Flutter android tutorial pdf

Perlis is an open source framework for creating high-quality, high-performance mobile applications in mobile operating systems - Android and iOS. It provides a simple, simple, powerful, efficient and easy to understand SDK that is easy to understand in Google's own language, a mobile application in darts. This tutorial walks through the basics of the flutter framework, the installation of the Flutter SDK, setting up Android Studio to develop flutter-based applications, the architecture of the flutter framework and developing all types of mobile applications using the flutter framework. Audience This tutorial is prepared for professionals who are eager to pursue a career in the field of mobile applications. This tutorial is intended to make you comfortable getting started with the flutter framework and various features. A prerequisite, this tutorial is based on the assumption that the reader already knows what the framework is and that the reader has a sound knowledge of object-oriented programming and a basic knowledge of Android framework and dart programming. If you're a beginner to any of these concepts, it's a good idea to explore the tutorial slot before you start with Flutter. The Flutter tutorial teaches you how to use the flutter framework to build mobile applications for iOS and Android. Choose from: How to build a layout using the layout mechanism of the flutter. Once you've learned the basic principles, build the layout of the sample screenshots. Expand the simple layout application created in the layout build of the flutter so that you can appappable the icons by adding interaction to the flutter app. Other ways to manage the status of widgets are also discussed. Flutter's animation describes the basic classes of flutter animation packages (controllers, animatables, curves, listeners, builders) and uses various aspects of the animation API to guide the tweed animation progress. Learn how to internationalize flutter apps to internationalize flutter applications. A guide through widgets and classes that allow your app to display content using your language and formatting rules. Update note: Nishant Srivastava updated this tutorial for Flutter 1.7 and Dart 2.4. The original tutorial was written by Joe Howard. Since the explosion of iOS and Android platforms 10 years ago, cross-platform development has been a goal across the mobile development world. The ability to write one app for both iOS and Android can save you a lot of time and effort for your company and your team. There are a number of tools that have been released for cross-platform development over the years, including web-based tools such as Adobe's PhoneGap, powerful frameworks like Microsoft's Jamarin, and new tools like Facebook's React Native. Each set of tools has pros and cons and has achieved a variety of successes in the mobile industry. The latest framework for entering the cross-platform arena is the flutter. Google. Flutter provides fast development cycles, fast UI rendering, unique UI design, and basic app performance on both platforms. The introduction to the Flutter Flutter app is written using the dart programming language, originally the current ECMA standard on Google. Darts share many of the same features as other modern languages, such as Cortines and Swift, and can be converted into JavaScript code. As a platform-to-platform framework, the flutter is most similar to native reactions because it allows for a responsive and declarative programming style. However, unlike react natives, flutterers don't need to use JavaScript bridges that can improve app launch time and overall performance. Darts achieve this using pre-time or AOT compilation. Another unique aspect of darts is that you can also use low-time or JIT editing. JIT compilation using Flutter improves development workflow with hot reload ing during development without a completely new build. As you can see in this tutorial, the flutter framework is built largely around the concept of widgets. In flutter, widgets are not only used for app viewing, but also for full screen and the app itself. In addition to cross-platform iOS and Android development, learning flutter provides a starting point for the development of the Fuchsia platform, an experimental operating system currently being developed by Google. In this tutorial, you can query the GitHub API for team members in your GitHub organization and build a flutter app that displays team member information in a scrollable list: iOS simulator, Android emulator, or both while using the app! In building an app, you can learn the following for flutter: import files and packages using the new Project Hot widget, reload the new project hot and create your own making network calls showing items on the list and you can also learn little darts along the way! :] Starting flutter development can be done on macOS, Linux or Windows. All editors are available as flutter tool chains, but there are IDE plug-ins for IntelliJ IDEA, Android Studio and Visual Studio Code that can facilitate the development cycle. This tutorial uses VS code. To set up development preference guidelines for setting up development machines with the Flutter framework, you can find them here. The basic steps vary depending on the platform, but in most cases: you can download the installation bundle for the development machine's operating system to get the latest reliable release of the flutter SDK extract connecting the installation bundle in the desired location, add a flutter tool to the path to install the flutter framework, including Dart, and install a flutter framework, including Ftod, to install futters if futters are insufficient. The instructions provided on the Flutter website are very well performed and you can easily set up your development environment on the platform you want. The rest of this tutorial assumes that you have set up vs code for flutter development and have solved a problem that Dr. Flutter found. You should also be able to follow pretty well if you are using Android Studio. You'll also need to run an iOS simulator, an Android emulator, or a provisioned iOS device, or an Android device set up for development. Note: To build and test on an iOS simulator or iOS device, you'll need to use macOS with Xcode installed. Create a new project in vs code with the flutter extension installed, and select the View ► command palette to open the command palette... Or hit Cmd Shift -P on MacOS or Ctrl Shift -P on Linux or Windows. Enter The Flutter: Return to hit a new project in the palette. Enter the name ghflutter for the project and hit the return. Select the folder to store the project, and then wait for the flutter to set up the project in the VS code. When the project is ready, the file main.dart opens in the editor. The VS code shows a panel on the left showing the project structure. There are folders for iOS and Android, as well as lib folders with code that includes main.dart and applies to both platforms. You can only work in the lib folder in this tutorial. Replace the code in main.dart with the following code: 'Package:flutter/material.dart'. Vienna Main () = > RunApp (GHFlutterApp)); Class GHFlutterApp statusless widget {@override buildcontext context) {BuildContext context) { Return MaterialApp (title: 'GHFlutter', Home: Scaffold (Appbar: AppBar): Appbar (Title: Text ('GHFlutter'), Main ('GHFlutter'), Center ('GHFlutter'), near the top ,Main () Function near the top to run the app to the operator => operator. There is one class for an application named GHFlutterApp. You can see here that the app itself is a stay-at-home widget. Most entities in the flutter app are state-in-state or state-of-the-state widgets. Create an app widget by redefining the widget build () method. Material Design is using the MaterialApp widget, which provides the following apps with the many components they need: To start this tutorial, right-click on the test file widget_test.dart in the project, select the delete option, and then turn off delete. If you're on macOS, start the iOS simulator. You can also use Android emulators on macOS, Linux, or Windows. If both the iOS simulator and the Android emulator are running, you can switch between them using the menu at the bottom right of the VS code window. Click the cross bug icon in the left panel to switch to debug view. There are no configurations defined at this time. Click No Configuration Select Add configuration. The VS code creates a launch.json file with the following details: You don't need to modify it for this tutorial. Everything is set now to attack F5 or select debugging to build and run the project, start debugging, or tap the Green Play icon. If the debug console is open and running on iOS, you can see that Xcode is used to build the project. If you're running on Android, you'll see the grads being called to perform the build. Here's an app running on the iOS simulator: and runs on android emulator: The slow mode banner you see indicates that the app is running in debug mode. You can stop running apps by clicking the Stop button to the right of the toolbar at the top of the VS code window: Click the icon in the upper left corner of the VS code, or select ► Explorer to return to project view. One of the best aspects of hot reload flutter development is that you can reload the app hot as you change. This is similar to android studio's instant run/enforcement changes. Build and run the app to run on a simulator or emulator. Without stopping the running application, change the application bar string to something else: AppBar (title: text ('GHFlutter app'), now click the hot reload button on the toolbar or simply save the main.dart file: within the second or two you should see the changes reflected in the running application: the hot reload function may not always work and the official documentation doesn't always work, it doesn't do a really good job, but it's a good time to build the UI overall. Instead of keeping all dart code in a single main.dart file, you can import the code from another class that you created instead of importing the file. You now see an example of importing strings, which is helpful when you need to localize a user-facing string. Correctly click a file called strings.dart in the lib folder and select a new file: Add the next import to the top of the class string { static string appTitle = GHFlutter; } main.dart import 'strings.dart'. Change the widget to use a new string class so that the GHFlutterApp class looks like this: Class GHFlutterApp is a statusless widget { build @override widget (BuildContext context) { Return MaterialApp (Title: Strings.appTitle, Home: Scaffold (Appbar: AppBar) Title: Text (String.appTitle), Text: Center: Text: Text: Text: Text: Text: Text: Text: Text: Text: Text: Text: Text} F5 Press to build and run the app and build and run the app, but use a string from the string file. Information, such as a static image of an image view. State pool: A widget that maintains dynamic information and interacts with state objects to do this. State-by-state and state-specific widgets are redrawn from the flutter app for all frames. To start creating your own widget, create a new class at the bottom of main.dart: Class GHFlutter expands state pool widget {create @override () = > GHFlutterState (); } You have created a StatefulWidget subclass and you override the createState() method to create a state object. Now add the GHFlutterState class above GHFlutterState: Class GHFlutterState expands the parameters and states of state<GHFlutter> {} GHFlutterStateGHFlutter. Add a build () override to GHFlutterState: build @override widget (BuildContext context) { { Build as follows(Create: @override widget build (BuildContext context context context) { Return scaffold (appBar: AppBar: AppBar) Title: Text (Strings.appTitle), Body: Text (Strings.appTitle), (Scaffold (Strings.appTitle), } Scaffolding is a container for material design widgets and widget stool hierarchy, each of which contains a body text. By updating GHFlutterApp to use the new GHFlutter widget as a home property, class GHFlutterApp uses the statusless widget {@override Widget BuildContext) { Return MaterialApp (title: Strings.appTitle, Home: GHFlutter() } Build and run the app, you can see the new widget in action: many have not changed yet, but now it is set to build a new widget. Get the 'package: http/http.dart' to http; The http package is not available. This is because it has not yet been added to the project. Go to the pubspec.yaml file and add 'dependencies' and 'cupertino_icons: ^0.1.2' below: cupertino_icons: ^0.1.2 # HTTP package http: ^0.12.0+2 Note: Check the indentation. The indentation of the http package declaration cupertino_icons maintain syassy as the package. Now, when you save pubspec.yaml files, the flutter extension in the VS code is commanded by the flutter pub. The flutter will get the declared http package and the package is available at main.dart. The light is now displayed on main.dart for imports that are not currently being used. The dart app is a single thread, but the dart is </GHFlutter>Run the code on another thread and run asynchronous code that does not block the UI thread using the asynchronous/standby pattern. GitHub will make asynchronous network calls to retrieve the list of team members. Add an empty list with properties at the top of GHFlutterState and add properties that hold text styles: var _members = []; Final _biggerFont = Consensu text style (font size: 18.0); The underline at the beginning of the name makes the members of the class private. To create an asynchronous HTTP call, add a _loadData() method to GHFlutterState: _loadData () Sync {String DataURL = http. Response = http.get (dataURL) waits. setState () {_members = json.decode (response.body); To let darts know that they are asynchronous, we've added _loadData() sync keywords, and there's also a standby keyword for the http.get() call that's blocked. You are using a dataUrl value that is set to the GitHub API endpoint to search for members for your Organization. When the HTTP call is complete, the call is forwarded to setState() which is executed synchronously on the UI thread. In this case, decode the JSON response and assign it to the _members list. Add an initState() override to GHState() that calls _loadData () when the state is reset: @override invalid initState() { super.initState(); _loadData (); } ListView Now you have a dart list of members now in use, and you need a way to show up in the list of UI. Darts can display data in a list by providing a list view widget. ListView works like RecyclerView on Android and UITableView on iOS, and recycles views as users scroll through lists to achieve smooth scrolling performance. _buildRow GHFlutterState () Added method: widget _buildRow (int i) { Return ListTile (title: text (${_members[i]login}, style: _biggerFont); Json returns the ListTile widget, which displays the login values parsed by JSON for Ith members, and is also using previously created text styles. Update ghflutterState's build method so that the console can be configured as ListView.builder: ListView.builder (padding: constellation EdgeInset.all (16.0), item count: _members.length, Item Builder: (Build context context context, int position) {return _buildRow (location); added padding, set the item count on the number of members (_buildRow. you can try a hot reload, but you may need to restart the whole message. Add divider to add separator to the add-on list, double the number of items and return the Divider widget when _members the list is out of position. Lytiton divider (); Final index = position ~/ 2; Return _buildRow (index); }) Be sure to miss *2 in the itemCount. Padding has now been removed from builders with dividers. ItemBuilder returns divider, calculates a new index by integer split, and builds row items using _buildRow(). To try hot reload and you need to add a separate to the list: if you want to add padding back to each row, you want to use a padding widget (within _buildRow) () : Widget _buildRow (int i) {return padding (padding: const EdgeInset.all (16.0), Child: ListTile (title: text ($_members [i]_biggerFont} ListTile is now a padded widget. You can see the padding of the column by hot reloading, but _members not in the partition. Reception: '$login'); } }) Members have login properties and constructors that throw errors if the login value is null. GHFlutterState has updated its _members declaration to list member objects var _members = <Member>[;; Update _buildRow () To use the login properties of member objects instead of using the map's login key title: text (${_members[i]login}, style: _biggerFont) Now _loadData () to set state() to update the decoded map to the member object and add it to the member list: setState() Final Member JSON = json.decode (response.body); (Var memberJSON of member JSON) {Final Member =Member JSON [login] ;),}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}} []],'_members}}}}},'},'};};}}} If you try to reload hot, but stop and press F5 to build and run the app, you may see the same screen as before. Now add that avatar to the member class and display it in the app. Update the member class to add an avatar Url property that cannot be null: class member { final string login; final string avatar Url; member (this.login, this.avatarUrl) {(login == null) {ThrowError (member login cannot be null) Reception: '$login'); } (Avatar Url == null) { throw argument error (the member's avatar Url cannot be null. Receive: '$avatarUrl') ;); } _buildRow updated with network image and CircleAvatar widget: widget _buildRow (int i) { return padding (padding: const EdgeInset.all (16.0), Child: ListTile (Title: Text (${_members[i]login}, Style: _biggerFont), Leading: CircleAvatar (Background Color) >Member >;),)); } If you set your avatar as the main property of ListTile, it will appear before the title within the row. You also used a color class to set the background color for the image. Now update _loadData () use the value avatar_url on the map when creating a new member: final member = member (member JSON [login], member JSON [avatar_url]); Use F5 to stop, build, and run your app. Member avatars are now displayed on each row: most of the code is in the main.dart file. To make your code cleaner, you can refactor widgets and other classes you've added to your file. Create a file called member.dart and ghflutter.dart in the lib folder. Go to member.dart and GHFlutterState and GHFlutter classes to ghflutter.dart. The import statement does not require a member.dart, but the import of ghflutter.dart is as follows: import 'dart:convert'; Get the 'package: http/http.dart' to http; Import 'Package:Flutter/Material.Dart'; Get 'member.dart'; Get 'strings.dart'. You also need to update the import in main.dart so that the entire file is configured as 'package:flutter/material.dart'. Importing 'ghflutter.dart'; Get 'strings.dart'. Vienna Main () = > RunApp (GHFlutterApp)); Class GHFlutterApp is a statusless widget {@override widget build (BuildContext context) { Return MaterialApp (title: strings.appTitle, Home: GHFlutter), } Hitting F5 to build and run the app doesn't change, but the code is now a little cleaner. :] Adding a theme allows you to easily add themes to your app by adding materialapp theme properties you create in main.dart: Class GHFlutterApp statusless widget @override (BuildContext context) {Return MaterialApp (title: Strings.appTitle, Theme: ThemeData: Primary Color); GHFl.green.shade.800} Uses green shades as material design color values for the subject. Build and make a new theme by hitting f5 running: most of the app screenshots have come from Android emulators. You can also run the final theme app in iOS Simulator: now that is what I call cross-platform! :] Where do I go from here? You can download a completed project using the download button at the top or bottom of this tutorial. You must be able to open the project in vs code or android studio. Just open the root folder and open it in the VS code. You must get the package before you can run the project. With Cmd +Shift+P, open the command palette and get flutter: package commands. To open your final project in Android Studio, open your existing Android Studio project from Welcome to Android Studio and select the root folder for your final project. Then you need to select Open The Flutter Settings and go to the environment settings to go to the flutter, and then go to the flutter to set the flutter SDK path to where the flutter git repository was replicated. If necessary In Android Studio, select Get Dependency on the 'Package is not running' line. In the Project panel, you can select Project View to view the file. If you're on macOS, you can run your app from Android Studio to an Android emulator or an iOS simulator. Try it, it's pretty cool! There's more to learn about flutter and darts. The best starting point is the main flutter page of flutter.dev. You'll find a lot of great articles and other information. See available widgets. Here is a similar guide for Android developers who switch using Flutter. Here's a similar guide for responsive native developers. Stay tuned for more flutter tutorials and screencasts on our Flutter page! Share feedback, results, or ask questions in the forums below or in the forums. I hope you enjoy educating you to start with the flutter! Raywenderlich.com you start your tools and libraries, newsletters are the easiest way to keep everything you need to know as a mobile developer up to date. Get weekly digestion of our tutorials and courses, and get a free in-depth email course as a bonus! 4.7/5 76 rating