Provider State Management in Flutter

Maaz Aftab · Follow
Published in CodeChai
5 min read · Feb 13, 2020

Flutter is a **declarative framework.** In contrast, to the imperative framework, Flutter does not allow to change the widget, which is mostly the UI component on screen, once it is defined. It is to make widget light weighted. **However**, the widget can be rebuilt with a different appearance or data in it, and the end result is same, **the change in UI.** For this, we use **Stateful widget**, and we call **setState()**, whenever there is some change occurred in state of widget. Calling **setState() t**ells the Framework that state of the widget is changed, and widget must be rebuilt, hence the widget gets rebuilt, which is same like changing the widget, however there is the difference of mechanism.

hundreds of different widgets, so there could be hundred different points where you have to take care of calling **setState()**, and managing state. Moreover, your front end logic will be scattered in different places in UI code. So, using this raw technique to manage state is not a good option, we have a better approach to manage state, which is not just Easy but Effective, called **Provider State Management**.

Provider

Provider State Management, which is recommended by Google a well, mainly provides you a central point to manage the state, and to write front end logic.

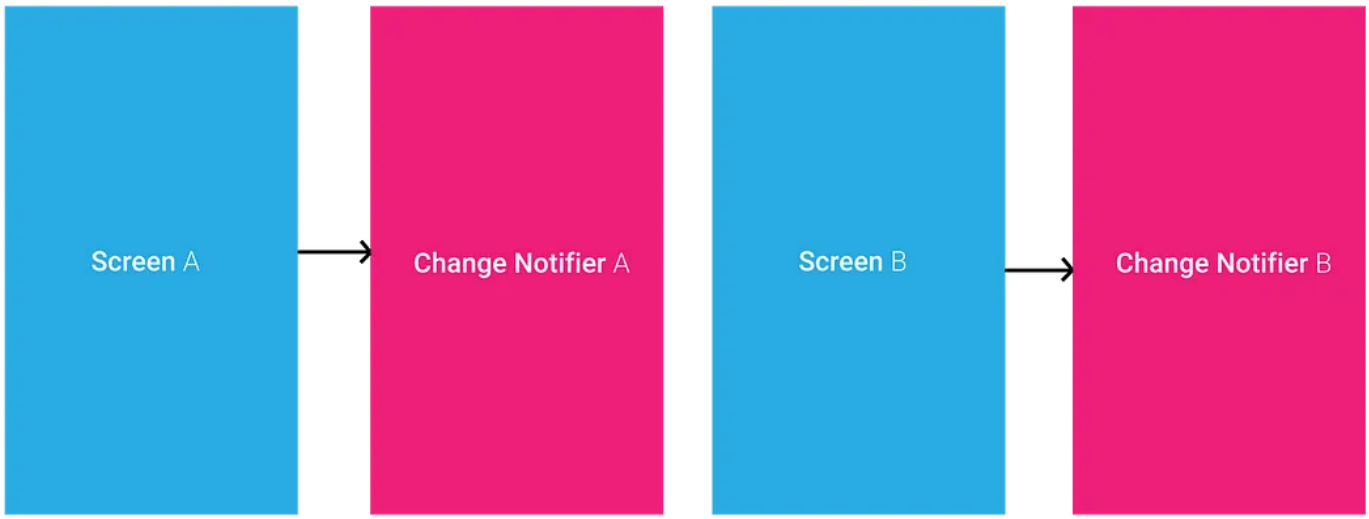
Provider Components

There are three components related to this Provider State Management that we need to understand.

1. ChangeNotifier.
2. ChangeNotifierProvider
3. Consumer

Now, there are different techniques to understand this provider approach, however, for the sake of simplicity, we are discussing the below variant of Provider State Management.

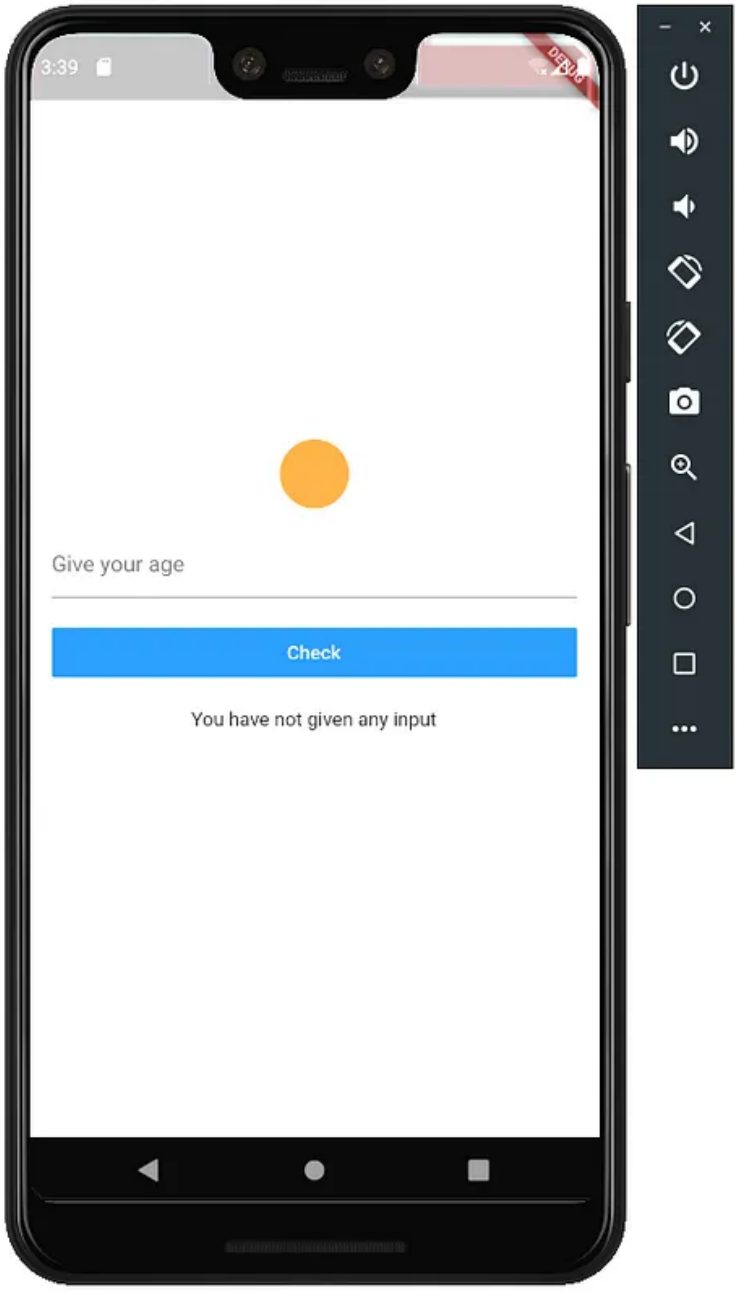
For Every Screen of Flutter application, we make a central point for managing state which is called a **ChangeNotifier,** which is just an ordinary class that extends **ChangeNotifier** class**.** It contains all the state data that are being used by different parts of corresponding screen. Now if something is a **ChangeNotifier**, and some of its data gets changed, it notifies the framework that the screen which is using the change notifier, needs to be rebuilt, since the corresponding screen's state, **ChangeNotifier**, is changed. If you have already worked on **MVVM** than you can think your screen as View and **ChangeNotifier** as **ViewModel** or **Controller** in case of **MVC**.



Variant of Provider Approach

For the implementation, Let's consider a scenario. Say a person wants to apply for driving license, and there is a very simple form that he has to fill using our Flutter mobile application in order to know if he is eligible for applying or not. The Criteria for eligibility is pretty simple, one must be at least 18 years old in order to be eligible.

So our Flutter application would be a single screen application, on which there would be a simple Form contains **TextField**. There are two extra widgets on screen as well, one is C**ircle**, which is **orange** by default, but after submitting form, if the user is eligible, then the circle would turn **green**, however if not, then **red**. Secondly, there is a **Text** widget at the bottom that will show the message that would change based on user input, however, by default it will show *"You have not give any input"*.



Application UI design

```dart
import 'package:flutter/material.dart';

class EligiblityScreen extends StatelessWidget {
  final ageController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        padding: EdgeInsets.all(16),
        child: Form(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Container(
                decoration: BoxDecoration(
                    shape: BoxShape.circle,
                    color: Colors.orangeAccent
                ),
                height: 50,
                width: 50,
              ),

              SizedBox(height: 16,),

              TextFormField(
                controller: ageController,
                decoration: InputDecoration(
                  hintText: "Give your age",

                ),
              ),
              SizedBox(height: 16,),
              Container(
                width: double.infinity,
                child: FlatButton(
                  child: Text("Check"),
                  color: Colors.blue,
                  textColor: Colors.white,
                  onPressed: (){},
                ),
              ),
              SizedBox(height: 16,),

              Text("You have not given any input")
            ],
          ),
        )
```

```
49          ),
50        );
51      }
52    }
```

EligiblityScreen.dart hosted with ❤ by GitHub view raw

Eligibility Screen Code with out Provider

Above is the Code for Eligibility Screen **without Provider (Not Final).** So, we have done with the UI part, and now we can make ChangeNotifer against above screen.

ChangeNotifier

Now let's make **ChangeNotifier** against above screen that would contain all the state data related to that screen.

```dart
import 'package:flutter/material.dart';

class EligiblityScreenProvider extends ChangeNotifier{
  String _eligiblityMessage = "";
  bool _isEligible;

  void checkEligiblity(int age){
    if(age >= 18)
      eligibleForLicense();
    else
      notEligibleForLicense();
  }

  void eligibleForLicense(){
    _eligiblityMessage = "You are eligible to apply for Driving License";
    _isEligible = true;

    //Call this whenever there is some change in any field of change notifier.
    notifyListeners();
  }


  void notEligibleForLicense(){
    _eligiblityMessage = "You are not eligible to apply for Driving License";
    _isEligible = false;

    //Call this whenever there is some change in any field of change notifier.
    notifyListeners();
  }

  //Getter for Eligiblity message
  String get eligiblityMessage => _eligiblityMessage;

  //Getter for Eligiblity flag
  bool get isEligible => _isEligible;


}
```

EligiblityScreenProvider.dart hosted with ❤ by GitHub view raw

Eligibility Screen Change Notifier Code

Now in the above **ChangeNotifier** that we have made against Eligibility Screen has just two state data in it first ***String*** *_eligibilityMessage,* which is showing message, after processing the user age, and, ***bool*** _isEligible flag, which is to change the color of the circle status indicator.

In **ChangeNotifier,** we made a method **checkEligibility()** which takes age as a parameter and after performing the logic it changes the state, i.e., if user is eligible then it will call **eligibleForLicense()** else it will call **notEligibleForLicense().**

The Most Important thing is calling **notifyListeners(),** whenever you change the state data in ChangeNotifier. If you will not call this method, the state change would not reflect in the UI. The method **notifyListeners()** tells flutter to rebuild the screen which is using that ChangeNotifier.

ChangeNotifierProvider

As we have made **EligibilityScreenProvider** ChangeNotifer against the **EligibilityScreen**, now we have to connect this provider with the screen, or we have to use this provider in EligibilityScreen**.** For this, we use ChangeNotifierProvider, a widget, that provides the instance of ChangeNotifer to the screen, and thus we can access state data of the ChangeNotifier in the screen. We just need to wrap the **EligibilityScreen** inside ChangeNotifierProvider for this, code is given below.

Consumer

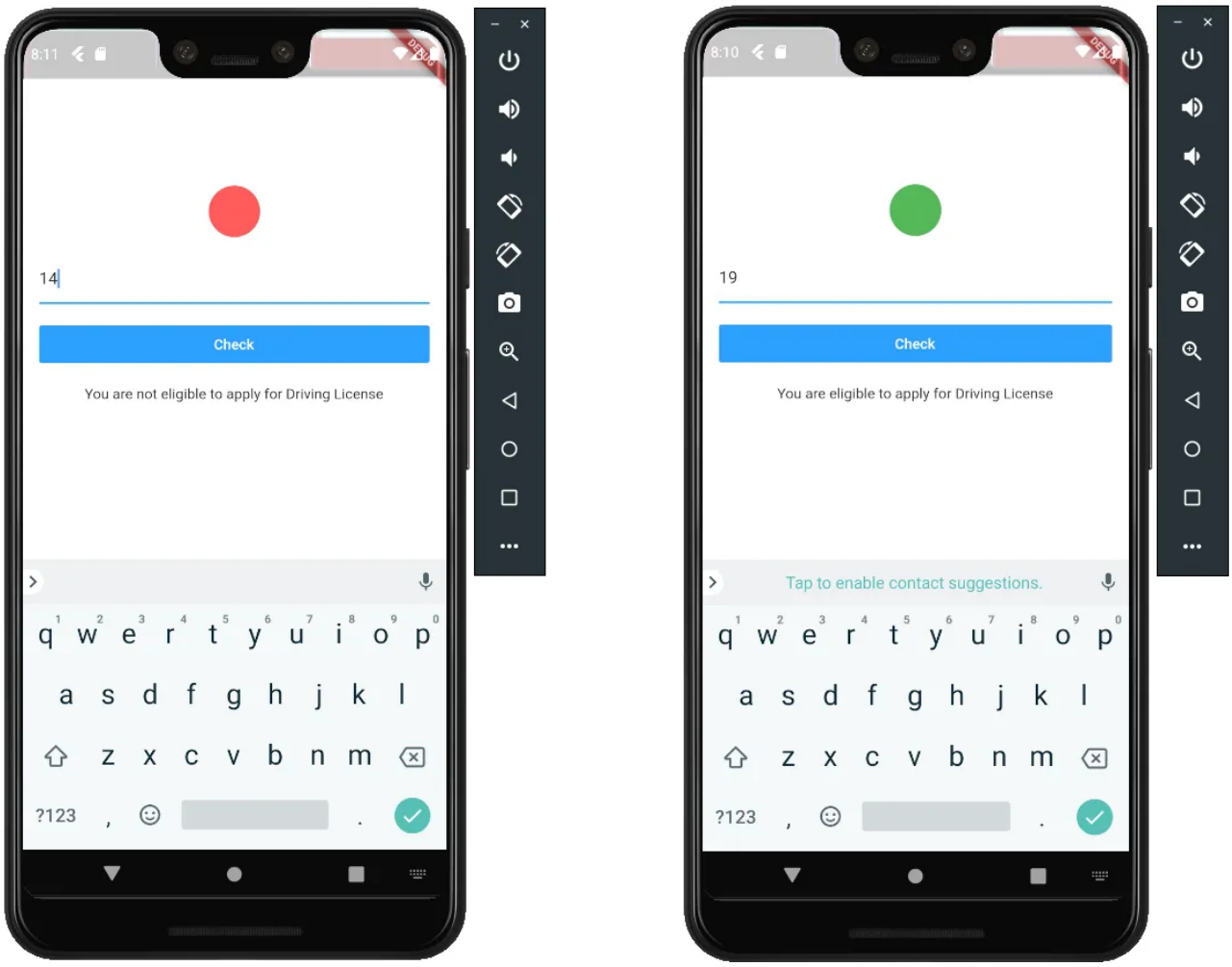
It is not necessary that all the part of the UI in **EligibilityScreen** will be using the state data from **EligibilityScreenProvider,** and thus needs to be rebuit. It may be possible that 50% of the UI of any screen, not in our case, not needs to be rebuilt. So, **Consumer**, which is a widget, allows to observe the state changes from ChangeNotifier in a particular part of UI, and thus only observing part of the UI will get re-rendered.

```dart
import 'package:flutter/material.dart';
import 'package:proivder_demo/states/EligiblityScreenProvider.dart';
import 'package:provider/provider.dart';

class EligiblityScreen extends StatelessWidget {
  final ageController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider<EligiblityScreenProvider>(
      create: (context) => EligiblityScreenProvider(),
      child: Builder(
        builder: (context) {

          return Scaffold(
            body: Container(
              padding: EdgeInsets.all(16),
              child: Form(
                child: Consumer<EligiblityScreenProvider>(
                  builder: (context, provider, child){
                    return Column(
                      mainAxisAlignment: MainAxisAlignment.center,
                      children: <Widget>[
                        Container(
                          decoration: BoxDecoration(
                              shape: BoxShape.circle,

                              //if isEligible is null then set orange color else if it is true
                              color: (provider.isEligible == null) ? Colors.orangeAccent : prov
                          ),
                          height: 50,
                          width: 50,
                        ),

                        SizedBox(height: 16,),

                        TextFormField(
                          controller: ageController,
                          decoration: InputDecoration(
                            hintText: "Give your age",
                          ),
                        ),
                        SizedBox(height: 16,),
                        Container(
                          width: double.infinity,
                          child: FlatButton(
                            child: Text("Check"),
                            color: Colors.blue
```

```
                    color: Colors.blue,
                    textColor: Colors.white,
                    onPressed: (){

                      //getting the text from TextField and converting it into int
                      final int age = int.parse(ageController.text.trim());

                      //Calling the method from provider.
                      provider.checkEligiblity(age);
                    },
                  ),
                ),
                SizedBox(height: 16,),

                Text(provider.eligiblityMessage)

              ],
            );
          },
        )
      ),
    ),
  );
  }
}
```

Eligibility Screen Code

At line 10, We have used ChangeNotifierProvider widget, so that we can access what's inside the ChangeNotifier. We need to give the type of your provider in type parameter of ChangeNotifierProvider **ChangeNotifierProvider<EligibilityScreenProvider>.**
At line 19, We have used **Consumer** widget, so that whatever inside it gets rebuilt whenever **notifyListener()** is called from **EligibilityScreenProvider** ChangeNotifier.
At Line 20, we have 3 thing in the builder method, the first one is **BuildContext context**, which we have in every builder, second the instance of **EligibilityScreenProvider provider**, which you can use to access the properties of EligibilityScreenProvider, and last one is just for the optimization, which you can skip for now.



Different Output on different input

Below is the Code of main.dart

```dart
import 'package:flutter/material.dart';
import 'package:proivder_demo/screens/EligiblityScreen.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(

        primarySwatch: Colors.blue,
      ),
      home: EligiblityScreen(),
    );
  }
}


```

main.dart hosted with ❤ by GitHub view raw

main.dart Code

Awesome So that's all from it. If you really found this article useful and you want to learn more from me, then you can head over to my YouTube channel **Easy Approach**, over there you can find hundreds of different videos on Flutter, and a comprehensive and easy Flutter video series.

Thank you!

Provider State Management Flutter State Flutter Widget

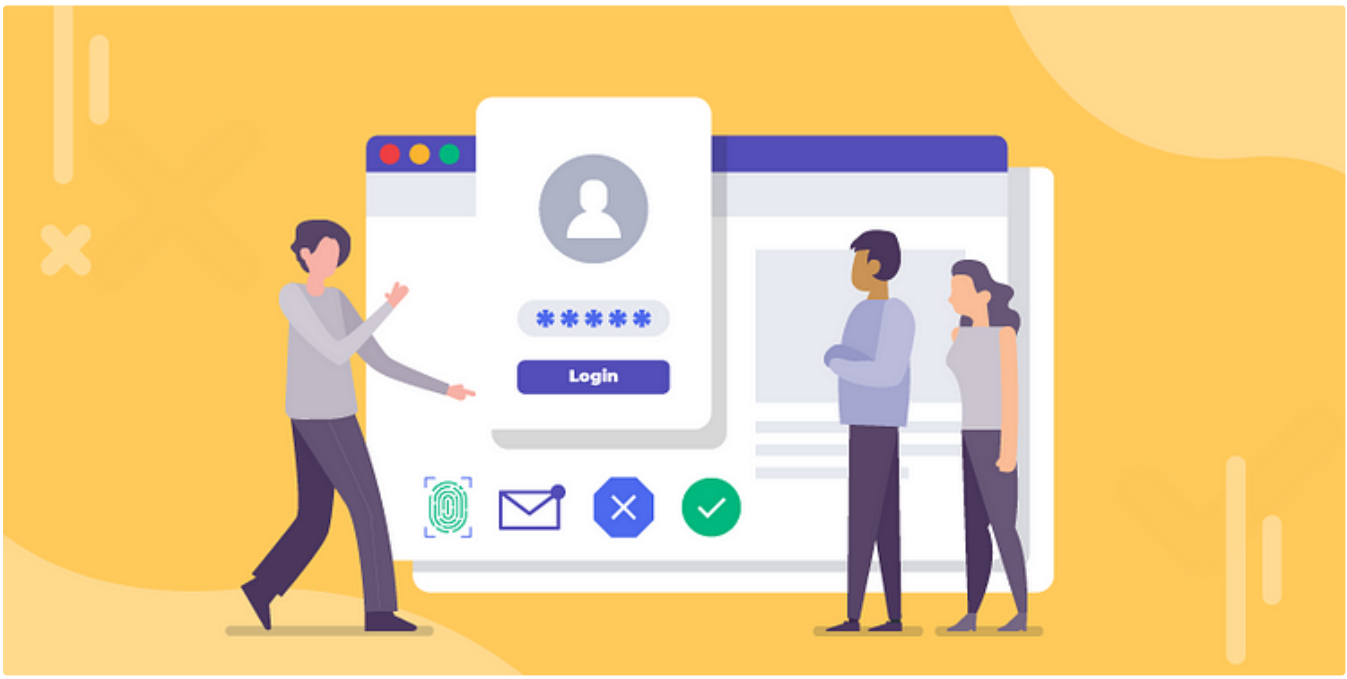
Follow

Written by Maaz Aftab

370 Followers · Writer for CodeChai

Founder Of Easy Approach, Flutter Lover, YouTuber, and a writer.

More from Maaz Aftab and CodeChai



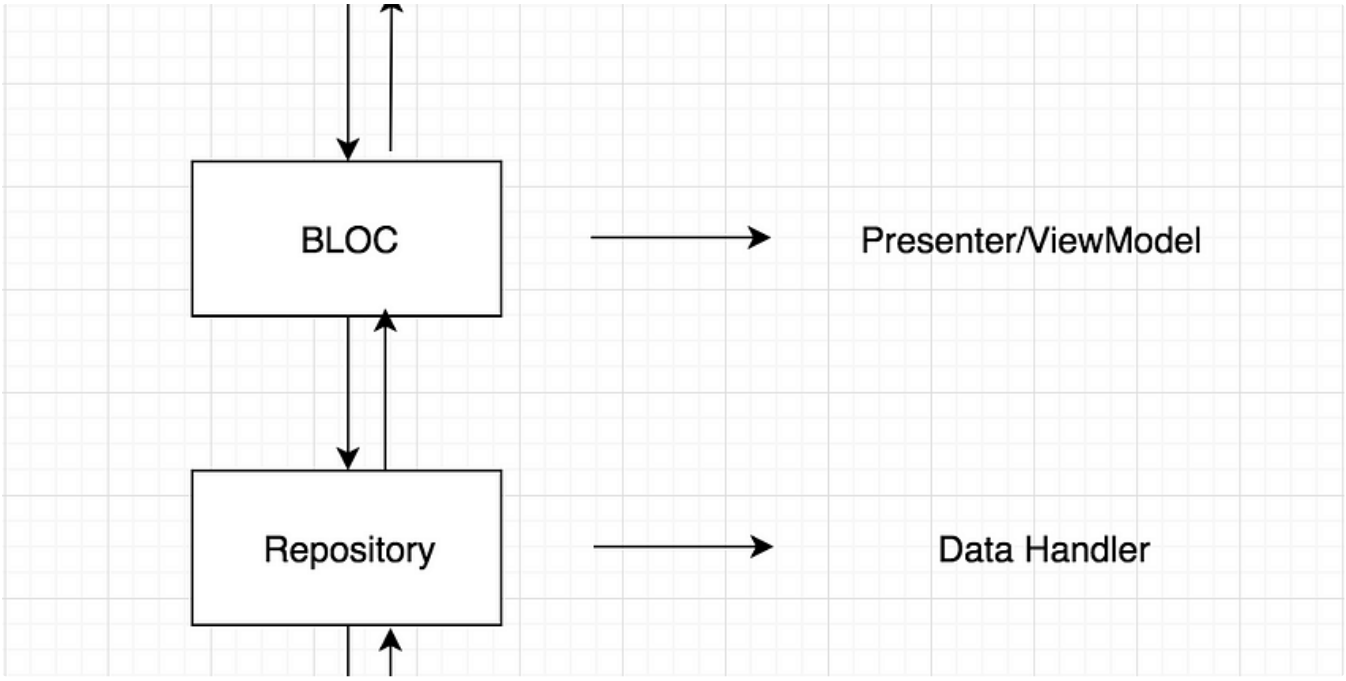
Maaz Aftab in CodeChai

Firebase User Authentication using Phone verification in Flutter

Mobile Application often requires User authentication in order to show the secured and relevant to the authenticated user.

7 min read · Jan 19, 2020

1K 9



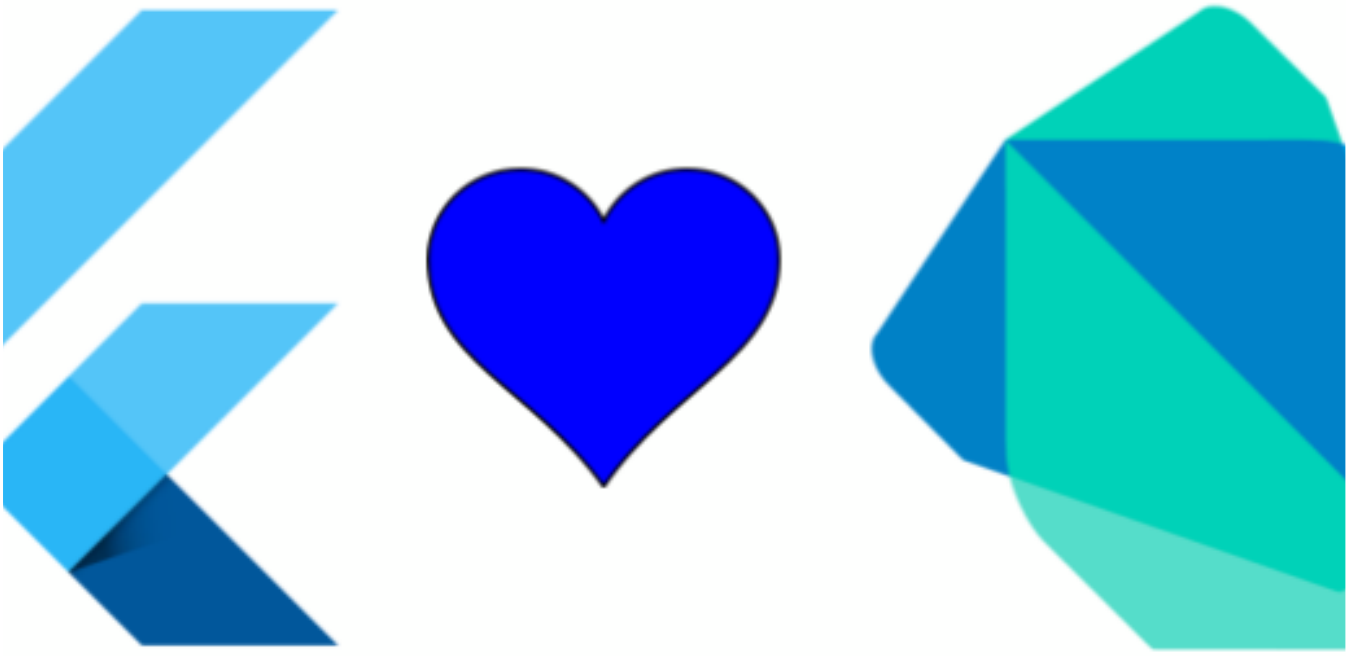
Sagar Suri in CodeChai

Architect your Flutter project using BLOC pattern

Hi Folks! I am back with another brand new article on Flutter. This time I will be talking and demonstrating to you “how to architect your...

11 min read · Aug 26, 2018

14.2K 76



Neila in CodeChai

How to use custom icons in Flutter

Many thanks to the developers of FlutterIcon!

2 min read · Sep 16, 2018

3.2K 23

Maaz Aftab in CodeChai

Bloc State Management with Easy Approach

In Flutter there are different options for managing states in which one is Bloc Pattern (Business Logic Component) that is one of the most...

5 min read · Feb 16, 2020

571 4

See all from Maaz Aftab

See all from CodeChai

Recommended from Medium

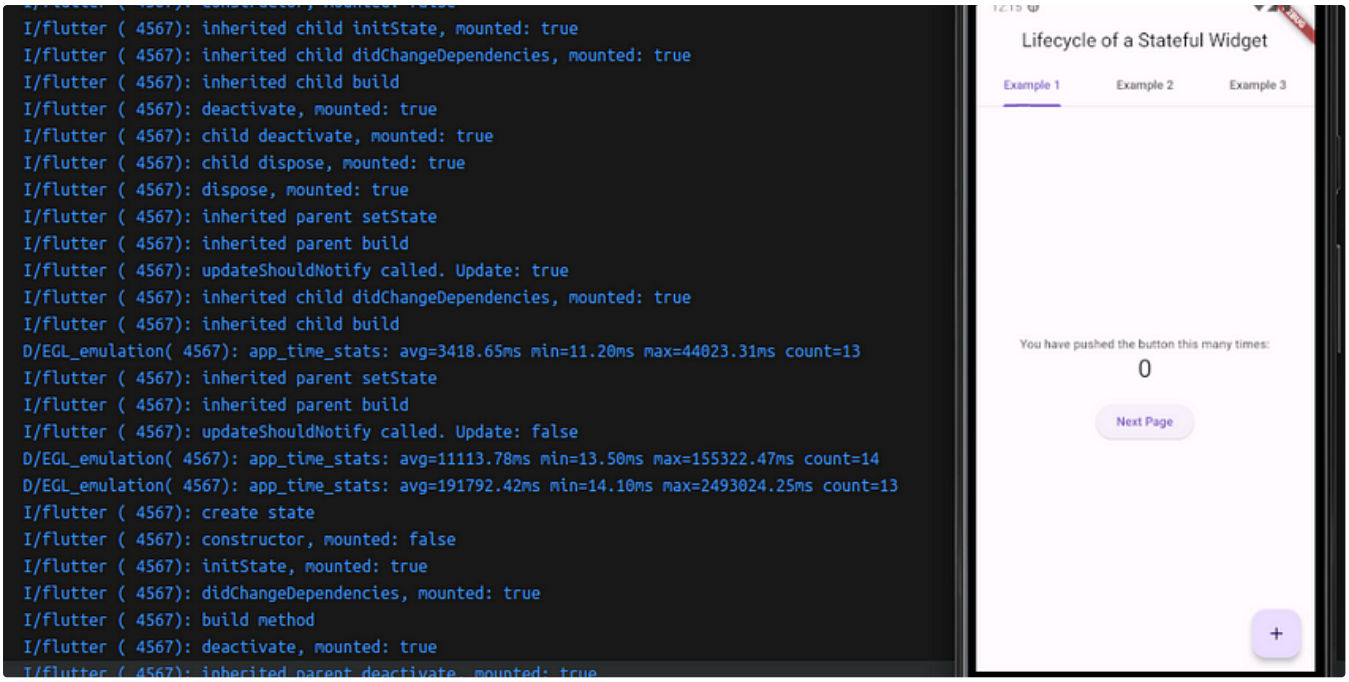
Nurettin Eraslan

Understanding state management with Provider in Flut

In Flutter, “provider” is a popular package used for state management and data sharing within an application. Provider allows you to share...

3 min read · Jul 9

23



Hadiyaaamir

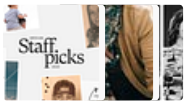
Lifecycle of a Stateful Widget

A stateful widget in Flutter is a component that can maintain state and update its appearance in response to changes. The lifecycle of a...

13 min read · Jun 20

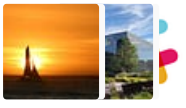
70 1

Lists



Staff Picks

537 stories · 545 saves



Stories to Help You Level-Up at Work

19 stories · 371 saves

Self-Improvement 101

20 stories · 1059 saves

Productivity 101

20 stories · 961 saves

Sebastian Buzdugan

Life without state management - Flutter

Heey, today I want to talk to you about a topic that might seem controversial or even heretical to some of you: life without state...

4 min read · Jul 14

22 1

Putu Guna in DSF Web Services Engineering

GetX — One of the Best Choices for State Management in Flutter Development

Many developers choose Flutter as a base language to create mobile applications. If you are interested in utilizing it, you may need to...

5 min read · Jul 5

8 1

Iqbal Shehzada

Flutter State Management Libraries in 2023

Flutter, a leading framework for mobile app development. It offers efficient state management, enabling developers to handle app state in a...

3 min read · Nov 8

5

Rutudhvaj Bodar

GetX Middleware

In GetX, middleware allows you to listen to route events and trigger actions accordingly. It provides a convenient way to add functionality...

2 min read · Jun 24

2 1

See more recommendations