is unbelievably satisfied to delete a lot of class and other code on your Sclotiupanheil. When you have many statistics, consider using the painting library. The painting codelab is here. More on the other architecture components codelabs codelabs of architecture, room, lavedat and vaumudal other architecture