[Connecting Flutter application to Localhost | by Seddiq Sorush | Medium](https://medium.com/@podcoder/connecting-flutter-application-to-localhost-a1022df63130)

**Connecting Flutter application to Localhost**

Connecting your flutter application is not an easy task, as it seems to be. Especially when you want to run your Flutter application, which is connected to the localhost on different platforms, like, iOS emulator, Android emulator or on a real device at the same time.

You can watch the tutorial on YouTube

**Running Flutter Application on iOS emulator**

iOS emulator uses the current machine localhost i.e 127.0.0.1 or localhost as the localhost. So, if you want to connect to localhost on iOS emulator you just need to point to the

[http://localhost](http://localhost/):SPECIFIC\_PORT // port 80 by default  
 or   
[http://127.0.0.1:SPECIFIC\_PORT](http://127.0.0.1:SPECIFIC_PORT/) .For Example  
<http://localhost:8000/api/users>

**Running Flutter Application on Android emulator**

Android emulator uses 10.0.2.2 for pointing to the localhost. So, if some user’s info is available on a url like , <http://localhost:8000/api/users> and you want to access it from your Flutter application which is running on Android emulator you need to write:

<http://10.0.2.2:8000/api/users>

**Running Flutter Application on a Real Device**

Localhost means something different on the real device, basically in means to look inside the current device.

To run a Flutter application which is connected to the localhost, on a real device, first the ***real device*** and the ***machine*** which acts as localhost should be connected on the ***same network.***

And then the application should be ***self hosted*** on the machine, for example, if you are using Laravel as your backend solution, and you want to access it from your Flutter application on a real device you need to run your Laravel Application as bellow:

$ php artisan serve --host 0.0.0.0 --port 8000

So, this well host this **Laravel** application on the current machine IP address, suppose your current machine IP address is 192.168.0.243 then you can access user’s info from your Flutter application running on the real device as follow:

http://192.168.0.243:8000/api/users

If you are having a pure **PHP** application for your backend then you can host the application with the following procedure:

cd path/to/your/app // Navigate where you application isphp -S 0.0.0.0:<PORT\_NUMBER>// For example  
php -S 0.0.0.0: 3000   
// Will host the app at http://192.168.0.243:3000

So, this will enable you to access the localhost from the real device as:

http://192.168.0.243:3000/api/users

Note: PORT\_NUMBER is an integer from 1024 to 49151.

For an **ASP.NET** core application as your backend you need to configure your ***Program.cs*** file as follow and then run your application:

var host = new WebHostBuilder()  
 .UseKestrel()  
 .UseContentRoot(Directory.GetCurrentDirectory())  
 .UseUrls("http://localhost:5000", "http://192.168.0.243:5000")  
 .UseIISIntegration()  
 .UseStartup<Startup>()  
 .Build();

So, you’ll be able to have access to the localhost from the real device as:

http://192.168.0.243:5000/api/users

Also you need to make sure the Firewall on windows allow you to connect. In addition, both your device and computer should be connected on **the same network**.

The good thing about self hosting is that, it can be accessed with same url from iOS emulator, Android emulator and on the real device.

**Conclusion**

Accessing the localhost from Flutter application is not the same on iOS emulator, Android emulator and the real device.

You can self host your application on an IP, an access it from all the platforms at the same time.

The key point to remember, when you want to connect your Flutter application to a real device is that, both the device and PC should be connected to the same network.

[Flutter Localization. While I was working on a multi-language… | by Seddiq Sorush | Medium](https://medium.com/@podcoder/flutter-localization-a39402757a42)

**Flutter Localization**

If your app might be deployed to users who speak another language then you’ll need to “internationalize” it. That means you’ll need to write the app in a way that makes it possible to “localize” values like text and layouts for each language or “locale” that the app supports. Flutter provides widgets and classes that help with internationalization and the Flutter libraries themselves are internationalized.

Flutter App Localization is an easy task but yet a tricky one, specially when you want to change the application’s language without restarting the application, Native Components Translation also RTL and LTR support .

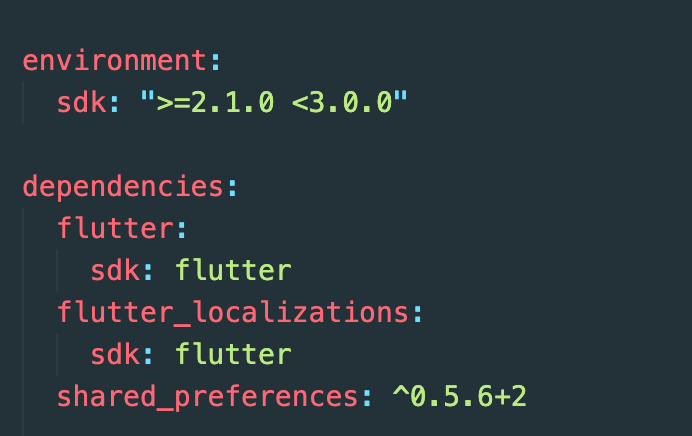
In this tutorial we will develop an application to support RTL and LTR languages, Translate Native Components like DatePicker, TimePicker and Dialogs, also persists the user’s selected language, so, next time when the user is opening the app, user’s preferred language is selected automatically.

You can watch video tutorial on PodCoder, YouTube

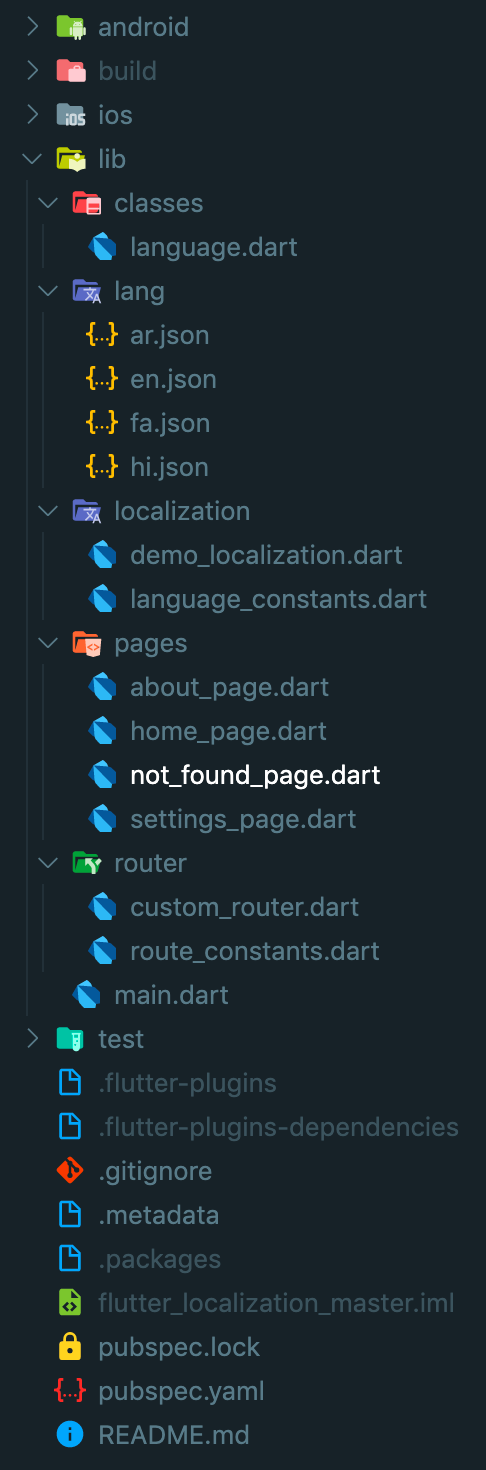
Create an empty flutter application by running the following command on your terminal

flutter create flutter\_localization\_demo

To localize a Flutter application,**flutter\_localizations** package is required, also, to persist the selected language I am using **shared\_preferences**, so, Open the the project in your favorite IDE, head to **pubspec.yaml** file and add these two package as follows, be careful of indentation.



The project structure in this tutorial is as follows.



To easy handle the language translation use JSON files. so, if you want to add support for a new language in future, you can accomplish this job by just by copying the default JSON file and translating it’s values to the new language and adding it to the main supporting languages.

In this tutorial we are developing an application to support two languages i.e English, Hindi→ LTR and Persian (Farsi ), Arabic → RTL . so, inside **lib\lang**directory create two JSON files i.e**en.json** , **fa.json, ar.json**and**hi.json**

en.json ( English language Translations)

fa.json ( Farsi Language Translations)

ar.json ( Contains Arabic language Translations)

hi.json ( Hindi language Translations)

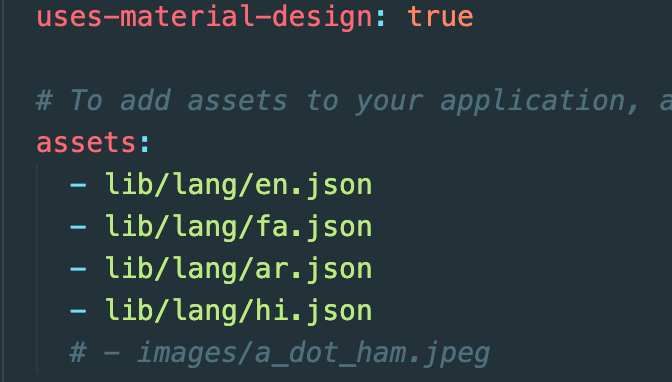
Look at the above JSON files, for both English and Persian languages keys are the same, only values have changed, which is the equivalent translation in the related language.

Now create **demo\_localization.dart** inside **localization** directory

AppLocalizations class has two methods, load method to load the data from the JSON files and map the data to a **Map<String, dynamic>** and **translate**method to accept a key and return its translated value.

Flutter localization uses delegates for this initialization, **\_DemoLocalizationDelegate** is also present in this file

To load the data from the JSON files present in **lib\lang** directory, first we need to load JSON files as an asset in**pubspec.yaml** file.



Now Create three pages **HomePage, SettingsPage**and **AboutPage**inside **lib\pages**

Home Page

About Page

Settings Page

In order to handle the localization easily create a **language\_constants.dart** file. Inside this file we will have all the constants and methods which we use to easily handle the localization.

language\_constants.dart

Now delete everything inside the main.dart file and paste the following code

main.dart

Notice the **main** class is a **StateFull widget**, and also has a **static** method called **setLocale** which accepts a new **Locale.**

static void setLocale(BuildContext context, Locale newLocale) async {   
 \_MyAppState state = context.findAncestorStateOfType<\_MyAppState>(); state.setLocale(newLocale);}

Once this method is called, first of all it will find the **state** of the **main** using ***context.findAncestorStateOfType<T>()***;

\_MyAppState state = context.findAncestorStateOfType<\_MyAppState>();

Then it will call a **state** method called **setLocale** present in the **main’s class** state, and passes the new Locale to it.

setLocale(*Locale* locale) {  
 setState(() {  
 \_locale = locale;  
 });  
}

In this way the **state** of the **main** class will **change** and it forces the Widget to **recreate** itself. So, it will change the overall application **Locale**.

In this way this method can be called from any Screen of the application by

Local newLocale = Locale('en', 'US');  
MyApp.setLocale(newLocale);

At the end Configure the Router to Navigate between the pages

Here is the link to the Github repository of the demo Application