Documentation:

Name: Mahmoud Ahmed Mostafa

Level :#4

Group:2cs

- **Some analysis and notes of my project:**

Light Control Application Documentation

Overview

The Light Control application is a Flutter application that demonstrates how to access and use the ambient light sensor data from a device. The app changes its background color and text color based on the ambient light intensity. When the light intensity is below a certain threshold, the background color becomes black, and the text color turns white, making it easier to read in low-light conditions.

Importing the Required Packages

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

The code imports the following Flutter packages:

- material.dart: Provides the Material widget library.
- async.dart: Contains asynchronous programming utilities, like StreamSubscription.
- light.dart: The light package is used to access the device's light sensor data.

Main Function and Entry Point

```
void main() {
  runApp(const MyApp());
}
```

The main() function serves as the application's entry point. It calls runApp() with an instance of MyApp, which is a StatelessWidget that sets up the basic app structure.

MyApp Widget

dart

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Light Control',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const Task(),
    );
  }
}
```

MyApp is a StatelessWidget that returns a MaterialApp with the following configurations:

- debugShowCheckedModeBanner: false: Hides the debug banner in the top-right corner.
- title: 'Light Control': Sets the app title.
- theme: ThemeData(primarySwatch: Colors.blue): Configures the app theme with a primary color of blue.
- home: const Task(): Sets the home widget to an instance of the Task StatefulWidget.

Task StatefulWidget

```
class Task extends StatefulWidget {
  const Task({Key? key}) : super(key: key);
  @override
  State<Task> createState() => _TaskState();
}
```

Task is a StatefulWidget that represents the main screen of the app. It creates an instance of _TaskState to manage its state.

_TaskState

```
class _TaskState extends State<Task> {
  int luxValue = 0;
  late Light light;
  late StreamSubscription subscription;
}
```

_TaskState manages the state of the Task widget, including the current light sensor value (luxValue), a Light object to access the sensor data, and a StreamSubscription to listen to sensor updates.

onData, stopListening, and startListening Methods

```
void onData(int lux) async {
  debugPrint("Lux value: $luxValue");
  setState(() {
    luxValue = lux;
  });
}
void stopListening() {
  subscription.cancel();
}
void startListening() {
  light = Light();
  try {
    subscription = light.lightSensorStream.listen(onData);
  } on LightException catch (exception) {
    debugPrint(exception.toString());
  }
}
```

These methods control the listening and handling of light sensor data:

- onData(int lux): Callback function that updates luxValue with the received light sensor data.
- stopListening(): Cancels the StreamSubscription and stops listening for light sensor updates.
- startListening(): Creates a Light object, subscribes to the lightSensorStream, and listens for light sensor updates.

initState and initPlatformState Methods

```
@override
void initState() {
  super.initState();
  initPlatformState();
}
Future<void> initPlatformState() async {
  startListening();
}
```

These methods initialize the state of the widget:

- initState(): Overrides the State class's initState() method and calls initPlatformState().
- initPlatformState(): Starts listening for light sensor updates using the startListening() method.

Build Method

```
@override
Widget build(BuildContext context) {
  return SafeArea(
    child: Scaffold(
      backgroundColor: luxValue < 20 ?  Colors.black :  Colors.white,
      body: Center(
        child: Text(
          'Hello World',
          style: TextStyle(color: luxValue < 20 ? Colors.white : Colors.black),
        ),
      ),
    ),
  );
}
```

The build method returns a SafeArea widget with a Scaffold child. The Scaffold has a background color and text color that change based on the current luxValue. If the value is below 20, the background color is black, and the text color is white. Otherwise, the background color is white, and the text color is black.

- ## Sensors used in my project(Ambient sensor):
- **What is ambient sensor in mobile?**

An ambient light sensor is a component in smartphones, notebooks, other mobile devices, automotive displays and LCD TVs. It is a photodetector that is used to sense the amount of ambient light present, and appropriately dim the device's screen to match it.

- **What is the ambient light sensor on my Android phone?**

The Ambient Light Sensor is responsible for a common feature on iPhones and Androiddevices called “Auto Brightness.” This sensor makes it possible for your phone to detect the lighting conditions around you and adjust the screen brightness accordingly.

- **Notes About :**

Accuracy

The accuracy of the light sensor data depends on the hardware of the device running the app. For devices with high-quality ambient light sensors, the app should be able to provide accurate light intensity readings and adjust the background and text colors accordingly.

Power Consumption

The app listens to the light sensor data continuously while active, which may result in increased power consumption. Implementing a mechanism to stop listening to the sensor updates when the app is not in the foreground can help conserve battery and resources.

Efficiency

The app efficiently manages its state using the StatefulWidget and separates concerns into different methods. However, the efficiency of the app depends on the device's hardware and the quality of the light sensor.

Reliability

The app includes error handling for LightException, ensuring it does not crash due to issues with the light sensor data stream. However, to further improve reliability, it is recommended to write unit and widget tests to ensure the application's correctness, especially if additional features or modifications are introduced in the future.

Advantages

1. The app demonstrates how to access and use ambient light sensor data from a device.
2. The background and text colors adjust based on light intensity, improving the reading experience in varying light conditions.
3. Error handling is included to prevent crashes due to issues with the light sensor data stream.

Disadvantages

1. The app's power consumption may be high due to continuous listening to the light sensor data.
2. The user interface is limited, only displaying a "Hello World" text without any controls or additional information.
3. The app does not provide a way to stop listening to the light sensor updates.
4. The threshold for changing background and text colors is hardcoded, limiting personalization.
5. The app lacks code comments and tests, making it more challenging to maintain and ensure reliability.