|  |  |
| --- | --- |
| LAB 5  INFT 3101-01 | Varun Singh (100865156)  Date submitted: 8-11-2024 |

Table of Contents

[**Topic Overview (40%)** 2](#_Toc182000842)

[**Definition** 2](#_Toc182000843)

[**Importance of the Topic in Mobile App Development** 2](#_Toc182000844)

[**Comparison with Related or Alternative Features** 3](#_Toc182000845)

[**Example Implementation (40%)** 4](#_Toc182000846)

[Example 1: A Timer application 4](#_Toc182000847)

[Example 2: Listener for theme change in the device/emulator 5](#_Toc182000848)

[Output: 6](#_Toc182000849)

[**Real World Scenario** 7](#_Toc182000850)

[Scenario Application 7](#_Toc182000851)

[**REFERENCES** 7](#_Toc182000852)

# **Topic Overview (40%)**

Topic of choice: **Responding to Widget Lifecycle Events.**

### **Definition**

A widget lifecycle in Flutter outlines several stages a widget goes through until it is removed from the widget tree. A widget with states has a lifecycle defined in Flutter. The lifecycle is controlled by a series of events and methods associated with the process, which allow the developers to act accordingly to the changes that occur in the widget's state. Some of the key lifecycle methods used in Flutter are defined below:

* ***initState****:* This method is called at the beginning of the cycle when the creation of the widget is initialized. As the method states initialize state. The usage of the method is to initialize and allocate resources such as network connections and animations.
* ***build****:* The build method is called immediately after the *initState* method or whenever there is a state change in the widget, and it needs to be drawn again.
* ***didChangeDependencies:*** As the method states, it is triggered when any dependencies on the inherited widget get changed such as data from any parent/ancestor widget. This is also called after *initState* while initializing the widget for the first time.
* ***didUpdateWidget:*** This is called whenever a configuration is changed.
* ***dispose:*** This method is used to remove the widget from the widget tree. This method is used by developers to clean and free up resources.

## **Importance of the Topic in Mobile App Development**

Some of the crucial reasons for understanding and managing widget lifecycles are listed below.

* **Managing Resources**

Since the widget involves several heavy tasks such as animations. The widget lifecycle methods are important for cleaning and freeing up resources.

* **Managing Widget’s State**

Developers need to maintain and manage different states of the widgets they use in the development of any application.

* **User Experience**

The control over different states of the widgets allows the developer to deliver a better user experience to the end user through the application. This is achieved through optimized responsiveness and the update/disposal of resources.

## **Comparison with Related or Alternative Features**

The Widget lifecycle concept in Flutter is shared among many frameworks like React Native and Android development.

* **React Native:**  The framework React Native has a similar concept to Flutter’s widget lifecycle called component lifecycle. The components lifecycle includes similar methods as there are in Flutter. Some examples of the component lifecycle methods include componentDidMount, componentDidUpdate, and componentWillUnmount. The major difference between these two is syntax and application scope.
* **Android:**  The applications in Android have a similar concept called Activity lifecycle. Some methods associated with activity lifecycle are onCreate, onPause, and onDestroy. The focus of the lifecycle is on activity management whereas Flutter’s widget lifecycle methods have more depth allowing more control over the state management.

**Advantages and Disadvantages of Using This Feature**

#### Advantages:

1. **Efficient Resource Management**: This feature leads to efficient resource management by providing the developer more control over different states of widgets and memory.
2. **Precision Control**: Developers can code to any specific state change of the widget and handle it accordingly. This feature increases the precision of the application.
3. **Optimized Performance**: Since the developers can manage different states of the widgets, this allows the developer to avoid any unnecessary computations/updates.

#### Disadvantages:

1. **Hard to learn:** Since understanding the widget lifecycle has a hard learning curve, it can be quite challenging for developers who are new to the field.
2. **Bug vulnerabilities:** Any false use of widget lifecycle methods can increase the application’s vulnerability to bugs. For example, the incorrect use of *dispose* method can lead to a memory leak.

# **Example Implementation (40%)**

### Example 1: A Timer application

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

#### Output:

A screenshot of a computer

Description automatically generated

#### Commentary

* initState: We initialize the timer widget using the initState method.
* setState: The variable \_counter is increased every second and the setState method is called to build the widget again.
* build: This method is used to redraw the widget with the new counter value.

#### Issues/ considerations to keep in mind

* Ensure that timer is disposed using dispose method as its going to go on and eventually crash.
* setState is being used to build the UI again. Use it cautiously to avoid any updates that are not required.

### Example 2: Listener for theme change in the device/emulator

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

### Output:

This application shows output whenever the theme of the emulator/device is changed using didChangeDependencies displaying either dark/light mode in the output window.

A black background with white text

Description automatically generated

#### Commentary

* didChangeDependencies: This is called when there is a change in the properties of the inherited widget
* build: Used to update/ draw the UI to display dark/light mode.

#### Issues/ considerations to keep in mind

* Avoid unnecessary rebuilds by didChangeDependencies
* Make sure to invoke only required updates.

## **Real World Scenario**

A real-world implementation of this topic can be used in an application used for tracking stock prices live. The requirements for the application will be to fetch real-time stock prices over different periods and display them to the user. This app highly demands the usage of widget lifecycle methods to stop updating when the widget is not being used or is not being seen. Methods like initState and dispose can be pretty useful here. initState will be used for starting the process of fetching prices and dispose can be used when the user drives away or shifts to some other application.

### Scenario Application

The example can also be applied to any applications that require real-time data fetching frequently. Some examples that can use these features are weather apps and news feed applications. The widget lifecycle methods can be useful to save and efficiently use the resources for an optimized performance.

# **REFERENCES**

Add interactivity to your Flutter app. (n.d.). <https://docs.flutter.dev/development/ui/interactive>

Internationalizing flutter apps. Flutter. (n.d.-a). <https://docs.flutter.dev/ui/accessibility-and-internationalization/internationalization>

Timer class abstract interface. Timer class - dart:async library - Dart API. (n.d.). <https://api.flutter.dev/flutter/dart-async/Timer-class.html>

*Add interactivity to your flutter app. Flutter. (n.d.-a).* [*https://docs.flutter.dev/development/ui/interactive#context-and-inheritedwidget*](https://docs.flutter.dev/development/ui/interactive#context-and-inheritedwidget)

Use themes to share colors and font styles. Flutter. (n.d.-e). <https://docs.flutter.dev/cookbook/design/themes>

Globalmateriallocalizations class abstract. GlobalMaterialLocalizations class - flutter\_localizations library - Dart API. (n.d.). <https://api.flutter.dev/flutter/flutter_localizations/GlobalMaterialLocalizations-class.html>