I use this widgets `cupertino_icons: ^1.0.2`

```
  flutter_screenutil: ^5.9.0
  flutter_spinkit: ^5.2.0
  firebase_core: ^2.24.2
  firebase_auth: ^4.15.3
  email_validator: ^2.1.17
  provider: ^6.1.1
  quickalert: ^1.0.2
  cloud_firestore: ^4.13.6
  get_it: ^7.6.4
  shared_preferences: ^2.2.2
  flexible_grid_view: ^0.0.2
  dots_indicator: ^3.0.0
  smooth_page_indicator: ^1.1.0
  carousel_slider: ^4.2.1
  firebase_storage: ^11.5.6
  cached_network_image: ^3.3.0
  uuid: ^4.2.2
  animated_bottom_navigation_bar: ^1.3.0
  file_picker: ^6.1.1
```

`flutter_screenutil` We are using for adapting screen and font size with this SDK. provides a set of utility functions to help with screen adaptation and size management in Flutter applications. It includes functions for adapting widget sizes based on different screen sizes and densities.

```
return ScreenUtilInit(
      designSize: const Size(360, 690),
      minTextAdapt: true,
      splitScreenMode: true,
      builder: (_, child) {
        return MaterialApp(
          debugShowCheckedModeBanner: false,
          home: child,
        );
      },
      child: const Splash(),
    );
```

```
flutter_spinkit    This package provides a collection of loading
indicators (spinners) that can be used to indicate that the application
```

is busy or loading data. It includes widgets for various types of spinners, such as rotating circles, fading cubes, and bouncing dots. Indicators in Flutter are visual elements displayed on your screen each time your application wants to indicate the status of the requested command. For example, a spinning circle indicates the loading progress if you are loading something on your application. Flutter loader can be in the form of text stating your progress percentage.

- Progress indicators

The progress indicator is a more detailed animation showing the percentage of the completed task.

- Refresh Indicators

When a refresh action is triggered in Flutter, such as when the user pulls down on a scrollable view to refresh the contents of that view, refresh indicators give the user a visual indication of what has happened. It will typically display a progress indicator as a spinning circular bar or some other animation to indicate that the refresh procedure is occurring.

`firebase_core` This package is a prerequisite for using other Firebase services in your Flutter application. It provides the necessary initialization and configuration for Firebase. It includes widgets and functions for initializing Firebase and accessing the Firebase app instance.

firebase_auth: This package provides the necessary tools for authentication using Firebase. It includes widgets for user registration, login, and management.

email_validator: This package provides a validator for email addresses. It includes a widget that can be used to validate email input fields.

provider: This package is a state management solution that allows you to share data between widgets efficiently. It includes widgets like Provider, Consumer, and Selector that help manage and access shared data.

quickalert: This package provides a simple way to display alert dialogs in your application. It includes a widget that can be used to show alert dialogs with custom content and buttons.

cloud_firestore: This package allows you to interact with the Cloud Firestore database provided by Firebase. It includes widgets for querying and managing data in Firestore.

get_it: This package is a simple service locator that allows you to register and retrieve dependencies in your application. It includes a widget called GetIt that can be used to access registered dependencies.

shared_preferences: This package provides a way to store and retrieve simple data (such as user preferences) on the device using key-value pairs. It includes a widget that can be used to access and manipulate shared preferences.

flexible_grid_view: This package provides a flexible grid view widget that allows you to create a grid layout with customizable properties. It includes a widget called FlexibleGridView that can be used to display items in a grid.

dots_indicator: This package provides a widget that displays a series of dots as an indicator for a paginated widget, such as a carousel or a page view.

smooth_page_indicator: This package provides a smooth and customizable page indicator widget for paginated views. It includes a widget called SmoothPageIndicator that can be used to display page indicators.

carousel_slider: This package provides a carousel widget that allows you to create a horizontally scrollable list of items. It includes a widget called CarouselSlider that can be used to display a carousel.

firebase_storage: This package allows you to interact with the Firebase Cloud Storage service for storing and retrieving files. It includes widgets for uploading and downloading files from Firebase Storage.

cached_network_image: This package provides a widget that can efficiently load and cache images from the network. It includes a widget called CachedNetworkImage that can be used to display images from URLs.

uuid: This package provides a way to generate unique identifiers (UUIDs) in your application. It includes a function that can be used to generate UUIDs.

animated_bottom_navigation_bar: This package provides an animated bottom navigation bar widget for navigating between different screens in your application. It includes a widget called AnimatedBottomNavigationBar that can be used to display a bottom navigation bar.

file_picker: This package allows you to pick files from the device's storage. It includes a widget called FilePicker that can be used to open a file picker dialog and select files.

now we explain part of code

```
 DefaultTabController(
      length: 3,
      initialIndex: 1,
      child: Scaffold(
        backgroundColor: ColorsUtil.badgeColor,
        body: Column(
          children: [
            const Padding(padding: EdgeInsets.only(top: 50)),
            Theme(
              data: Theme.of(context).copyWith(
                  colorScheme: Theme.of(context)
                      .colorScheme
                      .copyWith(surfaceVariant: Colors.transparent)),
              child: TabBar(
                controller: _tabController,
                isScrollable: false,
                labelColor: const Color.fromARGB(255, 0, 0, 0),
                unselectedLabelColor: Colors.grey,
                labelStyle:
                    const TextStyle(fontWeight: FontWeight.bold,
fontSize: 33),
                unselectedLabelStyle:
                    const TextStyle(fontWeight: FontWeight.w300,
fontSize: 22),
```

```
                indicatorColor: Colors.transparent,
                tabs: [
                  Container(
                    margin: const EdgeInsets.only(left: 10),
                    child: const Tab(
                      text: "Sign Up",
                    ),
                  ),
                  Container(
                    margin: const EdgeInsets.symmetric(horizontal: 20),
                    child: const Tab(
                      text: "Log in",
                    ),
                  ),
                  Container(
                    margin: const EdgeInsets.only(right: 10),
                    child: const Tab(
                      text: "Forget Password",
                    ),
                  ),
                ],
              ),
            ),
            Expanded(
              child: TabBarView(
                controller: _tabController,
                children: const [
                  Signup(),
                  Login(),
                  ForgetPassword(),
                ],
              ),
            ),
          ],
        ),
      ),
    );
```

DefaultTabController: This widget establishes the default controller for a tabbed view. It takes in the length parameter, which specifies the number of tabs, and the initialIndex parameter, which sets the initial active tab. In your case, there are three tabs, and the initial active tab is set to index 1 (the second tab).

Scaffold: This widget provides a basic structure for the screen layout. It includes properties such as backgroundColor, which sets the background color of the screen.

Column: This widget allows you to arrange its children vertically. In this case, the Column widget holds the entire content of the screen.

Padding: This widget adds padding to its child. In this case, it adds padding to the top of the screen.

Theme: This widget allows you to customize the theme of its subtree. In this case, it is used to make the TabBar background transparent.

TabBar: This widget displays a horizontal row of tabs. It takes in various properties such as controller, which is the tab controller associated with the TabBar, and tabs, which is a list of Tab widgets that represent each tab.

Tab: This widget represents a single tab in the TabBar. It takes in the text parameter, which specifies the text to be displayed on the tab.

TabBarView: This widget displays the content corresponding to each tab. It takes in the controller parameter, which is the tab controller associated with the TabBarView, and children, which is a list of widgets representing the content of each tab.

Expanded: This widget expands its child to fill the available vertical space. In this case, it is used to make the TabBarView occupy the remaining space in the Column.

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

class SelectedColor extends StatefulWidget {
  final void Function(Color) selectedColorCallBack;
  final List<int> colors;

  const SelectedColor({
    required this.colors,
    required this.selectedColorCallBack,
    Key? key,
  }) : super(key: key);
```

```
  @override
  State<SelectedColor> createState() => _SelectedColorState();
}

class _SelectedColorState extends State<SelectedColor> {
  int selectedIndex = -1;
  late List<Color> colors;

  @override
  void initState() {
    colors = widget.colors.map((e) => Color(e)).toList();
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return SizedBox(
      height: 39,
      child: ListView.builder(
        scrollDirection: Axis.horizontal,
        itemCount: colors.length,
        itemBuilder: (context, index) {
          return GestureDetector(
            onTap: () {
              if (selectedIndex == index) {
                selectedIndex = 0;
              } else {
                selectedIndex = index;
              }
              widget.selectedColorCallBack(colors[selectedIndex]);
              setState(() {});
            },
            child: Padding(
              padding: const EdgeInsets.only(right: 22),
              child: Container(
                width: 39,
```

```
              height: 39,
              decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(19),
                color: colors[index],
              ),
              child: Center(
                child: selectedIndex == index
                    ? const Icon(
                        Icons.check,
                        color: Colors.white,
                      )
                    : null,
              ),
            ),
          ),
        ),
      );
    },
  ),
 );
 }
}

class ProductDetailsPage extends StatelessWidget {
 final Map<String, List<int>> variants;

 const ProductDetailsPage({required this.variants});

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text('Product Details'),
   ),
   body: Column(
    children: variants.entries.map((e) {
     return Column(
      children: [
```

```
            Padding(
              padding: const EdgeInsets.symmetric(vertical: 11),
              child: Align(
                alignment: Alignment.topLeft,
                child: Text(
                  'SELECT ${e.key.toUpperCase()}',
                  style: TextStyle(
                    fontWeight: FontWeight.w500,
                    fontSize: 12,
                    color: Color(0xff515c6f).withOpacity(0.502),
                    letterSpacing: 1,
                  ),
                ),
              ),
            ),
          if (e.key == 'color')
            SelectedColor(
              colors: List<int>.from(e.value),
              selectedColorCallBack: (color) {
                // Handle selected color
                print('Selected color: $color');
              },
            ),
        ],
      );
    }).toList(),
  ),
 );
 }
}

void main() {
 runApp(MaterialApp(
   home: ProductDetailsPage(
     variants: {
       'color': [0xFF000000, 0xFFFFFFFF, 0xFF00FF00],
     },
```

```
    ),
  ));
}
```
to choice color