Practical 4

Introduction to Gestures

Introduction

• Gestures are primarily a way for a user to interact with a mobile (or any touch-based device) application.

• Gestures are generally defined as any physical action / movement of a user in the intention of activating a specific control of the mobile device.

• Gestures are as simple as tapping the screen of the mobile device to more complex actions used in gaming applications.

Some of the widely used gestures

• Tap − Touching the surface of the device with fingertip for a short period and then releasing the fingertip.

• Double Tap − Tapping twice in a short time.

• Drag − Touching the surface of the device with fingertip and then moving the fingertip in a steady manner and then finally releasing the fingertip.

• Flick − Similar to dragging, but doing it in a speeder way. • Pinch − Pinching the surface of the device using two fingers.

• Spread/Zoom − Opposite of pinching. Panning − Touching the surface of the device with fingertip and moving it in any direction without releasing

Panning − Touching the surface of the device with fingertip and moving it in any direction without releasing.

Gesture Detector

• Flutter provides an excellent support for all type of gestures through its exclusive widget, Gesture Detector.

• Gesture Detector is a non-visual widget primarily used for detecting the user's gesture.

• To identify a gesture targeted on a widget, the widget can be placed inside Gesture Detector widget. Gesture Detector will capture the gesture and dispatch multiple events based on the gesture

Gestures and the corresponding events

• Tap – onTapDown – onTapUp – onTap – onTapCancel

• Double tap – onDoubleTap

• Long press – onLongPress

• Vertical drag – onVerticalDragStart – onVerticalDragUpdate – onVerticalDragEndGestures and the corresponding events

• Tap – onTapDown – onTapUp – onTap – onTapCancel • Double tap – onDoubleTap

• Long press – onLongPress

• Vertical drag – onVerticalDragStart – onVerticalDragUpdate – onVerticalDragEnd

• Horizontal drag-– onHorizontalDragStart – onHorizontalDragUpdate – onHorizontalDragEnd

• Pan – onPanStart – onPanUpdate

Practical 5

Introduction to Themes

Introduction

• Themes are an integral part of UI for any application.

• Themes are used to design the fonts and colors of an application to make it more presentable.

• In Flutter, the Theme widget is used to add themes to an application.

• One can use it either for a particular part of the application like butons and navigation bar or define it in the root of the application to use it throughout the entire app. Creating a Theme Use the Theme widget to create a theme. In themes some of the properties that can be used are given below:

• Text Theme

• brightness

• primarycolor

• accentColor

fontFamily

Creating a Theme

```
MaterialApp(
 title: title,
 theme: ThemeData(
        brightness: Brightness.dark,
primaryColor: Colors.lightBlue[800],
accentColor: Colors.cyan[600],
 fontFamily: 'Georgia’,
textTheme: TextTheme
( headline1: TextStyle(fontSize: 72.0, fontWeight: FontWeight.bold),
headline6: TextStyle(fontSize: 36.0, fontStyle: FontStyle.italic),
 bodyText2: TextStyle(fontSize: 14.0, fontFamily: 'Hind'),
),
```

How to use it

```
Container(
color: Theme.of(context).accentColor,
 child: Text(
'Hello Everyone!',
style: Theme.of(context).textTheme.headline6,
 ),
 );
```

Themes for part of an application

To override the app-wide theme in part of an application, wrap a section of the app in a Theme widget

There are two ways to approach this:

– Creating a unique ThemeData

– Extending the parent theme

Creating unique Theme Data

If you don’t want to inherit any application colors or font styles, create a ThemeData() instance and pass that to the Theme widget.

```
 Theme(
// Create a unique theme with `ThemeData`
 data: ThemeData(
splashColor: Colors.yellow,
),
child: FloatingActionButon(
onPressed: () {}, child: const Icon(Icons.add),
 ),
 );
```

Extending the parent theme

Rather than overriding everything, it often makes sense to extend the parent theme. You can handle this by using the copyWith() method.

```
 Theme(
 // Find and extend the parent theme using `copyWith`. See the next
// section for more info on `Theme.of`.
data: Theme.of(context).copyWith(splashColor: Colors.yellow),
```

```
child: const FloatingActionButon(
onPressed: null,
child: Icon(Icons.add) ),);
```