Android studio first steps

Before starting to build an Android application, ITA's important to choose an IDE for programming. The official IDE for Android development is Android Studio. Android Studio is the best overall IDE to start. The download includes IDE support Googleâ ¢ s Android SDK, NDK, Java and Kotlin along with all the tools and emulators Android SDK needed. native Android applications can be written in Java or Kotlin and Android Studio it provides support for both languages. Others to consider are IDE IntelliJ IDEA or Eclipse. Once downloaded, installed and configured the environment, you can create the first Android project. EA ¢ s important to understand the structure of the project of an Android application. The 'src' folder contains all the source files. The folder contains images prime assets, strings and xml layouts that are compiled into an .apk file. The res folder also contains similar items like activity folder, but include alternative or subclasses of these resources to support the screen orientations, different languages, and operating system version. Each file in a directory res ID is a pre-compiled for quick access to these resources. build.gradle is also another important project files. You'll typically see two build.gradle files in the Android project. One is for the project (Project:), and the other is your app form (Form: app). You will have the most work with files build.gradle Module application to configure how Gradle tools behave and build your application in AndroidManifest.xml The manifest file describes the basics of the app and defines its components. Once the code and create the Android app, it is possible to interact with the application through an Android emulator or an Android physical device connected to the computer via USB. An Android emulator simulates a phone, tablet, or Android TV device to your computer. An emulator provides almost all the features of a real Android device, and you can configure them to emulate a specific manufacturer, operating system, and tools to meet your needs. An Android emulator is not a substitute for actual devices, and you should always test on actual devices before shipping to testers and app store market. Check the Android apps on real devices is imperative as the performance of the actual device, different versions of the operating system, the changes made by the manufacturer and firmware vectors can lead to unexpected problems with your application. Test on an actual device gives you a more accurate understanding of how users interact with your app. On the other hand, obtaining physical devices for testing is a logistical challenge. This is where the cloud testing comes into play. With the cloud testing, you can test your application on real devices that are accessible on the cloud. You can run a manual test or run automated tests to ensure the quality of your application. As an open platform, Android developers have a couple of choices for deploying their applications for users, customers, developers, and business associates. The most common market and Official App Android is the Google Play Store, which allows the publication of applications for a market with a wider audience. In addition to the Google Play Store, another popular app market for Android Apps is the Amazon App Store. For beta testing and get your pre-published Android marketplace, you can use the Google Play Console application to get your application in the hands of testers, or specific groups to provide valuable feedback. Before submitting your application to the Google Play store, you should understand a bit 'of Google Play Services and why © ita s important in Of an Android Application. Google Play Services first appearance in 2012 and is a platform (provided by Google), which provides a way to developers to access Google APIs, such as Google Play Services games, Google Maps, locations, announcements for mobile phones and Google Wallet . This section describes how to create a simple Android application. First, you learn to create a "hello, world!" project with Android Studio and execute it. run it. You create a new interface for the app that accepts the user's entry and switch to a new screen in the app to view it. Before starting, there are two fundamental concepts you need to understand Android apps: as they provide more entry points and how they adapt to different devices. The apps provide more input points. Android apps are built as a combination of components that can be recalled individually. For example, an activity is a type of app component that provides a user interface (UI). The "Main" activity starts when the user touches the icon of your app. You can also direct the user to a business from elsewhere, as per a notification or even from a different app. Other components, such as WorkManager, allow the app to perform background activities without a UI. After built your first app, you can learn more about the other app components to the fundamentals of applications. The apps adapt to different Android devices allows you to provide different resources for different devices. For example, you can create different layouts for different screen sizes. The system determines which layout use based on the screen size of the current device. If one of your app's features requires specific hardware, as a camera, you can query when the execution phase if the device has that hardware or not, and then disable the corresponding functions if it does not. You can specify that your app requires certain hardware so that Google Play will not allow app install on devices without them. After built your first app, more information about device configurations at the device's compatibility overview. Where to go from here with these two basic concepts in mind, you have two options. If you prefer to stay in the main documentation, which makes it easy to branch to other arguments for more information on the specific aspects of creating an app, you can proceed to the next lesson to create your first app. However, if you like to follow step-by-step tutorials that explains every step from beginning to end, then consider the Android basis during Kotlin. Content Tip: This CodeLab guides you through the writing of the first flutter app. You may prefer to try to write your first flutter app on the web. If you prefer a conductive-led version of this codeLab, check the following workshop: This is a guide to create your first flutter app. If you have familiarity with the object-oriented code and basic programming concepts such as variables, loops and conditional, you can complete this tutorial. You don't need prior experience with Dart, Mobile or Web programming. This codeLab is part 1 of a two-part CodeLab. You can find part 2 on Google Developers CodeLabs (as well as a copy of this codelab, part 1). What you build in part 1 implement a simple app that generates proposed names for a startup company. The user can select and deselect the names, saving the best. The code generates 10 names at a time lazily. How the user screamed, more names are generated. There is no limit to how far you can scroll a user. The animated gif shows how the app works to completing part 1. How to write a flutter app that seems natural on iOS, Android and the web basic structure of an application of flutter lying and using packages to extend The functionality using hot top-up for a faster development cycle How to implement a stateful widget How to create an infinite list in part in part 2 of this codeLab, add interactivity, change the app's theme and add the possibility of browsing In a new screen (called a flutter path). You need two software to complete this laboratory: the SDK Flutter and an editor. This CodeLab takes Android Studio, but you can use your favorite editor. You can run this codelab using one of the following devices: a physical device (Android or iOS) connected to the computer and set the iOS simulator (requires the installation of Xcode tools) the Android emulator (requires installation in Android Studio) A Browser (Chrome is for debugging) Flutter Each application is also created to fill in the web. In your IDE devices under the drop-down, or from the command line using the flutter device, you should now see Chrome and the listed Web server. The Chrome device starts automatically Chrome. The Web server starts a server that hosts the application so that you can charge from any browser. Use your Chrome device during development in order to use DevTools, and the web server when you want to test on other browsers. For more information, see Building a web application with flutter flutter and write your first application on the web. Step 1: Create the starter Flutter create a simple app, based on Flutter app models, using the instructions in the Getting Started with your first Flutter application. Name the project startup_namer (instead of flutter_app). Tip: If you donâ t see a new Flutter Projecta as an option in your IDE, make sure you have installed the plugin for Flutter and Dart. Youâ ¢ ll mostly edit lib / main.dart, where he lives Dart code. Replace the contents of lib / main.dart. Delete all the code from the lib / main.dart. Replace with the following code, which displays a Hello Worldâ ¢ in the center of the screen. Tip: When you paste the code into your application, the return can become distorted. You can fix this with the following tools flutter: Android Studio and IntelliJ IDEA: Right-click the code and select Reformat code with dartfmt. VS code: Right-click and select Document Format. Terminal: Run flutter . Run the application in the way the IDE describes. You should see both Android, iOS or web output, depending on the device. Android iOS Tip: The first time you run on a physical device, you can take a little 'dependents. Later, you can use hot charging for hotfixes. It also saves running a hot reloading if the application is running. When you are running an application directly from the console using flutter stroke, type r to perform hot charging. Remarks This example creates an application material. The material is a visual design language that is standard on mobile and web. Flutter offers a rich set of widgets material. EA ¢ s a good idea to have a user-material-design: the real voice in the beating section of the pubspec.yaml file. This will allow you to use more material characteristics, such as their default set of icons. The main () method uses the arrow (=>) notation. Use the arrow notation for functions of a single line or methods. The application extends StatelessWidget, which makes the application itself a widget. In Flutter, almost everything is a widget, including alignment, padding, and layout. The widget scaffold, the material from the library, provides a default app bar, and a property of the body that contains the tree widget for the home screen. The widget sub-tree can be very complex. A main widgetÃ ¢ s work is to provide a build () method that describes how to display the widget in terms of other, lower-level widget. The body of this example is made up of a Center widget that contains a child widget text. The Widget Widget Center aligns its sub-tree in the center of the screen. Step 2: Use an external package at this stage, youÃ ¢ ll start using an open-source package called english_words, which contains several thousand English words most used plus some utility functions. You can find the english_words package, as well as many other open source packages on pub.dev. The pubspec.yaml file manages the activities and dependencies for an application Flutter. In pubspec.yaml, add (3.1.5 or higher) to the dependencies list: @@ -8.4 +8.5 @@ 8 8 3 12 + ENGLISH_WORDS: ^ 4.0.0 When viewing the Pubspec.YAML file in Android StudioÃ ¢ s View of the editor, click Pub Get. This pulls the package in the project. You should see the following in the console: $ pub flutter run "Get flutter pub" in startup_namer ... finished process with the output code 0 Performing bar also get self-generate the pubspec.lock file with a list of all packages pulled into the project and their version version In lib / main.dart, import the new package: as typed, Android Studio offers suggestions for libraries to import. It therefore makes the import string in gray, making you know that the imported library is not used (so far). Use the English word package to generate the text instead of using the string Ã ¢ â,¬ å "hello worldÃ ¢ â,¬: @@ -9.14 +10.15 @@ 9 10 ã, class myapp extends withoutWIDGET { 10 11 Ã, @override 11 12 ã, Widget Build (BuildContext Context) {13 + Final Wordpair = Wordpair.random (); 12 14 Ã, Return materialapp (13 15 Ã, Title: 'Welcome to flutter', 14 16 Ã, House: Scaffold (15 17 Ã, AppBar: AppBar (16 18 Ã, Title: Const Text ('Welcome to Flutter') , 17 19 Ã,), 18 - Body: cost center (19 - child: text ("hello world"), 20 + body: center (21 + child: text (wordpair.ascascalcase), 20 22 Ã,), 21 23 Ã,), 22 24 Ã,); Note: Ã ¢ â,¬ Å "Sascal CaseÃ ¢ â,¬ (also known as Ã ¢ â,¬ Å" Upper Camel CaseÃ ¢ â,¬), means that every word In the string, including the first one, begins with a capital letter. So, Ã ¢ â,¬ Å "UpperCamelcaseÃ ¢ â,¬" seppercamelcaseÃ ¢ â,¬. If the app is running, hot recharge to update l 'App running. Every time you click on hot charging or save the project, you need to see a pair of different words, chosen randomly, in the execution app. This is because the combination of the word is generated all 'Interior of the build method, which runs every time Materialiapapp requires rendering or when it activates the flat Form in the flutter inspector. Android problems iOS? If your app is not running correctly, look for typographics. If you want to try some of Flutter debugging tools, check the Devtools suite of debug and profiling tools. If necessary, use the code in the following links to return to the track. Pubspec.yaml Lib / main.dart widgets Apolidi are immutable, which means that their properties cannot change - all values are definitive. State widgets maintain the status that could change throughout the widget. The implementation of a Stateful widget requires at least two classes: 1) A StatefulWidget class that creates an instance of 2) a status class. The STATEFULWIDGET class is, same, unchanged and can be thrown away and regenerated, but the state class persists on the widget life. In this step, you will add a status widget, RandomWords, which creates its status class, _andomWordSstate. You will then use Rommwords as a child inside the Myapp widget designed. Create the boiler code for a status widget. In lib / main.dart, place the cursor after all the code, insert return a couple of times to start on a new line. In your IDE, start typing Spalp. The editor asks if you want to create a state widget. Press RETURN to accept. The kettle code is displayed for two lessons and the cursor is positioned to enter the name of your state widget. Enter RandomWords as your widget name. The RandomWords widget makes little else next to create its status class. Once Randomwords are inserted as the Name of the Stateful widget, the IDE automatically updates the accompanying status class, naming it _andomWordSstate. By default, the name of the status class is set with a subbeat. Area code A identifier with an emphasizing applies privacy in DART language and is a better practical recommended for state objects. The IDE automatically updates the status class to extend the status , indicating that you are using a generic status class specialized for use with RandomWords. Most of the logic of the app resides here - keeps the status for the RandomWords widget. This class saves the list of generated word pairs, which grows infinitely while the user flows and, in part 2 of this laboratory, i They are the word pairs as the user adds or removes from the list by activating the heart icon. Both classes now look as follows: Class Randomwords extend in state of statuswind {@override _randomwordsstate creates () => _andomwordsstate (); } Class _RandomWordSstate extends status {@Override Widget Build (BuildContext Context) {Return {Return }} Update the build method () in _randomWordSstate: Remove the word generation code from MyApp Stuffing the changes displayed in the following Diff: @@ -10.7 +10.6 @@ 10 10 Ã ¢ Class MyApp extends in a way STATALEWIDGET {11 11 Ã, @override 12 12 Ã, Widget Build (BuildContext Context) {13 - Final Wordpair = Wordpair.random (); 14 13 Ã, Return MaterialiaPapp (15 14 Ã, Title: "Welcome to flutter", 16 15 Ã, House: Scaffold (@@ -18.8 +17.8 @@ 18 17 Ã, Title: Const Text ('Welcome a flutter '), 19 18 Ã ¢), 20 19 Ã, body: center (21 - child: text (wordpair.ascascalcase), 20 + child: randomwords (), 22 21 ã,), 22 22), 24 23 Ã,); 25 24 Ã,} Restart the app. The app should behave as before, view a pairing to a word each time you recharge hot or save the app. Tip: If you see a warning on a hot recharge that you may need to restart the app, consider the restart. The notice could be a false positive, but the restart of the app ensures that the changes are reflected in the user's user interface. If your app is not working properly, look for the typefrom. If you want to try some of Flutter debugging tools, consult the Devtools suite of debug and profiling tools. If necessary, use the code at the following link to return to the track. In this step, expand _andomWordSstate to generate and view a list of word combinations. Because the user scrolls the list (displayed in a ListView widget) grows infinitely. The ListView builder factory manufacturer allows you to build a list view lazily, upon request. Add a list _suggestions to class _andomwordsstate for saving suggested combinations. Also, add a variable _Biggerfont to make the size of the larger font. Subsequently, add an _Buildsuggestions () function to the _andomwordSstate class. This method creates the valid of ListView that displays the combination of the suggested word. The ListView class provides a Builder property, ITEMBUILDER, is a factory function and a callback function specified as an anonymous function. Two parameters have passed to the function - the BuildContext and the iterator line, I. The iterator starts at 0 and increases every time the function is called. Increased twice for any suggested word coupling: once per jubles, and once for the divider. This model allows the suggested list to continue to grow as you travel the user. Add a function _Buildsuggestions () to the _andomwordsstate class: The callback itembuilder is called once by suggested word combination and place each suggestion in a fill line. For even lines, the function adds a review line for the combination of the word. For odd lines, the function adds a dividers widget to visually separate the entries. Note that the divider may be difficult to see on smaller devices. Add a high pixel dividers widget before each line in the list. The expression I ~ / 2 divides the for 2 and returns a whole result. For example: 1, 2, 3, 4, 5 becomes 0, 1, 1, 2, 2. This calculates the actual number of Word combinations in the valid list, minus dividers widgets. If you have reached the end of the available words, generals 10 more and add them to the list of suggestions. The _Buildsuggests () function of calls _buildrow () once by pair of words. This function displays each new torque in a log, which allows you to make the most attractive lines in the next step. Add a function _BuildRow () to _RandomWordSstate: in the _RandomWordSstate class, update the build method () to use _Buildsuggestions (), rather than call the word generation library directly. (The scaffold implements the Visual design of the base material.) Replace the body of the method with the highlighted code: in the MyApp class, update the build method () by changing the title and modifying the home widget to be a RandomWords widget: @@ -10, 15 +10.8 @@ 10 10 Ã ¢ Class MyApp extends Statunitensewidget {11 11 Ã, @override 12 12 Ã ¢ Ã, Widget Build (BuildContext Context) {13 13 Ã, Return MaterialApp (14 - Title: "Welcome to flutter ", 15 - Home: Scaffold (16 - AppBar: AppBar (17 - Title: Const Text ('Welcome to Flutter'), 18 -), 19 19 Body: Center (20 - Child: Randomwords (), 21 -), 22 -), 14 + Title: 'Name Startup Generator', 15 + Home: RandomWords (), 23 16 Ã,); 24 17 Ã,} Restart the app. You should see a list of word combinations, no matter how far it flow. Android problems iOS? If your app is not working properly, look for the typefrom. If you want to try some of Flutter debugging tools, consult the Devtools suite of debug and profiling tools. If necessary, use the code at the following link to return to the track. Profile or release Run IMPORTANT: Do not test your app's performance with debugging and hot reload enabled. Until now, you are performing your app in debug mode. The Debug mode offers performance for useful developer performance such as hot reloading and phase debugging. It is not unexpected to see lens performance and janky animations in debug mode. Once you are ready to analyze the performance or release your app, you would like to use the flutter Ã ¢ â,¬ Ã ¢ â,¬ Å "ProfileÃ ¢ â,¬ Ã ¢ â,¬ Å" release "construction method. For Further details, consult the flutter construction methods. IMPORTANT: If you are worried about the size of your app's package, see Measurement of your App's size. Upcoming the app from the 2 congratulations! You wrote an interactive flutter application Which runs both on iOS and Android. In this codelab, you "VE: has created a flutter app from scratch. DART code Written. Take advantage of an external library and third parties. Used hot charging for a faster development cycle. Implemented a state widget. Created an infinite sliding list at Pigia. If you want to extend this app, go to Part 2 on the Google Developers CodeLabs website, where you add the following features: Implement the interactivity by adding a clickable heart icon to save your favorite combinations. Implementing browsing a new route by adding a new screen containing saved favorites. Change the color of the theme, make a whole-white app. app.