

GetX

- تعد **GetX** من المكتبات التي تقدم أداء عالي وسهولة بالاستخدام والصيانة وخاصة للمبتدئين ودقة ومنظمة للخبراء.
- تعتبر **Ecosystem** كامل يشمل العديد من الخدمات الهامة منها:

State Management.1

Routing .2

Localization .3

Dependency injection .4

- فهي آمنة وثابتة ومحدثة بشكل مستمر وتعتمد **GetX** على ثلاثة مبادئ أساسية:

1- الأداء: يركز **GetX** على الأداء العالي والمنظم والحد الأدنى من استهلاك الموارد في الذاكرة. فلا يستخدم الـ **Streams** أو **ChangeNotifier**.

2- إنتاجية: يستخدم **GetX** صيغة سهلة وبسيطة وممتعة للمطور.

سيوفر ساعات من التطوير والصيانة في المستقبل .
فبشكل عام ، يجب أن يهتم المطور بإزالة controllers من الذاكرة بعد الانتهاء من استخدامها، ولكن مع **GetX** تم إزالتها بشكل تلقائي.

إذا كنت تريد الاحتفاظ بها في الذاكرة، فيجب استخدام خاصية "permanent: true"

في **dependency** الخاصة بال controller . وبذلك ستكون أقل عرضة لخطر الاعتماد غير الضروري على الذاكرة. وعادة يكون **dependency** بوضع **lazy** بشكل افتراضي (اي لا يتم حقنه الا عند الحاجة لاستخدامه).

GetX

3- التنظيم: يسمح نظام GetX بفصل كامل لكل من: view, presentation logic, business logic, dependency injection, navigation clean code فيكون الكود منظم وسهل القراءة والصيانة، لا تحتاج لوجود context للتنقل بين المسارات، لذلك لست مضطراً لأنخذ ال widget tree بعين الاعتبار. وبالتالي يمكنك الاستغناء من مبدأ الوراثة.

لن تحتاج إلى ال widget tree داخل ال MultiProviders لحقن ال Controllers الخاصة بك. لهذا الغرض فإن GetX تستخدم ميزة الحقن الخاصة بها مما يؤدي إلى فصل ال dependency injection عن ال view بشكل كامل.

- لا يعد GetX ضخماً وبطيئاً وذلك لأنه يفصل بين وظائفه أيضاً، أي أنه في حال تم استخدامه لل routing فقط لن يتم تشغيل compile لبقية الوظائف مثل ال State Management .GetX
- في النهاية سيبقى Flutter يدعم GetX طالما أنFlutter موجود. و ال code الذي يُكتب بالنسبة ل GetX يمكن استخدامه نفسه في android, ios, windows, mac,linux



GetX

3- التنظيم: يسمح نظام GetX بفصل كامل لكل من: view, presentation logic, business logic, dependency injection, navigation clean code فيكون الكود منظم وسهل القراءة والصيانة، لا تحتاج لوجود context للتنقل بين المسارات، لذلك لست مضطراً لأنخذ ال widget tree بعين الاعتبار. وبالتالي يمكنك الاستغناء من مبدأ الوراثة.

لن تحتاج إلى ال widget tree داخل ال MultiProviders لحقن ال Controllers الخاصة بك. لهذا الغرض فإن GetX تستخدم ميزة الحقن الخاصة بها مما يؤدي إلى فصل ال view عن ال dependency injection بشكل كامل.

- لا يعد GetX ضخماً وبطيئاً وذلك لأنه يفصل بين وظائفه أيضاً، أي أنه في حال تم استخدامه لل routing فقط لن يتم تشغيل compile لبقية الوظائف مثل ال State Management .GetX
- في النهاية سيبقى Flutter يدعم GetX طالما أنFlutter موجود. و ال code الذي يُكتب بالنسبة ل GetX يمكن استخدامه نفسه في android, ios, windows, mac,linux



Getx

- تقدم GetX ميزة routing بين الصفحات بطريقة مبسطة وسهلة الاستخدام وبأداء عالي .

• يجب اتباع الخطوات التالية:

1- إضافة ملف pubspec.yaml get package dependencies:

get: ^4.6.6

2- تضمين المكتبة ضمن الكود البرمجي

```
import 'package:get/get.dart';
```

3- استخدام `GetMaterialApp` بدلًا من `MaterialApp` وهذا يُجب التنوية وتوضيح الفرق بشكل مبسط

GetMaterialApp && MaterialApp

باختصار class هو GetMaterialApp لديه نفس خصائص properties - functions- من حيث MaterialApp properties (abilities) من دون أي تعديل عليها..، مضافاً له خاصية بال Get مثل: (...getPages, initialBinding)

ملاحظة : عند استخدام get ک StateManagement لا داعي لاستخدام dependency management نكتفي ب GetMaterialApp

(تكون `routes`, `snack bars`, مهمّة لـ `GetMaterialApp` `internationalization`, `bottom sheets`, `dialogs`



GetX

4- إضافة خاصية `getPages` وإسناد مصفوفة من الـ `pages` لها.

```
GetMaterialApp(  
    getPages:[  
        GetPage( name: "/page1",  
                 page: Page1( )),  
        GetPage( name: "/page2",  
                 page: Page2( ))  
    ],)
```

5- استخدام التعليمات الموجودة بالجدول أدناه للانتقال بين الصفحات عند حدث معين :

Flutter Navigator	GetX
<code>Navigator.of(context).push(MaterialPageRoute(builder: (context) => Page()));</code>	<code>Get.to(Page());</code>
<code>Navigator.of(context).push Replacement(MaterialPageRoute(builder: (context) => Page()));</code>	<code>Get.off(Page());</code>
<code>Navigator.of(context).pop();</code>	<code>Get.back();</code>
<code>Navigator.of(context).push AndRemoveUntil(MaterialPageRoute(builder: (context) =>Page()), (route) => false);</code>	<code>Get.offAll(Page());</code>
<code>Navigator.of(context).maybePop();</code>	<code>Get.maybePop();</code>

GetX

```
Navigator.of(context).push  
Named("/page1");
```

```
Get.toNamed("/page1");
```

```
Navigator.of(context).  
pushReplacementNamed(  
"/page2");
```

```
Get.offNamed("/page2");
```

6- يمكن تمرير arguments عند الانتقال لصفحة معينة على الشكل التالي :

- Get.to(SecondScreen(), arguments: 123);
- Get.to(SecondScreen(), arguments: 456.789);
- Get.to(SecondScreen(), arguments: 'Hello World');
- Get.to(SecondScreen(), arguments: ['Apple', 'Banana', 'Cherry']);
- Get.to(SecondScreen(), arguments: {'name': 'John Doe', 'age': 30});
- User user = User(id: 1, name: 'Jane Doe');
Get.to(SecondScreen(), arguments: user);



GetX

7- تستطيع إضافة animation لانتقال كالتالي:

```
Get.toNamed('/homeScreen', transition:  
Transition.rightToLeftWithFade),
```

وإضافة ميزات أكثر للanimation

```
Get.toNamed('/homeScreen',  
transition: Transition.custom(  
animation: Tween<double>(  
begin: 0,  
end: 1).animate(  
CurvedAnimation(  
parent: Get.keyCurrent!.animation!,  
curve: Curves.easeInOut)),  
duration: Duration(seconds: 1),  
reverseDuration: Duration(seconds: 1),  
curve: Curves.easeInOut,  
opacity: 0.5,  
left: 50,  
right: 50,  
up: 50,  
down: 50, )  
)
```



State Management

تعتبر GetX من المكتبات السهلة والبسيطة التي تستخدم من أجل عمليات ال state management مما يجعلها خياراً ممتازاً للمطوريين الذين يبحثون عن طريقة فعالة للقيام بال state .management

بشكل بسيط وحسب ما هو متعارف عليه بالنسبة لل state فأنـت في البداية بحاجة إلى ما يسمى .GetxController

ما هو GetxController؟ وماذا يحتوي؟ ولماذا يتم استخدامه؟

هو جزء أساسي من إطار العمل GetX في Flutter، يحتوى على جميع الأمور الخاصة بال logic من متغيرات وتوابع والتي يمكن الوصول إليها من قسم ال view، يتم استخدامه من أجل ال state management والتخزين المؤقت والعمليات المتعلقة بالبيانات داخل التطبيق.

وهذا مثال بسيط يوضح كيفية عمل ال controller ومحظياته:

```
● ● ●  
1 class CounterController extends GetxController {  
2     int count = 0;  
3  
4     void increment() {  
5         count++;  
6         update();  
7     }  
8  
9     void decrement() {  
10        count--;  
11        update();  
12    }  
13 }
```

ملاحظة: update() هي الدالة المسؤولة عن عمليات التحديث التي تظهر للمستخدم في UI.



State Management

يتميز **GetBuilder** عن غيره بأنه أسرع وأقل استهلاك لل**Memory** ويستخدم لتحديث جزء معين من الشاشة عند حدث معين.

```
Row(  
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
    children: <Widget>[  
        FloatingActionButton(  
            onPressed: () => controller.increment(),  
            child: const Icon(Icons.add),  
        ),  
        FloatingActionButton(  
            onPressed: () => controller.decrement(),  
            child: const Icon(Icons.remove),  
        ),  
    ],  
)
```

كود الأزرار

```
GetBuilder(  
    init: CounterController(),  
    builder: (controller) => Text('Count: ${controller.count}'),  
)
```

كود Text

: init

المسؤول عن تحديد ال controller الذي سيقوم بإجراء التغييرات في ال widget التي يعتبر ال GetBuilder بمثابة parent لها.

: builder

عبارة عن دالة مسؤولة عن القيام بعملية ال widgets rebuild لـ controller التي تحتويها وذلك في حال حصول تحديث للبيانات الخاصة بال controller.



State Management

هل هناك خاصية بال life cycle خاصة GetxController

نعم يوجد life cycle initial تبدأ بال initial وتنتهي بال close، حيث يمكن الاستفادة من كل مرحلة من مراحل حياته GetxController

: **onInit()** •

تقابـل الدـالـة (initState()) وتنـفذ عـنـدـمـا نـقـوم بـعـمـلـيـة init لـلـ GetxController

: **onReady()** •

تنـفذ بـعـد تـنـفيـذ الدـالـة (build()) الـتـي تـحـتـوي عـلـى شـيـء خـاص بـ GetxController

: **onClose()** •

تـكـافـئ الدـالـة (dispose()) وتنـفذ عـنـد نـهـاـيـة حـيـاة الـ GetxController



State Management

:Getx & Obx

- يعتمدان على مبدأ Stream بمعنى أي تغييرات تتم بالcontroller مباشرة سيستمع لها ال view فيتم تحديث الشاشة تلقائياً.
- الذي يميزهما بأنهما reactive مع التطبيق وتحسين من كفائته.
- ضمن ال controller يتم تمييز observable variables بإضافة أنواع خاصة بها:

RxInt - RxDouble - RxString - RxBool -)

(RxMap<k,v> - RxList<T> ...ex

يتم تعريفها على النحو التالي: RxInt count = 1.obs;

 **ملاحظة:** تدل obs أيضاً على أن المتغير من نوع observable

- لدينا المثال التالي:
عمر counter عند الضغط على زر increment يتم زيادة العدد عشر مرات بمقدار 1 كل ثانية وبالتالي يتم تحديث الشاشة تلقائياً عند كل زيادة وبالمثل عداد النقصان ينقص ال controller من قيمته الحالية ليصل إلى قيمة صفر.
- ضمن ال controller تم كتابة logic من توابع الزيادة والنقصان .count والمتغير
- ضمن view بالبداية تم حقن ال controller على النحو التالي:

```
● ● ●  
1 var controller = Get.put(CounterController());
```

```
1 class CounterController extends GetxController {  
2     RxInt count = 0.obs;  
3  
4     void increment() async {  
5         for (int i = 0; i < 10; i++) {  
6             await Future.delayed(const Duration(seconds: 1),);  
7             count++;  
8         }  
9     }  
10  
11    void decrement() async {  
12        for (int i = count.value; i > 0; i--) {  
13            await Future.delayed(const Duration(seconds: 1));  
14            count--;  
15        }  
16    }  
17 }  
18
```

ملاحظة : يتم إضافة كلمة **value**. بعد المتغير وذلك للإشارة إلى القيمة الحالية للمتغير من نوع **observable** ولا تشير إلى المتغير بحد ذاته.

ضمن الأزرار تم استدعاء توابع **increment** و **decrement** بال**controller** أخيراً تم استخدام دالة **Obx** عند الجزئية (**widgtes**) التي تستقبل تغييرات من الـ **controller** وتتحدى تلقائياً..



```
1 Column(  
2   mainAxisAlignment: MainAxisAlignment.center,  
3   children: <Widget>[  
4     //TODO: Here add the Obx Function  
5     Obx(() => Text('Count: ${controller.count.value}')),  
6     const SizedBox(height: 20),  
7     Row(  
8       mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
9       children: <Widget>[  
10      FloatingActionButton(  
11        onPressed: () => controller.increment(),  
12        child: const Icon(Icons.add),  
13      ),  
14      FloatingActionButton(  
15        onPressed: () => controller.decrement(),  
16        child: const Icon(Icons.remove),  
17      ),  
18    ],  
19  ),
```

GetBuilder

- 1- بمجرد استدعاء دالة (update()) الموجودة ضمن class معين وذلك عند حدث ما في أي مكان بالتطبيق يتم rebuild لكافة GetBuilder التي تستمع لهذا controller .class
- 2 - غير تفاعلية (فهي تحتاج حدث معين واستدعاء دالة update حصراً لتحديث الشاشة) ولا تعتمد مبدأ Stream.
- 3 - أداء أفضل واستهلاك موارد أقل.
- 4 - تحتاج أن تستمع ل controller معين <HomeController>

Obx && Getx

- 1- عند تحديث قيمة المتغير في controller يتم عمل rebuild للأجزاء الشاشة التي فقط تستخدم هذا المتغير أو الدوال التي تستخدمه (عند تحديث قيمة المتغير count سيتم فقط إعادة بناء العداد بالشاشة وإذا كان هناك دالة مجموع تستخدم count سيتم تحديث الجزء الخاص بها أيضاً)
- 2 - تفاعلية reactive تعتمد مبدأ Stream
- 3 - أداء أسوأ وأبطأ واستهلاك ذاكرة أكبر
- 4 - إن Obx لا داعي لأن تستمع ل controller معين مثل GetX<HomeController> أن تستمع لأكثر من controller داخلها ويتم تعريفهم في أعلى الكود (الحقن)

Dependency Injection

يوجد للحقن العديد من المزايا أهمها هو عمل initial لل controller والسماح للمطور بالوصول من ال view إلى محتويات ال controller.

: put () -1

تعتبر هذه الدالة مسؤولة عن حقن ال controller مباشرة بحيث يمكنك استخدام جميع محتويات ال controller من متغيرات وتوابع.

```
1 class CounterController extends GetxController {  
2     int count = 0;  
3  
4     void increment() {  
5         count++;  
6         update();  
7     }  
8  
9     void decrement() {  
10        count--;  
11        update();  
12    }  
13 }
```

Controller Class

```
1 class _HomePageState extends State<HomePage> {  
2     CounterController controller = Get.put(CounterController());  
3     @override  
4     Widget build(BuildContext context) {  
5         return Scaffold(  
6             appBar: AppBar(title: const Text('Simple Counter')),  
7             body: Center(  
8                 child: Column(  
9                     mainAxisAlignment: MainAxisAlignment.center,  
10                    children: <Widget>[  
11                        GetBuilder<CounterController>(  
12                            builder: (controller) => Text('Count: ${controller.count}'),  
13                        ),  
14                        const SizedBox(height: 20),  
15                        Row(  
16                            mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
17                            children: <Widget>[  
18                                FloatingActionButton(  
19                                    heroTag: "btn1",  
20                                    onPressed: () => controller.increment(),  
21                                    child: const Icon(Icons.add),  
22                                ),  
23                                FloatingActionButton(  
24                                    heroTag: "btn2",  
25                                    onPressed: () => controller.decrement(),  
26                                    child: const Icon(Icons.remove),  
27                                ),  
28                            ],  
29                        ),  
30                    ],  
31                ),  
32            ),  
33        );  
34    }  
35}
```

Injection with Put()

Dependency Injection

: lazyPut () -2

كمبدأ عمل تعتبر هذه الدالة قريبة جداً من الدالة put ولكن ما يميزها أنه لا يتم حقن ال controller بشكل مباشر فور تنفيذ التعليمة lazyPut() وإنما عند الحاجة له فقط، فهو يتم تهيئته للحقن فيما بعد.

: find () -3

نستخدم هذه الدالة من أجل الوصول من ال view إلى controller قد تم حقنه مسبقاً ولا زالت حياته مستمرة ضمن التطبيق أو أنه مهياً للحقن

ملاحظة: عند استخدام دالة lazyPut في نفس الصفحة التي تستخدم controller يتم عمل lazyPut مباشرة وذلك فور انشاء الصفحة وتنفيذ تعليمة create.

```
1 class _HomePageState extends State<HomePage> {
2     @override
3     void initState() {
4         Get.lazyPut<CounterController>(() => CounterController());
5         super.initState();
6     }
7
8     @override
9     Widget build(BuildContext context) {
10        return Scaffold(
11            appBar: AppBar(title: const Text('Simple Counter')),
12            body: Center(
13                child: Column(
14                    mainAxisAlignment: MainAxisAlignment.center,
15                    children: <Widget>[
16                        GetBuilder<CounterController>(
17                            builder: (controller) => Text('Count: ${controller.count}'),
18                        ),
19                        const SizedBox(height: 20),
20                        Row(
21                            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
22                            children: <Widget>[
23                                FloatingActionButton(
24                                    heroTag: "btn1",
25                                    onPressed: () {
26                                        Get.find<CounterController>().increment();
27                                        print(Get.find<CounterController>().count);
28                                    },
29                                    child: const Icon(Icons.add),
30                                ),
31                                FloatingActionButton(
32                                    heroTag: "btn2",
33                                    onPressed: () => Get.find<CounterController>().decrement(),
34                                    child: const Icon(Icons.remove),
35                                ),
36                            ],
37                        ),
38                    ],
39                ),
40            ),
41        );
42    }
43}
```

Dependency Injection

بارمترات دوال Put و lazyPut

:permanent - 1

هو عبارة عن بارامتر خاص بالدالة put يتم استخدامه لحفظ قيم المتغيرات عندما يتم التنقل بين الشاشات.

permanent: true يتم حفظ القيم
permanent: false تعود القيم إلى حابتها الإبتدائية

:fenix - 2

.lazyPut يشبه بارامتر permanent ولكنه خاص بالدالة

:tag - 3

يتم استخدام هذا البارامتر من أجل تمييز مجموعة ال objects التي تم إنشاءها من نفس ال controller باستخدام تعليمات الحقن.

مثال:

```
Get.put<UserSession>(UserSession('User1'), tag: 'user1');  
Get.put<UserSession>(UserSession('User2'), tag: 'user2');
```

// الوصول إلى جلسة المستخدم الخاصة

```
var user1Session = Get.find<UserSession>(tag: 'user1');
```

في هذا المثال، تم إنشاء 2 objects مختلفين من ال UserSession، تم إنشاء Tag مختلف ('user1' و 'user2'). يمكن الوصول إلى كل object بشكل فردي باستخدام الدالة Get.find مع تمرير ال Tag المناسب كعلامة.

هذا الأسلوب يفيد بشكل خاص في الحالات التي تحتاج فيها إلى إدارة عدة objects من نفس ال class، مثل الحفاظ على حالات مختلفة من نفس ال class.

Binding

١ - عند التوجه إلى صفحة جديدة باستخدام `:GetPages`

```
Yellow Red Green
GetPage(
  name: SplashScreen.routeName,
  page: () => const SplashScreen(),
  binding: SplashBinding()
),
```

٢ - عند تشغيل ال `application` مباشرةً:

```
Yellow Red Green
GetMaterialApp(
  home: const SplashScreen(),
  initialBinding: RootBinding(), // هنا يتم الحقن بشكل مباشر
  getPages: [
    GetPage(
      name: SplashScreen.routeName,
      page: () => const SplashScreen()),
    GetPage(
      name: HomeScreen.routeName,
      page: () => const HomeScreen(),
      children: [
        GetPage(
          name: CategoryScreen.routeName,
          page: () => const CategoryScreen()),
        GetPage(
          name: AboutScreen.routeName,
          page: () => const AboutScreen()),
        GetPage(
          name: CartScreen.routeName,
          page: () => const CartScreen()),
      ],
    );
);
```

Binding

٣- من خلال التنقل بشكل مباشر من صفحة إلى صفحة:



```
Get.offAll(  
() => const InitialScreen(),  
binding: InitialScreenBindings(),  
)
```

- بعد الإطلاع على هذه الطرق الثلاث لعمليات الحقن أصبح بإمكانك أن تتحكم أكثر في اللحظة التي سيكون فيها ال controller جاهز للعمل، بالإضافة إلى عزل عملية الحقن عن الكود الخاص بال view كما ذكر في البداية.



Localization GetX

من الميزات المهمة التي تدعمها GetX الـ Localization وذلك لجعل تطبيقك يدعم عدة لغات.. هناك مجموعة خطوات بسيطة عليك اتباعها:

1. إنشاء ملف `.locale.dart`.
2. إنشاء `class MyLocale` والذي سيضم جميع النصوص المترجمة إلى اللغات التي سيتضمنها تطبيقك، وهذا الـ `class` يجب أن يرث من `class Translations` والذي هو معرف أساساً ضمن المكتبة `GetX`.
3. نقوم بعمل `override` لـ `get keys`

`Map<String, Map<String, String>> get keys`

```
import 'package:get/get.dart';

class MyLocale extends Translations {
  @override
  Map<String, Map<String, String>> get keys => {
    'en': { 'homepage': 'homepage' },
    'ar': { 'homepage': 'الصفحة الرئيسية' };
}
```



Localization GetX

في حال كنت ترغب فقط بالترجمة بالإعتماد على لغة الجهاز فإن function GetX تحتوي على `Get.deviceLocale` بإمكانها الحصول عليها وهي:

`Get.deviceLocale`

```
GetMaterialApp(  
  locale: Get.deviceLocale,  
  translations: MyLocale( ),  
)
```

وهنا مثال لاستخدام المفردات ضمن ui

```
AppBar(  
  title: Text( 'homepage'.tr ),)
```

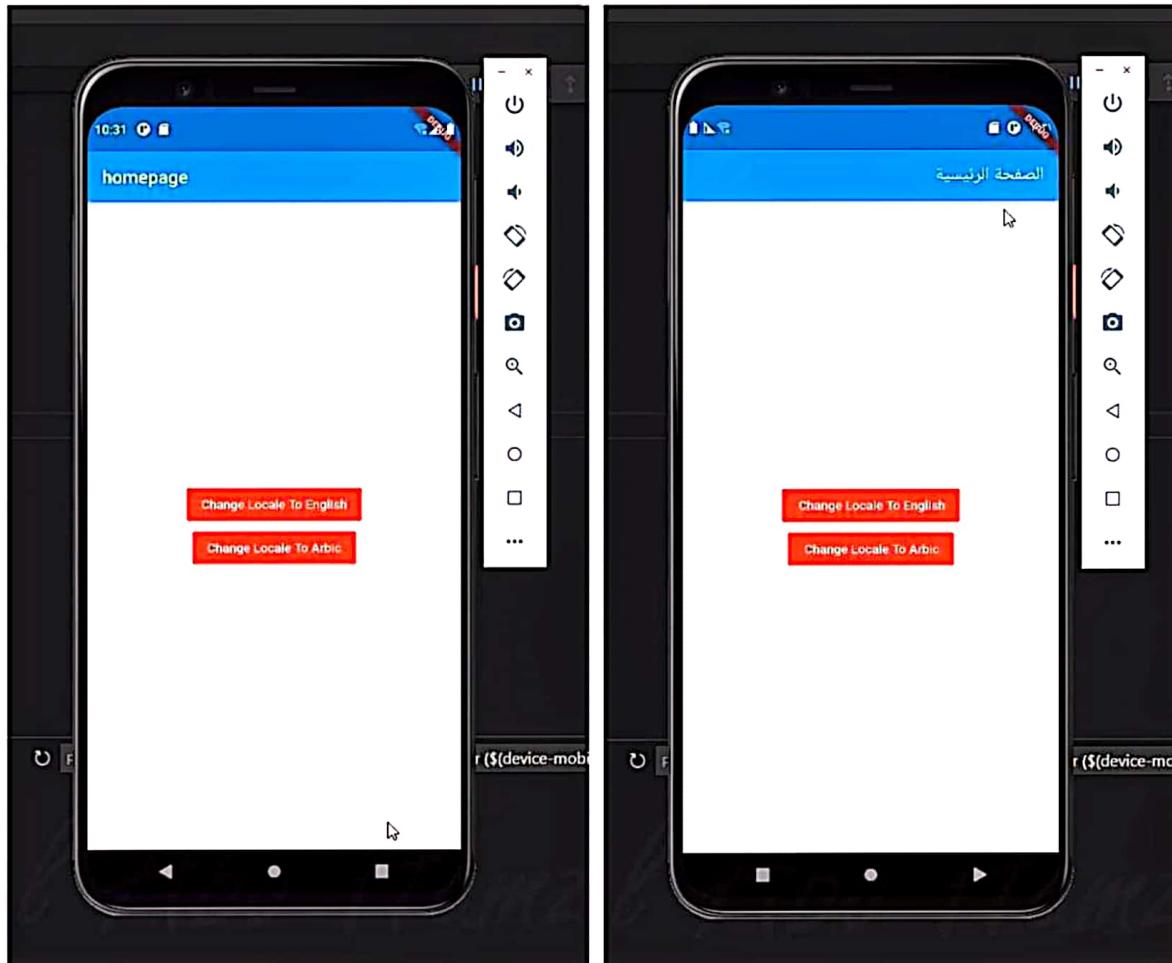
حيث أن `.tr` هي خاصة بالlocalization في GetX بحيث يجب وضعها بعد key الكلمة التي تريد ترجمتها حسب اللغة المضافة.

 GetX

Localization GetX

English

العربية



ملاحظة :

نلاحظ أنه عند اختلاف اللغة بين العربية والإنكليزية فإن اتجاه **widgets** يختلف حسب كل لغة (من اليمين لليسار للعربية ومن اليسار لليمين للإنكليزية).

ملاحظة :

ملاحظة: يجب أن يكون ال **key** نفسه بين اللغتين كما هو موضح في المثال السابق.
. 'homepage'

GetX

Localization GetX

من أجل تبديل اللغات من داخل التطبيق يتم اضافة **class** يحوي تابع يقوم بالتبديل عند حدث معين .

```
class MyLocaleController extends GetxController{
    void changLang(String codeLang) {
        Locale locale= Locale(codeLang);
        Get.updateLocale(locale);
    }
}
```

 **ملاحظة :**

هذا الكود عبارة عن ال **controller** الخاص باللغة حيث يتم حقنه من أجل استخدامه في ال **presentation layer** وهذا ما سنتكلم عنه في المنشورات القادمة.....

بعد حقن ال **controller** يمكن استخدام دالة **changeLang** للتبديل الى اللغة العربية وذلك عند الضغط على **button** معين .

```
MaterialButton(
    color: Colors.red,
    textColor: Colors.white,
    child: Text('اللغة العربية'),
    onPressed: () => controllerLang.changLang('ar'),
)
```

