Evaluating State Management Solutions for Bluetooth Button Integration in the Social Buddy Bot App

Project 7/8: Integration beacons with Social-Buddy

General Information

Project Code:	**CMI-TI-23-TINPRJ0478**
Project Goal:	
Team Members:	Ahmet Oral (1023107) - Author Khizer Butt (1052313) Nguyen Do (1057048) Terrence Zhong (1028516)
Skills Coach:	Sandra Hekkelman
Technical Coach:	Alexander Slaa
Submission Occasion:	First Attempt

Contents

Summary

ROTTERDAM UNIVERSITY OF APPLIED SCIENCES/CMI

This research explores the best state management solution for implementing a Bluetooth button in the Social Buddy Bot app, which assists elderly users by connecting them with caregivers. The study compares different state management approaches to ensure the button's functionality across the app. The conclusion identifies the provider package as the most effective solution because of its ease of use and minimal impact on performance.

Introduction

Background

This research is conducted to enhance the Social Buddy Bot app, which helps elderly users stay in touch with caregivers. The app includes a Bluetooth button to notify caregivers when medication is taken. The challenge is to ensure this button works across the entire app. This document is authored by Ahmet Oral to find the best state management solution for this feature.

Glossary

Specialist Terms

- **State Management**: The process of managing the state, or data, of an application throughout its lifecycle.
- **State**: The current condition or data within an application at any given moment.
- **State Management Solution**: A framework or library used to manage the state of an application, ensuring data consistency and reliability.
- **Provider Package**: A state management solution in Flutter that simplifies data allocation and disposal, reducing boilerplate code and improving scalability.
- **ChangeNotifier**: A class in Flutter's Provider package that notifies listeners when the state it holds changes.
- **Bloc**: A state management pattern in Flutter that separates the presentation layer from the business logic, often used with streams and reactive programming.
- **Redux**: A predictable state container for JavaScript apps, often used with Flutter through packages like `flutter_redux`, providing unidirectional data flow and immutable state.
- **Performance**: The efficiency and speed of an application, including factors like response time and memory usage.
- **Usability**: The ease of use and accessibility of an application for both developers and end-users.
- **Boilerplate Code**: Repetitive and redundant code that must be written to set up a basic structure or functionality.
- **Complexity**: The level of intricacy or difficulty in understanding and implementing a solution or concept.
- **Widget Tree**: The hierarchical structure of widgets in a Flutter application, defining the layout and behavior of UI elements.
- **Streams**: A sequence of asynchronous events, often used in Flutter for handling data flow and communication between different parts of an application.
- **Immutable State**: State that cannot be changed after it is created, ensuring predictability and preventing unexpected modifications.
- **Global App State**: Data or information that is accessible and consistent throughout an entire application, typically used for managing user authentication, preferences, or settings.

Problem Statement

Description of the Problem

The current challenge is to ensure that the Bluetooth button's functionality is consistent throughout the app. Without proper state management, the button's status is only retained on the page it is connected to, causing usability issues.

Importance

The primary users affected are elderly individuals who rely on timely notifications for their medication intake, and caregivers who need reliable updates. The issue impacts the app's usability, making it critical to find a solution that ensures the button's functionality throughout the app.

Research Objective

The goal is to identify and implement a state management solution that allows the Bluetooth button's status to be maintained across all pages of the app.

Refined Assignment Description

Research Question

Main Question: What is the best state management solution to ensure the Bluetooth button works across the entire Social Buddy Bot app?

Sub-questions:

1. What are the existing state management solutions in Flutter?
2. How does each state management solution handle global state?
3. What are the performance implications of each solution?
4. How user-friendly and maintainable is each solution?

Theoretical Framework

Existing Problems and Solutions

Various state management solutions like provider, bloc, and redux [1] are commonly used in Flutter apps. Literature suggests that while bloc offers robust state management, it can be complex to implement. Provider is praised for its simplicity and efficiency, making it a suitable candidate for this research.

Methodology

Research Method

The research will employ experimental methods, comparing the provider, bloc and redux state management solutions.

Approach

1. **Literature Review**: Study the documentation and use cases of each state management solution.
2. **Implementation**: Implement the Bluetooth button feature using each solution.
3. **Testing**: Evaluate the implementations based on performance, usability, and maintenance.
4. **Comparison**: Compare the solutions against predefined requirements such as ease of implementation and performance.

Requirements

- **Performance**: Minimal impact on app performance.
- **Usability**: Easy to use for developers and maintain.

Implementation Code

The prover package:

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

class BluetoothButtonState extends ChangeNotifier {
  bool _hasTakenMedication = false;

  bool get hasTakenMedication => _hasTakenMedication;

  void updateMedicationStatus(bool status) {
    _hasTakenMedication = status;
    notifyListeners();
  }
}

class BluetoothButton extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ElevatedButton(
      onPressed: () {
        context.read<BluetoothButtonState>().updateMedicationStatus(true);
      },
      child: Text('Medication Taken'),
    );
  }
}
```

The Bloc package:

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

// Event
class MedicationTaken extends BlocEvent {}

// State
class BluetoothButtonState {
  final bool hasTakenMedication;

  BluetoothButtonState(this.hasTakenMedication);
}

// Bloc
class BluetoothButtonBloc extends Bloc<MedicationTaken, BluetoothButtonState>
{
  BluetoothButtonBloc() : super(BluetoothButtonState(false));

  @override
  Stream<BluetoothButtonState> mapEventToState(MedicationTaken event) async* {
    yield BluetoothButtonState(true);
  }
}

class BluetoothButton extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return BlocBuilder<BluetoothButtonBloc, BluetoothButtonState>(
      builder: (context, state) {
        return ElevatedButton(
          onPressed: () {
            context.read<BluetoothButtonBloc>().add(MedicationTaken());
          },
          child: Text('Medication Taken'),
        );
      },
    );
  }
}
```

The Redux package:

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

import 'package:flutter_redux/flutter_redux.dart';
import 'package:redux/redux.dart';

// Actions
enum MedicationAction { takeMedication }

// Reducer
bool medicationReducer(bool hasTakenMedication, dynamic action) {
  if (action == MedicationAction.takeMedication) {
    return true;
  }
  return hasTakenMedication;
}

// Initial State
bool initialState = false;

// Store
final store = Store<bool>(
  medicationReducer,
  initialState: initialState,
);

class BluetoothButton extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return StoreConnector<bool, VoidCallback>(
      converter: (store) {
        return () => store.dispatch(MedicationAction.takeMedication);
      },
      builder: (context, callback) {
        return ElevatedButton(
          onPressed: callback,
          child: Text('Medication Taken'),
        );
      },
    );
  }
}
```

Results

Comparative Analysis

Solution	**Performance**	**Usability**
Provider	High	Easy
Bloc	High	Moderate
Redux	High	Moderate

All three state management solutions, Provider, Bloc, and Redux, demonstrate high performance levels, ensuring minimal impact on app responsiveness. However, Provider distinguishes itself in terms of usability due to its simplicity and intuitive design. Provider simplifies data allocation and disposal, reduces boilerplate code, and integrates seamlessly with Flutter's widget tree. This ease of use makes Provider the preferred choice for developers, especially those unfamiliar with complex state management patterns like streams in Bloc or the strict architecture of Redux. Therefore, Provider remains the most user-friendly option for integrating the Bluetooth button in the Social Buddy Bot app.

References

[1] K. Makadiya, „medium.com," medium, 11 may 2022. [Online]. Available: https://medium.com/dhiwise/flutter-state-management-what-to-choose-provider-bloc-or-redux-214160adbae4.

For the formulation of the research question, ChatGPT provided valuable assistance in refining and structuring the inquiry. Additionally, ChatGPT aided in the generation of code snippets for implementing the Bluetooth button feature using various state management solutions in Flutter.

Change Log

Version	Date	Changes
1	02/06/2024	First version