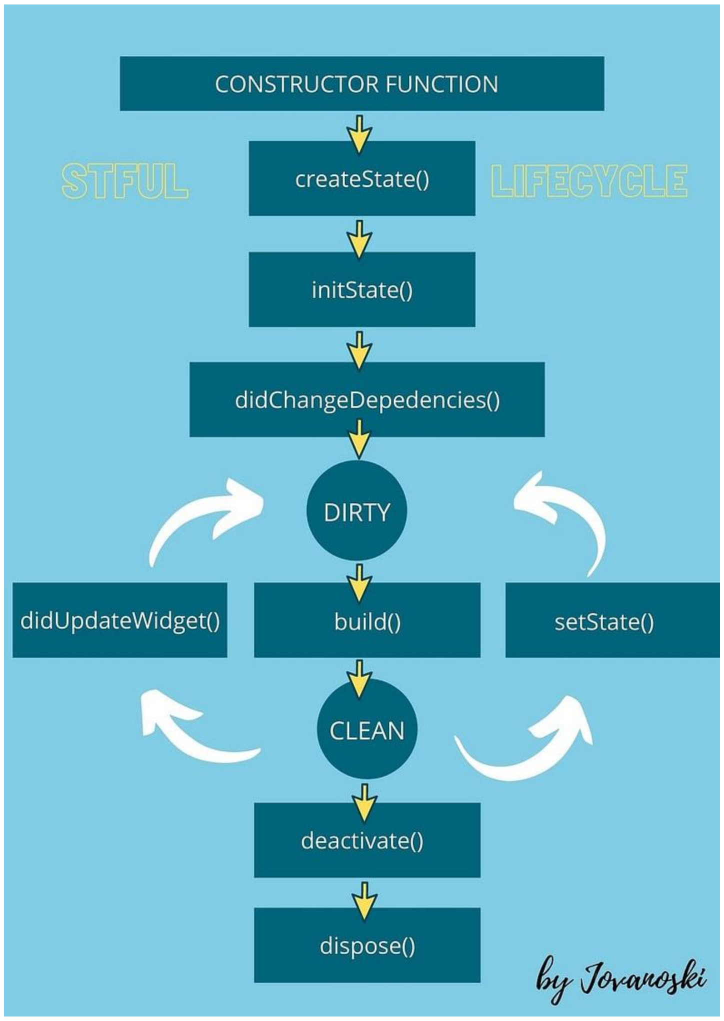
Task 8 – Report

Report to show details of stateful life cycle of stateful widget

Stateful widgets are an essential component in Flutter for building dynamic and interactive user interfaces. In this report, we will explore the various stages of the stateful widget's lifecycle and understand how it behaves in different situations. By understanding the lifecycle, developers can efficiently manage the state and provide a smooth user experience.



Widget Initialization:

createState(): When a stateful widget is created, the framework calls the createState() method. This method returns an instance of the associated State class, which manages the widget's mutable state.

```
class MyWidget extends StatefulWidget {
  @override
  _MyWidgetState createState() => _MyWidgetState();
}
```

Mounting:

initState(): After creating the state object, the framework calls the initState() method. It is called only once and is used for initializing the state and subscribing to any necessary resources.

didChangeDependencies(): This method is called immediately after initState(), and it provides an opportunity to handle any changes in the widget's dependencies. For example, if the widget depends on an InheritedWidget, this method will be called when the inherited widget changes.

```
class _MyWidgetState extends State<MyWidget> {
  @override
  void initState() {
    super.initState();
    // Perform initialization tasks here
  }
}
```

Updating:

didUpdateWidget(): Whenever the parent widget rebuilds and supplies a new instance of the stateful widget, the framework calls didUpdateWidget() method. It allows you to compare the old and new widget instances and perform any necessary updates.

```
class _MyWidgetState extends State<MyWidget> {
  @override
  void didUpdateWidget(MyWidget oldWidget) {
    super.didUpdateWidget(oldWidget);
    // Compare oldWidget with widget and perform updates if needed
  }
}
```

Rendering:

build(): The build() method is called whenever the framework determines that the widget needs to be rebuilt. It is responsible for creating the widget's user interface based on the current state.

```
class _MyWidgetState extends State<MyWidget> {
  @override
  Widget build(BuildContext context) {
    return Container(
      // Widget UI representation based on the current state
    );
  }
}
```

User Interaction:

User interactions, such as tapping a button or scrolling a list, can trigger changes in the widget's state. When the state is updated, the framework calls the build() method again to reflect the changes visually.

```
class _MyWidgetState extends State<MyWidget> {
  int counter = 0;

  void incrementCounter() {
    setState(() {
      counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Text('Counter: $counter'),
        ElevatedButton(
          onPressed: incrementCounter,
          child: Text('Increment'),
        ),
      ],
    );
  }
}
```

deactivate(): If a stateful widget is removed from the tree but might be inserted again later, the framework calls the deactivate() method. It allows the widget to clean up any resources before being removed from the tree.

```
class _MyWidgetState extends State<MyWidget> {
  @override
  void deactivate() {
    // Clean up resources before the widget is removed from the tree
    super.deactivate();
  }
}
```

dispose(): When a stateful widget is permanently removed from the tree, the framework calls the dispose() method. It is the last opportunity to release any resources, subscriptions, or animations held by the widget.

```
class _MyWidgetState extends State<MyWidget> {
  @override
  void dispose() {
    // Release resources, subscriptions, or animations held by the widget
    super.dispose();
  }
}
```