FLUTTER

Prepare by: Aamir Pinger

/AamirPinger
/AamirPingerOfficial
/AamirPinger

[Click for Flutter Course Playlist]

Made by Google

Flutter is Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, desktop, and embedded devices from a single codebase.

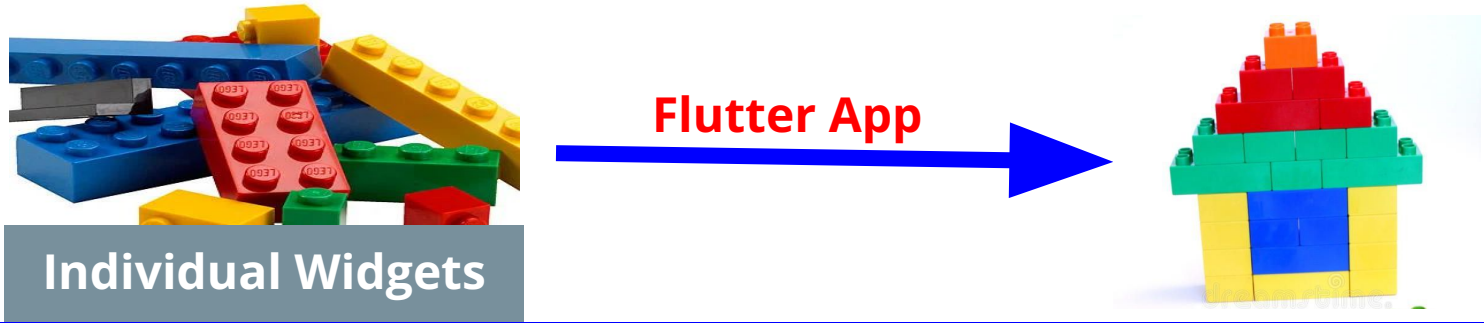
Flutter

- Flutter is a an open-source software development toolkit from Google for building cross-platform apps.
- Flutter is **NOT** a language.
- Flutter uses Dart as its programming language.
- Dart is an object-oriented language.

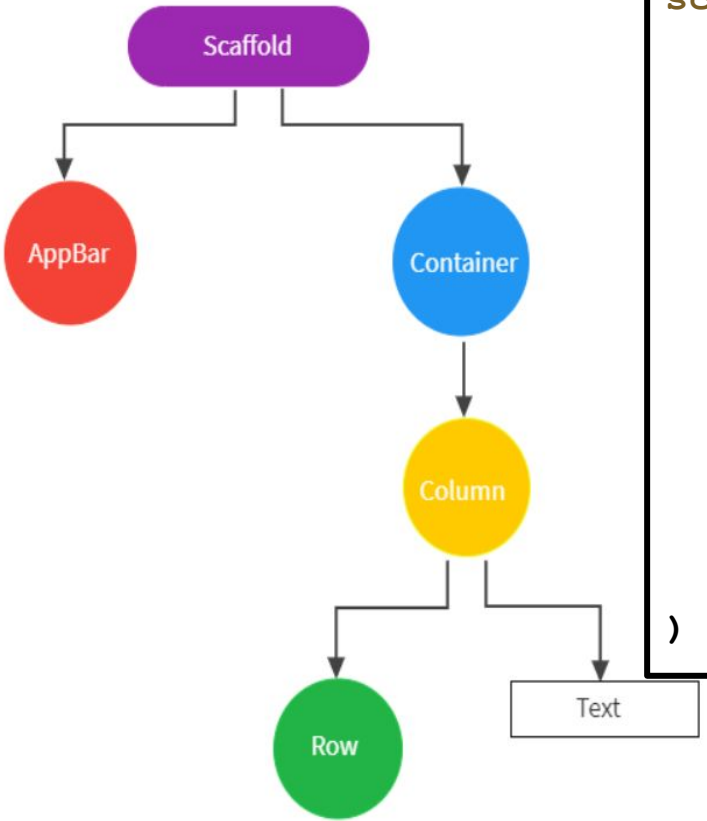
Flutter gives you the ability to write once and run anywhere.

Flutter

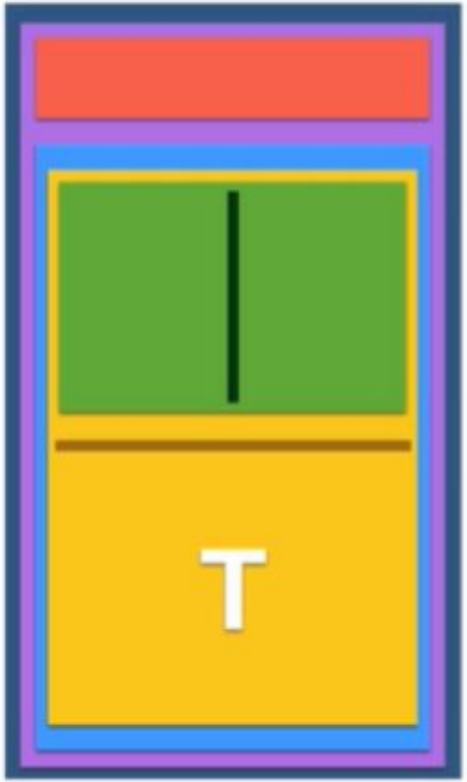
- Flutter apps consist of a series of packages, plugins and widgets.
- Flutter app are build widgets.
- Widgets are building blocks for any flutter app.
- Widgets are collection of reusable UI elements (buttons, text inputs, sliders, and so on) that you can personalize for your own needs.
- Development ease with hot reload.
    - This lets you update your code and see the changes live without re-deploying it.



Flutter - Widgets Tree

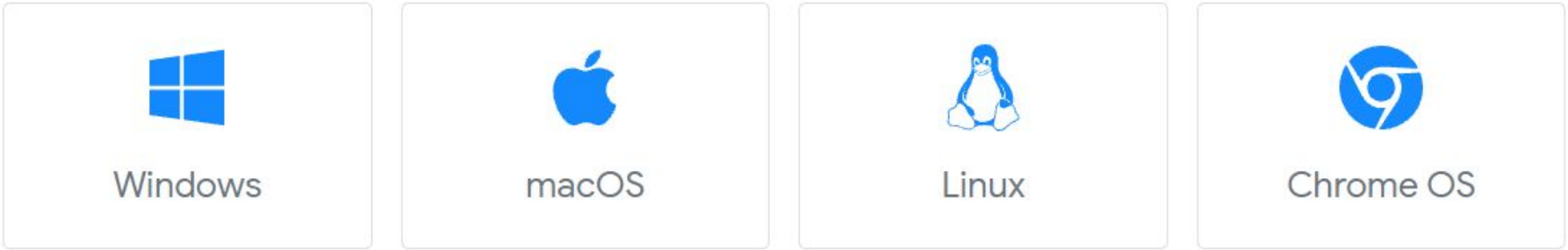



```
scaffold(
  appBar: AppBar(),
  body: Container(
    child: Column(
      children: [
        Row( ... ),
        Text(),
      ]
    )
  )
)
```



Source: https://iq.opengenus.org/getting-started-with-flutter-development/

Installation



https://flutter.dev/docs/get-started/install

Prerequisite

Dart Slides Link

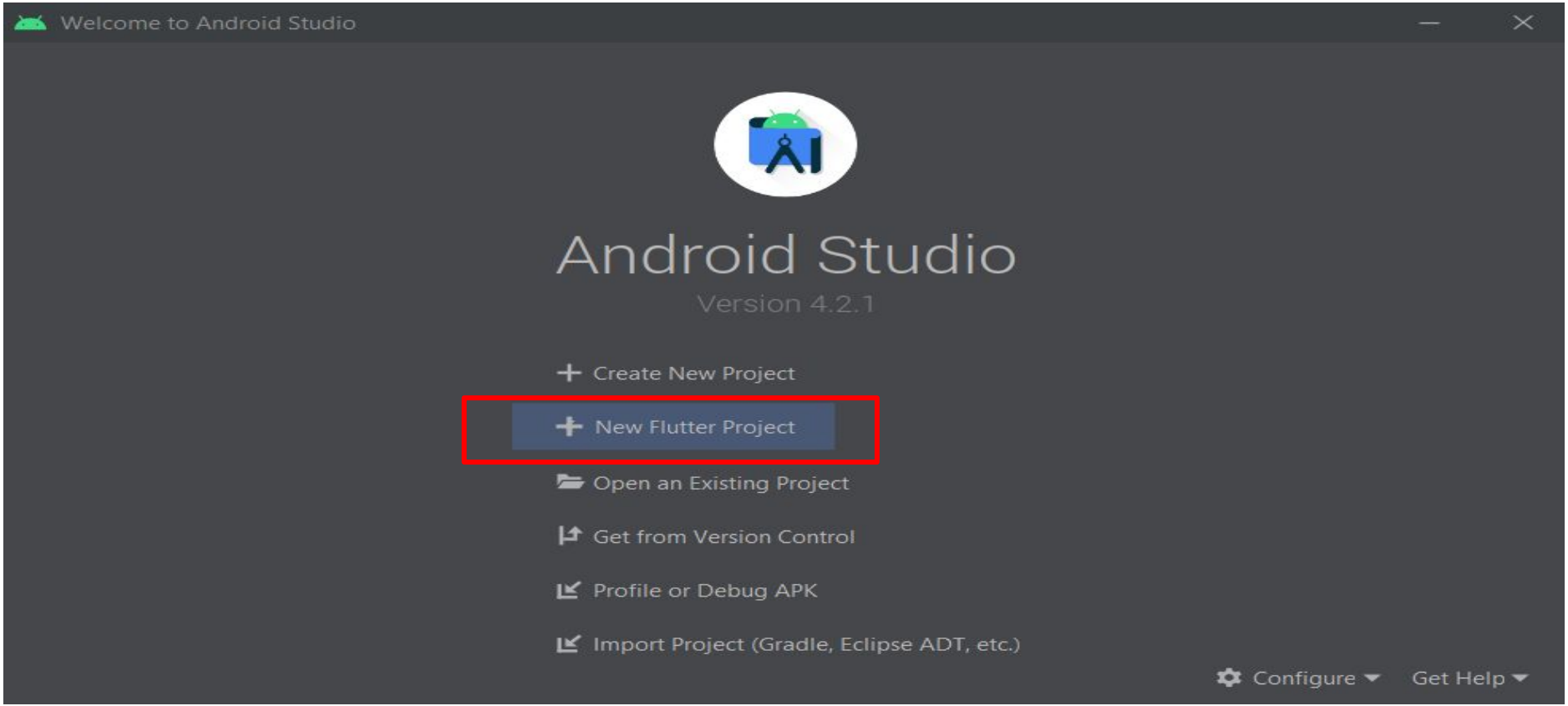
Installation

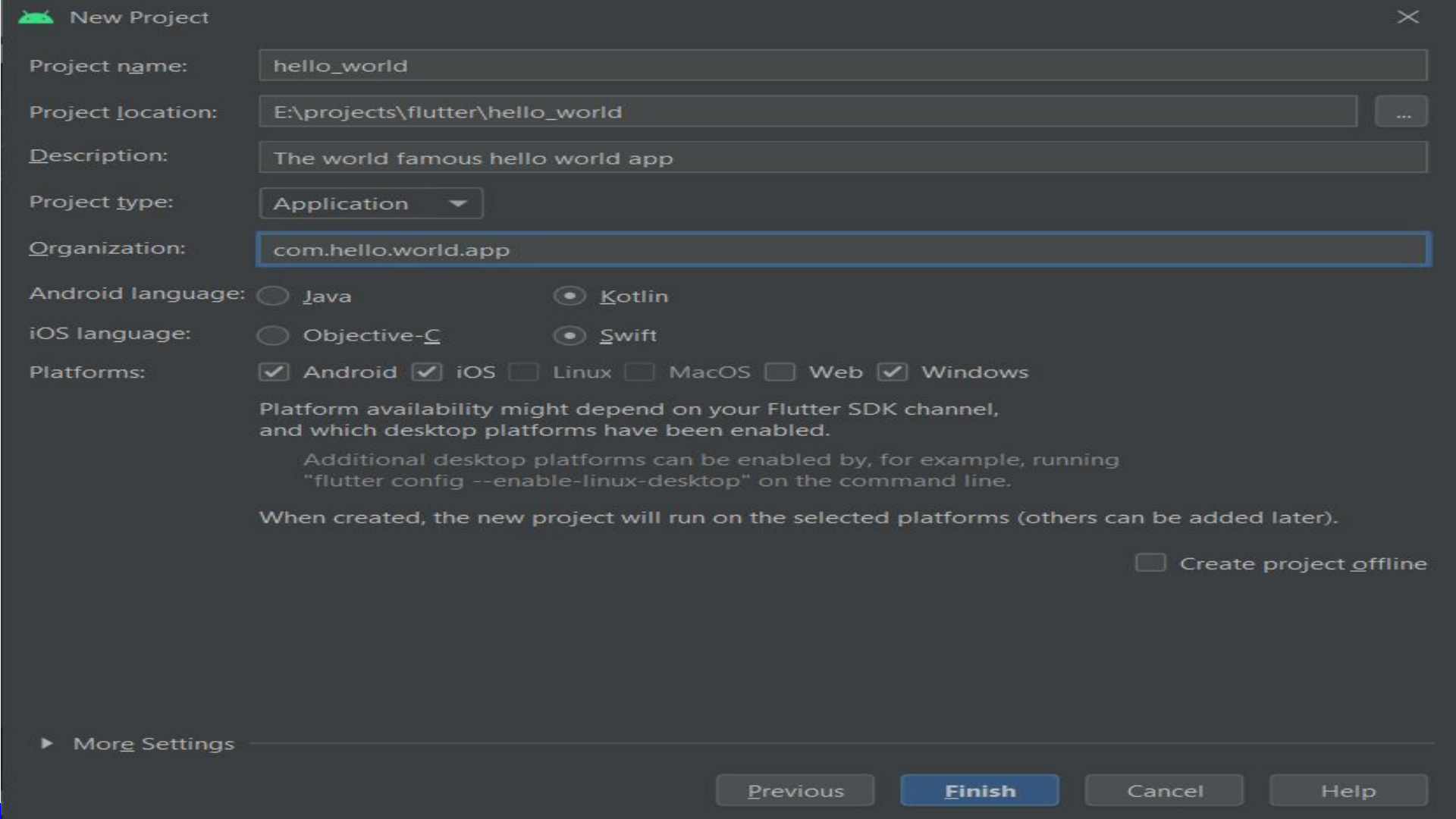
- Download and install Flutter SDK
- Update the PATH variable
- Run Flutter Doctor
- Install Android Studio
- Set up your Android device
- Set up the Android emulator
- Agree to Android Licenses
- Install Dart and Flutter extension at Android Studio
- Run Flutter Doctor again

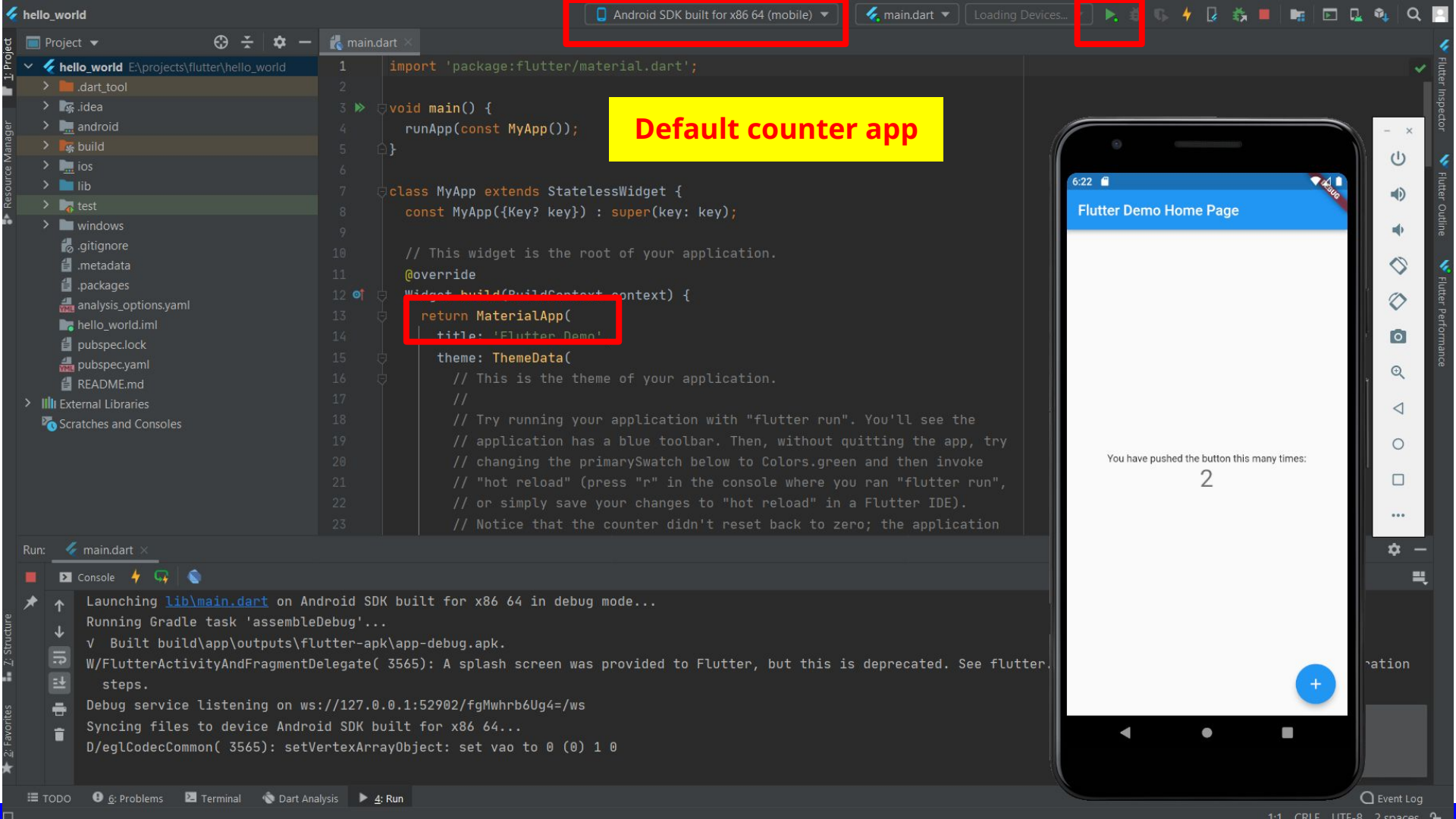
Material.io

- Flutter apps are composition of widgets
- Button, image, background, text, and even the whole base container is widget in Flutter.
- Off course, custom widgets can be made for everything.
- But reinventing the wheel, instead we should realign the wheel.
- Google have provided us material widgets library which most of the project heavily relies too.
- Check the component section at **Material.io**

Let’s start creating world famous hello world app




New Project
Project name:
hello_world
Project location:
E:\projects\flutter\hello_world
...
Description:
The world famous hello world app
Project type:
Application
Organization:
com.hello.world.app
Android language:
Java
Kotlin
iOS language:
Objective-C
Swift
Platforms:
Android
iOS
Linux
MacOS
Web
Windows
Platform availability might depend on your Flutter SDK channel,
and which desktop platforms have been enabled.
Additional desktop platforms can be enabled by, for example, running
"flutter config --enable-linux-desktop" on the command line.
When created, the new project will run on the selected platforms (others can be added later).
Create project offline
More Settings
Previous
Finish
Cancel
Help


hello_world
Android SDK built for x86 64 (mobile)
main.dart
Loading Devices...
Project
hello_world E:\projects\flutter\hello_world
.dart_tool
.idea
android
build
ios
lib
test
windows
.gitignore
.metadata
.packages
analysis_options.yaml
hello_world.iml
pubspec.lock
pubspec.yaml
README.md
External Libraries
Scratches and Consoles
main.dart
import 'package:flutter/material.dart';
void main() {
runApp(const MyApp());
}
class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);
// This widget is the root of your application.
@override
return MaterialApp(
title: 'Flutter Demo'
theme: ThemeData(
// This is the theme of your application.
//
// Try running your application with "flutter run". You'll see the
// application has a blue toolbar. Then, without quitting the app, try
// changing the primarySwatch below to Colors.green and then invoke
// "hot reload" (press "r" in the console where you ran "flutter run",
// or simply save your changes to "hot reload" in a Flutter IDE).
// Notice that the counter didn't reset back to zero; the application
Default counter app
Run:
main.dart
Console
Launching lib\main.dart on Android SDK built for x86 64 in debug mode...
Running Gradle task 'assembleDebug'...
√ Built build\app\outputs\flutter-apk\app-debug.apk.
W/FlutterActivityAndFragmentDelegate(3565): A splash screen was provided to Flutter, but this is deprecated. See flutter.
steps.
Debug service listening on ws://127.0.0.1:52902/fgMwhrb6Ug4=/ws
Syncing files to device Android SDK built for x86 64...
D/eglCodecCommon(3565): setVertexArrayObject: set vao to 0 (0) 1 0
6:22
DEBUG
Flutter Demo Home Page
You have pushed the button this many times:
2
TODO
6: Problems
Terminal
Dart Analysis
4: Run
Event Log

Hello World App

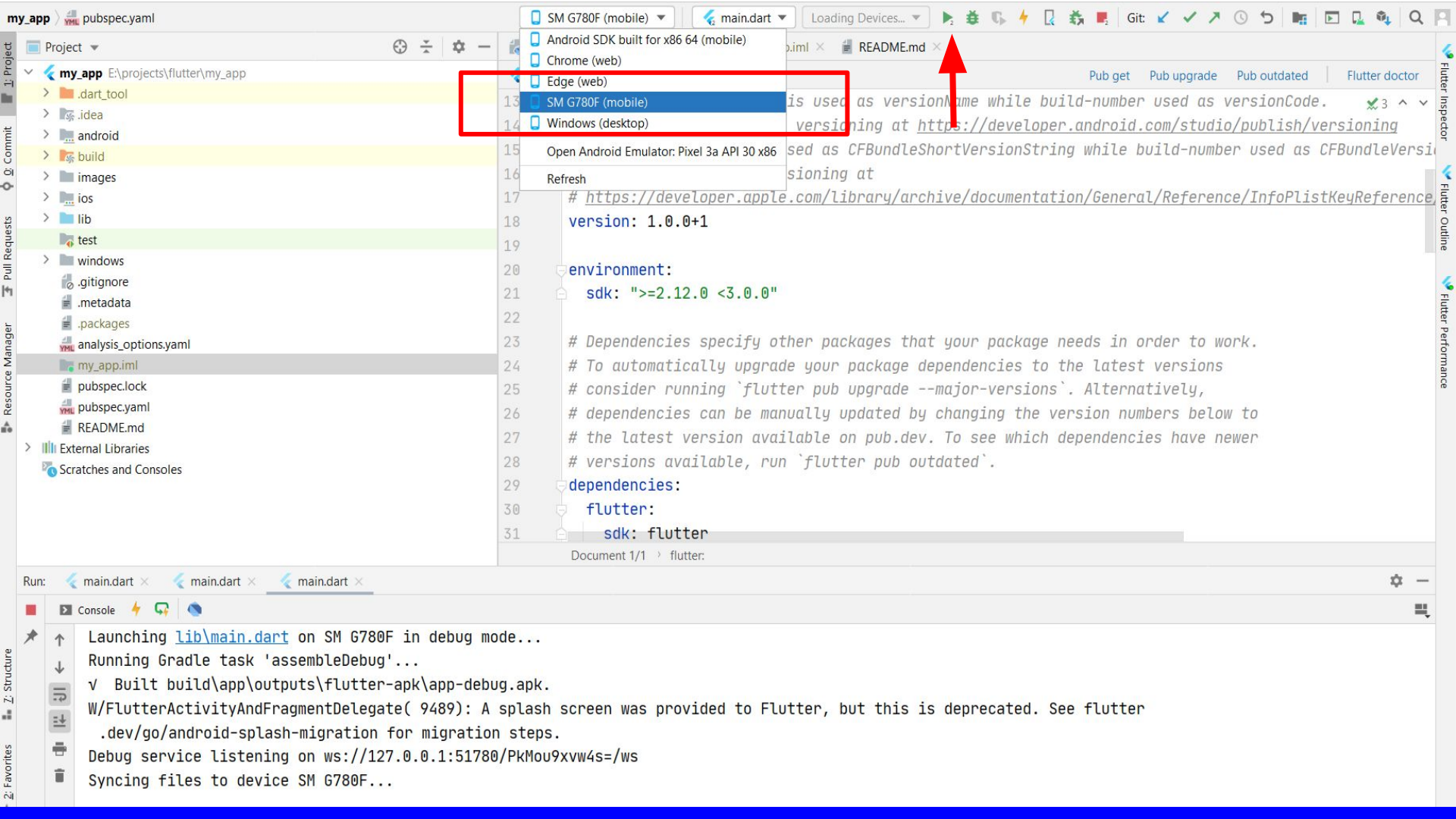
```dart
import 'dart:ui';
import 'package:flutter/material.dart';
void main() {
 runApp(MaterialApp(
   home: Center(
     child: Text(
       'Hello world.',
     ),
   ),
 ));
}
```

Deploy on physical Device using USB

- Goto Android Settings →
- About Phone →
- Software Information →
- Tap approx 7 times at Build Number
- Developer Mode Enable message will appear
- Go back to Settings →
- Developer options →
- Enable USB Debugging

Deploy on physical Device using

- Goto Android Settings →
- About Phone →
- Software Information →
- Tap approx 7 times at Build Number
- Developer Mode Enable message will appear
- Go back to Settings →
- Developer options →
- Enable USB Debugging →
- Connect the mobile to system with USB cable


my_app
pubspec.yaml
SM G780F (mobile)
main.dart
Loading Devices...
Git:
Android SDK built for x86 64 (mobile)
Chrome (web)
Edge (web)
SM G780F (mobile)
Windows (desktop)
Open Android Emulator: Pixel 3a API 30 x86
Refresh
Project
my_app E:\projects\flutter\my_app
.dart_tool
.idea
android
build
images
ios
lib
test
windows
.gitignore
.metadata
.packages
analysis_options.yaml
my_app.iml
pubspec.lock
pubspec.yaml
README.md
External Libraries
Scratches and Consoles
README.md
Pub get
Pub upgrade
Pub outdated
Flutter doctor
https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference
version: 1.0.0+1
environment:
sdk: ">=2.12.0 <3.0.0"
Dependencies specify other packages that your package needs in order to work.
To automatically upgrade your package dependencies to the latest versions
consider running `flutter pub upgrade --major-versions`. Alternatively,
dependencies can be manually updated by changing the version numbers below to
the latest version available on pub.dev. To see which dependencies have newer
versions available, run `flutter pub outdated`.
dependencies:
flutter:
sdk: flutter
Document 1/1
flutter:
Run:
main.dart
main.dart
main.dart
Console
Launching lib\main.dart on SM G780F in debug mode...
Running Gradle task 'assembleDebug'...
√ Built build\app\outputs\flutter-apk\app-debug.apk.
W/FlutterActivityAndFragmentDelegate(9489): A splash screen was provided to Flutter, but this is deprecated. See flutter
.dev/go/android-splash-migration for migration steps.
Debug service listening on ws://127.0.0.1:51780/PkMou9xvw4s=/ws
Syncing files to device SM G780F...

Scaffold( )

Scaffold( )

- The Scaffold is a widget in Flutter used to implements the basic material design visual layout structure.
- It works as a base wrapper for our all other widgets in the application
- It contains almost everything we need to create a functional and responsive Flutter apps.
- This widget is able to occupy the whole device screen.
- It provides many widgets or APIs for showing Drawer, SnackBar, BottomNavigationBar, AppBar, FloatingActionButton, and many more.
- The Scaffold class is a shortcut to set up the look and design of our app.
- Let’s try that!

Source: https://www.javatpoint.com/flutter-scaffold

Simple centered Image App

Repo: https://github.com/aamirpinger/flutter-slide-code/

Git Branch Name: `example_2_simple_centered_image`

Bundling image

- Previously, we added a network image, that requires internet.
- We can bundle any image as a asset to the app
- That means the image will be part of app as local asset
- This will make image loading fast and without internet
- Be careful!, as it will increase app bundle size.

Repo: https://github.com/aamirpinger/flutter-slide-code/

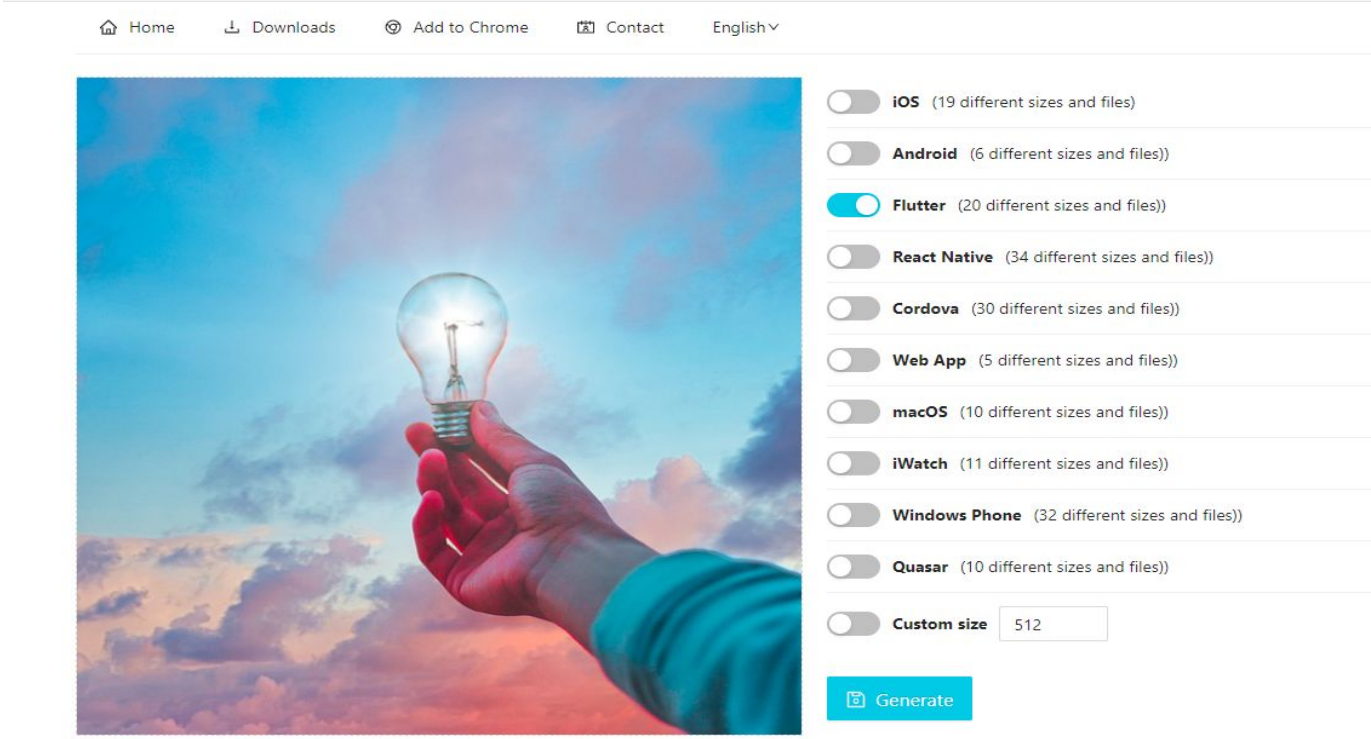
Git Branch: example_3_bundling_image_as_asset

Git Diff: #1

App Icon

App Icon

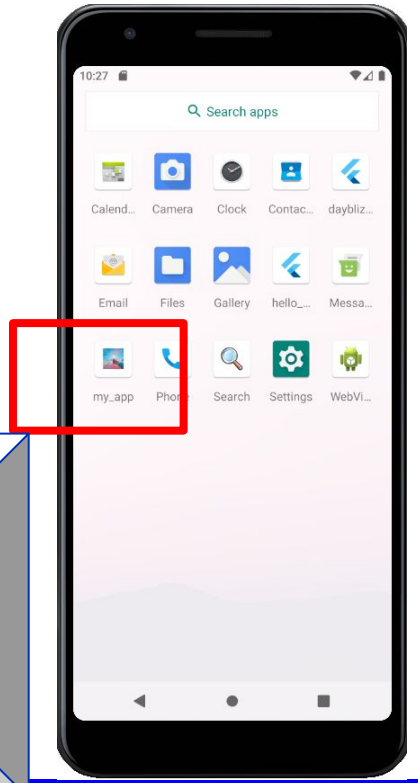
- We will be using https://www.iconify.pro/ to generate icon files



App Icon

- After extracting the zip file, folders from android folder will go to **\android\app\src\main\res**

- The files from ios folder will go to **\ios\Runner\Assets.xcassets\AppIcon.appiconset**
- That's it

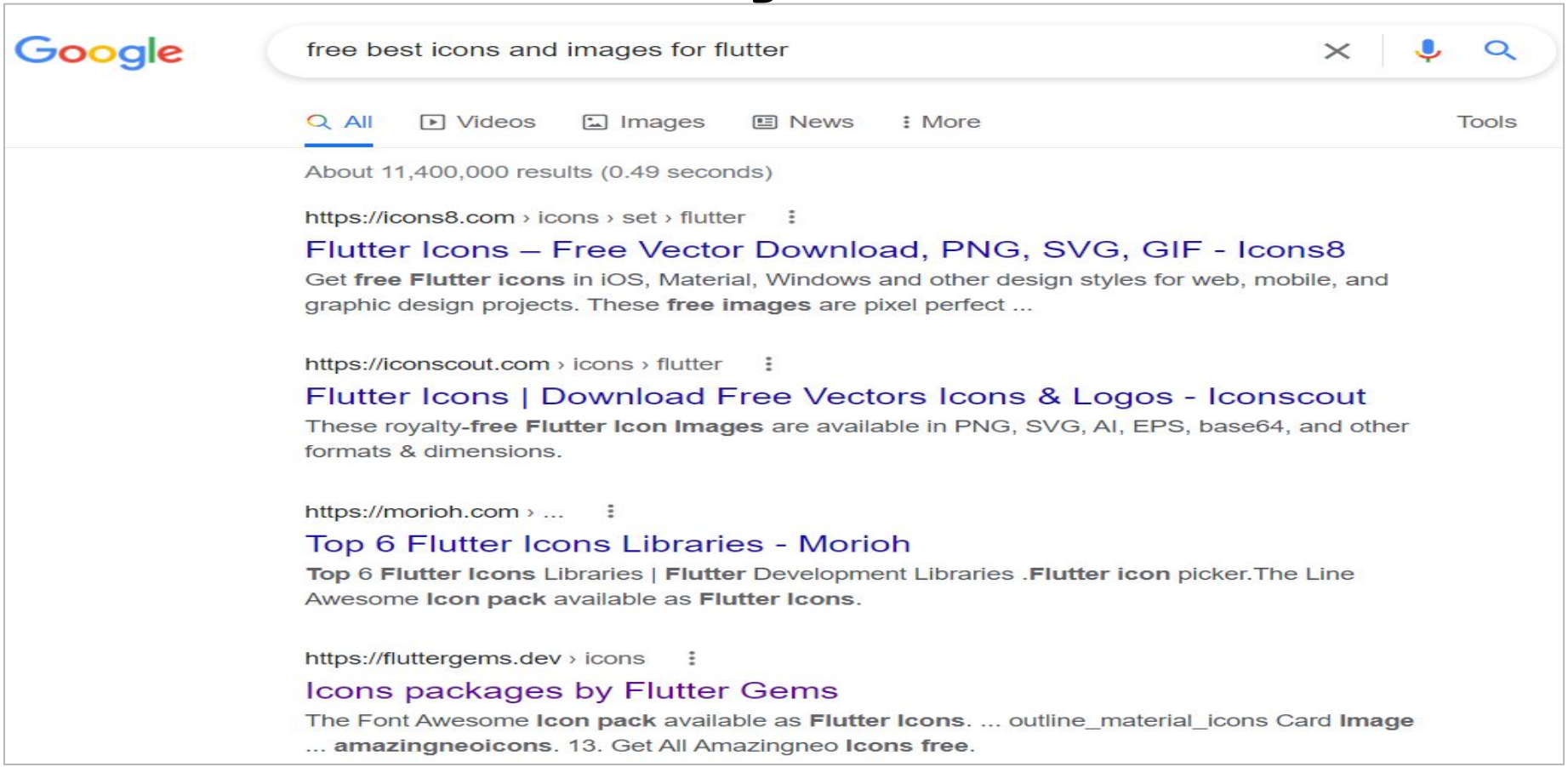


Repo: https://github.com/aamirpinger/flutter-slide-code/

Git Branch: app_icons

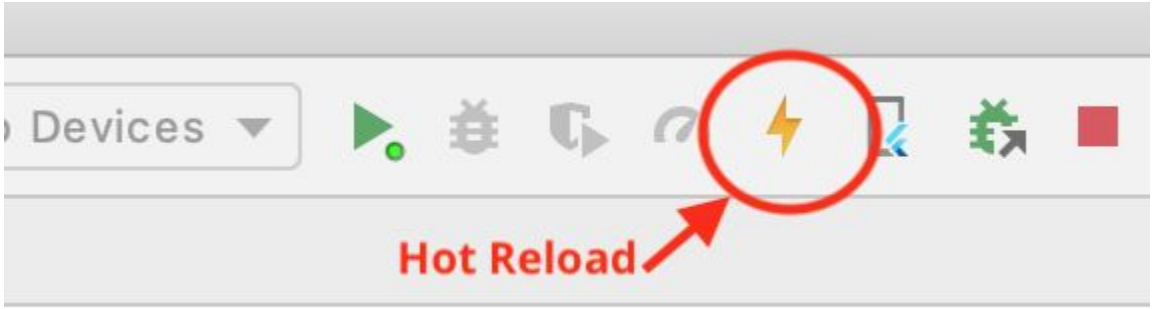
Git Diff: #2

Free to use icons and images



Hot reload

Hot reload



- Hot reload is supported by almost every mobile development frameworks.
- Flutter also provide hot reload option.
- Hot reload allows you to make updates to the code and reflect the change almost instantly to the UI
- Behind the scene flutter that rebuild the widget tree, but touches only section that have changed
- This all happens without having to reload state or recompile your app.
- Currently hot reload feature is not updating the app on device or simulator.

Hot reload

- To get the hot reload feature work we need to restructure our code by using stateless or stateful custom widgets
- Instead of pass widget tree in **runApp(),** we create a new custom widget and pass that custom widget to runApp.

```
// Current way

import 'package:flutter/material.dart';

void main() {
 runApp(MaterialApp(
   home: Text('My Flutter App'),
 ));
}
```

```
   );
 }}
```

Hot Reload

Repo: https://github.com/aamirpinger/flutter-slide-code/

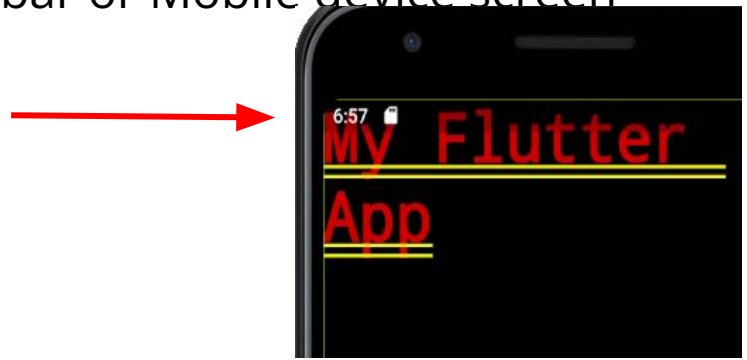
Git Branch: feature/hot_reload

Git Diff: #3

SafeArea( )

SafeArea( )

- Whenever we create a base widget it is by default align to top left corner
- Sometimes it not even visible because of Appbar or Mobile device screen design.
- SafeArea can solve this issue.



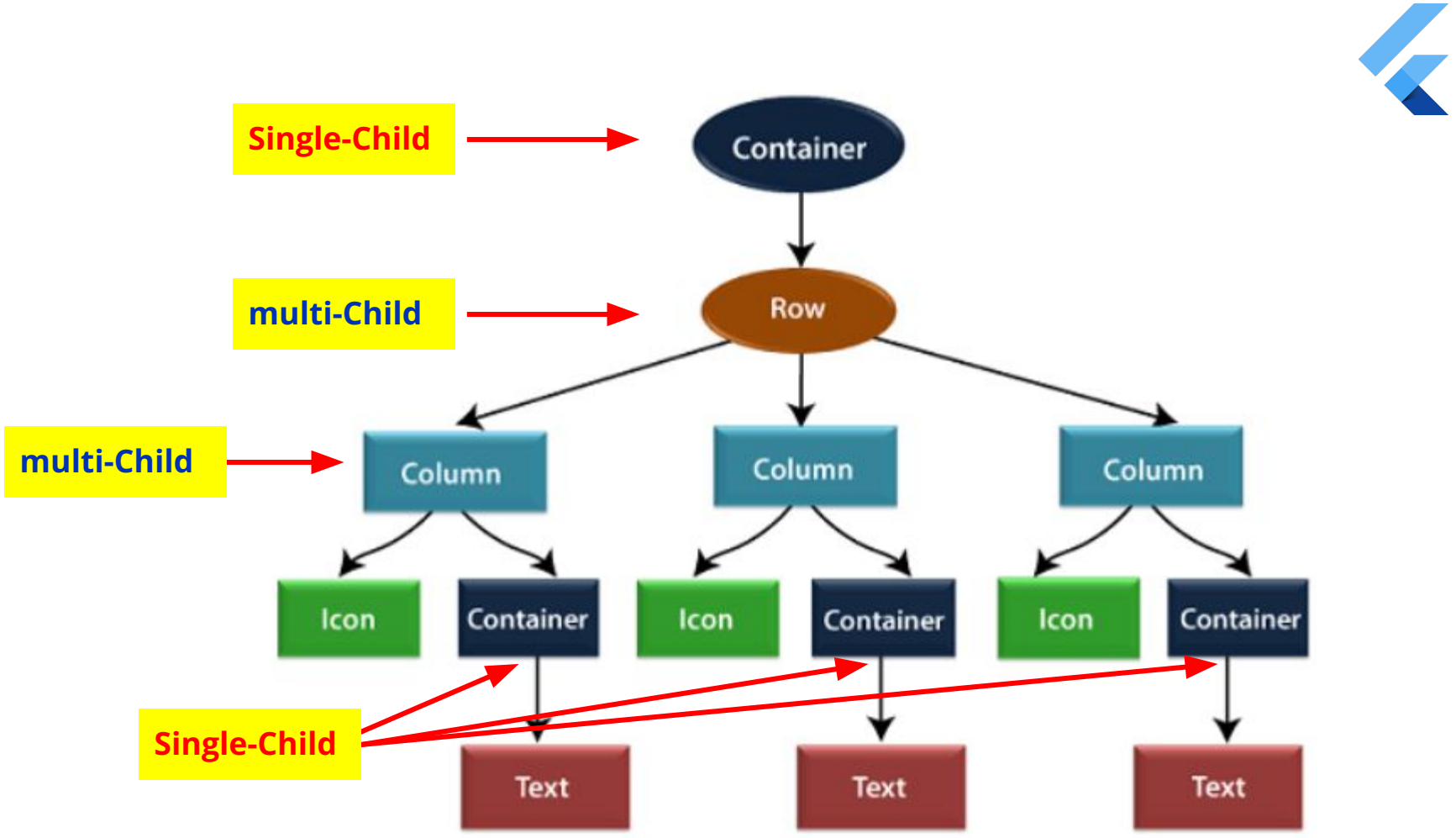

```
MaterialApp(
 home: SafeArea(child: Text('My Flutter App')),
);
```

Layout widgets

Layout Widgets

- Everything in flutter is a widget.
- Flutter allows us to create a layout by composing multiple widgets to build more complex widgets.
- Layout widget can be categories into two types:
  - Single Child Widget
    - Widget that has only one child.
  - Multiple Child Widget
    - Widget may have multiple childs.

https://flutter.dev/docs/development/ui/widgets/layout



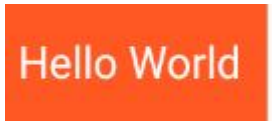
Source: https://www.javatpoint.com/flutter-layouts

Container( )

Container( )

- Container is a widget class that allows us to customize the child widget.
- It is mainly used to add borders, padding, margins, background color, and many more.

```
body: SafeArea(
 child: Container(
   child: Text('Hello World'),
   padding: EdgeInsets.symmetric(vertical: 20, horizontal: 12),
   margin: EdgeInsets.only(left: 10, top: 20),
   color: Colors.deepOrange,
   width: 150,
   height: 70,
 ),
),
```



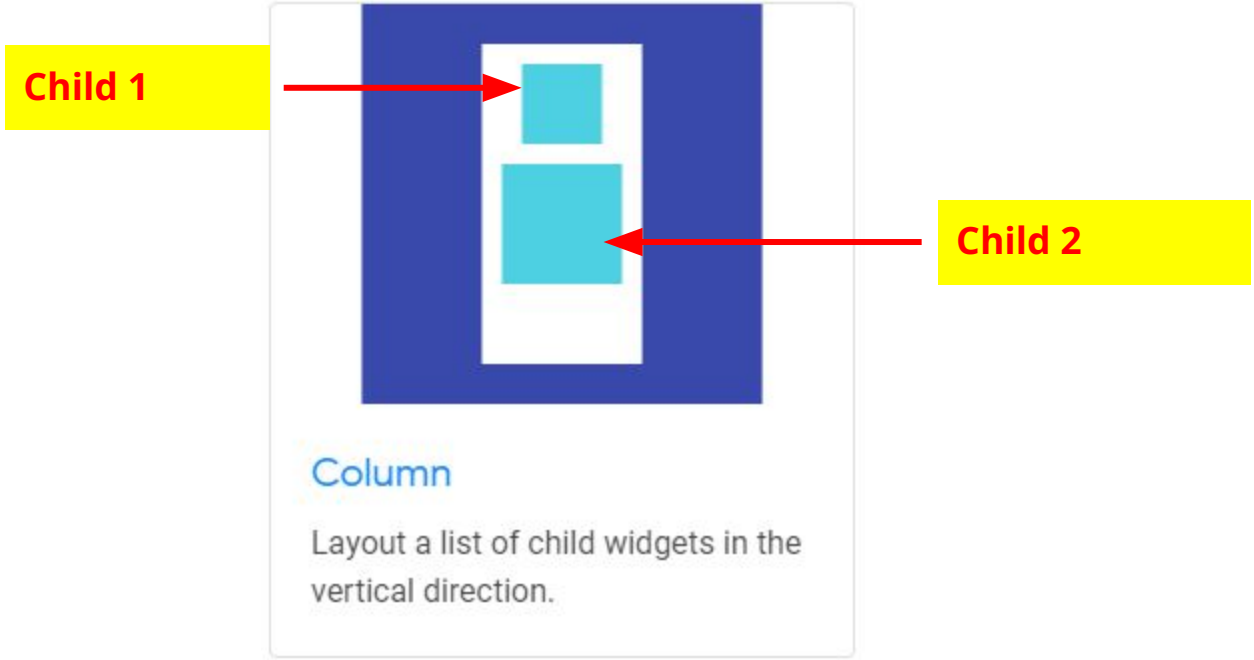
Git Branch: container_widget

Git Diff: #4

Row( ) & Column( )

Column( )

- Column is a Multi-child layout widgets.

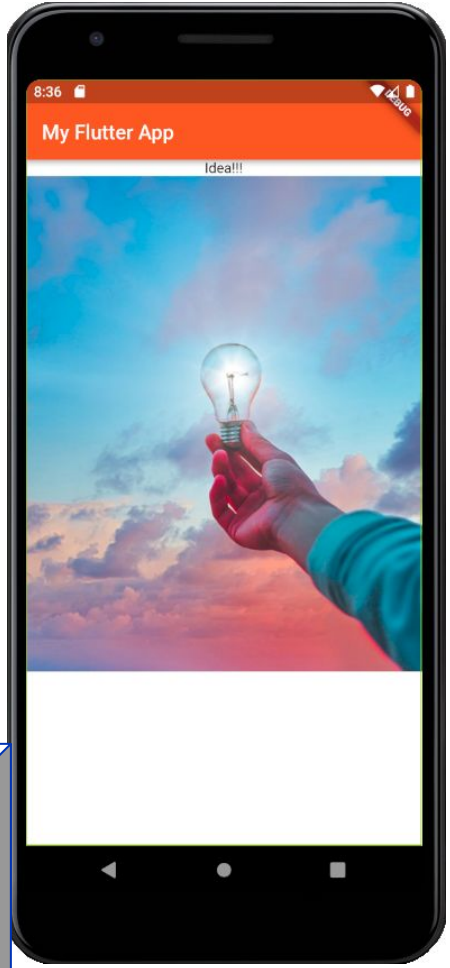


Columns( )

```
body: SafeArea(
  child: Column(
    children: [
      Container(
        child: Text(
          'Idea!!!',
        ),
      ),
      Image(
        image: AssetImage('images/bulb.jpg'),
      ),
    ],
  ),
),
```

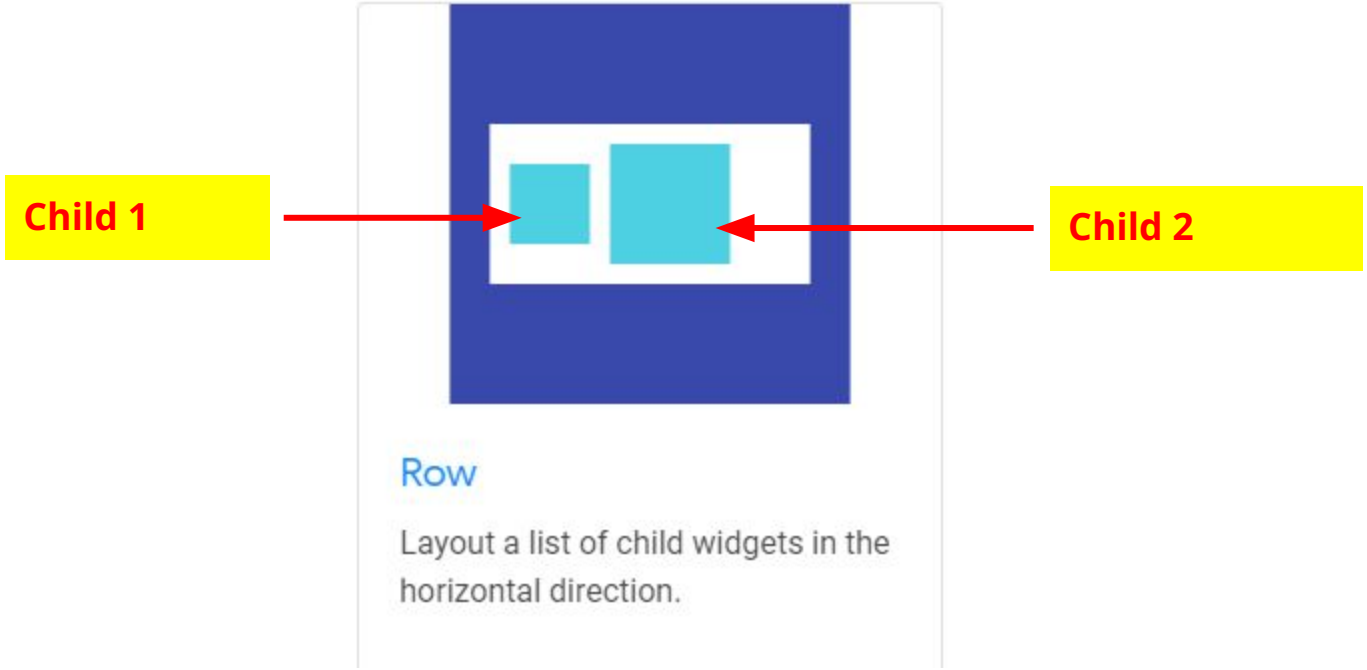
Git Branch: column_widget

Git Diff: #5



Row( )

- Row is also a Multi-child layout widgets.

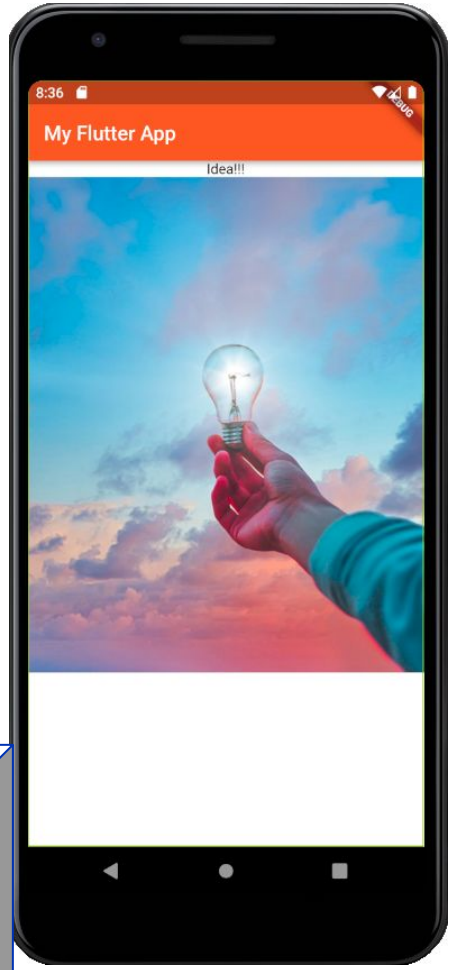


Row( )

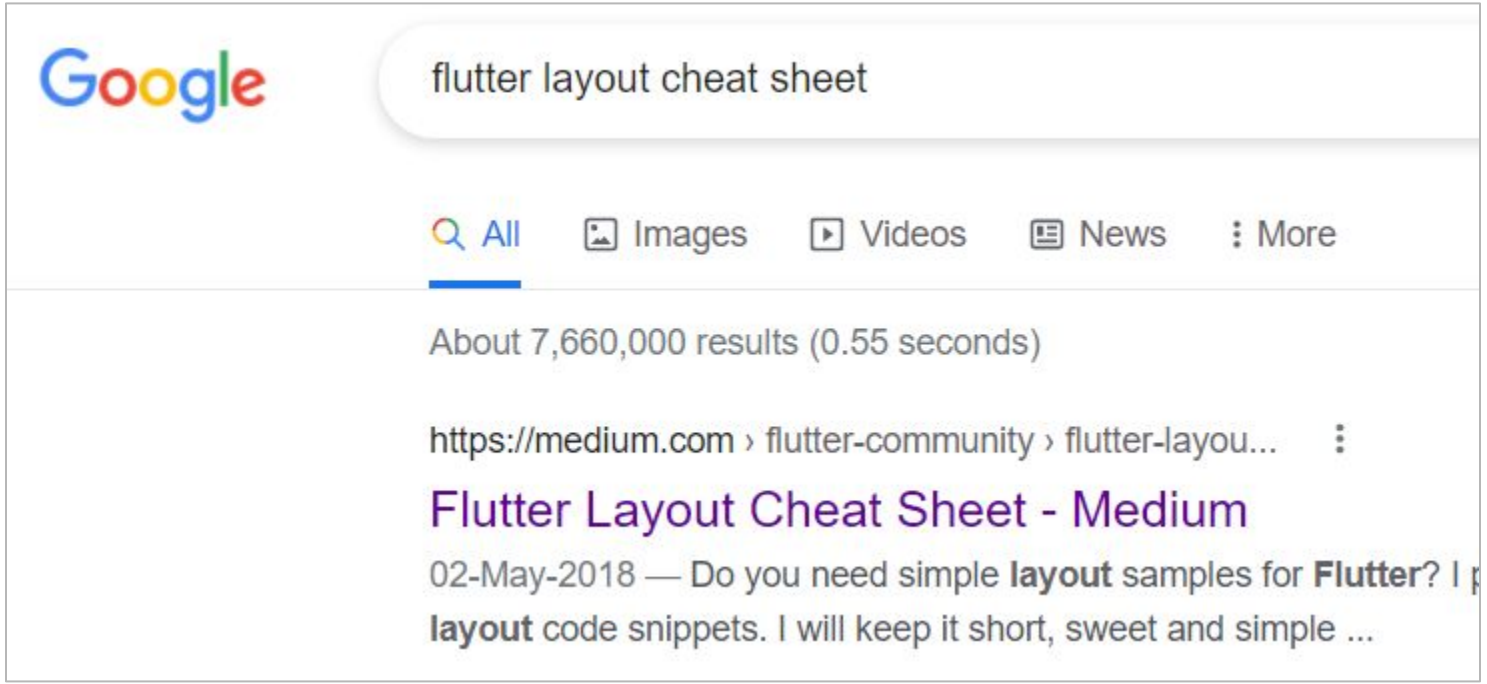
```
body: SafeArea(
  child: Column(
    children: [
      Container(
        child: Text(
          'Idea!!!',
        ),
      ),
      Image(
        image: AssetImage('images/bulb.jpg'),
      ),
    ],
  ),
),
```

Git Branch: row_widget

Git Diff: #6



A must practice task, practice every thing from the article.



Direct Link

Round Shaped Image

Round Shaped Image

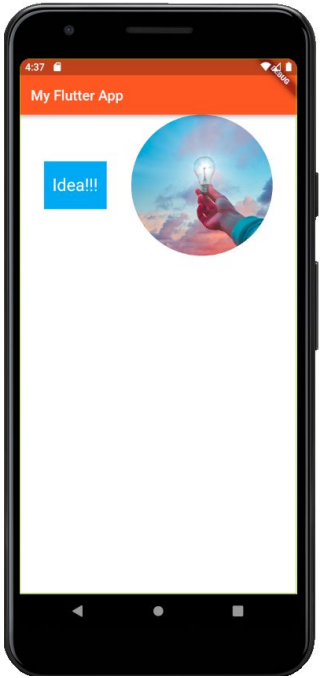
- Image can be rounded in various ways
- Container( ) can be decorated with round shaped box decorated image
- If the image is avatar, the CircleAvatar class can be used.

Git Branch: rounded_image

Git Diff: #8

Git Branch: circleAvatar

Git Diff: #9

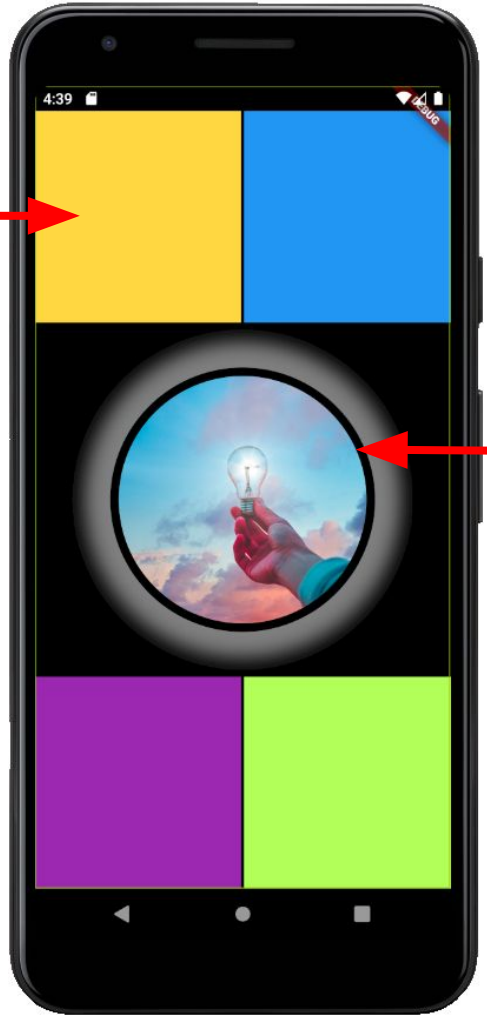


Exercise - (Rows & Columns)

All 4 boxes

Width: 195

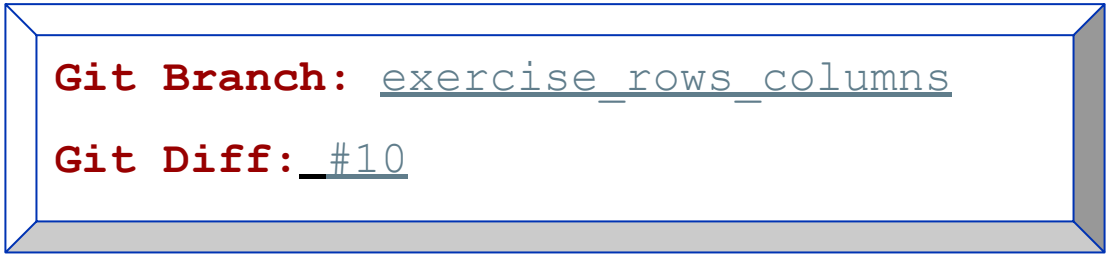
Height: 200



- borderRadius: 150

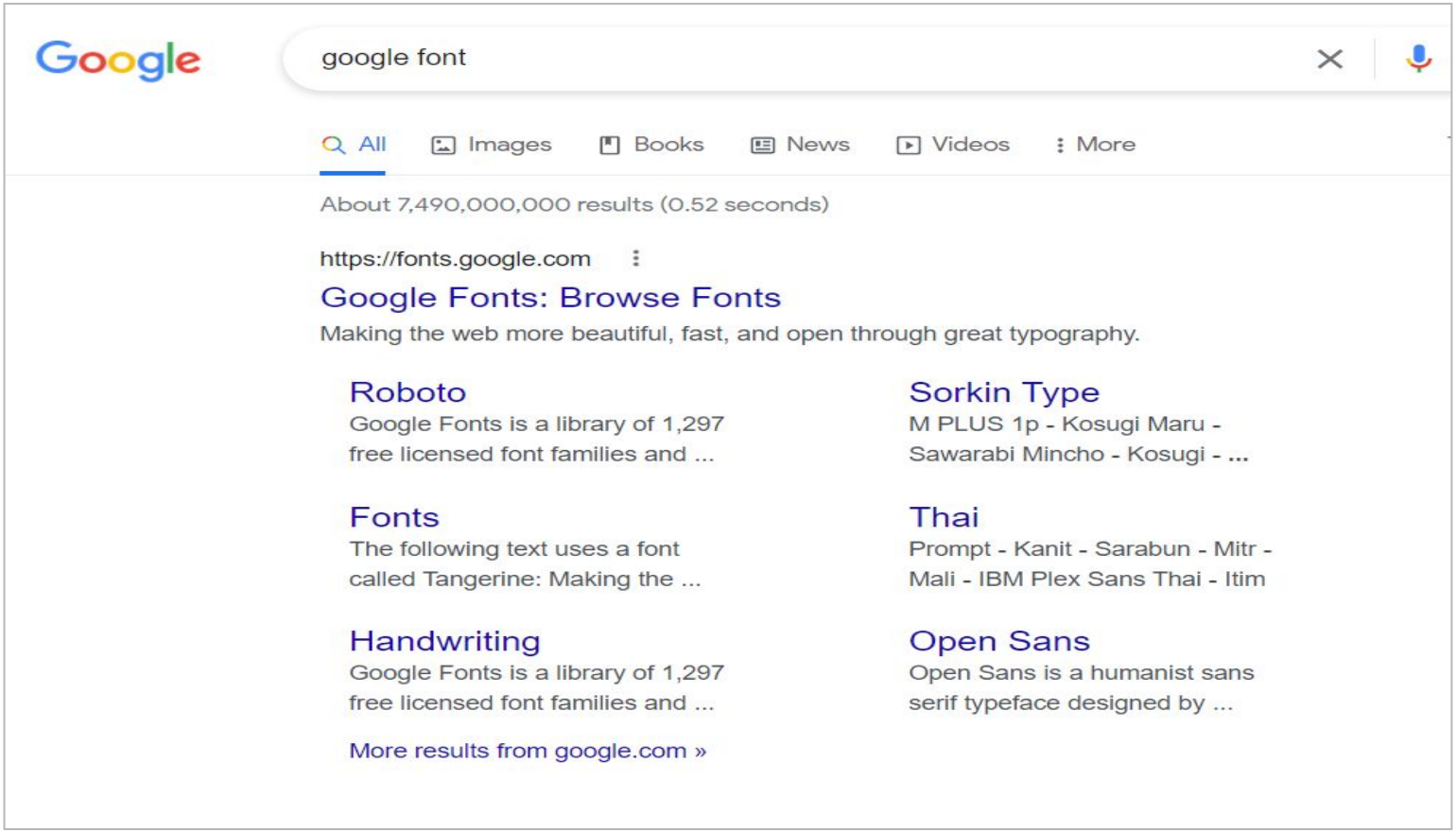
- border width: 8

Solution - (Rows & Columns)



External Font

Download a font



External Font

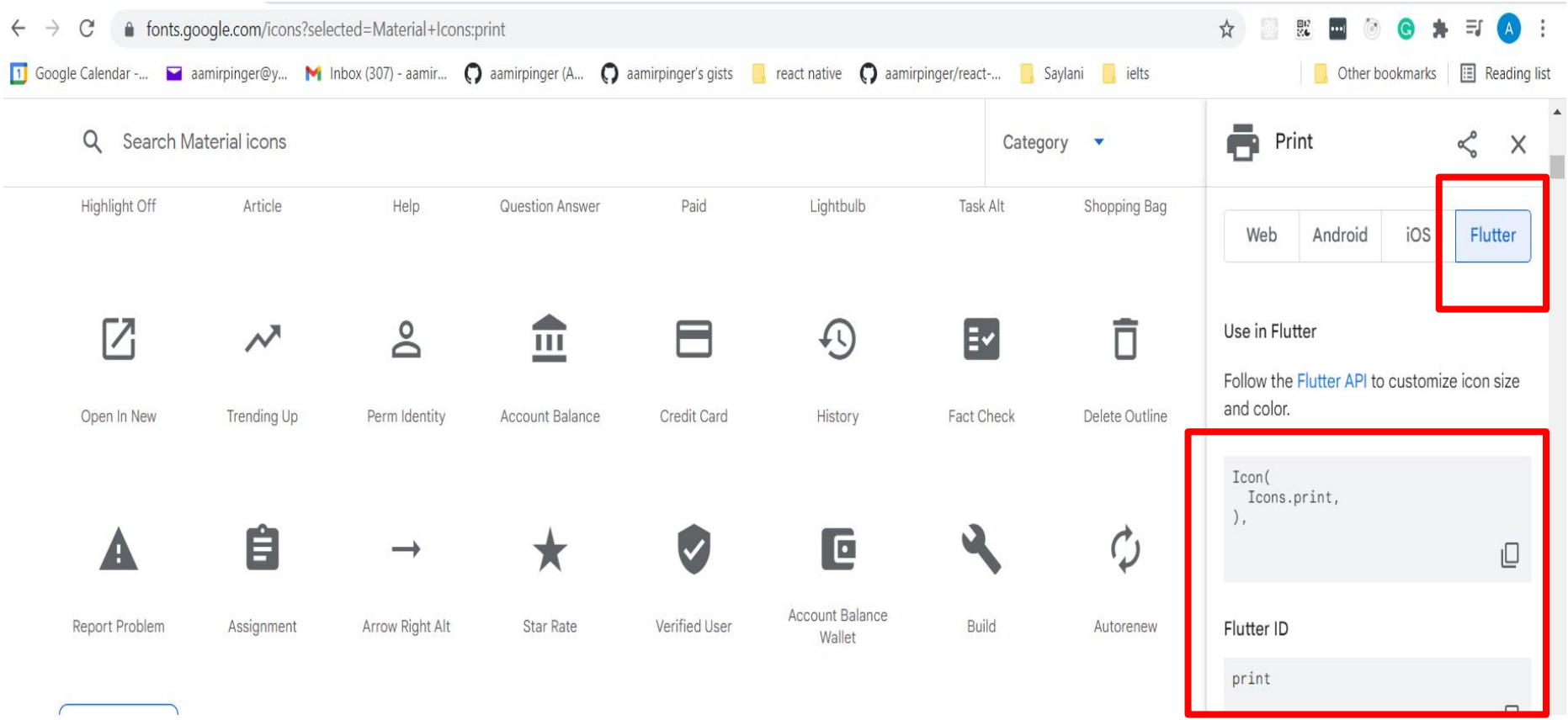
- Google provide us 100s of free fonts.
- Download the font.
- Create a fonts folder like we created images folder
- Extract the .ttf file from downloaded zip file into fonts folder
- Add entry in pubspec.yaml (Be sure about indentation)
- Use it in the App

Git Branch: custom_font

Git Diff: #11

Icons

https://fonts.google.com/icons



Icons

- Google provides tons of icons
- Icons are drawn at run time, styling it and scaling up is no problem

```
Icon(
 Icons.star_border,
 color: Colors.red,
 size: 50,
)
```

Git Branch: material_icons

Git Diff: #12

Card( )

Card title

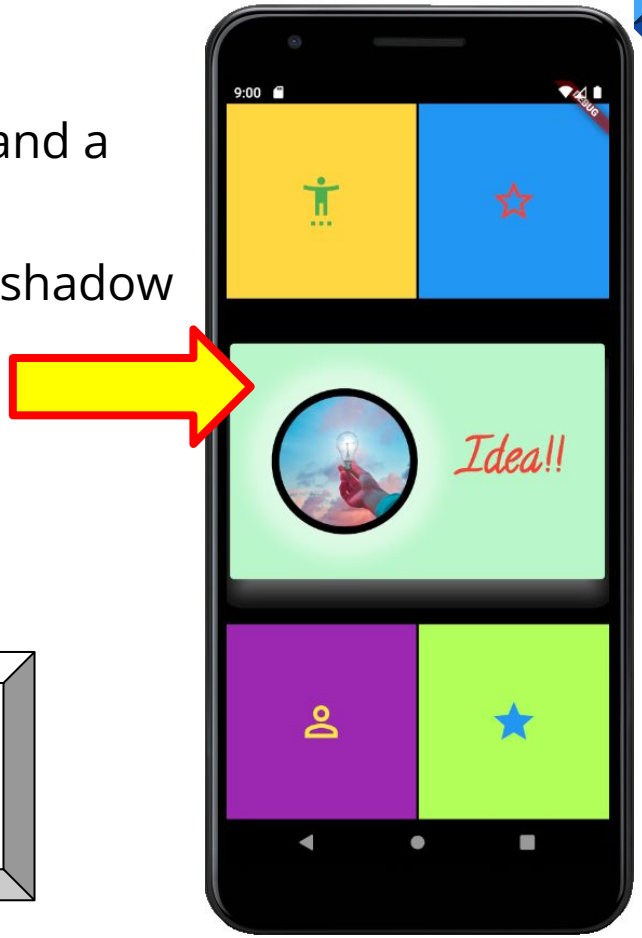
Secondary text

Greyhound divisively hello coldly wonderfully marginally far upon excluding.

ACTION 1 ACTION 2

Card( )

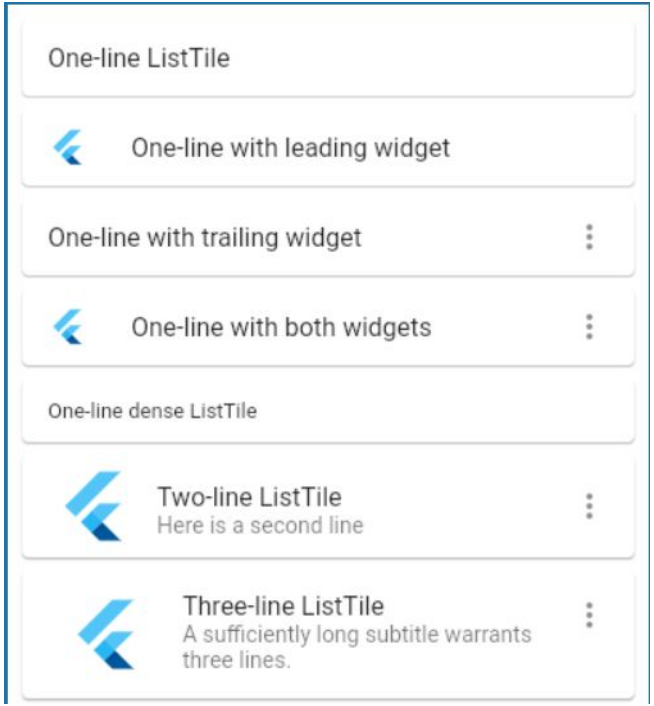
- Card widget provides panel with round corners and a slight elevation on the lower side.
- Easily customizable properties like color, shape, shadow color etc.



Git Branch: card_sizedBox_widget

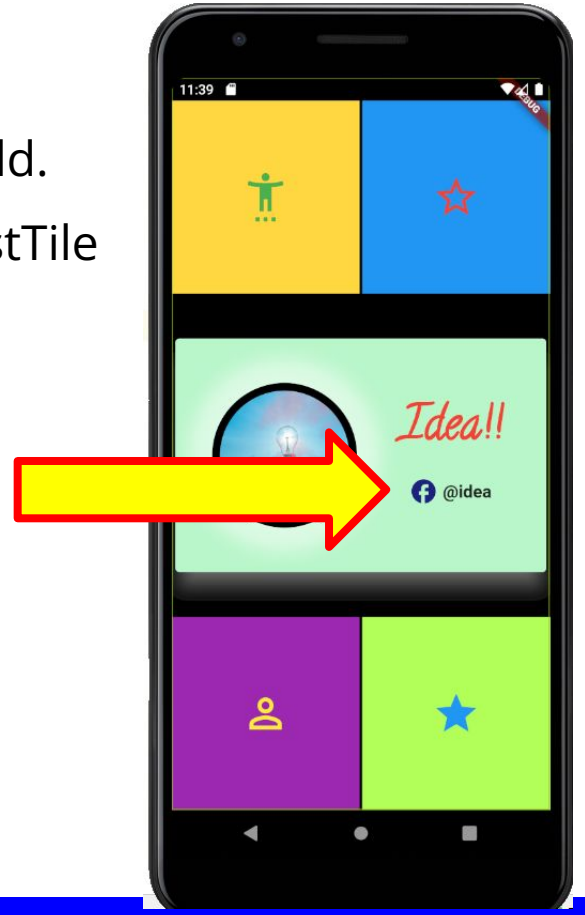
Git Diff: #13

ListTile( )


One-line ListTile
One-line with leading widget
One-line with trailing widget
One-line with both widgets
One-line dense ListTile
Two-line ListTile
Here is a second line
Three-line ListTile
A sufficiently long subtitle warrants three lines.

ListTile( )

- Provide an easy way to display text in a small panel
- listTile( ) automatically add default padding for its child.
- Mainly there are four parts that is auto handled by listTile
  - Leading → e.g. icon
  - Title → e.g. Text
  - Trailing → e.g. menu action button



Git Branch: listTile

Git Diff: #14

Exercise - Widget for Testimonial

Widget for Testimonial

- Use you own color theme
- For " " **font_awesome_flutter** library can be used

Git Branch: testimonial_widget_exercise

Git Diff: #15

Buttons

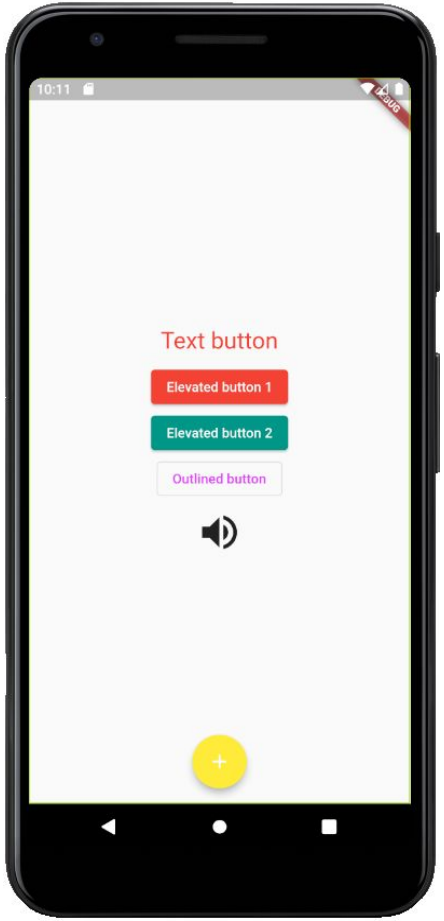
Buttons

- Flutter provides many types of buttons, some are
  - Text Button
  - Elevated Button
  - Outlined Button
  - Icon Button
  - Floating Action Button

Good Read:
https://material.io/components/buttons/flutter#contained-button

Git Branch: buttons

Git Diff: #16



Stateful Widget

Stateful Widget

- A widget is either stateful or stateless.
- Stateless widgets are immutable,
- In simple words, Stateless widgets cannot change their state during the runtime of the app, which means the widgets cannot be redrawn while the app is in action.
- If a widget can change—when a user interacts with it, for example—it’s stateful.

Source1: https://flutter.dev/docs/development/ui/interactive
Source2: https://medium.com/flutter-community/flutter-stateful-vs-stateless-db325309deae

Stateful Widget

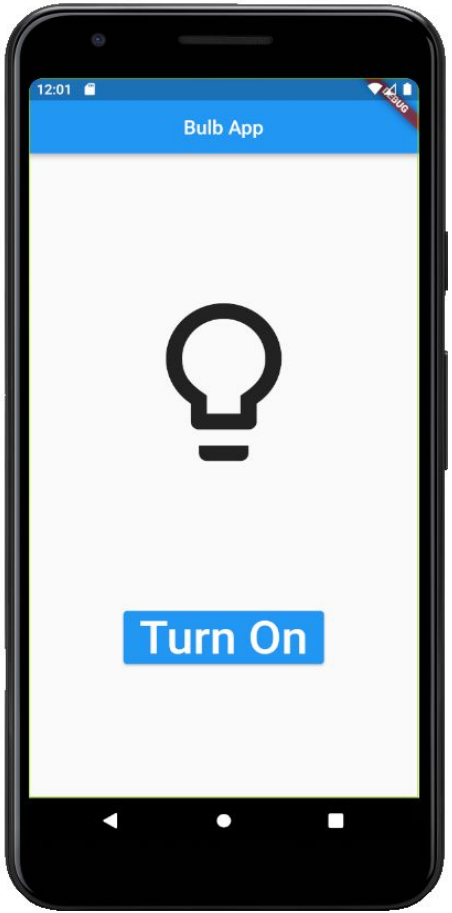
- For example, in a counter app, you have two widgets a text widget and a button widget.
- Text widget should show updated counter value when button is pressed.
- In other words, on button press, app should increase the number and redraw Text widget with new number.
- A stateful widget is dynamic: for example, it can change its appearance in response to events triggered by user interactions or when it receives data.
- A widget’s state is stored in a State object, separating the widget’s state from its appearance.
- setState() method help updating the state and redraw the UI.

Stateful Widget

- A widget's state is stored in a State object, separating the widget's state from its appearance.
- setState() method help updating the state and redraw the UI.
- **PRO Tip**: type **stful** in editor and press enter to create a stateful widget.

Git Branch: stateful_bulb_widget

Git Diff: #17

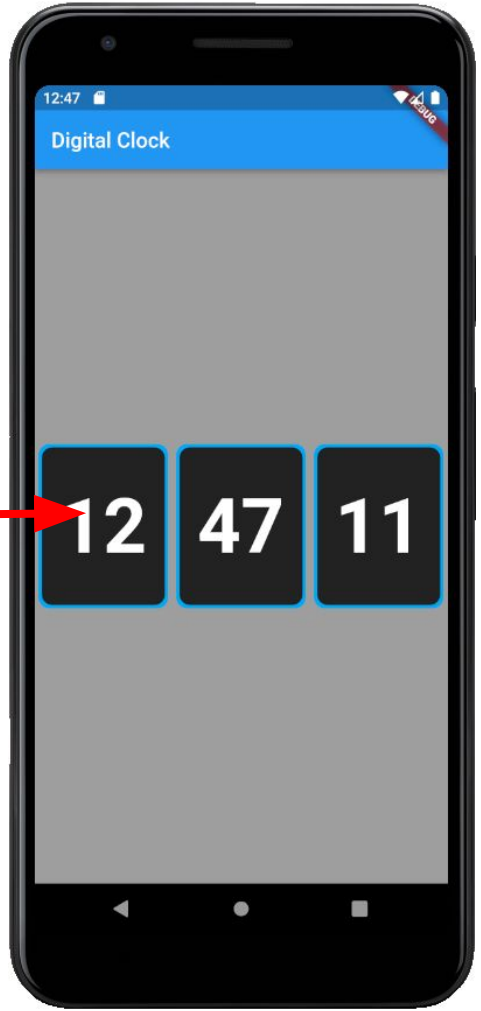


Exercise - Digital Clock

Exercise - Digital Clock

Tips:

- Use StateFul Widget
- Use Future.delay with second: 1
- Use variables for repeated code



Git Branch: digital_clock

Git Diff: #18

Exercise - FlashCards

Guess the Capital City

Exercise - FlashCards - Guess the Capital City

Tips:

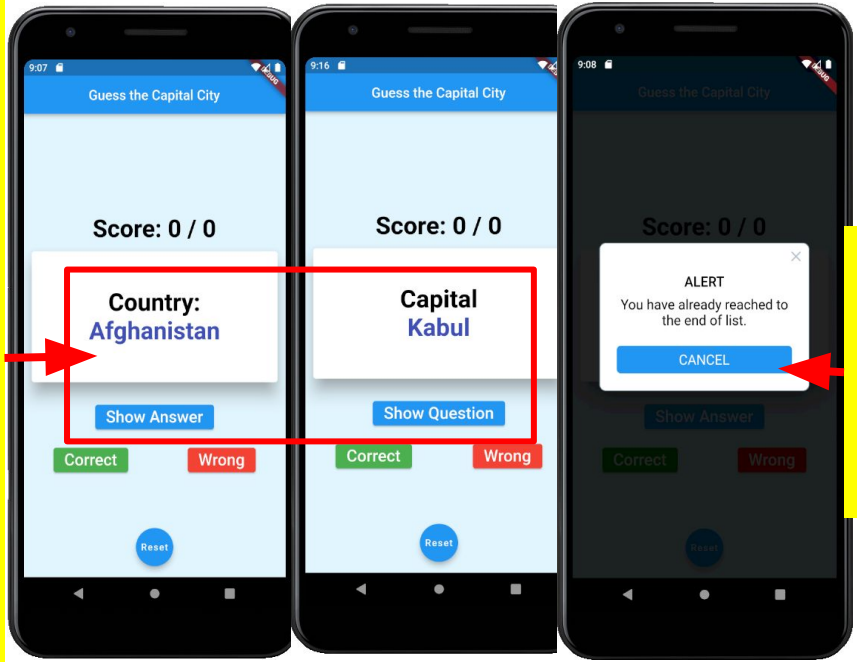
- **Score**: Score / Total Attempted

- **Show Answer:** Display Capital Name

- **Correct**: on press increase the score and total attempted

- **Wrong:** on press increase only total attempted

- **Reset**: Reset score, Country index, and total attempted



Tips:

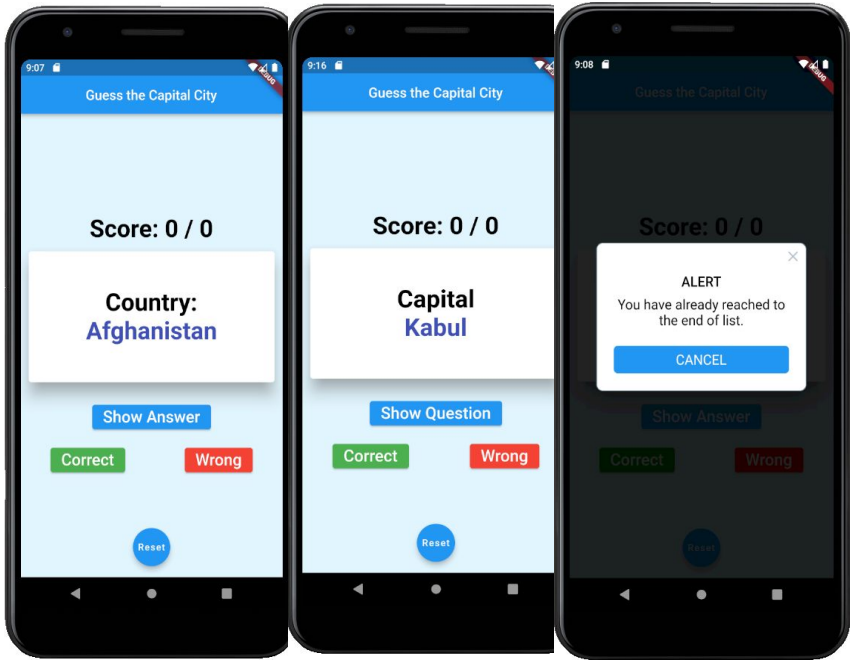
- Use **rflutter_alert** library

- Show alert on end of list

Exercise - FlashCards - Guess the Capital City

Git Branch: flash_card_app

Git Diff: #19



Modularizing the code

Modularizing the code

- It is never considered a good practice to repeat the code.
- Widgets should be broke into smaller pieces.
- Data should be accessed through relevant class rather than accessed directly.

Git Branch: flash_card_app_refactored

Git Diff: #20

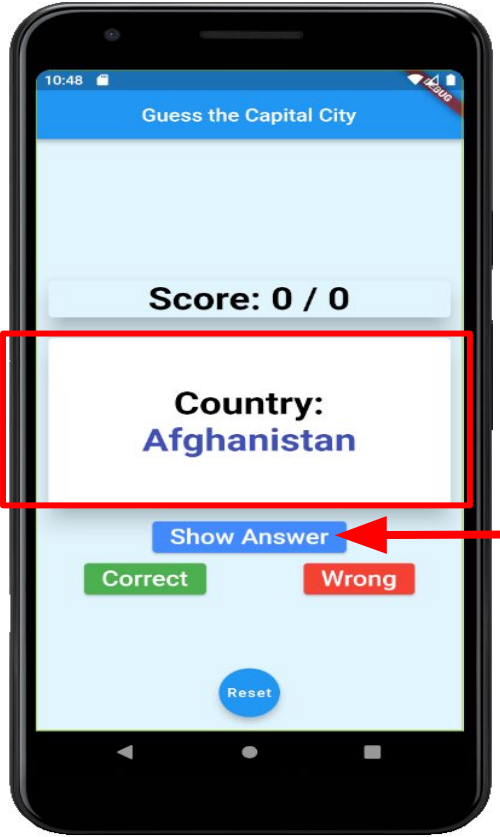
Gesture Detector

Gesture Detector

- There are many widget that do not have onPressed methods, e.g. Card, container etc.
- Let’s think of a previous example where you want to toggle question or answer on tap on the card itself.
- To achieve this, flutter provides us a gestureDetector Widget.
- This widget got numerous properties
- Must visit documentation link **https://api.flutter.dev/flutter/widgets/GestureDetector-class.html**

Gesture Detector

By using gestureDetector, we will toggle question and answer

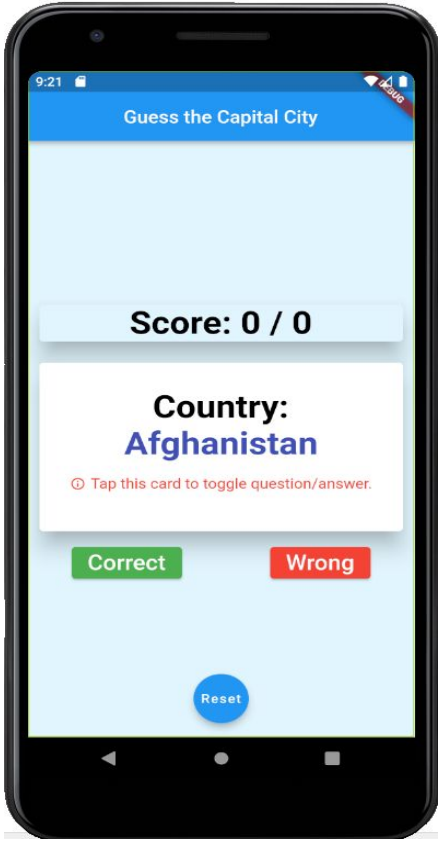


We will remove this button

Gesture Detector

Git Branch: example_gesture_detector

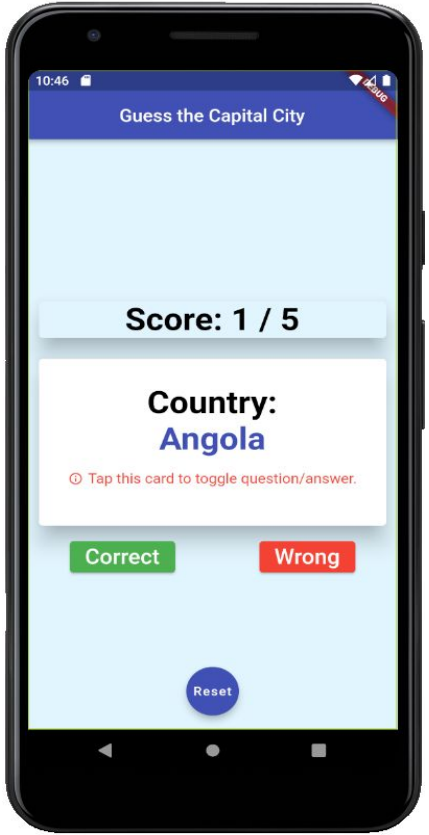
Git Diff: #21



Flutter Themes

Flutter Themes

- One of the best practices is to use Flutter Theme instead of providing colors to individual objects.
- This helps in writing less code and easily maintenance.
- Above all, colors stay consistent at all screens.



Good read:
https://flutter.dev/docs/cookbook/design/themes

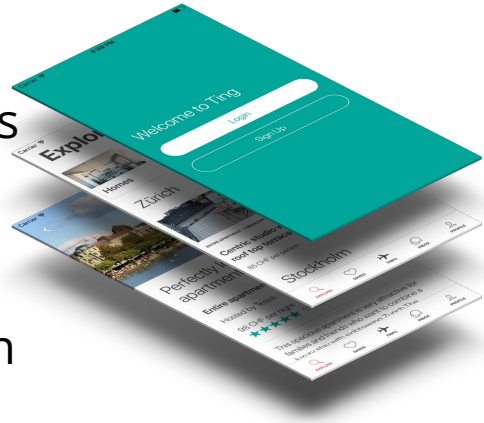
Git Branch: flutter-themes

Git Diff: #22

Navigation

Navigation

- Moving around different screens is made easy by Flutter.
- Screens are referred as Routes in navigation
- e.g. Changing screen from Main screen to Contact Us screen is simply called in Flutter as Navigating to contact us route.
- Behind the scene, Flutter maintains a route stack
- When we move to new screen, it simply pushes that screen to the top of stack.
- When back button is pressed Flutter simply pops out the current screen from the stack and show next screen in line.

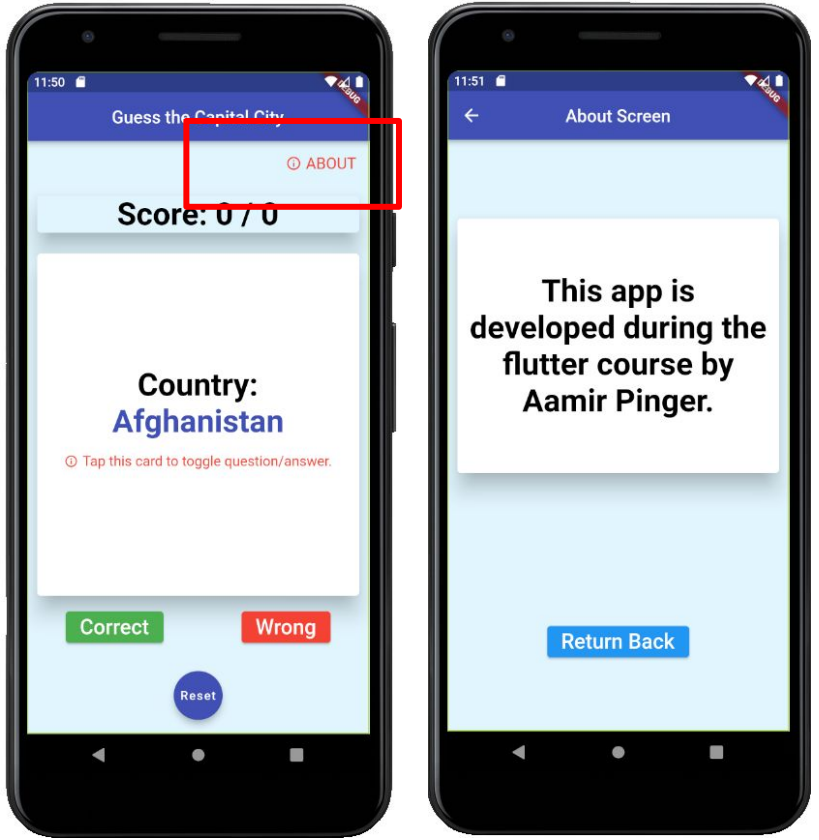


Good read: https://docs.flutter.dev/cookbook/navigation/navigation-basics

Navigator Push and Pop

Navigator Push and Pop

- Let's add About link at the top and on tap we will navigate to a new screen.
- On the About screen we will provide a button to return back to previous screen.



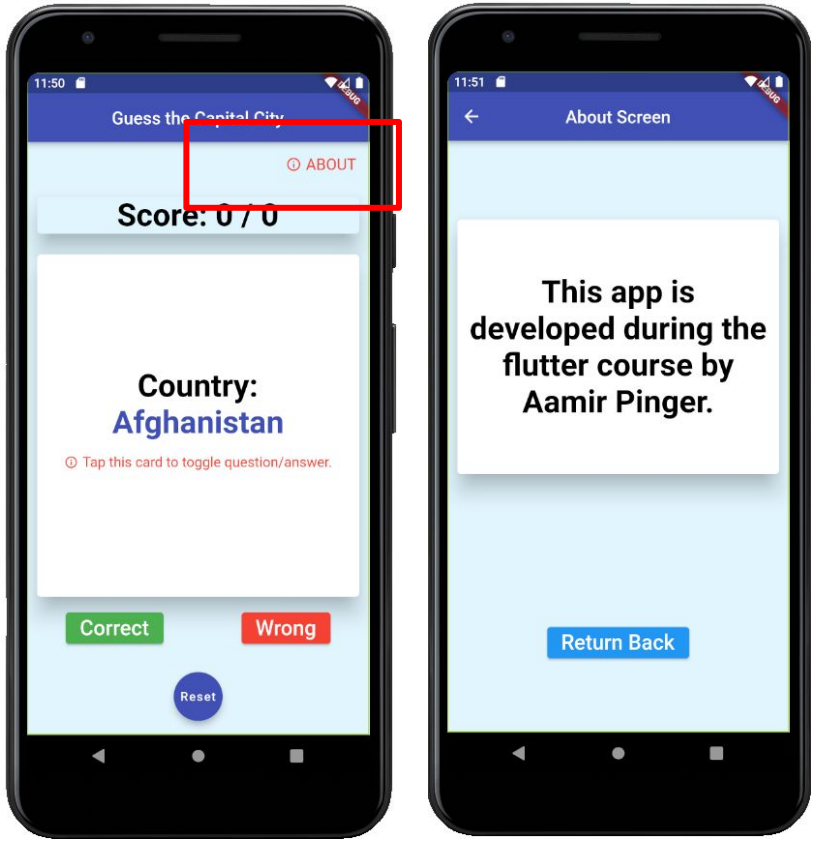
Git Branch: basic_navigation

Git Diff: #23

Navigator Named Route

Navigator Named Route

- Most of the time app need to navigate to the same screen from many parts of the app.
- This approach can result in code duplication.
- The solution is to define a named route, and use the named route for navigation.



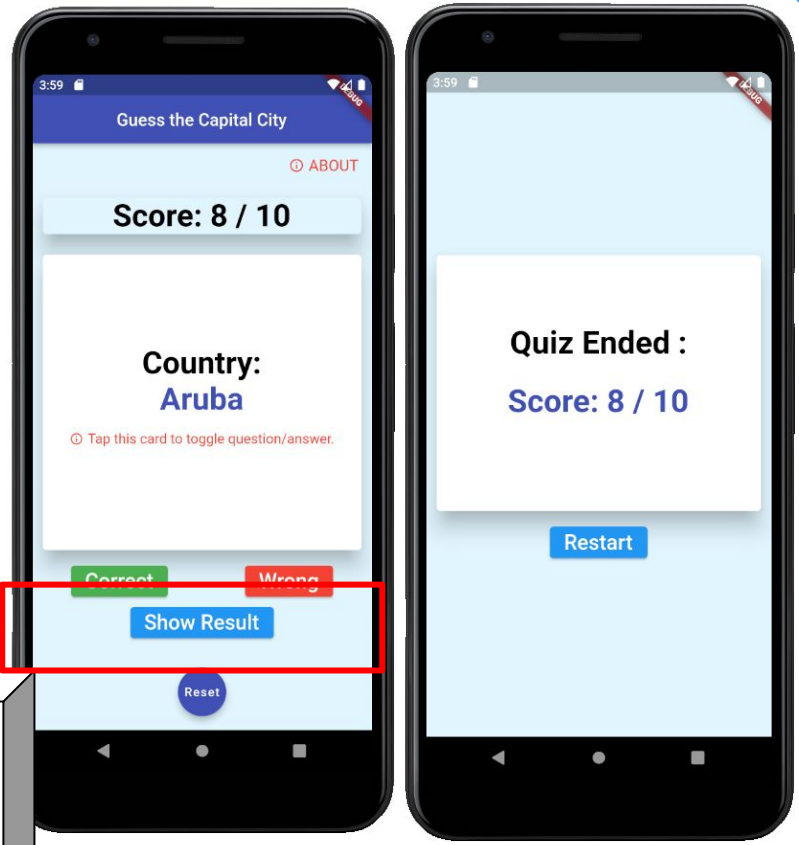
Git Branch: name_route

Git Diff: #24

Passing data on navigation

Passing data on navigation

- Let's add finish test button and on tap move back to results screen showing score.
- On the result screen we will provide a button to retry the test that we move user back to quiz screen.
-



Git Branch: passing_data_to_routes

Git Diff: #25

Refactoring routes

Route name as static class property

- It is a general practice to add a static variable **routeName** in every screen.
- Routes should be maintained in **separate file.**
- Instead of **MaterialApp's routes** property, we will maintain all the routes in **onGenerateRoute**

Git Branch: routes_refactored

Git Diff: #26

Folder structure

Folder structure

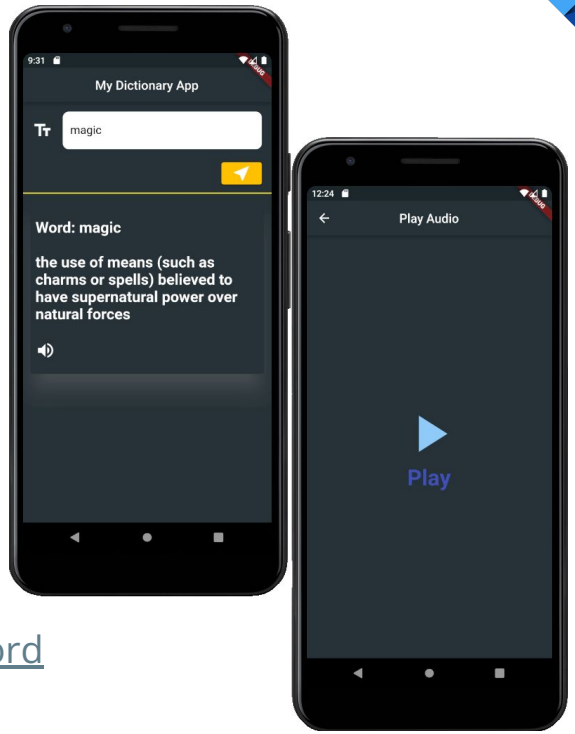
- A good folder structure is very important to keep code maintainable.
- There could be many good folder structures to segregate files based on there functionality.
- For example, all the screens related files should go into the screens folder and all the custom widgets related files should go into the widgets folder.
- We will refactoring the code by
  - Separating **files into relevant folders**
  - Creating **separate file** for **Theme**
  - Creating a **AppStrings** class for all the hard coded text used into the application

```
Git Branch: refactoring_code_folder_structure
Git Diff: #27
```

Dictionary App

Dictionary App

- We will be now creating a dictionary app that will
  - Take a input from user (TextField)
  - Make an API call to get word meaning etc (http)
  - Random word on App load (life cycle events)
  - Audio player for pronunciation audios
- APIs
  - Word meaning: https://www.dictionaryapi.com
  - Random Word: https://random-word-api.herokuapp.com/word
  - Audio: https://media.merriam-webster.com
  - How to find audio docs: https://dictionaryapi.com/products/json



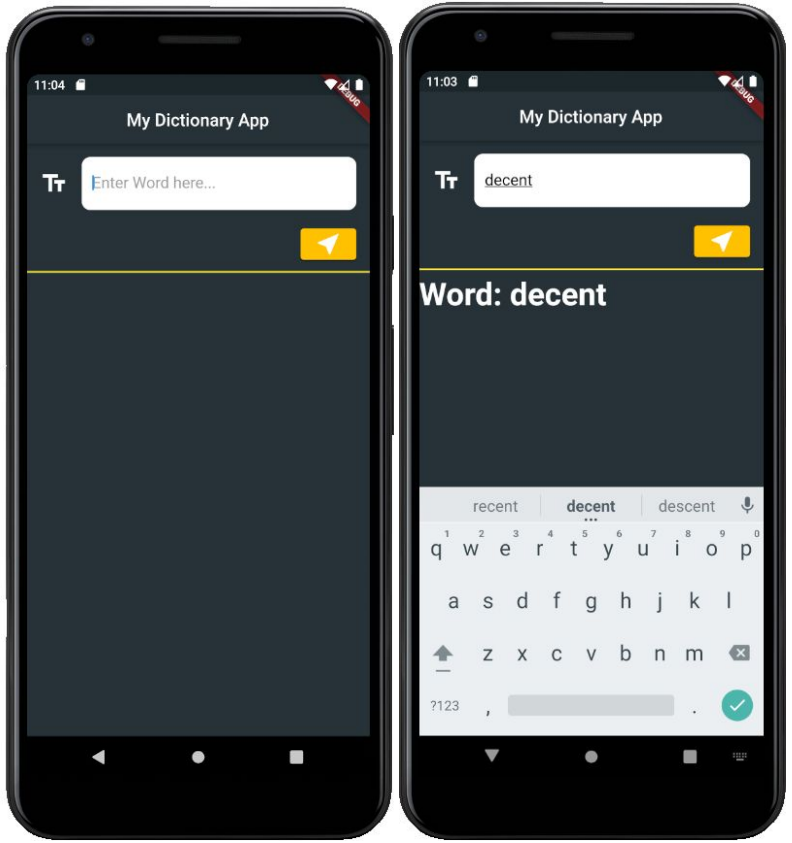
Text Input

Text Input

- On type the word will be displayed on UI.

Git Branch: text_field

Git Diff: #28

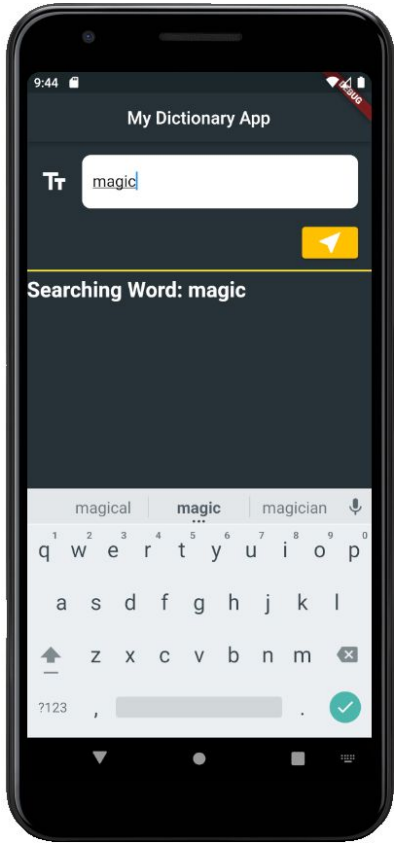


HTTP API call

HTTP API call

- On button tap, api call will fetch the data.
- Once data is fetched, print it in console.

```
Run: main.dart
Console
Performing hot reload...
Syncing files to device Android SDK built for x86 64...
Reloaded 4 of 640 libraries in 500ms.
I/flutter ( 3784): fetching data
I/flutter ( 3784): magic
I/flutter ( 3784): the use of means (such as charms or spells) believed to have supernatural power over natural
I/flutter ( 3784): https://media.merriam-webster.com/audio/prons/en/us/mp3/m/magic001.mp3
I/flutter ( 3784): fetching data complete
```



Git Branch: dictionary_http_api_call

Git Diff: #29

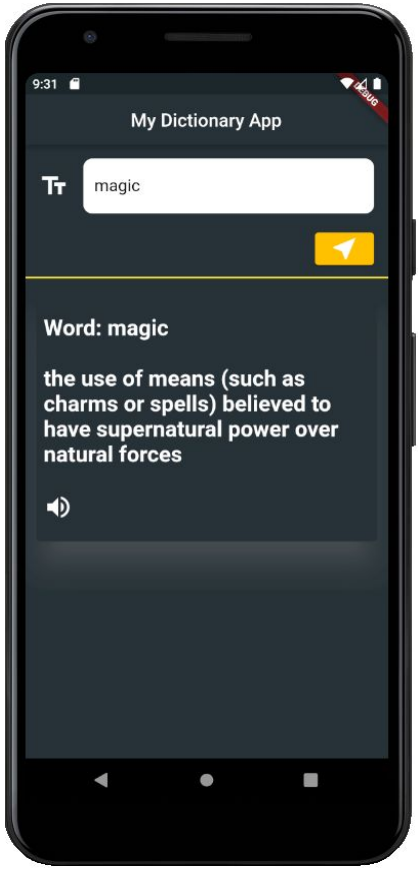
Display Fetched Data

Display Fetched Data

- Update UI when data is received.

Git Branch: dictionary_app_update_ui

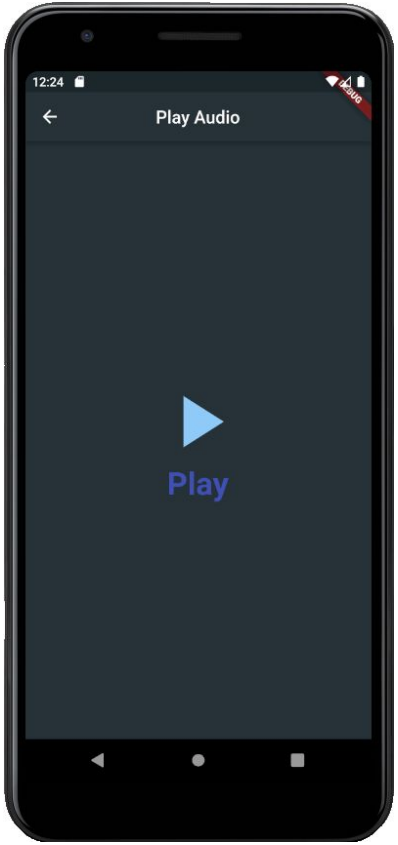
Git Diff: #30



Play Audio

Play Audio

- On click on navigate to playAudio route to play audio
- Click on should play a sound.
- Library hint: just_audio



Git Branch: dictionary_app_audio_player

Git Diff: #31

Next

Displaying random word on App Load

But Let’s first understand lifecycle events.

Displaying random word on App Load

- Functionality is to load a random word from one API and call another API to get the meaning.
- We will be using https://random-word-api.herokuapp.com/word to get the random word and usual API to get the meaning.
- Let's discuss the way to add a functionality that could run once on app load.
- One way could be **DictionaryScreen's build method,** but problem is it will recall the API whenever setState will update the UI.
- Flutter provides us many lifecycle event at which we can do stuff life above.
- **Let's first understand lifecycle events.**

Widget Lifecycle events

Widget Lifecycle events

- As we already know There are two types of widgets in Flutter.
  - Stateless Widgets
  - Stateful Widgets
- **Stateless Widgets** are immutable, meaning they cannot change once rendered on screen, meaning they cannot be changed on the app if data is changing.
- **Stateful Widgets** are mutable, meaning they maintain their state, meaning they do re-render when change occurs. They are Dynamic!

Widget Lifecycle events

- **Stateful Widgets** passes through different lifecycle events.
- This happens when it renders first time or re-renders because of state change.
- Each lifecycle event is attached with one specific method, and we can override them too!

Widget Lifecycle events

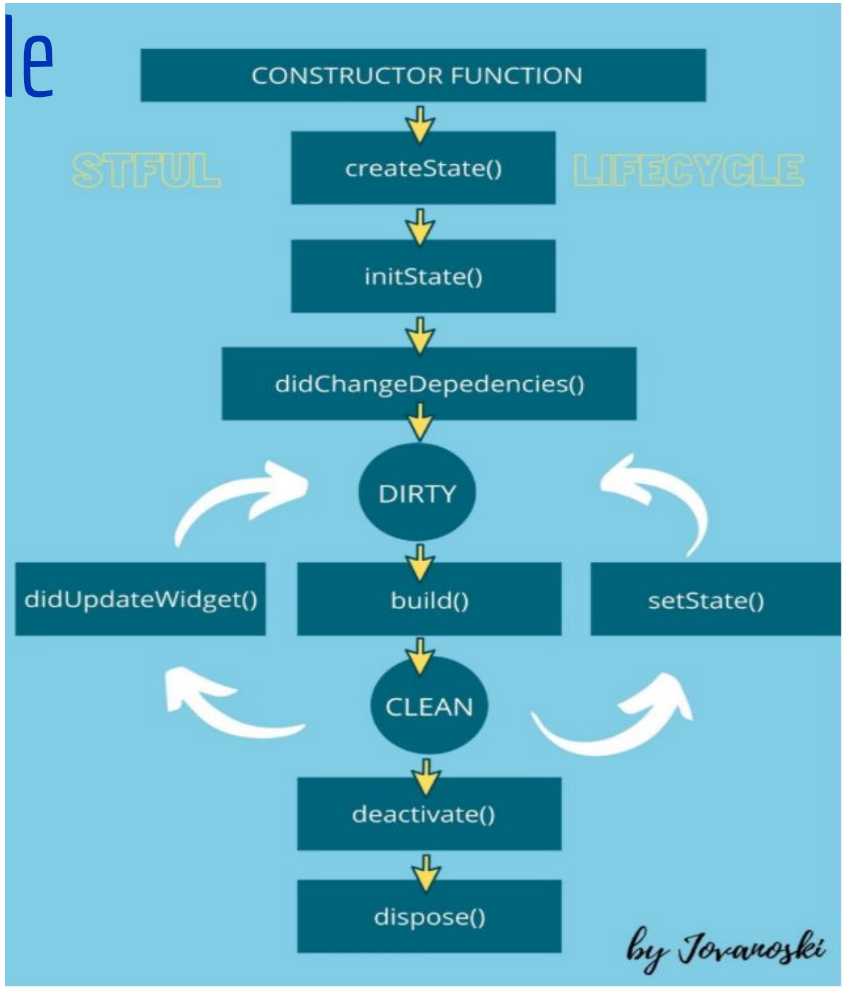
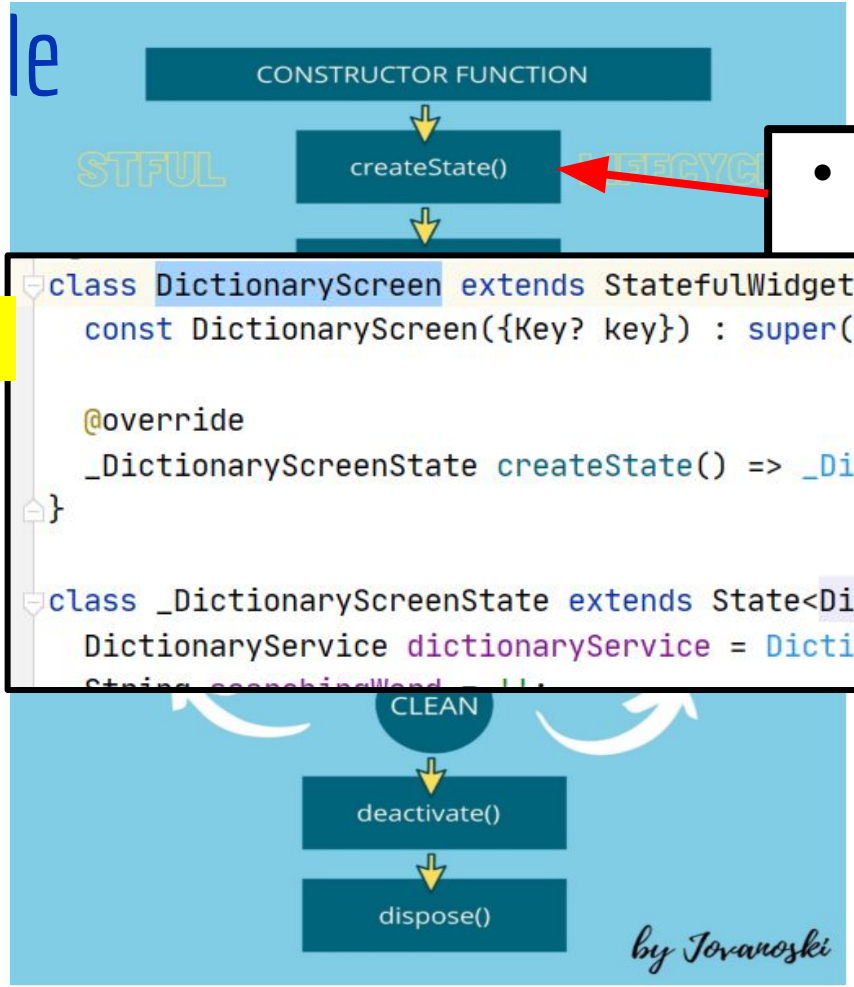


Image by Jelena Jovanoski.

Source: https://betterprogramming.pub/stateful-widget-lifecycle-a01c44dc89b0

Widget Lifecycle events



- Required method for stateful widget.

Example

```
class DictionaryScreen extends StatefulWidget {
  const DictionaryScreen({Key? key}) : super(key: key);

  @override
  _DictionaryScreenState createState() => _DictionaryScreenState();
}

class _DictionaryScreenState extends State<DictionaryScreen> {
  DictionaryService dictionaryService = DictionaryService();
```

Image by Jelena Jovanoski.

Source: https://betterprogramming.pub/stateful-widget-lifecycle-a01c44dc89b0

Widget Lifecycle events

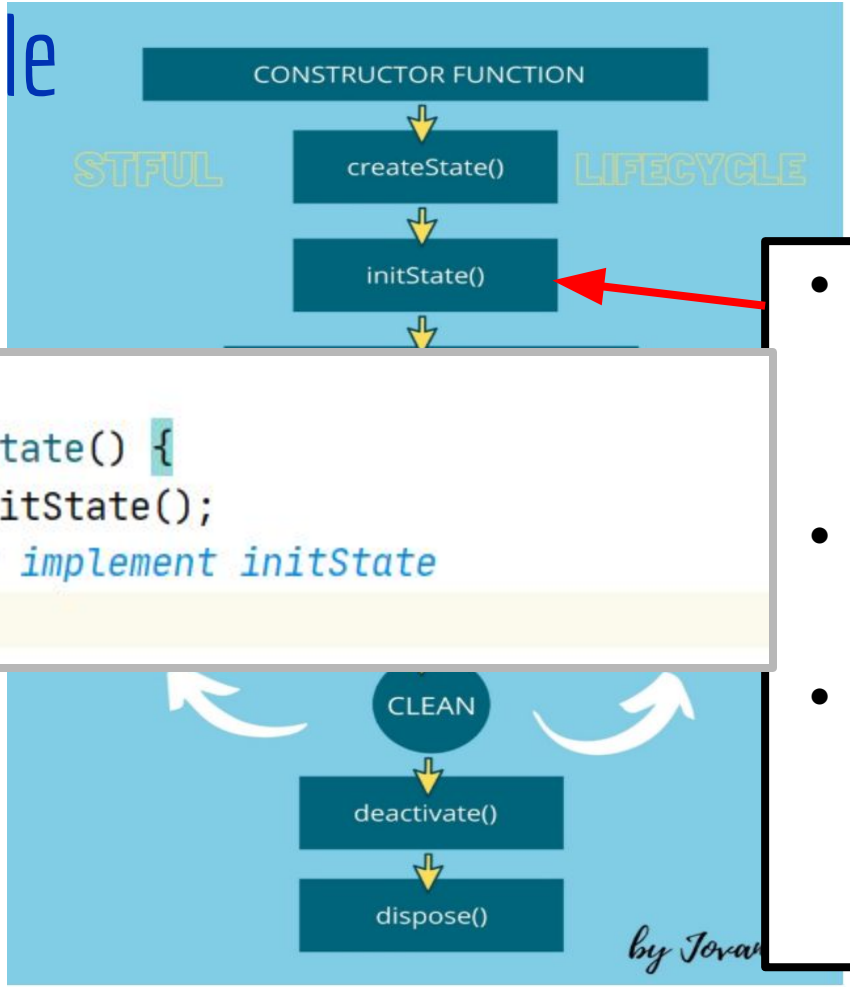


Image by Jelena Jovanoski.

Example

```
@override
void initState() {
  super.initState();
  // TODO: implement initState
}
```

- When the object is inserted into the tree this method is automatically executed after the class constructor.
- Is called only once, when the state object is created for the first time.
- Tip: Use this method to manage HTTP requests and subscribe to streams or any other object that could change the data on this widget.

Source: https://betterprogramming.pub/stateful-widget-lifecycle-a01c44dc89b0

Widget Lifecycle events

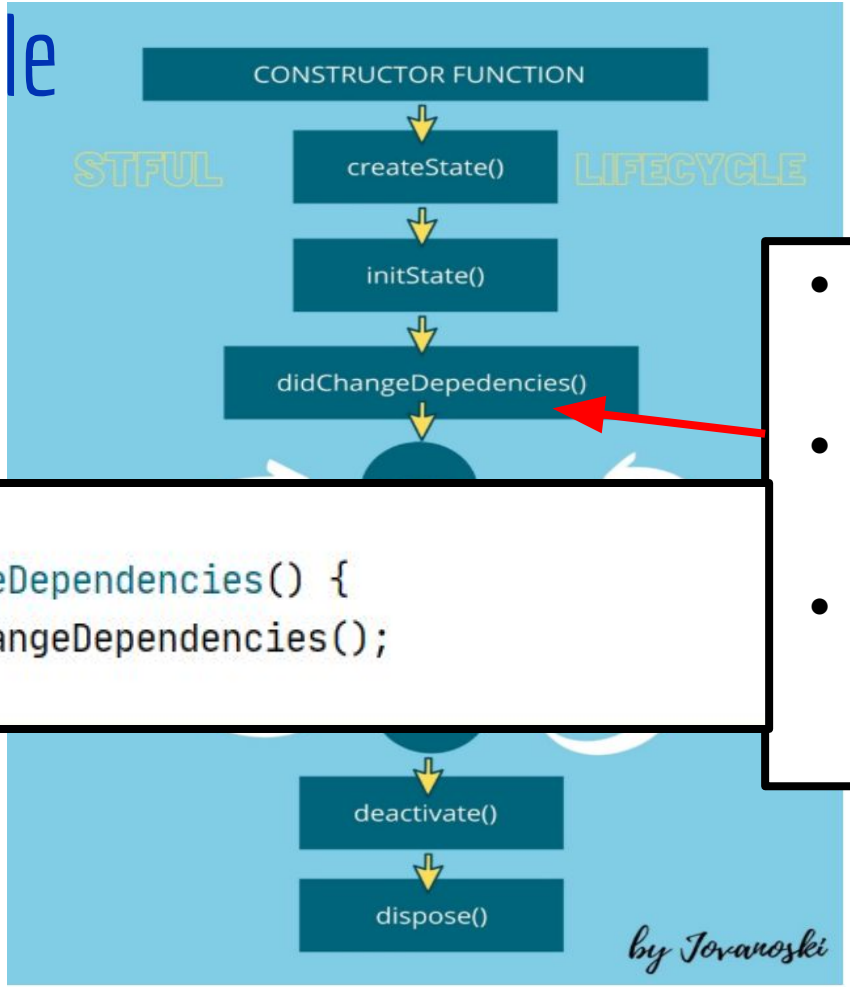


Image by Jelena Jovanoski.

- This method gets called immediately after the initState().
- It will also be called when an object that a widget depends on changes.
- Build method is always called after this method, that is why this method is rarely used.

Example

```
@override
void didChangeDependencies() {
  super.didChangeDependencies();
}
```

Source: https://betterprogramming.pub/stateful-widget-lifecycle-a01c44dc89b0

Widget Lifecycle events

Example

```
@override
void didUpdateWidget(covariant MyWidget oldWidget) {
  super.didUpdateWidget(oldWidget);

  if (oldWidget.VARIABLE_NAME != widget.VARIABLE_NAME) {
    // TODO: implement didUpdateWidget
  }
}
```

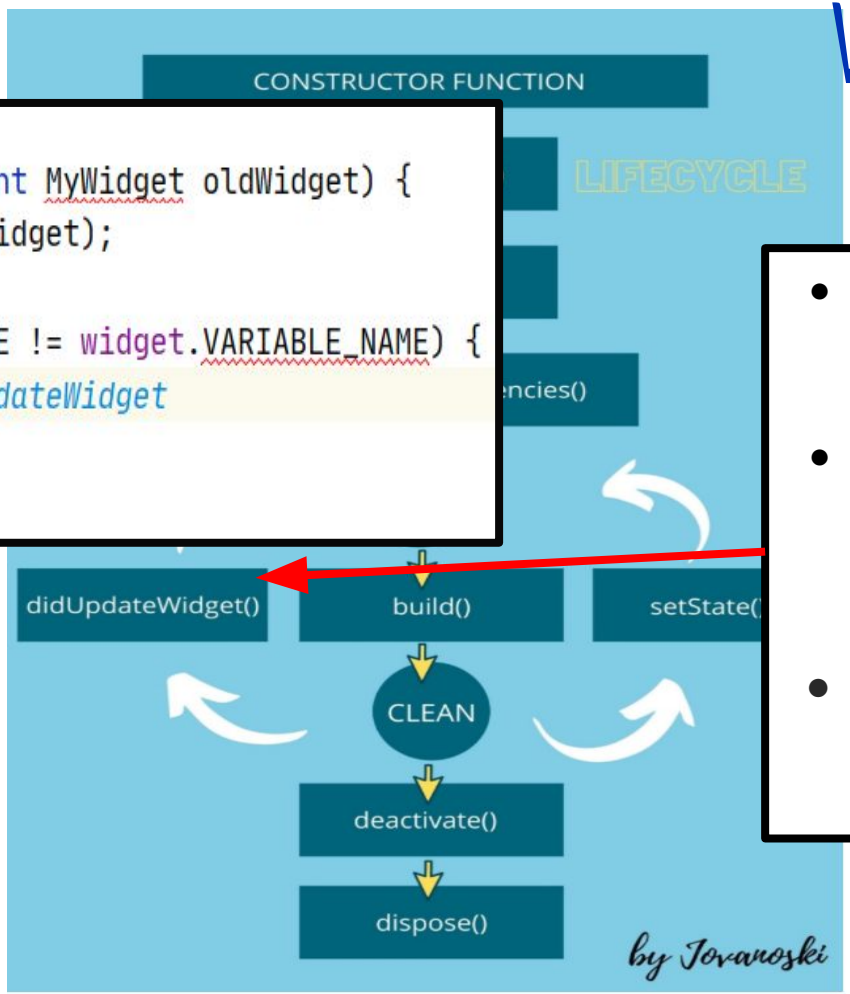


Image by Jelena Jovanoski.

- This method is invoked when value of constructor variables are change
- If the parent widget changes its configuration and has to rebuild this child widget.
- Use this method if you need to compare the new widget to the old one.

Source: https://betterprogramming.pub/stateful-widget-lifecycle-a01c44dc89b0

Widget Lifecycle events

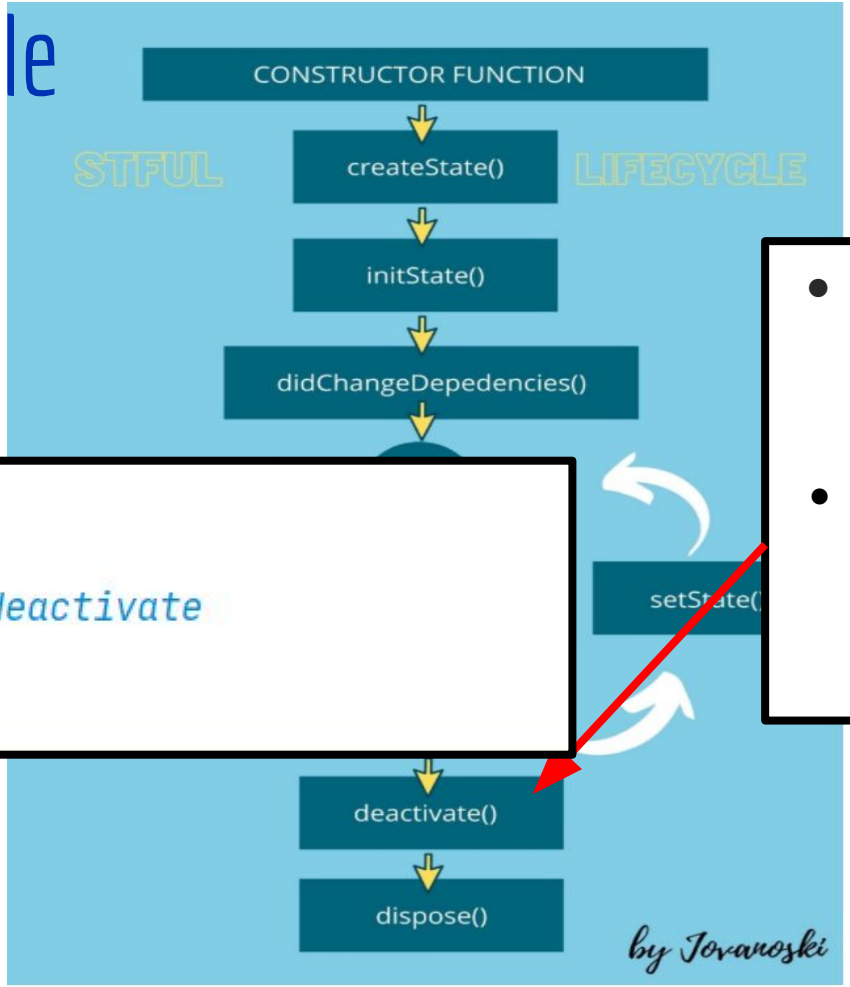


Image by Jelena Jovanoski.

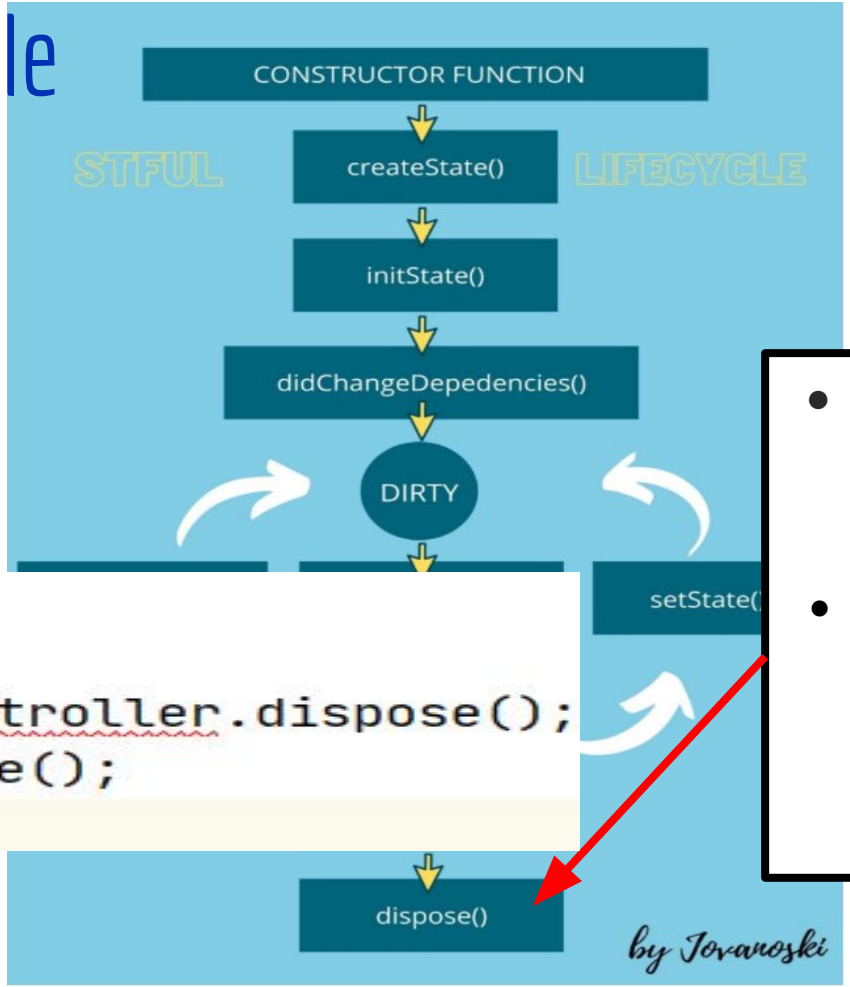
Example

```
@override
void deactivate() {
  // TODO: implement deactivate
  super.deactivate();
}
```

- This method will be called some state is removed from the tree, **temporarily or permanently.**
- In some cases, the framework will reinsert the State object into another part of the tree

Source: https://betterprogramming.pub/stateful-widget-lifecycle-a01c44dc89b0

Widget Lifecycle events



Example

```
@override
dispose() {
  animationController.dispose();
  super.dispose();
}
```

- This method will be called some state is removed from the tree, **permanently.**
- We use this method when we remove permanently like should release resource created by an object like stop animation etc.

Source: https://betterprogramming.pub/stateful-widget-lifecycle-a01c44dc89b0
Source2: https://medium.flutterdevs.com/explore-widget-lifecycle-in-flutter-e36031c697d0

Displaying random word on App Load

- Coming back to our app feature!
- Functionality is to load a random word from one API and call another API to get the meaning.
- We will be using https://random-word-api.herokuapp.com/word to get the random word and usual API to get the meaning.
- **This will happen on app load event**.

- DictionaryScreen's **initState()** method will be suitable for this.

Displaying random word on App Load

-

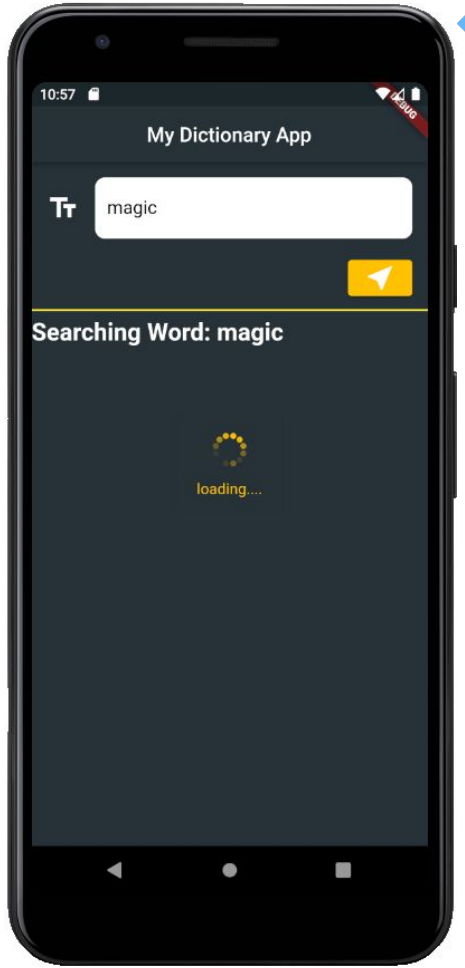
Git Branch: dictionary_app_init_state

Git Diff: #32

Loader

Loader

- We will be using **flutter_easyloading** library
  https://pub.dev/packages/flutter_easyloading



Git Branch: dictionary_app_loader

Git Diff: #33

Splash Screen

Splash Screen

- We will be using **flutter_native_splash** library
  https://pub.dev/packages/flutter_native_splash

pubspec.yaml

Flutter commands | Pub get | Pub upgrade | Pub outdated

```
38    just_audio: ^0.9.18
39    flutter_easyloading: ^3.0.3
40
41  dev_dependencies:
42    flutter_test:
43      sdk: flutter
44    flutter_native_splash: ^1.3.1
45
46  flutter_native_splash:
47    color: "#263238"
48    image: assets/images/book.png
49    android: true
50    ios: true
```

Splash Screen

Steps:

1. Save image into **assets/images** folder
2. Config the splash screen in **pubspec.yaml** file as shown in previous slide.
3. Goto terminal, goto project's root folder
4. Execute command: **flutter clean**
5. Execute command: **flutter pub get**
6. Execute command: **flutter pub run flutter_native_splash:create**
7. Rebuild the package

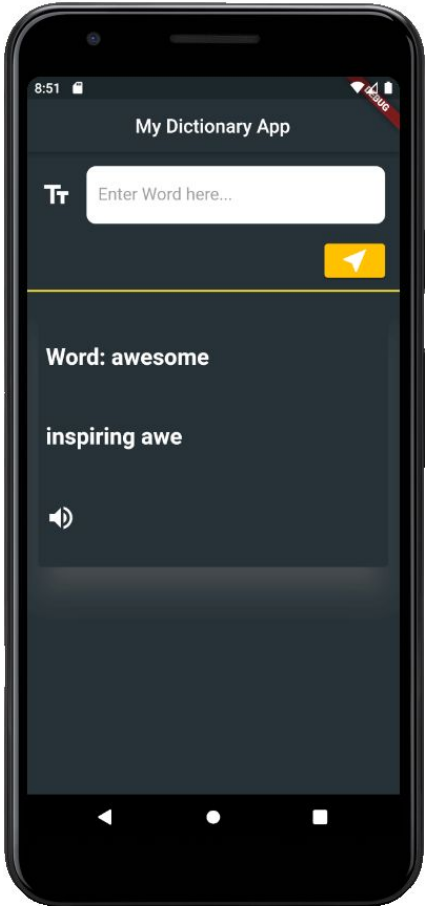
Git Branch: dictionary_app_splash_screen

Git Diff: #34

Text Editing Controller

Text Editing Controller

- One last thing to do for better UX is to clear out the search text field on button press.
- We will need a controller that will help as clear the text field.



Git Branch: dictionary_app_text_controller

Git Diff: #35

Assignment

Weather App

- Create a beautiful weather app
  - On load it should check device current location and get the weather for that location
    - TIP: Use **geolocator** package
    - TIP: For location user permission is must so **read Usage section** of above package carefully.
    - TIP: Must handle Exceptions if user do not give permission or location is off.
  - User can type city name to get weather.
    - TIP: you can use https://openweathermap.org/api or any other of your choice
    - Must check Exception if city weather is not found
  - Weather must be in Celsius
    - TIP: add **&units=metric** in the API URL to get weather in Celsius
  - OPTIONAL:
    - Give two buttons,
      - one for refresh current location and its weather
      - Second for converting weather in Fahrenheit, READ: **Units of measurement** in API Documentation.

There is no code in my repository, do it on your own.

Notice Board App

Notice Board App

- We will be creating a notice board app, and this app will have following features.
- We will be using Firebase Auth for user authentication and Firebase Store to store data.
- Screens:
  - Register User Screen
  - Login User
  - A list view of all the notices → instance update UI on new message
  - A Textfield to add a new notice
  - Push notification on new message received
  - Drawer menu
  - Filter the notices by sender → a filter modal with list widget
  - About Screen with Animation (Typewriter effects)
  - Logout user

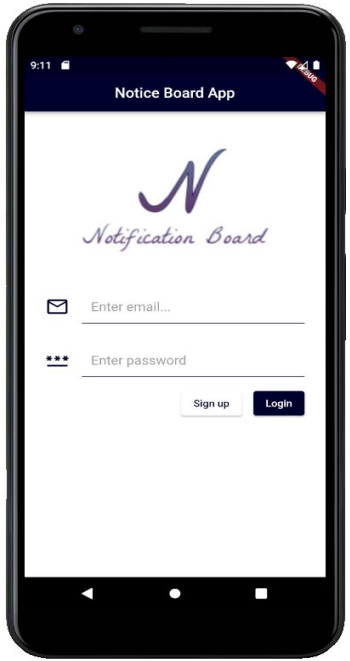
Notice Board App - Basic Skeleton

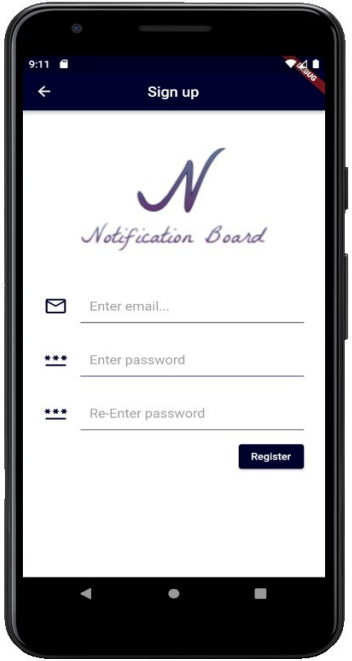
Git Branch:

notification_app_basic_skeleton

Git Diff:

#36





What is Firebase?

Firebase

- Firebase is a Backend-as-a-Service (Baas).
- It provides developers with a variety of tools and services to help them develop quality apps.
- It is built on Google’s infrastructure.
- Some of the firebase services are
  - Email & password, Google, Facebook, and Github authentication
  - NoSql database
  - Realtime data
  - Push notification
  - Web hosting
  - Cloud functions
- Most of the services can be used for free (ofcourse, quota applies!)

Setting up Firebase project

Setting up Firebase project

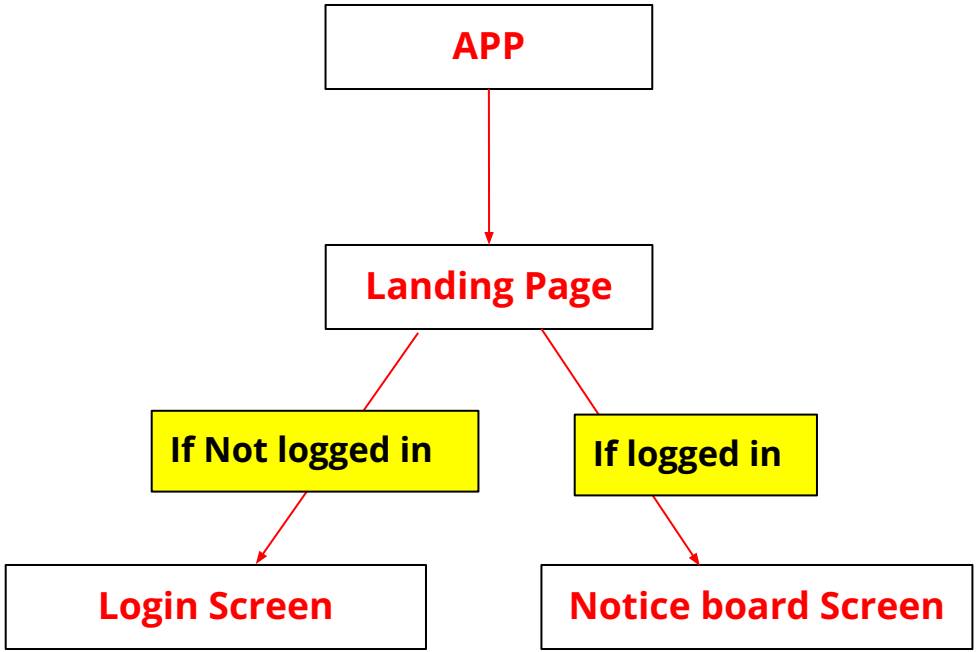
- First of all **create a project** at https://console.firebase.google.com/
- Using firebase console **Create an App** of Android, iOS, Web as required
- Setup **FlutterFire** in the app code, https://firebase.flutter.dev/docs/overview
  - Add dependencies in **pubspec.yaml**
    - firebase_core (Always required)
    - firebase_auth
    - Cloud_firestore
- Find and replace all the app name with the app name added at firebase e.g. **com.flutter.notification.app**
- In your app/build.gradle**, increase** the **minSdkVersion** to **23**
- Stop everything and rebuild the apk from android studio

Auth flow in Notice Board App

Auth flow

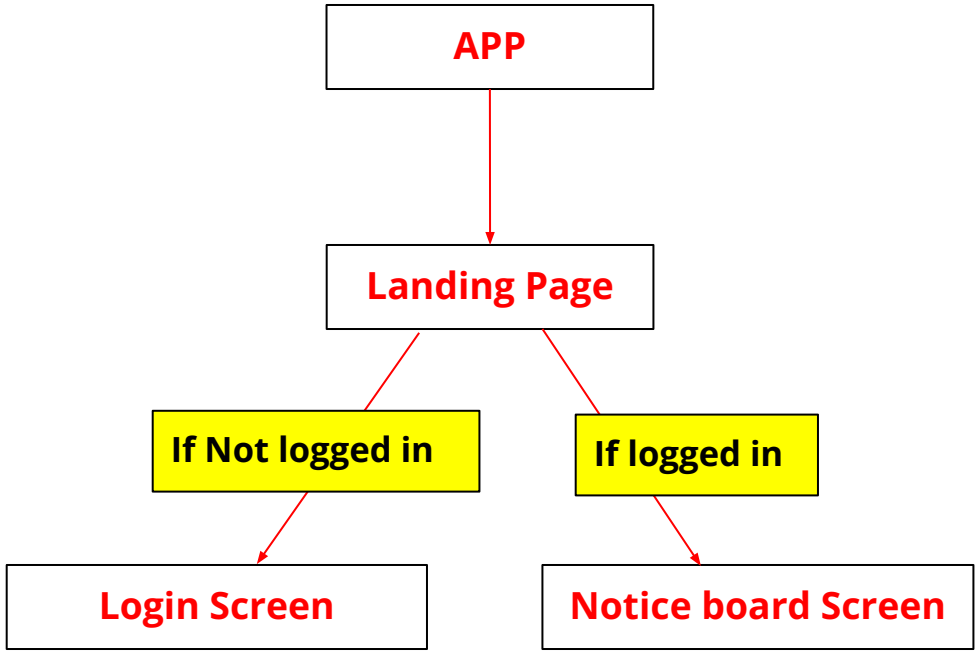
What screen to show first?

- We need to structure a code in a way that when app starts it lands to a place where we it can decide what screen to show.
- If user is logged in then Notice Board Screen.
- Otherwise login screen.
- For this we will create a **Landing Page Widget**



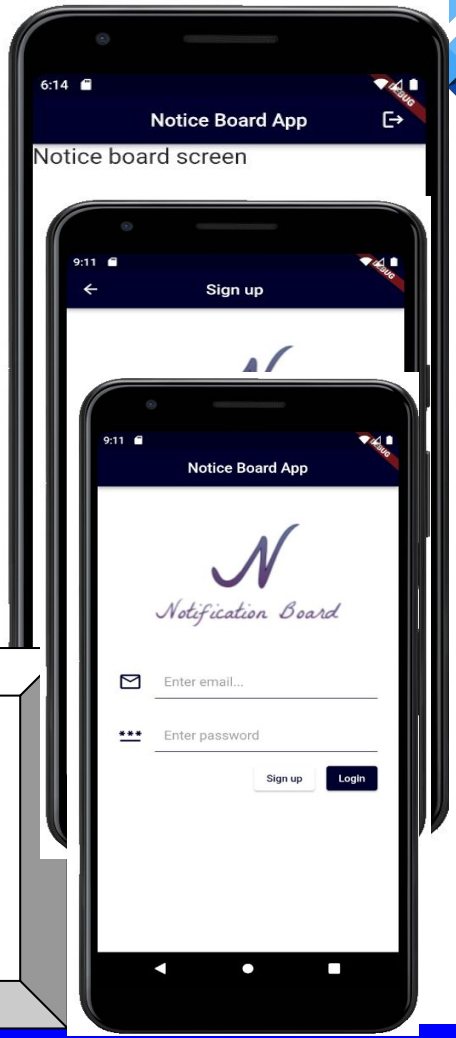
Things we will create in this step

- Auth class
  - signInWithEmail
  - SignOut
  - authStateChanges
  - currentUser Property
- Landing Widget with stream builder.



Notice Board App - Basic Skeleton

- Created AuthService class (singleton)
- Integrated Signup function
- Firebase login integration
- Firebase signout integration



Git Branch:

notificaiton_app_firebase_integration

Git Diff: #37

Cloud Firestore + Flutter

Cloud Firestore + Flutter

- We will be storing notices data into firestore’s collection
- Firestore is a NoSQL document database.
- It is scalable and high performant.
- We will be using **cloud_firestore sdk** for flutter with our notice board app

NoSql Vs Sql

SQL vs NOSQL

SQL Table

id	fname	lname	gender
1	Aamir	Pinger	Male
2	Irfan	Ali	Male

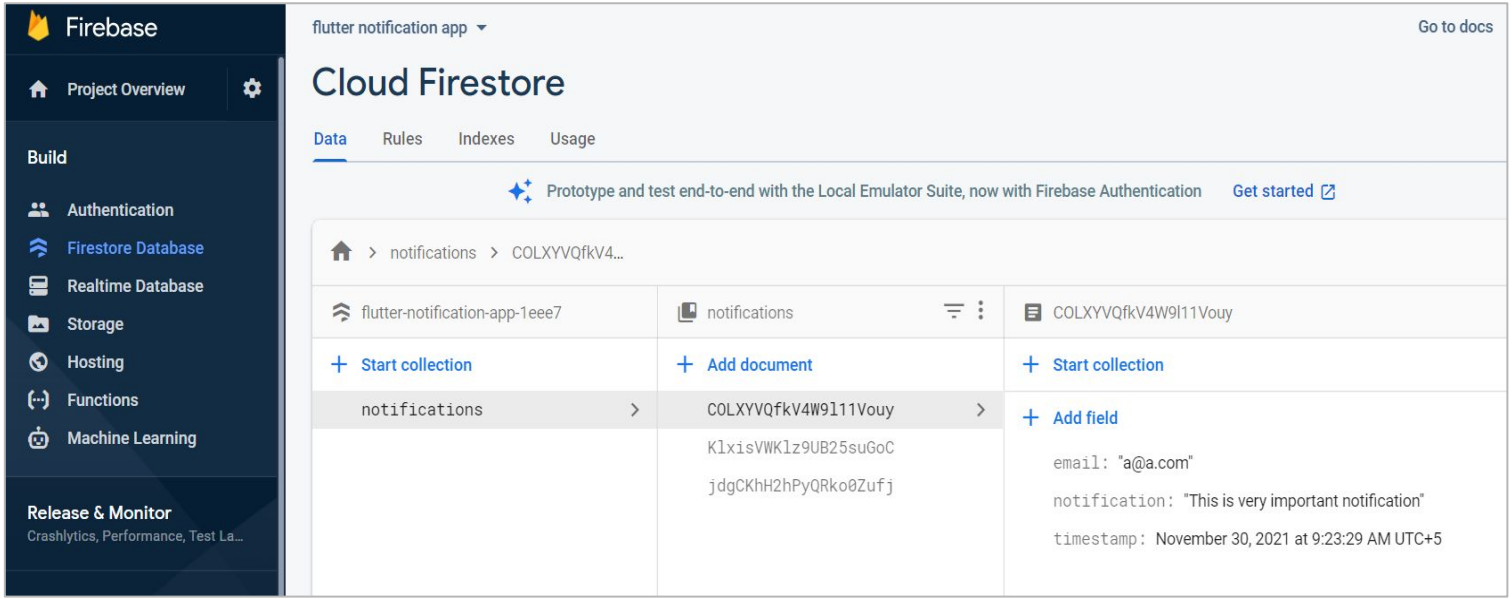
SQL	NoSQL
Table	Collection
Row	Document
Column	Field
Joins	Embedded documents, Linking

NoSQL Collection

```
{
    {
      "id": "1",
      "fname": "Aamir",
      "lname": "Pinger",
      "gender": "Male"
    },
    {
      "id": "2",
      "fname": "Irfan",
      "lname": "Ali",
      "gender": "Male"
    }
}
```

Add notifications

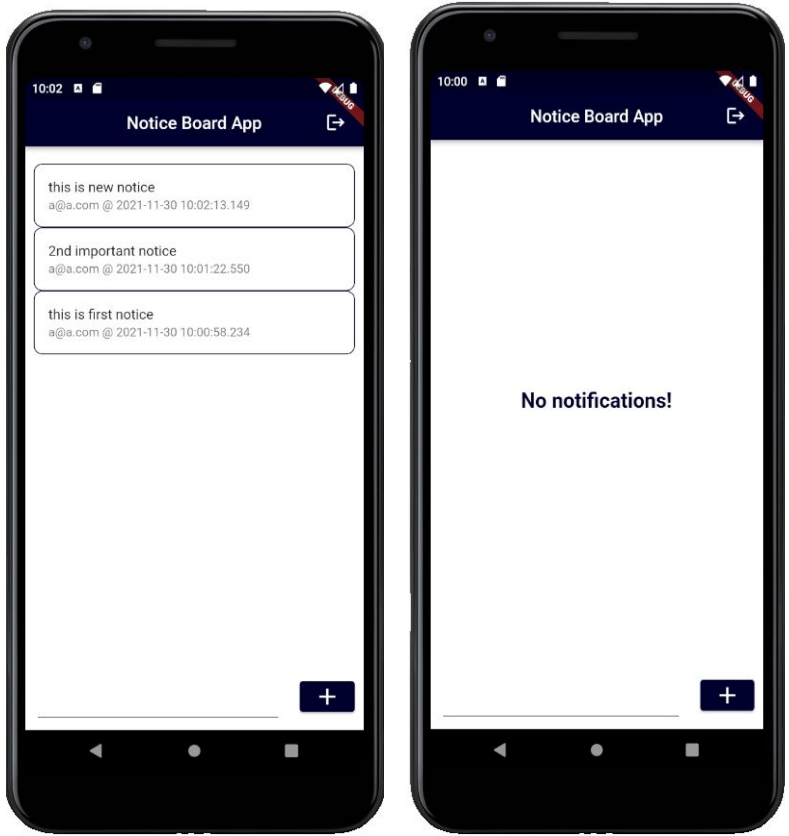
Add notifications



Git Branch: notice_board_app_add_notification

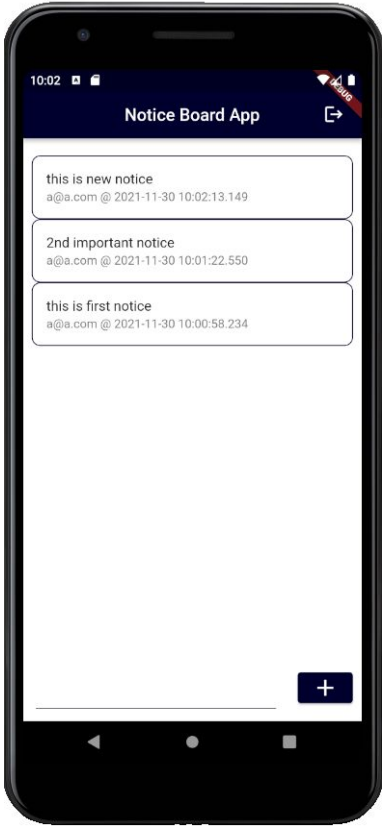
Git Diff: #38

Listing notifications



Listing notifications

- We will be using **ListView** and **ListTile** to display notices at UI
- We will also use **streamBuilder** to listen for any changes into the database, e.g. addition of new records
- **Stream Builder** is a Widget that builds itself based on the latest snapshot of interaction with a Stream.
- **Snapshot** is the xerox of your data, in our case it is the data within the notification collection



Git Branch: notice_board_app_listing_notices

Git Diff: #39

Assignment

Notice App - Assignment

- Add a delete option of notice.
- Add a tailing delete icon in notice bubble for this functionality.
- Icon option should only appear to those notices where creator and logged in user are same.
- A Alert should appear with the ok button and the text “Notice deleted successfully!”
- Extra points if you add edit functionality as well.
  TIP: can you Alert Widget for edit input

Drawer Menu

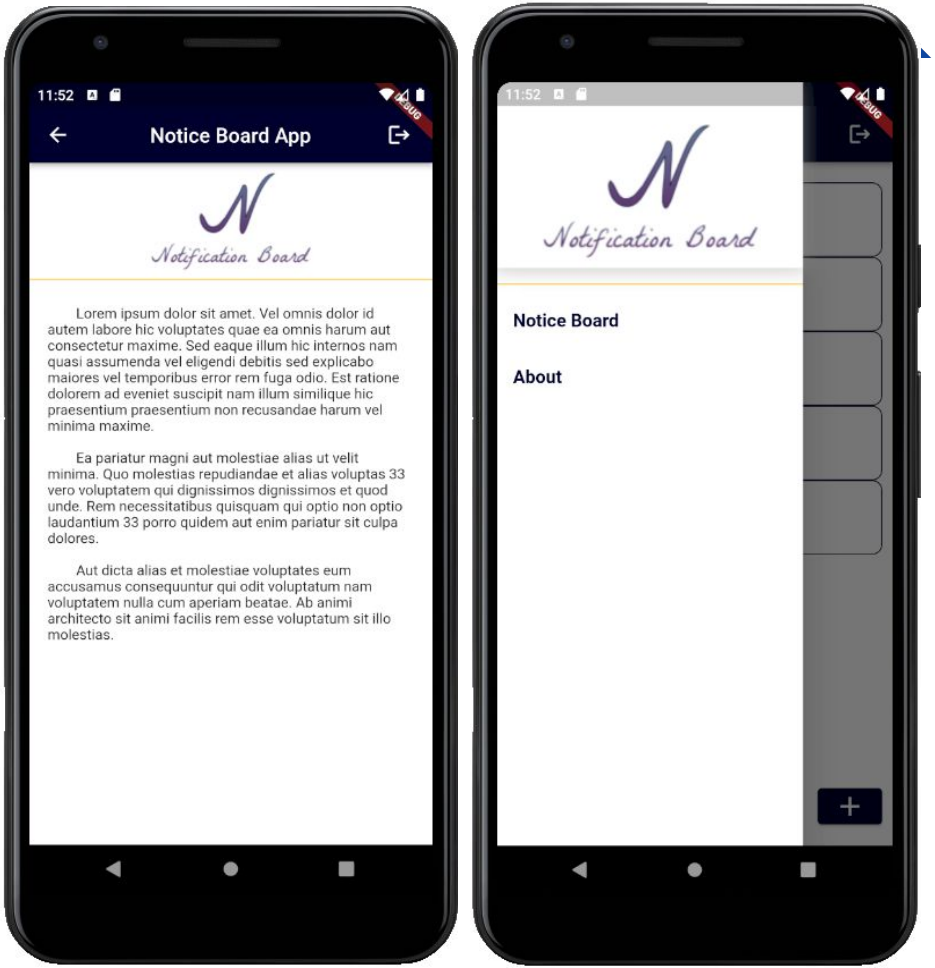
Drawer Menu

- We will be adding a About Screen and a Drawer Menu.

Git Branch:

notice_app_drawer_menu

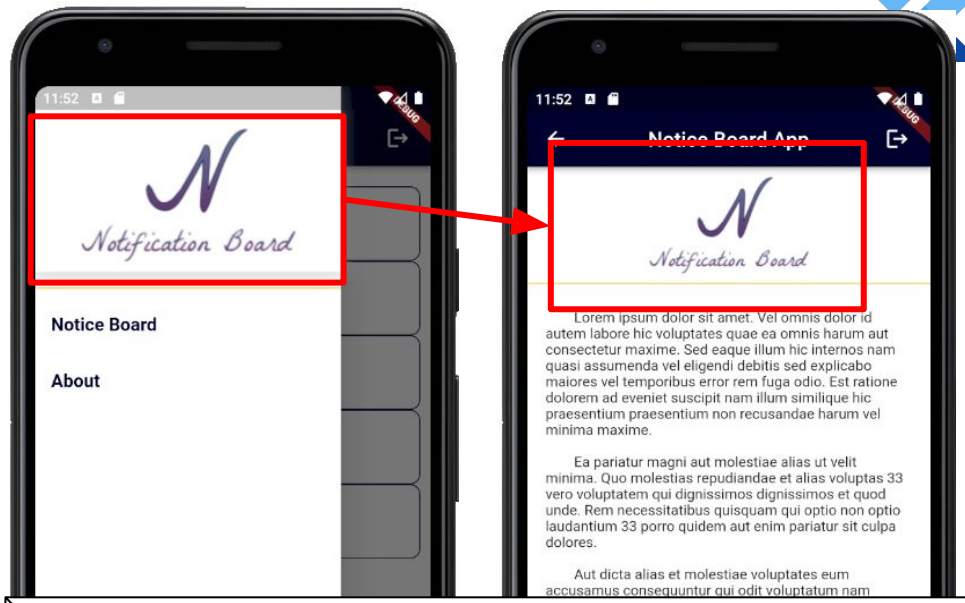
Git Diff: #40



Hero Animation

Hero Animation

- Hero Animation help as transite and animatedly transform widgets.
- It can be done to any widget by simply wrapping it with Hero Widget.
- By providing a **same tag** property to Hero Widget of starting widget and ending widget, flutter handles all itself.
- Let's try Hero animation on logo at drawer menu and about screen.



Git Branch:

notice_board_app_hero_animation

Git Diff: #41

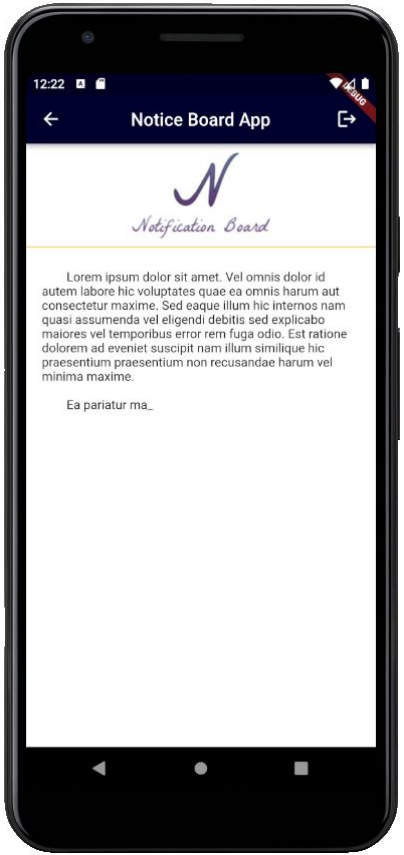
Text Animation Library

Text Animation Library

- Let’s add some typewriter text effect on About Screen.
- We will be using **animated_text_kit** package.

Git Branch: notice_board_app_text_animation

Git Diff: #42



Other Animation Libraries

Good read:

Flutter cookbook:

https://docs.flutter.dev/cookbook#animation

Blog:

https://www.topcoder.com/thrive/articles/top-five-animation-packages-to-use-in-your-next-flutter-app

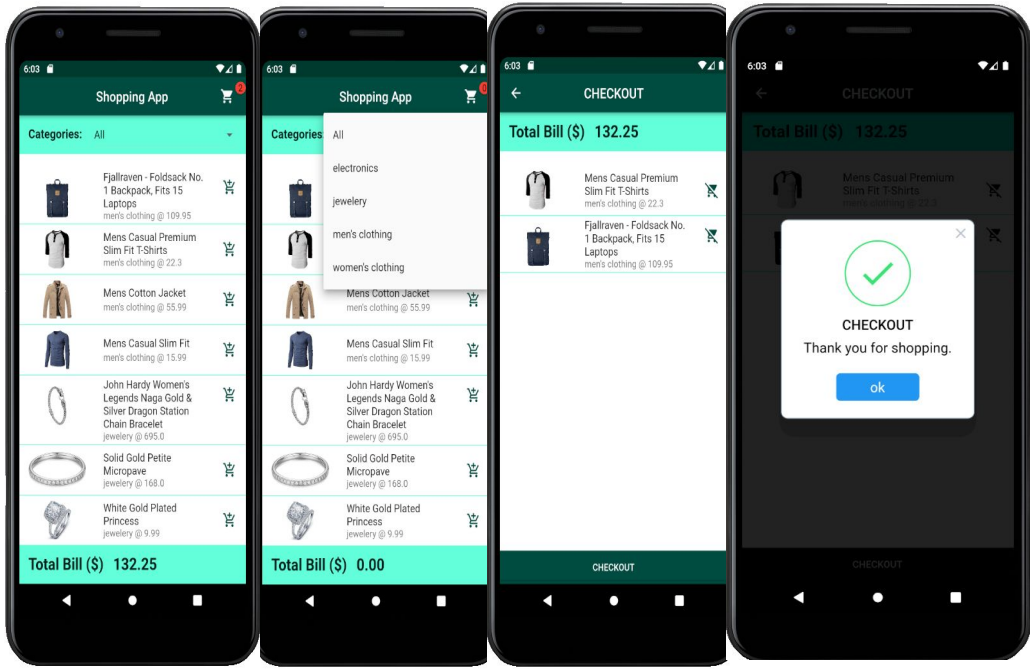
Other Animation Libraries

- Carousel_slider
- Photo_view
- Flutter_slidable
- Percent_indicator
- Lottie
- Pin_code_fields
- Flutter_typeahead
- Table_calendar
- Charts_flutter
- Page_transition
- Expandable
- Flutter_swiper
- Syncfusion_flutter_charts
- Persistent_bottom_nav_bar

Assignment - Shopping Cart App

Shopping Cart App

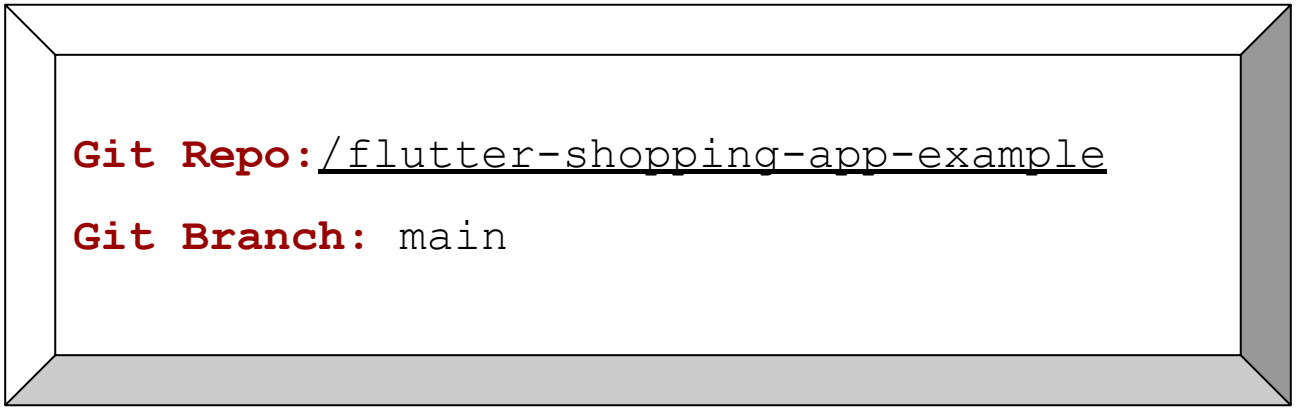
- Create a shopping cart app with all the knowledge we have learn.
- **More info at next slide.**



Shopping Cart App

- This app will have 2 screens
  - Main screen
  - Checkout Screen
- We will be using an API to get products and its categories
- On initial load all product will be displayed, on category selection, product list will update.
- Main screen cart icon will show selected item count in a badge, **package: badges**
- On checkout, cart will get empty and user will redirect back to main screen.

Solution



State management

setState

Riverpod

BLoC / Rx

Redux

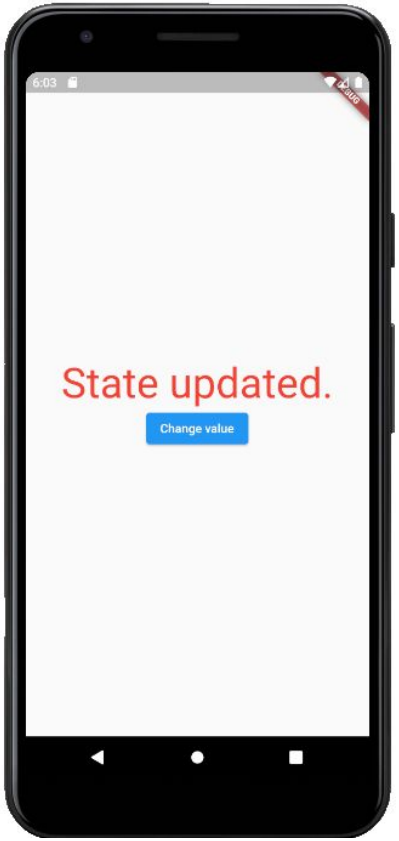
GetIt

Provider

MobX

Basic Lifting state up Example

- Let's create a small example having deep widget tree
- We will lift the state up and pass the state to the child widgets.
- We will also pass updateState method to the last widget to modify state managed by top level widget

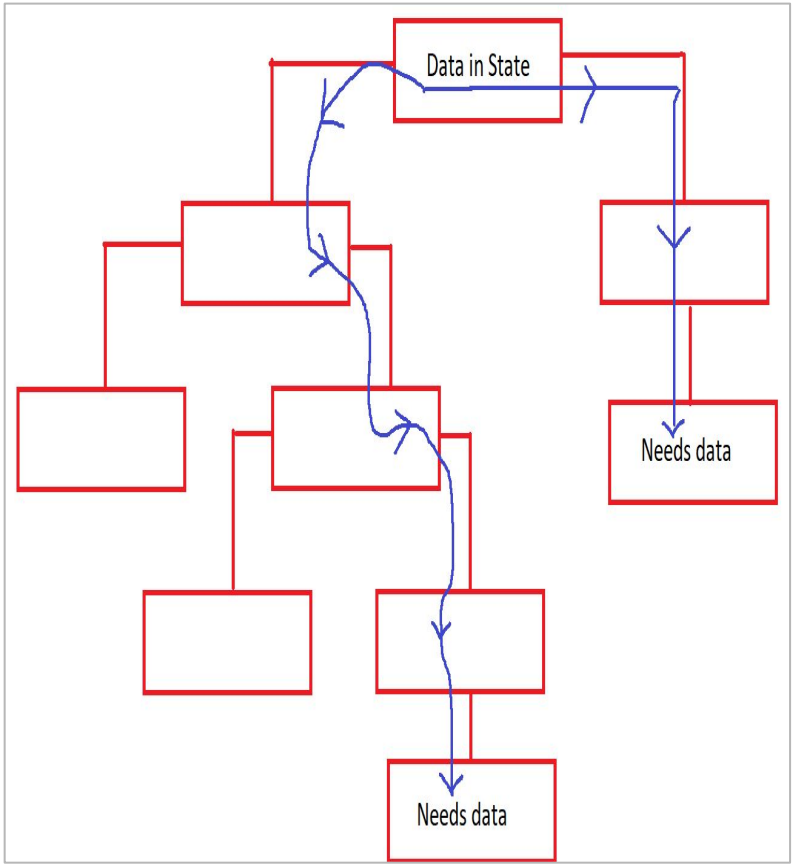


Git Branch: basic_lift_state_up_example

Git Diff: #43

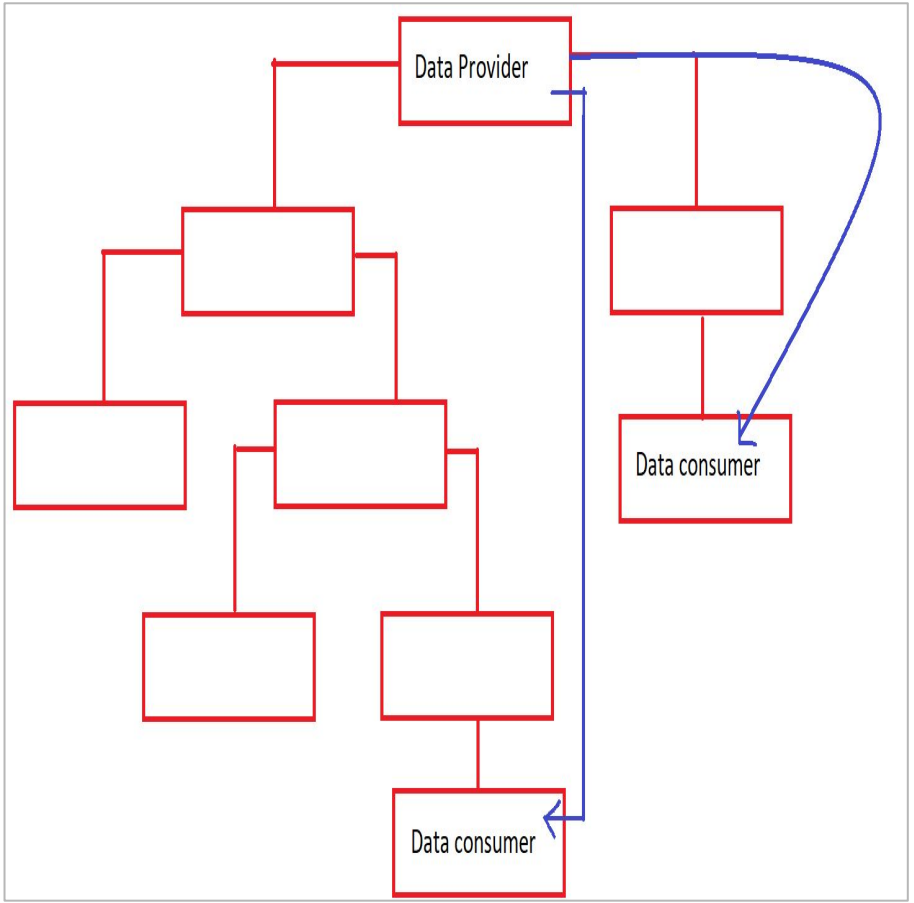
State management

- In the previous code for shopping cart example, we managed the state by **Lifting state up.**
- **Lifting state up** means that you keep the state at most appropriate top level widget and pass to its child and grandchild where it is need.
- We also witnessed that we had to use setState again at cartScreen widget on removal of item so that the Bill Amount get updated.
- This approach is fine for very small level app but not for big scale apps.

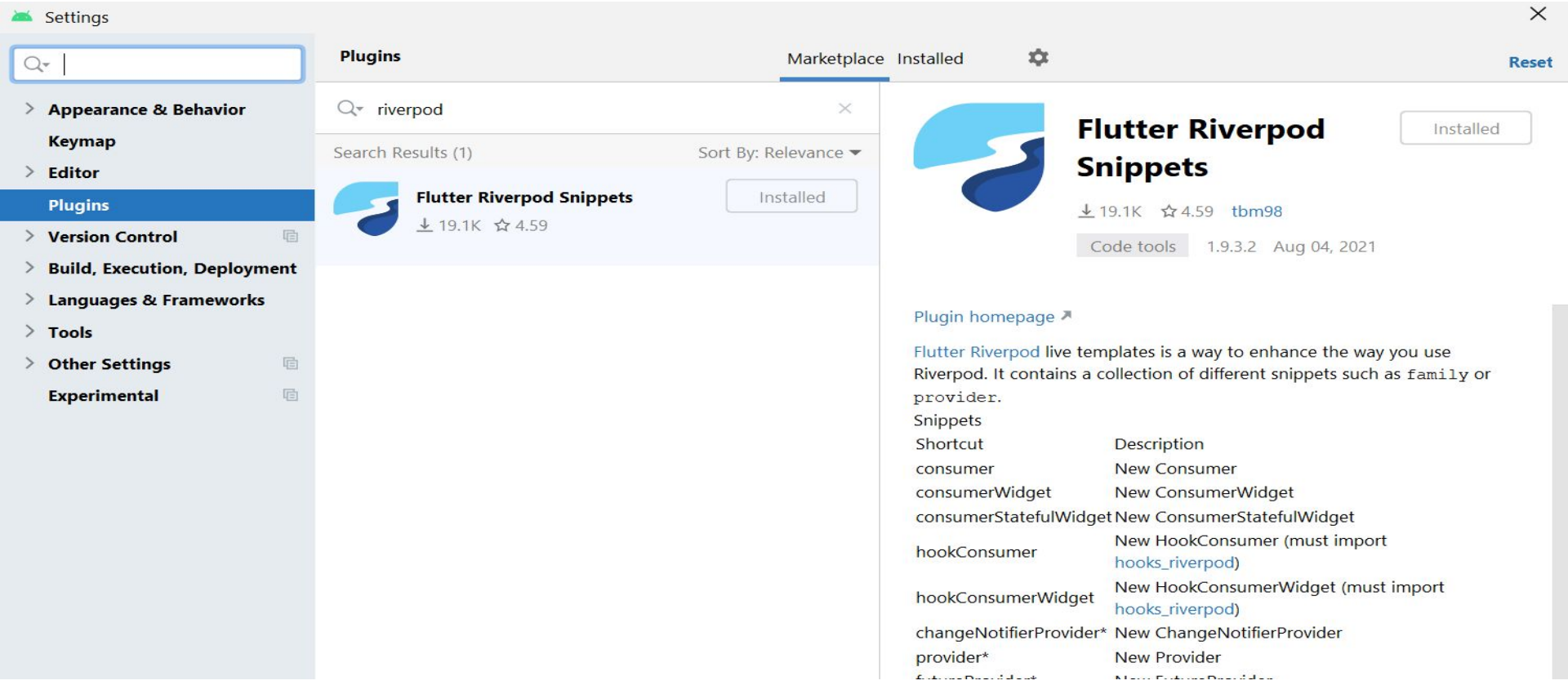


State management

- To solve this issue we will be using **Riverpod** a popular Flutter state management library.
- Riverpod is a state management package released by **Remi Rousselet** (the creator of Provider).
- Rousselet got the word Riverpod by rearranging the letters of the word “Provider.”
- Riverpod is the upgraded form of Provider package with lots of enhanced capabilities.



Flutter Riverpod Snippet Plugin



Providers

- Providers are the most important part of a Riverpod application.
- A provider is an object that encapsulates a piece of state and allows listening to that state.
- Allows easily accessing that state in multiple locations.
- Providers are a complete replacement for patterns like Singletons, Service Locators, Dependency Injection or InheritedWidgets.

Providers

- Simplifies combining this state with others.
- Enables performance optimizations. Providers ensure that only what is impacted by a state change is recomputed.
- Any provider can be overridden to behave differently during app tests, which allows easily testing a very specific behavior.
- Easily integrate with advanced features, such as logging or pull-to-refresh.

Types of Provider

- **Provider**
  - This is pretty much the most basic kind of Provider. It can hold any type of data but it isn't very flexible.
  - You can't change it from the outside, like somewhere in your Widget's build method.

To declare the state

```
final globalValueProvider = Provider<String>((ref) => "Hello from Provider");
```

To access the state

```
ref.watch(globalValueProvider);
```

Source: https://codewithandrea.com/articles/flutter-state-management-riverpod/

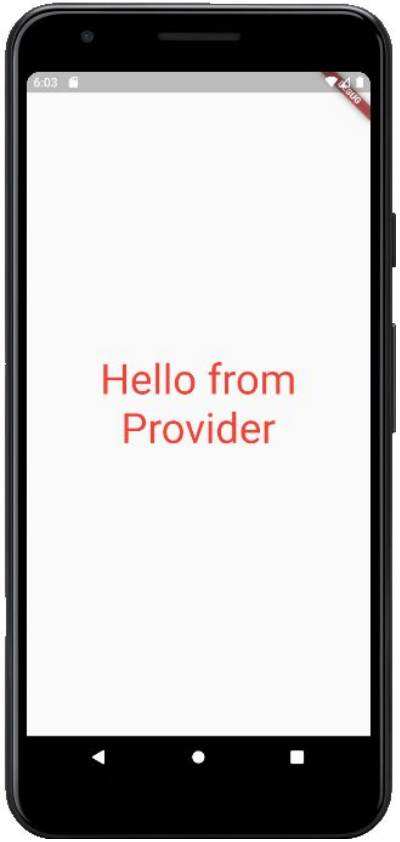
Accessing data using Riverpod

- Package to add: **flutter_riverpod**

Git Branch:

riverpod_accessing_global_provider_value

Git Diff: #44



Types of Provider

- **StateProvider**
  - It's not much more flexible also, but you can actually change its value from the outside.

To declare the state

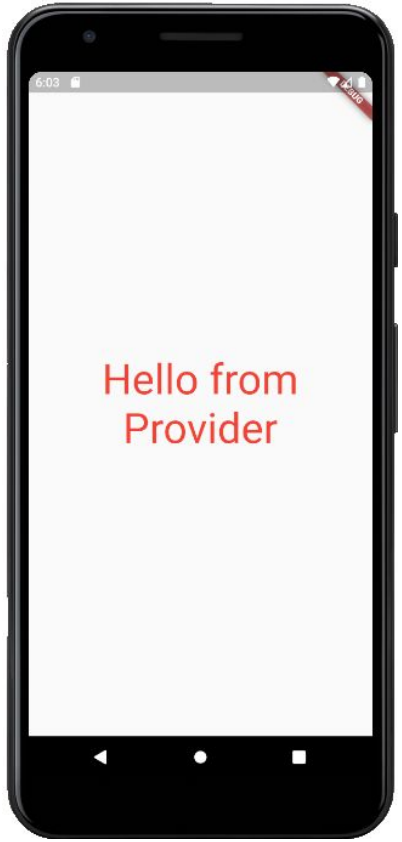
```
final globalValueProvider = StateProvider<String>(
                (ref) => "Hello from Provider"
);
```

To access/modify the state

```
ref.read(globalValueProvider.notifier).state = "Hello from Aamir Pinger."
```

Source: https://codewithandrea.com/articles/flutter-state-management-riverpod/

Modifying global data using Riverpod



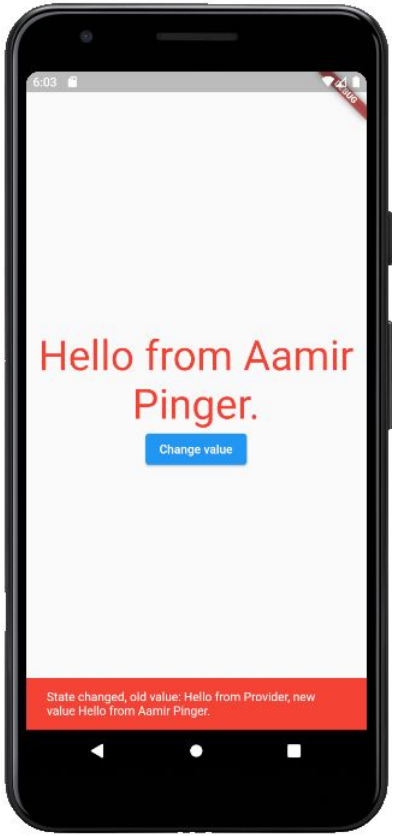
Git Branch: riverpod_updating_state

Git Diff: #45

Listening to state change using Riverpod

Git Branch: riverpod_listening_to_state_change

Git Diff: #45



Types of Provider

- **StateNotifierProvider**
  - Provider and StateProvider are sufficient for simple use cases like the counter example.
  - But in more complex apps we often need to store some state along with some business logic outside our widget classes.
  - The general idea is that you first create a class that holds some kind of state.
  - Then you create a StateNotifier that holds that state and makes updates/manipulations (update notifications are sent when you manipulate state)

Source: https://codewithandrea.com/articles/flutter-state-management-riverpod/

State model

```dart
// To create a StateNotifier you first need to create class/model to hold the state

class MyState {
 MyState({required this.value});

 final String value;

 MyState copyWith(String? newValue) {
   return MyState(value: newValue ?? value);
 }
}
```

StateNotifier

```dart
class MyValue extends StateNotifier<MyState> {
 MyValue() : super(MyState(value: 'This is initial state.'));

 // assign state a new value, in the following case new MyState instance.
 // we can also directly return MyState instance instead of state.copyWith
 void update(String newValue) => state = state.copyWith(newValue);

 // we can many methods to help state manipulation
}
```

StateNotifierProvider

```
final valueProvider =
    StateNotifierProvider<MyValue, MyState>((ref) => MyValue());
```

Modify the state

To modify

```
onPressed: () {
 ref
     .read(valueProvider.notifier)
     .update("Hello from State Notifier Provider.");
},
```

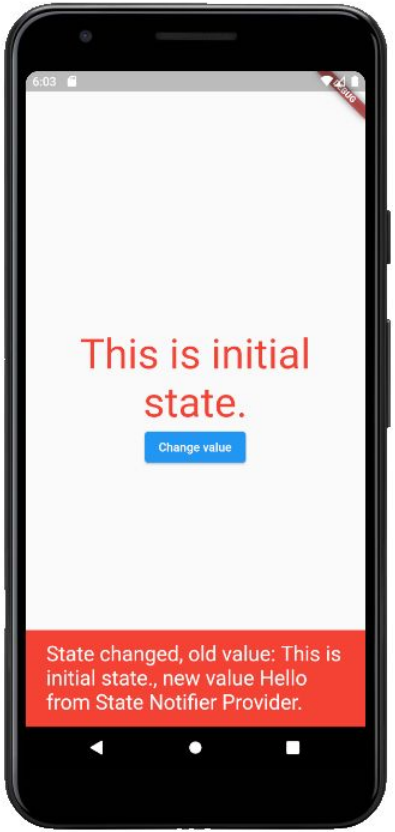
To listen

```
ref.listen<MyState>(valueProvider, (previous, current) {})
```

State management - Riverpod StateNotifierProvider

Git Branch: riverpod_state_notifier_provider

Git Diff: #46

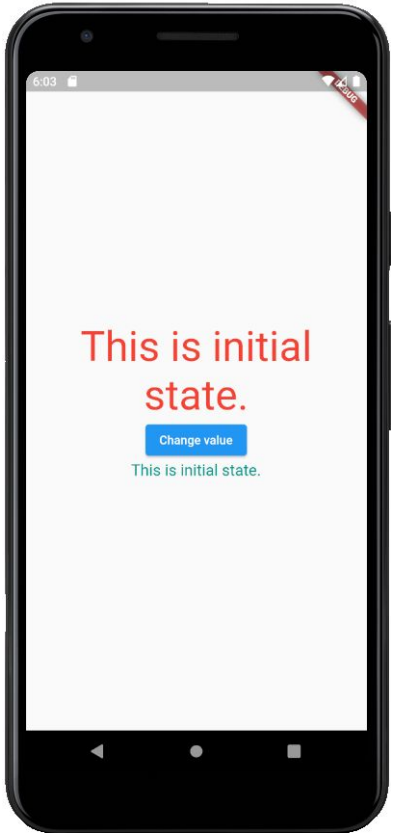


State management - Riverpod with Stateful widget

Git Branch:

riverpod_accessing_state_stateful_widget

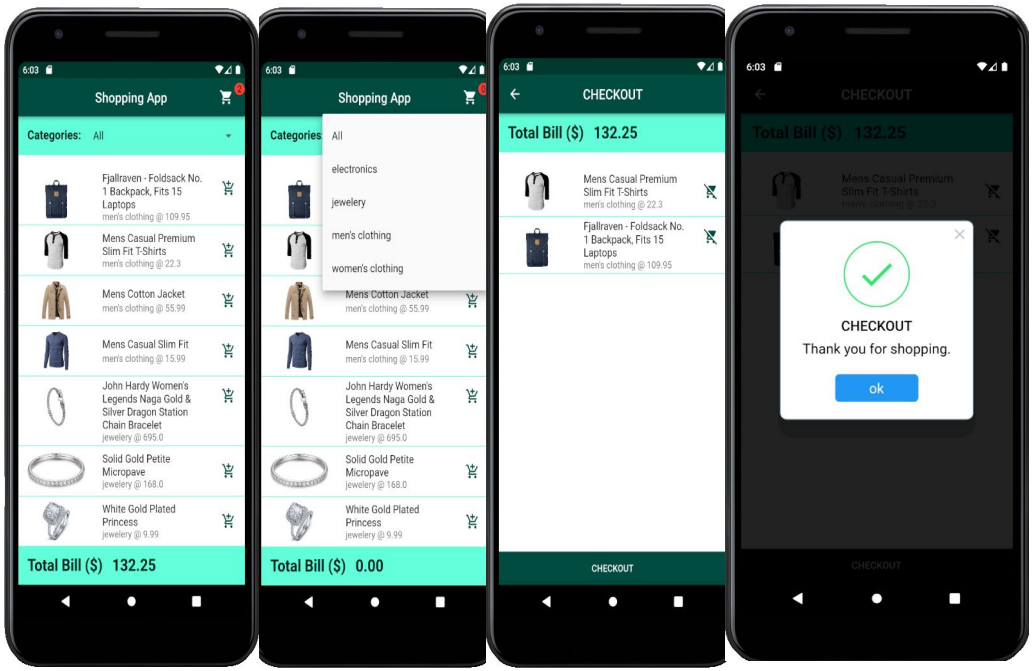
Git Diff: #48



Assignment - Transform
Shopping Cart App
With Riverpod

Shopping Cart App

- While creating shopping Cart App, we have witnessed the complexities of passing values from one screen to another screen.
- Modify this app and use Riverpod to manage the data.
- Keep categories, product list, and selected products in separate states
- Access everything from state



BONUS - Assignment

TODO App

With Riverpod

Assignment - Todo App

- Similar to Notice Board App, create a To Do app
- Take task details and due date/time as user input.
- User input field MUST BE on separate screen/modal
- Display similar to notice board but if task due date is passed, text colors should turn into red.
- Only tasks added by logged in user should be displayed.
- Task should be sorted as ascending order on due date.

Thank you!

Thank you

https://youtu.be/vLtwQMKPuH0

/AamirPinger

/AamirPingerOfficial

/AamirPinger