Introduction to Mobile Development- Lab 5

Handling Gestures in Flutter

By Ali Mujtaba- 100792180

A logo for a college

Description automatically generated

Ali Mujtaba- 100792180

INFT3101- Durham College

Professor Mohammed Shamas

November 5th , 2024

Contents

[What is Flutter and How Does It Work? 3](#_Toc181854095)

[What are Gestures in Flutter? 4](#_Toc181854096)

[1.1: How GestureDetector Works with Gestures 4](#_Toc181854097)

[1.2: Why Gestures Matter in Mobile App Development 4](#_Toc181854098)

[1.3: Comparison with Gesture Handling in Other Frameworks 4](#_Toc181854099)

[1.4: Advantages and Limitations of Flutter’s Gesture System 5](#_Toc181854100)

[1.5: Key Gesture Features in Flutter 5](#_Toc181854101)

[Coding Examples: 6](#_Toc181854102)

[Example 1 6](#_Toc181854103)

[Example 2: 7](#_Toc181854104)

[Errors while building: 10](#_Toc181854105)

[Real World Example 10](#_Toc181854106)

[References 11](#_Toc181854107)

## What is Flutter and How Does It Work?

Flutter is Google’s open-source toolkit for crafting high-quality, cross-platform applications for mobile, web, and desktop—all from a single codebase. Imagine a UI framework that brings your app vision to life across iOS, Android, and more, without any of the hassle of separate development for each platform. Powered by Dart, Flutter uses widgets for everything in the user interface. Here’s how it works:

**Widget-Based Design**: In Flutter, everything you see on the screen is a widget—from layout elements and buttons to the finer details of text and images. Widgets make building, nesting, and combining UI elements both straightforward and powerful.

**Reactive UI**: Flutter's reactive style lets the app update instantly in response to user actions or data changes. This responsiveness is crucial for crafting interactive apps.

**Skia Rendering Engine**: Flutter uses a high-performance rendering engine called Skia, so it doesn’t rely on native platform widgets. This independence enables Flutter apps to achieve the same look and feel across both Android and iOS platforms, all while looking beautifully native on both.

**Compiled to Native Code**: Flutter apps are compiled to native machine code, which ensures top-notch performance and smooth, native-quality experiences.

In short, Flutter makes it easy to build stunning, responsive apps without worrying about platform-specific code. With the power of Dart under the hood, Flutter’s framework is ideal for creating intuitive, interactive, and cross-platform applications.

## What are Gestures in Flutter?

In Flutter, gestures represent the ways users interact with an app through touch: tapping, swiping, pinching, dragging, just to name a few. The GestureDetector widget is like Flutter’s built-in listener for these interactions. It provides a wide range of callback functions such as onTap, onSwipe, onPan, and onLongPress that let developers add quick responses to users’ touches. Gestures add the fun and functionality in Flutter apps, turning simple screens into fully interactive experiences.

### 1.1: How GestureDetector Works with Gestures

The GestureDetector widget is at the heart of gesture handling in Flutter. When you wrap a widget in a GestureDetector, it becomes touch-aware, meaning it can respond to user actions with specific functions. Here’s a quick breakdown:

**GestureDetector:** A widget that listens for specific touch actions (gestures) like taps, swipes, and drags.

**Callback:** A function that’s triggered when a gesture is detected, like onTap for a tap or onPanUpdate for dragging.

When a user interacts with a widget wrapped in a GestureDetector, Flutter recognizes the gesture and activates the assigned callback, creating an interactive experience.

1.2: Why Gestures Matter in Mobile App Development

Gestures are critical to making apps feel alive and intuitive. On mobile devices, touch is the primary way users navigate, explore, and interact with an app. Flutter’s gesture system is designed to offer smooth, consistent experiences across both iOS and Android—no extra code required for each platform. This consistency enhances the user experience and simplifies the development process.

For example, in a photo app, gestures might allow users to zoom in or out with a pinch or swipe through their photo gallery. By adding such interactions, developers create a more natural, enjoyable experience.

### 1.3: Comparison with Gesture Handling in Other Frameworks

Flutter’s GestureDetector provides a streamlined approach to handling gestures compared to other frameworks:

**React Native:** Uses PanResponder and third-party libraries like react-native-gesture-handler. It’s more complex to configure and can require a bit more setup for fluid gesture handling compared to Flutter’s built-in system.

**Native iOS (Swift) and Android (Kotlin):** Native platforms use UITapGestureRecognizer on iOS and GestureDetector on Android, both of which are powerful but platform-specific. This means writing separate code for each platform to handle gestures, whereas Flutter’s system is unified.

Flutter’s gesture handling is more straightforward, saving developers time and effort on cross-platform projects while delivering smooth, consistent gestures.

### 1.4: Advantages and Limitations of Flutter’s Gesture System

**Advantages**

* **Cross-Platform Consistency:** Gestures are uniform across iOS and Android, which streamlines both development and user experience.
* **Easy to Implement:** GestureDetector makes adding gestures to an app a breeze, reducing complexity in gesture-heavy apps.
* **Highly Customizable:** Flutter’s gesture system supports a variety of touch interactions and allows developers to create unique gestures for specific tasks.
* **Smooth and Responsive:** Flutter’s high-performance rendering ensures smooth animations and interactions even on complex layouts.

**Limitations**

* **Customization Complexity:** While Flutter supports custom gestures, setting them up can require additional work, especially for very unique interactions.
* **Performance Overhead:** Complex gesture handling on screens with heavy layouts can slightly impact performance, particularly on older devices.
* **Learning Curve:** For developers new to Flutter or gesture handling, mastering the gesture system might take time.

### 1.5: Key Gesture Features in Flutter

**Pre-Built Gesture Callbacks**: The GestureDetector widget comes with a host of built-in callbacks such as onTap, onLongPress, and onPanUpdate, covering a variety of common touch interactions.

**Custom Gesture Creation:** Developers can create unique gesture recognizers for complex or app-specific touch interactions, adding flexibility for tailored user experiences.

**Multi-Gesture Support:** Flutter’s GestureDetector can handle combined gestures, like dragging and zooming, enabling complex interactions with ease.

**Animation Triggers:** Flutter makes it simple to pair gestures with animations, allowing developers to bring a touch of dynamism to apps. Using AnimationController, it’s easy to trigger animations in response to user gestures, adding depth to the app’s interface.

In a nutshell, Flutter’s gesture system is a powerful tool for building interactive mobile applications, offering everything from simplicity to advanced customization. It’s easy to set up, consistent across platforms, and works seamlessly with Flutter’s animations.

## Coding Examples:

### Example 1

In this example, a text widget will be wrapped in GestureDetector, which will listen for an OnTap event. Once it receives a tap, it will call a function, called \_changeColor, which will be used to toggle the colour of the text, in this case between black and green. The US spelling for colour is being used in the code, as that is universally how you control colour’s in flutter.

**Source Code:**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**Output:**

Before tap

A screenshot of a computer

Description automatically generated

After tap

A white background with green text

Description automatically generated

### Example 2:

In this example, we will allow the user to move a coloured box around the screen, using onPanUpdate, and enlarge it using onDoubleTap. We will also use the clamp method, constraining the box size between 100-300 pixels.

**Source Code:**

**A computer screen shot of a program code

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**Output:**

Before any gestures

A screenshot of a computer

Description automatically generated

After drag

A screenshot of a computer

Description automatically generated

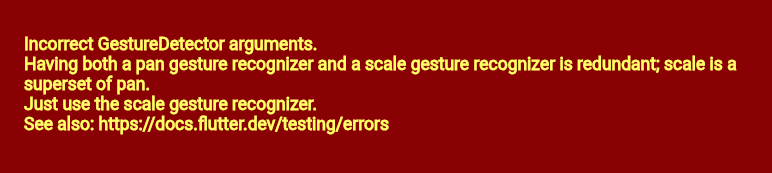
After double tapping twice

A screenshot of a computer

Description automatically generated

## Errors while building:

Originally I had a different idea for my second example. I thought of creating a screen with a box, which the user could drag using onPanUpdate, and resize using onScaleUpdate. In the first round of developing it, I did not read through the documentation enough to know that pan and scale are supersets of each other, as a result of this, I received the following error upon compilation.

****

In the second round of development, I couldn’t figure out how to resize the box even though I had the code to allow the user to do so, (I expected it to work like resizing an image in MSWord). It turns out, you need multitouch support to pinch the object, which isn’t supported on dartpad. As a result, I ended up scrapping that example and using onDoubleTap to enlarge the box, and onPanUpdate to move it.

## Real World Example

Scenario: Flappy Bird Clone in FlutterUI

In a Flutter-based Flappy Bird game, gestures are integral to the gameplay, providing an intuitive way for the player to interact with the game. In this case, the player taps the screen to make the bird "flap" and rise against gravity. This is achieved using the **onTap** gesture, which triggers a function that applies upward velocity to the bird each time the screen is tapped. This simple interaction allows the player to control the bird's movement and navigate through obstacles in the game.

Additionally, the game can incorporate more gestures for other interactions, such as swiping to navigate through the menus or double-tapping to restart the game after a crash. These gestures enhance the user experience by offering intuitive and immediate feedback, making the game feel dynamic and responsive. For example, after colliding with an obstacle, a double-tap gesture could be used to restart the game, providing a seamless transition from gameplay to the start screen.

## References

Flutter Documentation. (n.d.). *Gestures in Flutter*. Retrieved from https://flutter.dev/docs/development/ui/advanced/gestures

Flutter Documentation. (n.d.). *GestureDetector class*. Retrieved from https://api.flutter.dev/flutter/widgets/GestureDetector-class.html

Dart Documentation. (n.d.). *Dart Language Tour*. Retrieved from https://dart.dev/guides/language/language-tour

Medium. (2021). *Flutter Gesture Recognizers: Everything You Need to Know*. Retrieved from https://medium.flutterdevs.com/flutter-gesture-recognizers-everything-you-need-to-know-9d43d747c7a5

Dev.to. (2020). *Mastering Gestures in Flutter*. Retrieved from https://dev.to/flutter/mastering-gestures-in-flutter-3lgp

DartPad. (n.d.). *DartPad - Online tool for trying Flutter code*. Retrieved from <https://dartpad.dev/>

GitHub. (n.d.). *Flutter GitHub Repository*. Retrieved from <https://github.com/flutter/flutter>

GitHub. (n.d.). *Dart GitHub Repository*. Retrieved from <https://github.com/dart-lang/sdk>