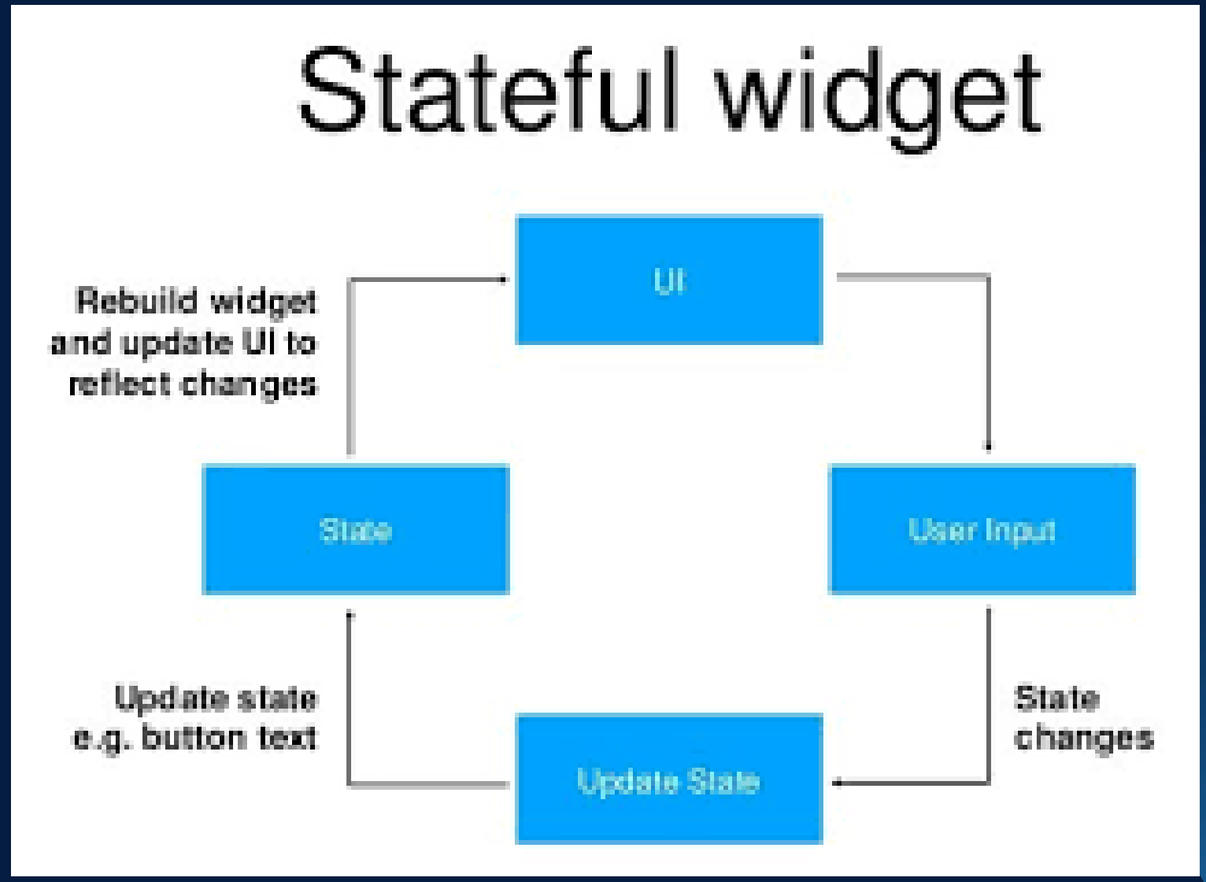
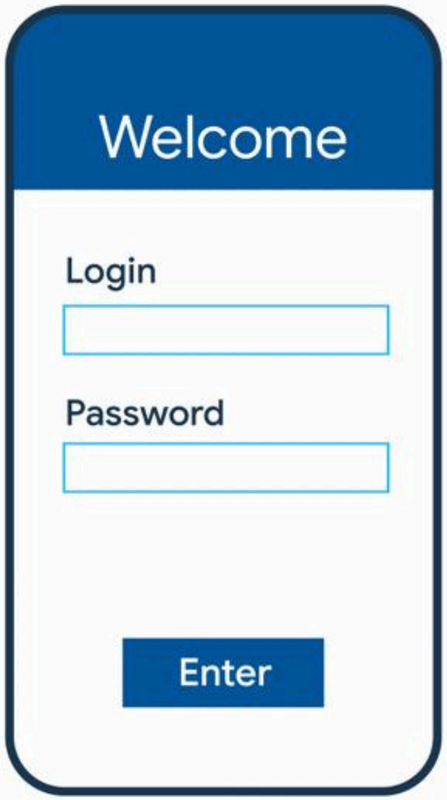
CS4039
SOFTWARE FOR
MOBILE DEVICES

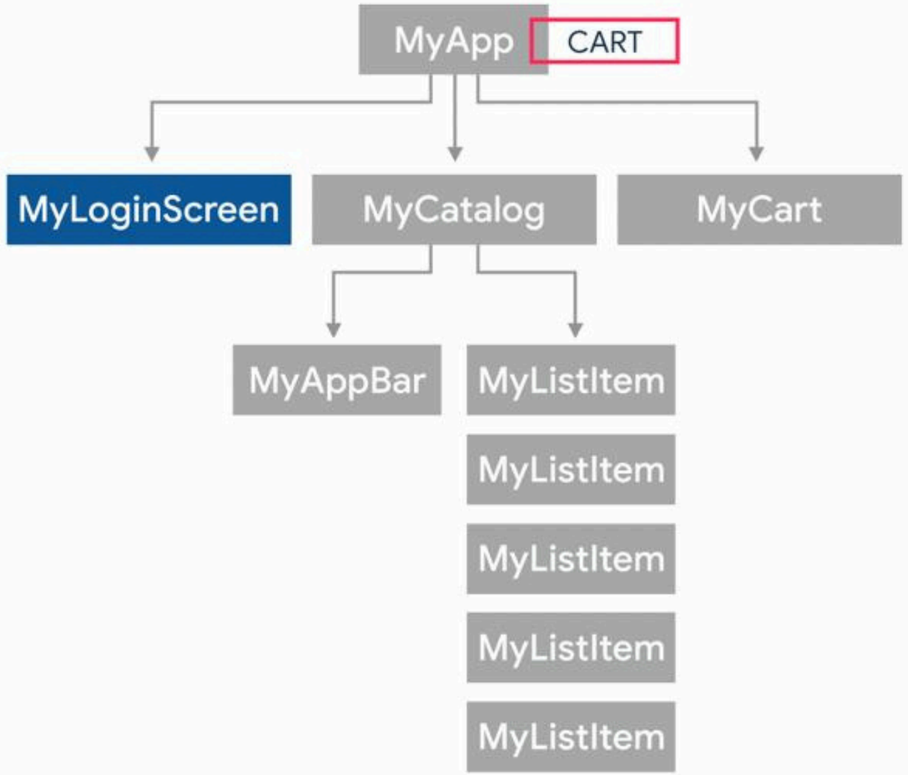
STATE MANAGEMENT

STATE MANAGEMENT

- Creating stateful widgets
- Maintaining states for each
- widget manually.
- Using simple setState()
- Is there a better way to
- manage it ?

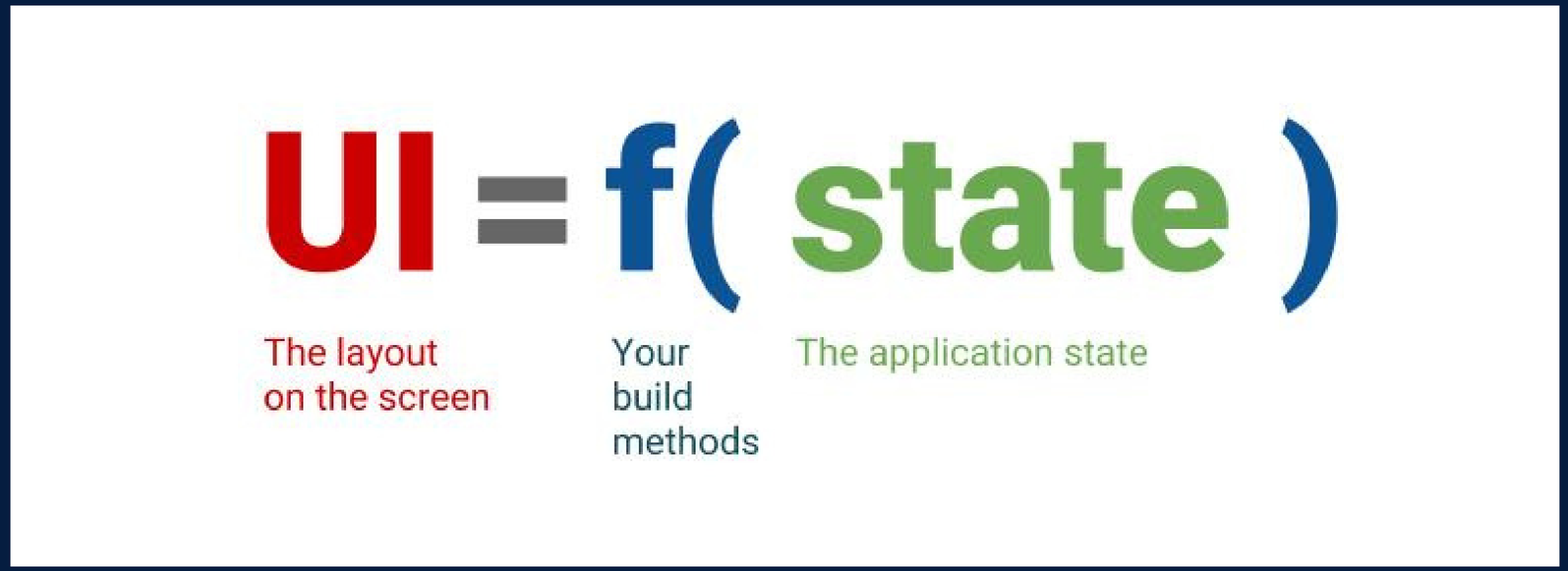



Welcome
Login
Password
Enter


MyApp
CART
MyLoginScreen
MyCatalog
MyCart
MyAppBar
MyListItem
MyListItem
MyListItem
MyListItem
MyListItem

START THINKING DECLARATIVELY

Flutter is declarative. This means that Flutter builds its user interface to reflect the current state of your app.



EPHEMERAL STATE

- **Ephemeral :: lasting for a very short time.**
- **_index is ephemeral state.**

```
class MyHomepage extends StatefulWidget {
  const MyHomepage({super.key});

  @override
  State<MyHomepage> createState() => _MyHomepageState();
}

class _MyHomepageState extends State<MyHomepage> {
  int _index = 0;

  @override
  Widget build(BuildContext context) {
    return BottomNavigationBar(
      currentIndex: _index,
      onTap: (newIndex) {
        setState(() {
          _index = newIndex;
        });
      },
      // ... items ...
    );
  }
}
```

APP STATE

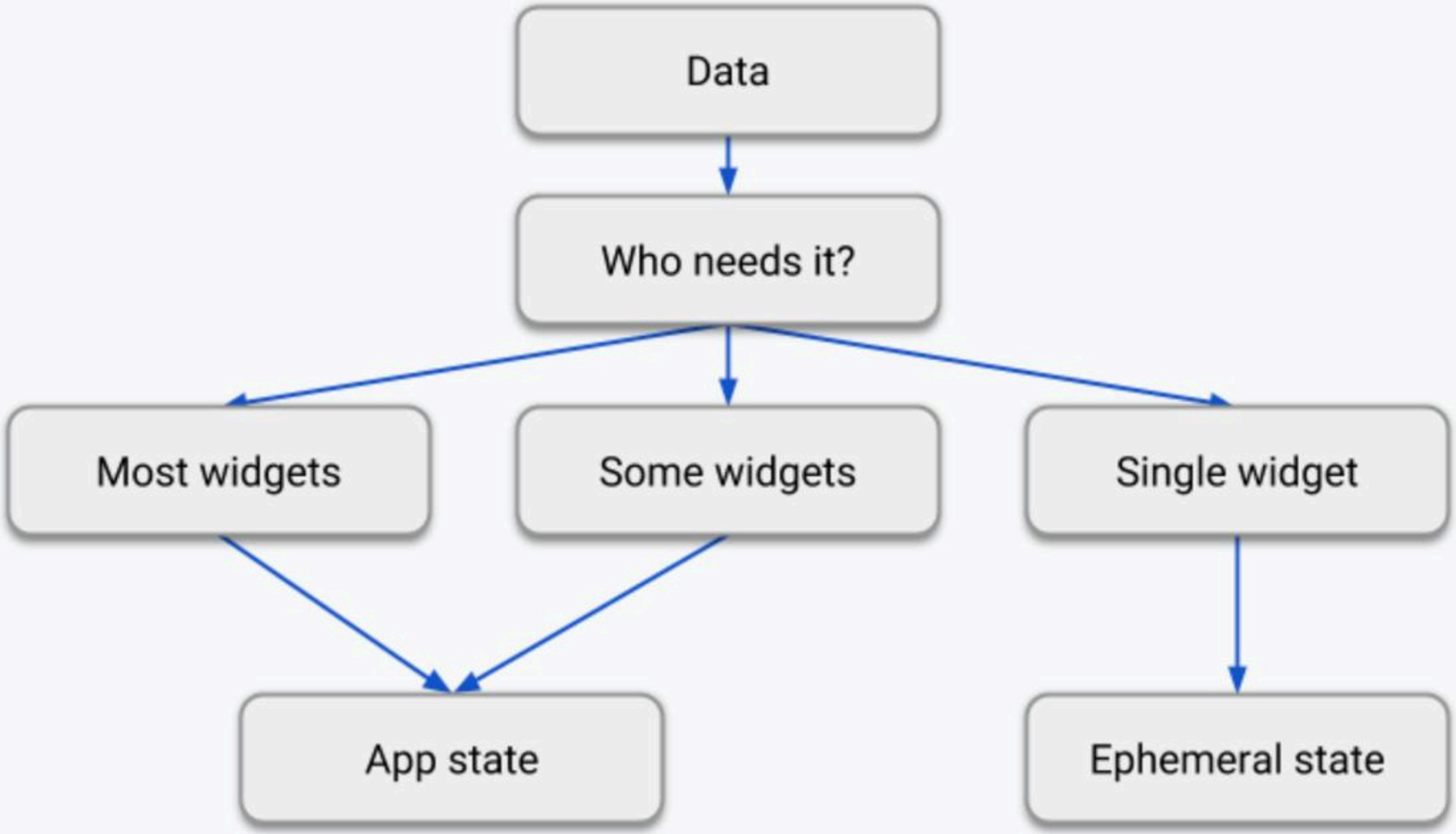
- State that is not ephemeral, that you want to share across many parts of your app, and that you want to keep between user sessions, is what we call application state (sometimes also called shared state).

EXAMPLES OF APPLICATION STATE

- User preferences
- Login info
- Notifications in a social networking app
- The shopping cart in an e-commerce app
- Read/unread state of articles in a news app

There is no clear-cut rule...

"THE RULE OF THUMB IS: DO WHATEVER IS LESS AWKWARD."


Data
Who needs it?
Most widgets
Some widgets
Single widget
App state
Ephemeral state

ARCHITECTURAL PATTERNS

- Why we use architecture patterns?
- Its Advantages
- Does it make your app scalable ?
- Is it helpful when you work with a team developer ?

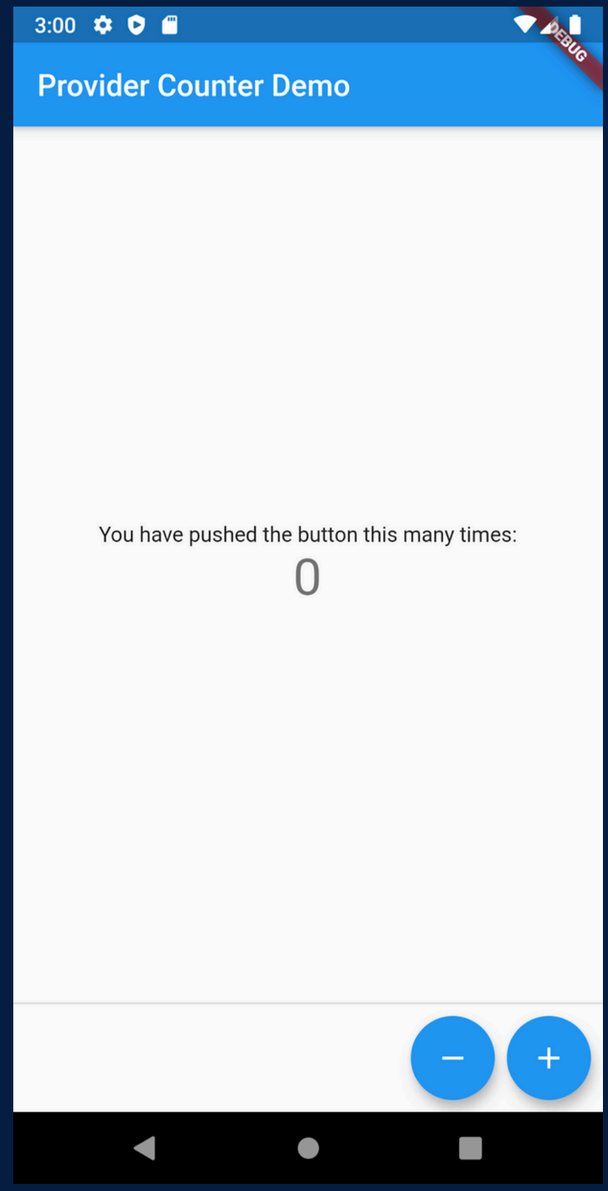
FLUTTER PROVIDER

- An implementation of observer
- pattern
- In which consumer waits of
- observable to emit changes
- using callbacks..

MAIN CONCEPTS

- Notifier
- Provider
- Consumer

CODE WALK THROUGH FOR PROVIDER IN A SIMPLE COUNTER APPLICATION

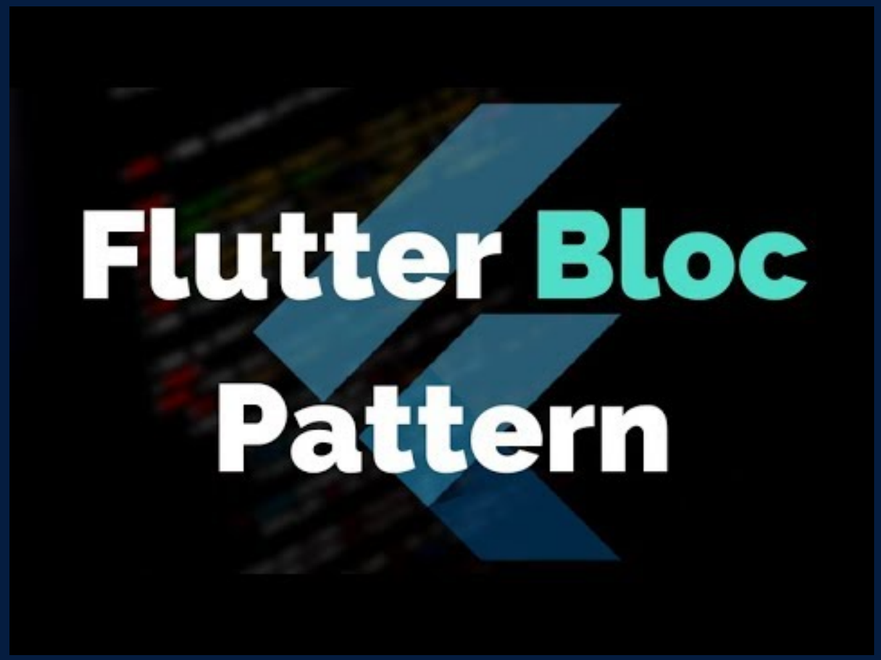


WHAT IS REACTIVE PROGRAMMING

- Streams
- Async*
- yield

BLOC

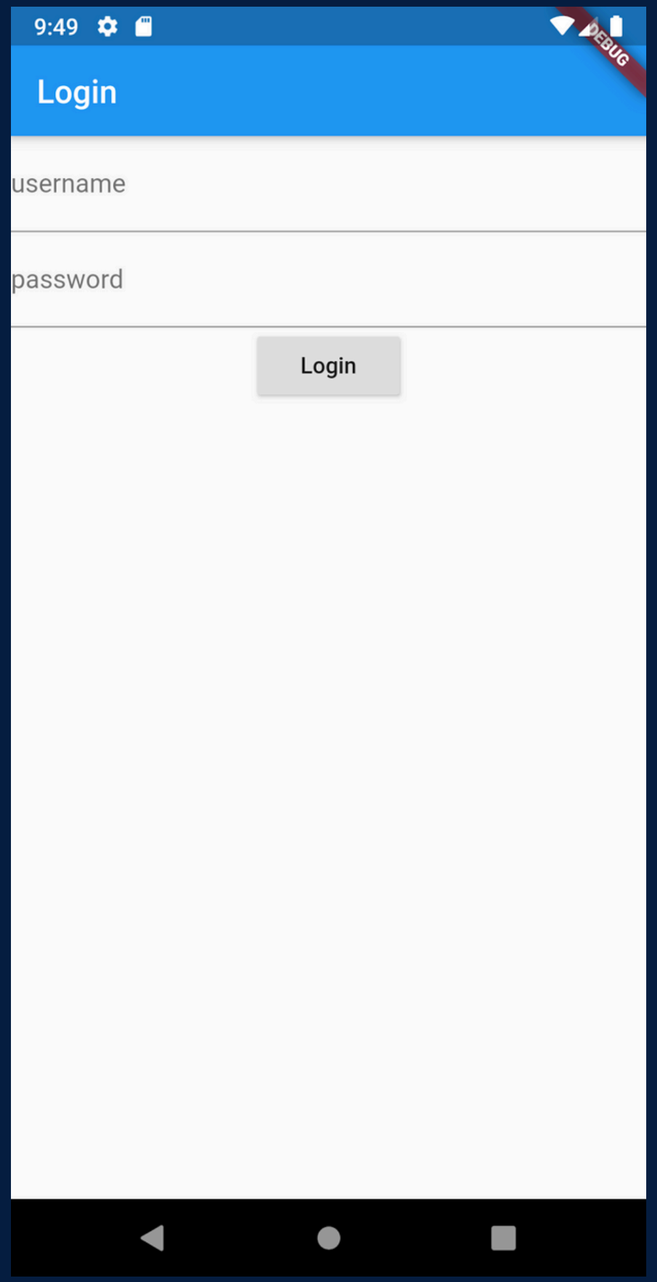
- A reactive support architecture to update states based on streams

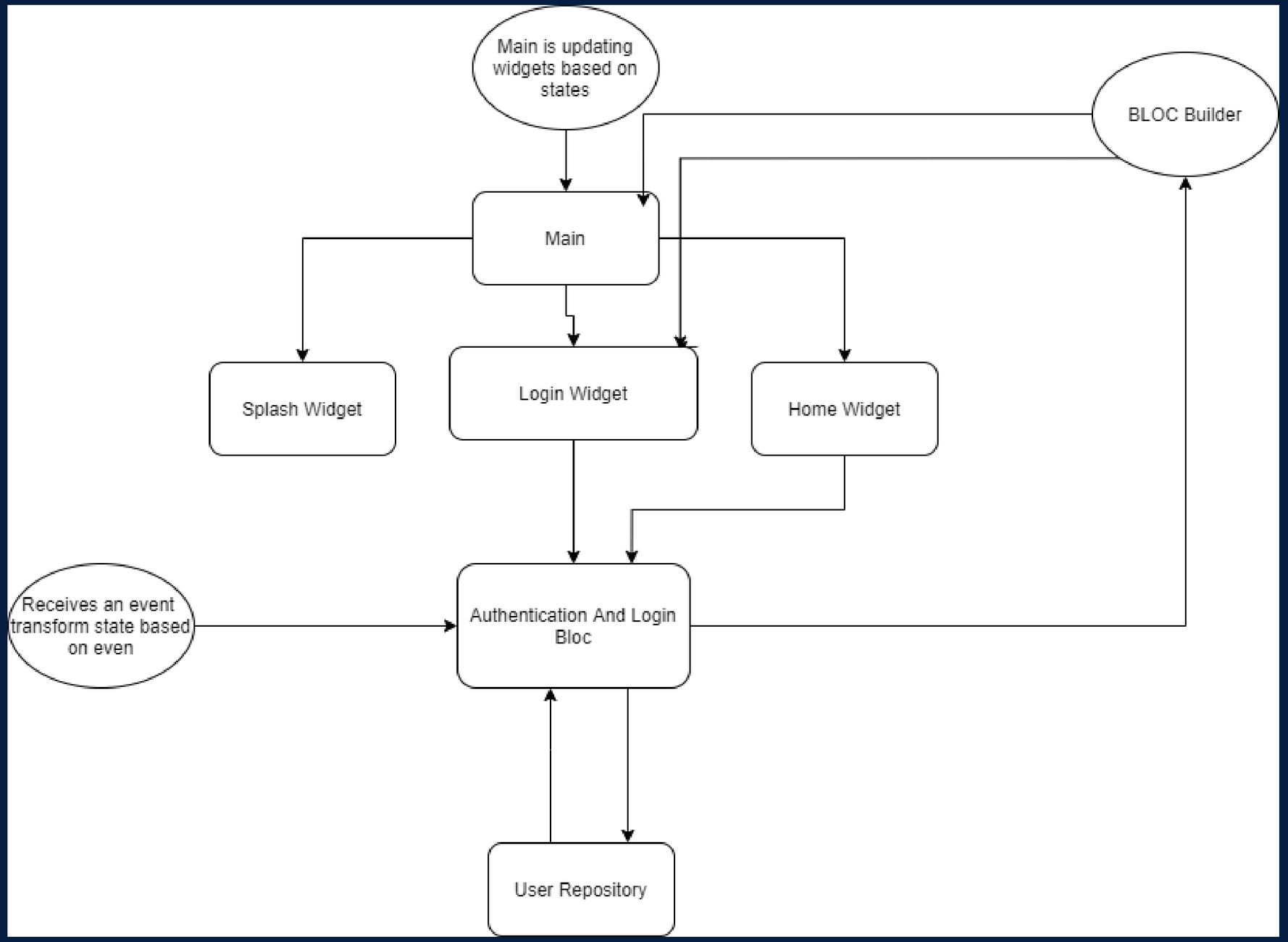


MAIN CONCEPTS

- Event
- States
- Bloc
- Bloc Builder
- Repository

CODE WALK THROUGH FOR BLOC CREATING LOGIN FLOW




Main is updating widgets based on states
BLOC Builder
Main
Splash Widget
Login Widget
Home Widget
Receives an event transform state based on even
Authentication And Login Bloc
User Repository