Title: Changing Widgets in Response to User Input

Student Name: Talent Nyota

Course Code: INFT 3101

Institution Name: Durham College

Date: 08/11/2024

DURHAM COLLEGE
SUCCESS MATTERS

08-11-2024

Table of Contents

08-11-2024

Introduction

Flutter is a free, open-source toolkit for building apps that work on mobile, web, and desktop platforms from a single set of code. It is well-known for its reactive programming approach, which makes it easy to create dynamic user interfaces that respond to user actions or automatic updates.

Changing Widgets in Response to User Input

Definition:
In Flutter, updating widgets based on user interactions means changing a widget's state as users interact with the app. This is a fundamental aspect of Flutter's design, using Stateful Widgets to track and manage these changes.

Importance:

This capability is essential for creating interactive and easy-to-use interfaces. It allows apps to be more responsive and engaging, enhancing user interaction and satisfaction.

Comparisons:

Unlike other UI frameworks that need a full page reload to reflect changes, Flutter's design supports smooth and quick updates. Flutter performs particularly well compared to frameworks like React Native because it compiles directly to native code, improving performance, especially during complex state changes.

Advantages and Limitations:

Advantages:

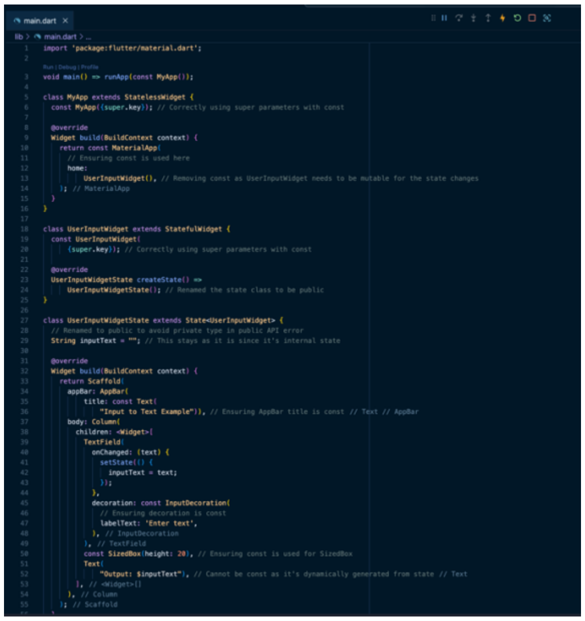
- Allows for highly interactive user interfaces.
- Enhances user experience by providing immediate feedback.
- Reduces the need for full page reloads or refreshes.

Limitations:

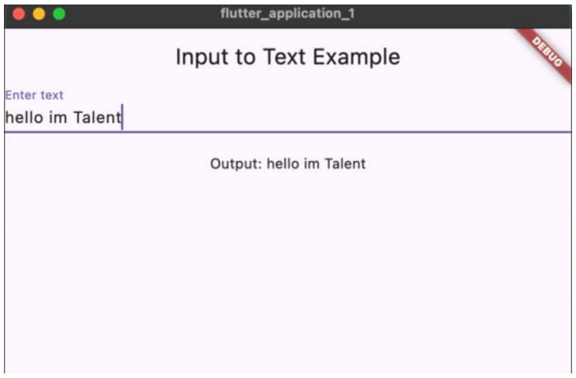
- Managing state in large applications can become complex.
- Overuse of Stateful Widgets can lead to decreased performance if not handled efficiently.

Code Examples

This example demonstrates updating a Text widget based on the user input from a Text Field.



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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key}); // Correctly using super parameters with const

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      // Ensuring const is used here
      home:
          UserInputWidget(), // Removing const as UserInputWidget needs to be mutable for the state changes
    ); // MaterialApp
  }
}

class UserInputWidget extends StatefulWidget {
  const UserInputWidget(
      {super.key}); // Correctly using super parameters with const

  @override
  UserInputWidgetState createState() =>
      UserInputWidgetState(); // Renamed the state class to be public
}

class UserInputWidgetState extends State<UserInputWidget> {
  // Renamed to public to avoid private type in public API error
  String inputText = ""; // This stays as it is since it's internal state

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
          title: const Text(
              "Input to Text Example")), // Ensuring AppBar title is const // Text // AppBar
      body: Column(
        children: <Widget>[
          TextField(
            onChanged: (text) {
              setState(() {
                inputText = text;
              });
            },
            decoration: const InputDecoration(
              // Ensuring decoration is const
              labelText: 'Enter text',
            ), // InputDecoration
          ), // TextField
          const SizedBox(height: 20), // Ensuring const is used for SizedBox
          Text(
              "Output: $inputText"), // Cannot be const as it's dynamically generated from state // Text
        ], // <Widget>[]
      ), // Column
    ); // Scaffold
```



Input to Text Example

Enter text

hello im Talent

Output: hello im Talent

Commentary:

- UserInputWidget is a StatefulWidget that reacts to user inputs.
- TextField takes user input, triggering onChanged.
- setState updates the UI with the new text in the Text widget.

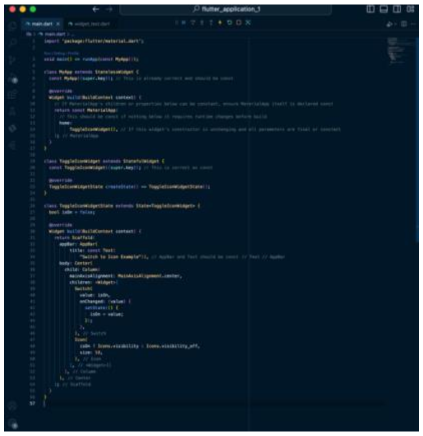
Issues or Considerations:

- Ensure state updates are efficient to avoid UI lag.

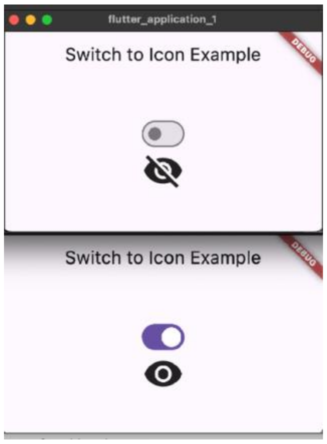
08-11-2024

2. Switch to Icon Example

This example toggles an icon based on a Switch.



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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key}); // This is already correct and should be const

  @override
  Widget build(BuildContext context) {
    // If MaterialApp's children or properties below can be constant, ensure MaterialApp itself is declared const
    return const MaterialApp(
      // This should be const if nothing below it requires runtime changes before build
      home:
          ToggleIconWidget(), // If this widget's constructor is unchanging and all parameters are final or constant
    ); // MaterialApp
  }
}

class ToggleIconWidget extends StatefulWidget {
  const ToggleIconWidget({super.key}); // This is correct as const

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

class ToggleIconWidgetState extends State<ToggleIconWidget> {
  bool isOn = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
          title: const Text(
              "Switch to Icon Example")), // AppBar and Text should be const // Text // AppBar
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Switch(
              value: isOn,
              onChanged: (value) {
                setState(() {
                  isOn = value;
                });
              },
            ), // Switch
            Icon(
              isOn ? Icons.visibility : Icons.visibility_off,
              size: 50,
            ), // Icon
          ], // <Widget>[]
        ), // Column
      ), // Center
    ); // Scaffold
  }
}
```



Commentary:

- ToggleIconWidget changes the icon based on the Boolean state is On.
- The Switch widget toggles the state, which updates the Icon widget accordingly.

Issues or Considerations:

- Managing multiple stateful widgets can require careful state management strategies.

Real-World Scenario

A shopping app where users can toggle their favorite items. Using the "Switch to Icon" example, each product can have a heart icon that users tap to add the item to their favorites. This immediate visual feedback enhances user experience by providing a smooth and responsive interface.
08-11-2024

References

Flutter Official Documentation: Flutter.dev
Flutter for Beginners by Alessandro Biessek
Mastering Flutter” by Simone Alessandria

08-11-2024