droidcon academy

Mastering Bloc and Cubit State Management with Architectural Patterns in Flutter

Rahul Agarwal

Flutter Developer & Trainer

in in/rahul-connect

Course Overview

- Importance of State Management in building responsive and efficient apps.
- Challenges with using setState for state management.
- State Management using Bloc.
- Understand the concept of events and states in the context of Bloc.
- BlocBuilder and BlocConsumer for efficiently building UI based on Bloc states.
- Explore the Equatable package to streamline equality checks.
- Explore Cubit simplified state management approach.
- Building an Expense Tracking app.
- Implement fl_chart package to visualize transaction data using Pie charts.
- Discuss Best practices for using Flutter Bloc for optimizing performance.

droidcon academy

Getting Started

Rahul Agarwal

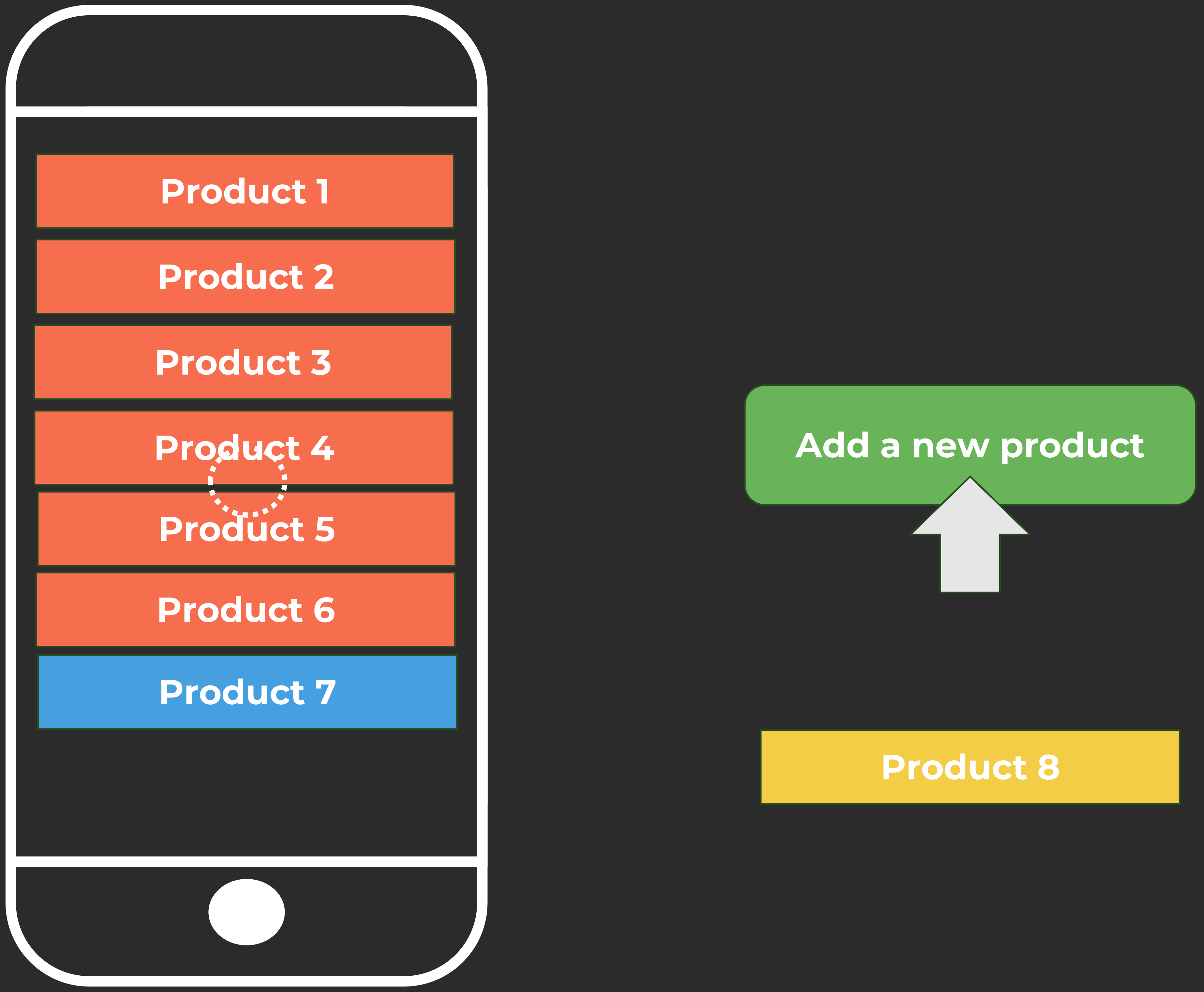
Flutter Developer & Trainer

in in/rahul-connect

Section Overview

- Understand State and State Management in Flutter
- Discuss the significance of effective state management
- Explore the Hands-On Starter Project
- Identify challenges associated with using setState

What is State and State Management?


Product 1
Product 2
Product 3
Product 4
Product 5
Product 6
Product 7
Add a new product
Product 8

The Problem with setState

- setState() triggers full subtree rebuilds, impacting performance in complex UIs.
- Limited to managing local state within StatefulWidget, leading to cumbersome state passing.
- Mixing UI and business logic in one class with setState() results in less maintainable code.
- Debugging is challenging due to setState()'s full rebuilds, prolonging development cycles.
- setState() for asynchronous operations like network requests can lead to nested callbacks.
- Extensive setState() usage makes unit testing and integration testing more challenging.

Section Summary

- Understood state and state management in Flutter
- Discussed the significance of effective state management
- Explored the Hands-On Starter Project
- Identified challenges associated with using setState

Creating and Implementing Bloc

Up Next

Creating and Implementing Bloc

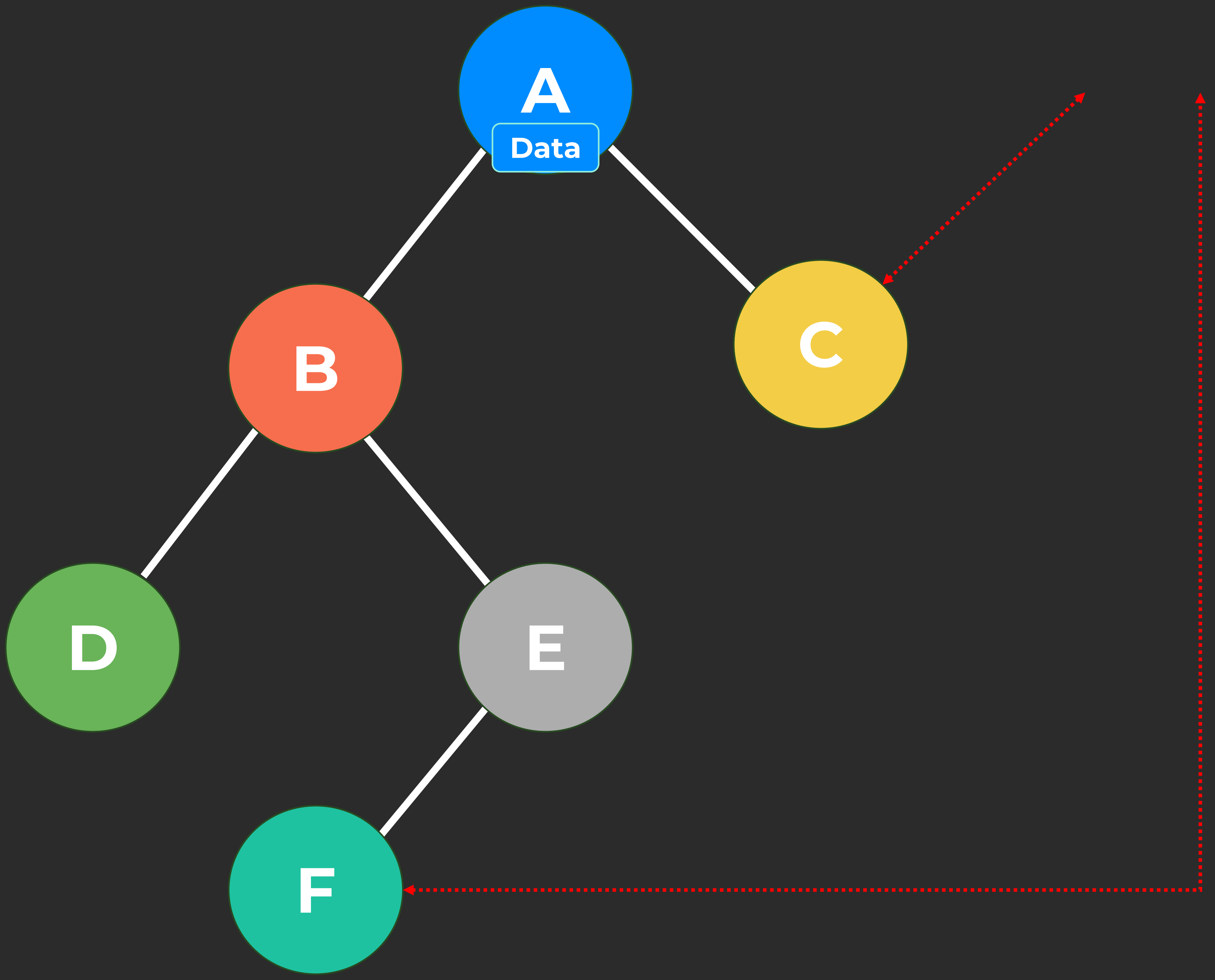
Rahul Agarwal
Flutter Developer & Trainer
in in/rahul-connect

Section Overview

- Introduction to Bloc State Management
- Importance and Advantages of Bloc
- Understanding Events and States
- Implementation of First Bloc

What is Bloc?

- Powerful state management library for Flutter.
- Follows event-driven architecture.
- UI elements emit events triggering state changes.
- Blocs process state changes in a clean and organized manner
- Advantages:
  - Separates business logic from UI, leading to cleaner code
  - Promotes code reuse and modularity
  - Facilitates easier unit testing for business logic
  - Handles asynchronous operations efficiently
  - Provides predictable state management
  - Supported by a strong community with ample resources


A
Data
B
C
D
E
F

Events and States

- Data flow in Bloc is managed through events and state.
- This separation ensures clarity and predictability in your application's data flow.
- Events:
  - Represent inputs to a Bloc
  - Trigger Bloc responses
  - Examples: user interactions, API calls
  - Result in state changes
- States:
  - Outputs of a Bloc
  - Represent UI conditions or data configurations.
  - Result from event processing
  - Emitted by Bloc and consumed by UI

How to trigger or add events

Approach 1 : Using BlocProvider : You can access your Bloc using the BlocProvider.of<BlocName>(context) method. This approach is useful when you want to assign your Bloc instance to a variable and use it multiple times. Here’s how you can do it:

```
var blocVariable = BlocProvider.of<BlocName>(context)
blocVariable.add(EventName());
```

How to trigger or add events

Approach 2 : Using context.read : Alternatively, you can directly add an event to the Bloc using the context.read<BlocName>().add(EventName()) method. This approach is concise and straightforward, making it ideal for one-time event dispatching. Here’s an example:

```
var blocVariable = BlocProvider.of<BlocName>(context)
context.read<BlocName>.add(EventName());
```

Section Summary

- Discussed Bloc State Management
- Highlighted its significance and benefits
- Explored Events and States concept
- Demonstrated creation and implementation of initial Bloc

Managing Data Flow Between Bloc Components

Up Next

Managing Data Flow Between Bloc Components

Rahul Agarwal

Flutter Developer & Trainer

in in/rahul-connect

Section Overview

- Passing Data within Events and States
- Using Model Classes for Efficient Data Handling
- Handling Complex Data Interactions

Section Summary

- Learned data passing within events and states
- Utilized model classes for object passing
- Gained understanding of complex data interactions in apps

Enhancing Bloc with Equatable and Bloc Observers

Up Next

droidcon academy

Enhancing Bloc with Equatable and Bloc Observers

Rahul Agarwal

Flutter Developer & Trainer

in in/rahul-connect

Section Overview

- Understanding the Equatable Package
- Implementing Equatable in Bloc
- Introduction to Bloc Observers
- Implementation and Benefits of Bloc Observers

Use of Equatable Package

The Equatable package is used to compare objects and check if they have equal value. While it's possible to do this manually by overriding the hashCode and == operator methods, this approach can become time-consuming as your app grows and you have multiple variables and properties to manage.

```
final user1 = User("Rahul");
final user2 = User("Rahul");

final isEqual = user1 == user2;
// We will get isEqual as false even if they are objects with same value
```

Section Summary

- Discussed significance and problem-solving of Equatable
- Implemented Equatable in Bloc
- Introduced and implemented Bloc Observers

Introduction to **Cubit** and **Architectural** Pattern

Up Next

droidcon academy

Introduction to Cubit and Architectural Pattern

Rahul Agarwal

Flutter Developer & Trainer

in in/rahul-connect

Section Overview

- Explore the concept of Cubit and its advantages
- Implement Cubit, replacing Bloc to understand differences
- Discuss the concept of Architectural Patterns
- Learn about effective folder and file structure

What is Cubit?

- Lightweight state management solution in Flutter, simpler than Bloc.
- Part of the Bloc library, manages state without the need for events.
- Advantages:
  - Simplified API
  - Less Boilerplate
  - Direct State Emission
  - Ideal for Simple State Management
  - Seamless Integration

What is Architectural Pattern?

The Architectural Pattern we will be using :

```
lib/ (Root directory for your Flutter application)
•  bloc/cubit
•  config/
      - routes/
      - themes/
•  data/
      - models/
      - repositories/
•  presentation/
      - screens/
      - widgets/
•  utils/
      - constants.dart
•  main.dart
```

Here's a breakdown of the Architecture :

- BLoC/Cubit:
  - Manages business logic and state with BLoC or Cubit pattern
- Config:
  - Contains route definitions and theme configurations
- Data:
  - Handles data operations, including models and repositories
- Presentation:
  - Manages UI components, screens, and widgets
- Utils:
  - Contains utility functions and constants
- Main.dart:
  - Entry point of the Flutter app

Advantages of a Good Architecture Pattern

- Modularity:
  - Easier to understand, maintain, and extend with isolated components
- Scalability:
  - Scales gracefully with organized code and managed dependencies
- Maintainability:
  - Clear separation of concerns for localized changes and reduced bugs
- Testability:
  - Decoupling promotes easier unit testing and ensures expected behavior
- Reusability:
  - Encapsulated logic reduces duplication and speeds up development

droidcon academy

Section Summary

- Explored the concept of Cubit and its advantages
- Implemented Cubit, replacing Bloc to understand differences
- Discussed the concept of Architectural Patterns
- Learnt about effective folder and file structure

Building an Expense Tracker Application

Up Next

droidcon academy

Building an Expense Tracker Application

Rahul Agarwal

Flutter Developer & Trainer

in/rahul-connect

Section Overview

- Setting Up Expense Tracker Project
- Implementing the discussed Architectural Pattern
- Crafting UI for Home Screen

Section Summary

- Initiated the Expense Tracker project setup
- Organized files and folders according to discussed architectural pattern
- Developed UI for the Home Screen

Designing the Recent Transactions UI

Up Next

droidcon academy

Designing the Recent Transactions UI

Rahul Agarwal

Flutter Developer & Trainer

in in/rahul-connect

Section Overview

- Designing Recent Transactions UI
- Creating Layout for Transactions
- Organizing Transactions by Categories
- Enhancing UX with intl Package for Date Formatting

Section Summary

- Designed UI for displaying recent transactions
- Developed layout to differentiate income and expenses
- Organized transactions by various categories
- Utilized intl package for improved date formatting and user experience

Building the Add Transaction Screen

Up Next

Building the Add Transaction Screen

Rahul Agarwal

Flutter Developer & Trainer

in in/rahul-connect

Section Overview

- Building the Add Transaction Screen
- Inputting Transaction Details
- Implementing Named Routing and onGenerateRoute

Section Summary

- Developed the Add Transaction Screen
- Enabled users to input transaction details
- Utilized named routing and onGenerateRoute for seamless navigation

Working with Hive Database

Up Next

Working with Hive Database

Rahul Agarwal

Flutter Developer & Trainer

in in/rahul-connect

Section Overview

- Benefits and Installation of Hive
- Generating Type Adapters for Efficient Data Storage
- Implementing Bloc for Add Transaction Screen
- Code Challenge to reinforce learning

What is Hive?

- NoSQL database for local data storage.
- Enables easy storage and retrieval of data on the device, even offline.
- Utilizes "boxes" for data storage, similar to tables in SQL databases.
- Advantages:
  - Offline storage
  - Key-value store
  - Fast and efficient
  - Flexible data types

Section Summary

- Discussed benefits and installation process of Hive
- Generated type adapters for efficient data storage
- Implemented Bloc for seamless integration with Hive Database
- Engaged in a code challenge to reinforce learning

Home Screen Data Handling with Cubit

Up Next

Home Screen Data Handling with Cubit

Rahul Agarwal

Flutter Developer & Trainer

in in/rahul-connect

Section Overview

- Fetching Transactions and Calculating Summary Data
- Implementation of Cubit for Home Screen
- Code Challenge to reinforce learning
- Adding Functionality for Transaction Deletion

Section Summary

- Implemented methods to fetch transactions and calculate summary data
- Created and implemented a Cubit tailored for the Home Screen
- Engaged in a code challenge to reinforce learning
- Added functionality for seamless transaction deletion

Creating and Integrating Stats Screen with Bloc

Up Next

droidcon academy

Creating and Integrating Stats Screen with Bloc

Rahul Agarwal

Flutter Developer & Trainer

in in/rahul-connect

Section Overview

- Creating the Stats Screen
- Integrating Bloc for State Management
- Crafting Transaction Statistics for Specific Period

Section Summary

- Developed Stats Screen for Expense Tracker App
- Integrated Bloc for efficient state management
- Implemented transaction statistics for specific periods

App Personalization and Bloc Best Practices

Up Next

App Personalization and Bloc Best Practices

Rahul Agarwal

Flutter Developer & Trainer

in in/rahul-connect

Section Overview

- Enhancing Stats Screen with Pie Chart
- Updating App Launcher Icon and Name
- Discussing Bloc Best Practices

Best Practices for using Bloc

- One State Bloc Pattern:
  - Use a single state class to reduce boilerplate and simplify state management
- Use Equatable:
  - Implement “==” operator and hashCode for comparing complex objects
- Static Code Analysis:
  - Employ tools like dartanalyzer or linters to catch issues early
- Use RxDart:
  - Implement event debouncing with debounceTime for rapid input handling
- Asynchronous Operations:
  - Gracefully handle API calls and database interactions, emitting loading, success, and error states

Best Practices for using Bloc

- Explore Freezed:
  - Use libraries like freezed for generating immutable state classes and efficient state comparison
- Use Sealed Unions:
  - Utilize sealed_unions for modeling state transitions in your bloc
- Documentation:
  - Document bloc classes, methods, and events for clarity and collaboration
- Testing:
  - Implement thorough unit tests for blocs to ensure stability and reliability
- Version Control:
  - Use Git for effective version control, collaboration, and code review

Section Summary

- Improved Stats Screen by incorporating a pie chart
- Enhanced app appearance by updating launcher icon and name
- Discussed Bloc Best Practices

Course Summary

- Importance of State Management in building responsive and efficient apps.
- Challenges with using setState for state management.
- Gained proficiency in state management with Bloc.
- Understood the concept of events and states in the context of Bloc.
- BlocBuilder and BlocConsumer for efficiently building UI based on Bloc states.
- Explored the Equatable package to streamline equality checks.
- Explored Cubit simplified state management approach.
- Built an Expense Tracking app.
- Implemented fl_chart package to visualize transaction data using Pie charts.
- Discussed Best practices for using Flutter Bloc for optimizing performance.

Thank You!

Rahul Agarwal

Flutter Developer & Trainer

in in/rahul-connect