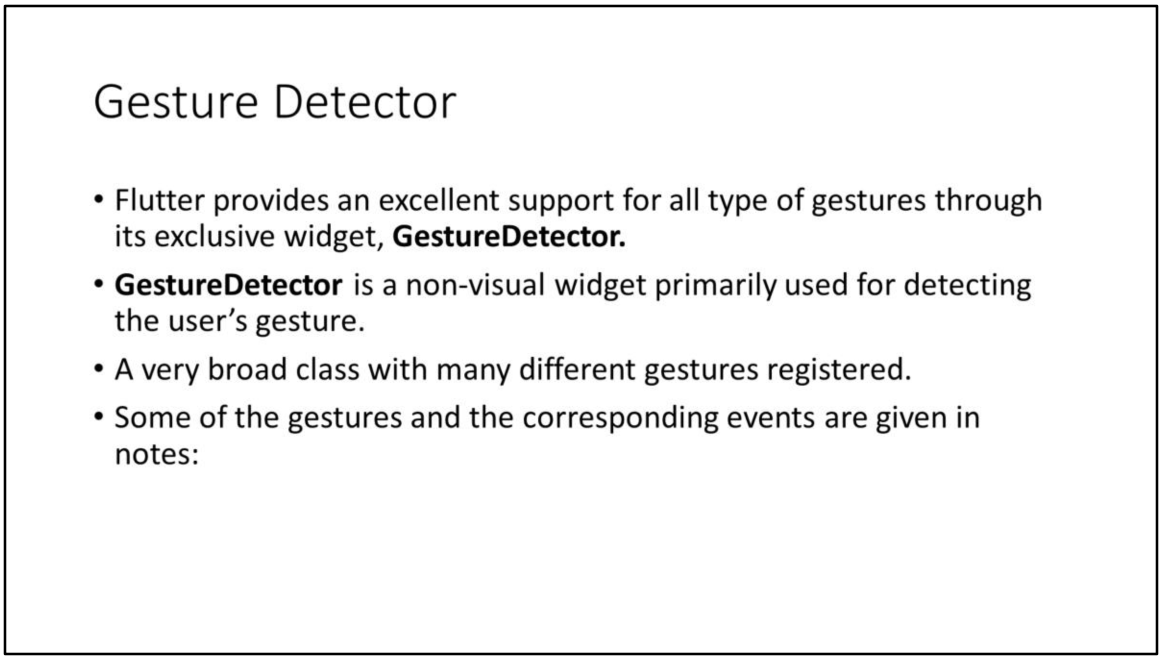
Flutter – Event Handling

Flutter Gestures

- 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.
  - **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 the fingertip



Gesture Detector

- Flutter provides an excellent support for all type of gestures through its exclusive widget, **GestureDetector.**
- **GestureDetector** is a non-visual widget primarily used for detecting the user's gesture.
- A very broad class with many different gestures registered.
- Some of the gestures and the corresponding events are given in notes:

- Tap
  - onTapDown
  - onTapUp
  - onTap
  - onTapCancel

- Double tap
  - onDoubleTap

- Long press
  - onLongPress

- Vertical drag
  - onVerticalDragStart
  - onVerticalDragUpdate
  - onVerticalDragEnd

- Horizontal drag
  - onHorizontalDragStart

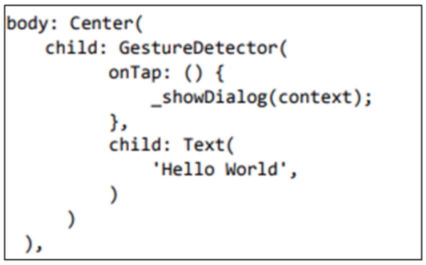
    - onHorizontalDragUpdate
    - onHorizontalDragEnd

- Pan
    - onPanStart
    - onPanUpdate
    - onPanEnd

Inkwell

- A rectangular area of a Material that responds to touch.
- Same as GestureDetector, but it shows ripple effect which is not provided by GestureDetector.
  - onTap
  - onPress
  - etc

Sample Code

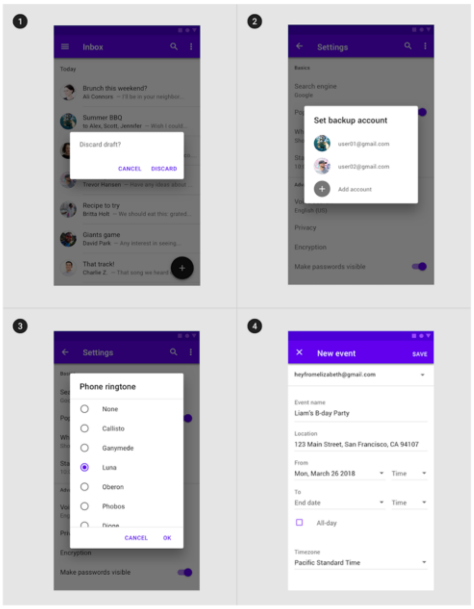


```
body: Center(
    child: GestureDetector(
          onTap: () {
              _showDialog(context);
          },
          child: Text(
              'Hello World',
          )
      )
  ),
```

- _showDialog is method, just being called inside onTap

Dialogs

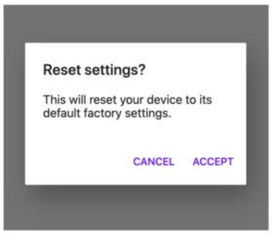
- A dialog is a type of modal window that appears in front of app content to provide critical information or ask for a decision. Dialogs disable all app functionality when they appear, and remain on screen until confirmed, dismissed, or a required action has been taken.
- There are four types of dialogs:
  - 1. Alert,
  - 2. Simple,
  - 3. Confirmation,
  - 4. Full-screen



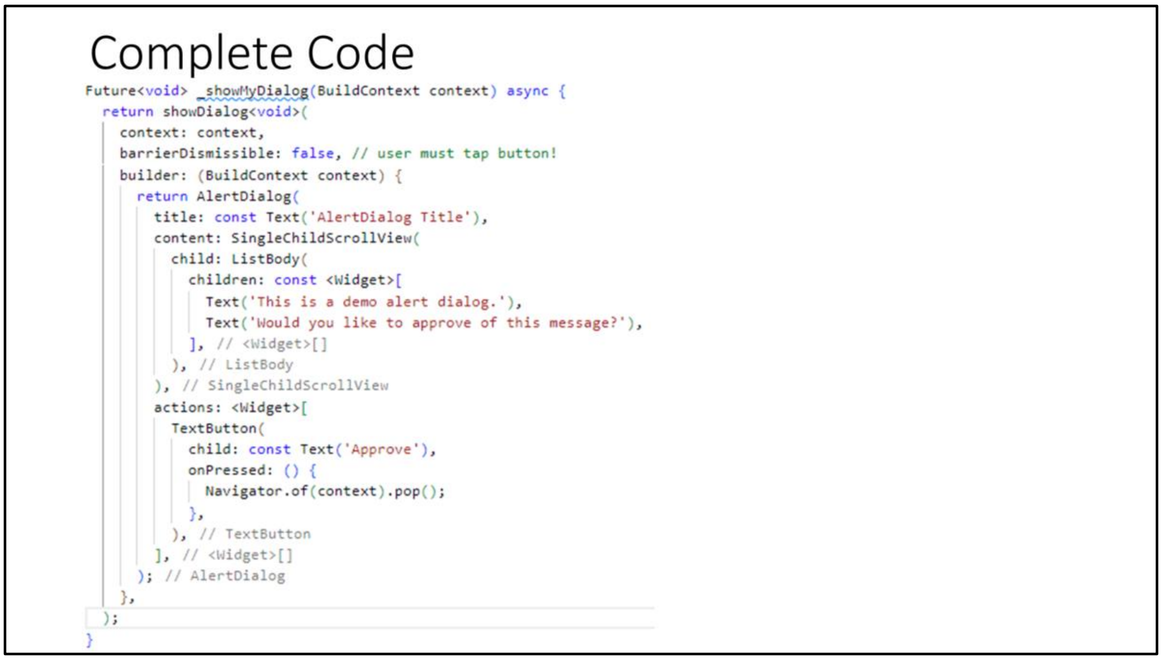
Alert dialog

- Alert dialogs interrupt users with urgent information, details, or actions.

```
AlertDialog(
  title: Text('Reset settings?'),
  content: Text('This will reset your device to its default factory settings.'),
  actions: [
    FlatButton(
      textColor: Color(0xFF6200EE),
      onPressed: () {},
      child: Text('CANCEL'),
    ),
    FlatButton(
      textColor: Color(0xFF6200EE),
      onPressed: () {},
      child: Text('ACCEPT'),
    ),
  ],
)
```


Reset settings?

This will reset your device to its default factory settings.



Complete Code

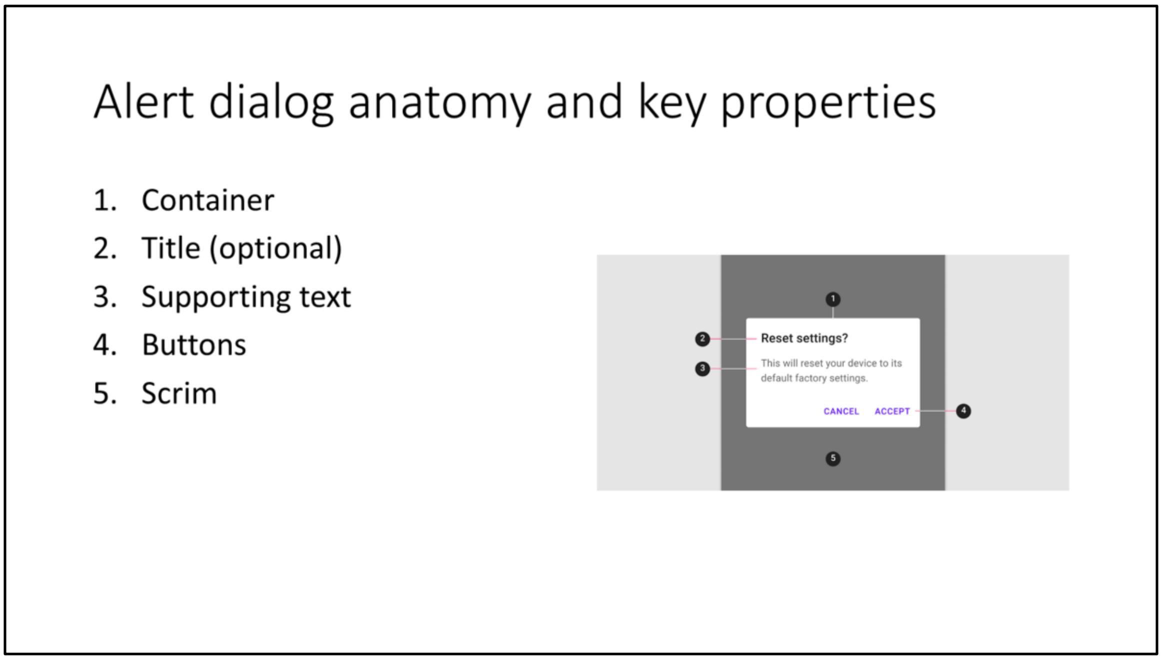
```
Future<void> _showMyDialog(BuildContext context) async {
  return showDialog<void>(
    context: context,
    barrierDismissible: false, // user must tap button!
    builder: (BuildContext context) {
      return AlertDialog(
        title: const Text('AlertDialog Title'),
        content: SingleChildScrollView(
          child: ListBody(
            children: const <Widget>[
              Text('This is a demo alert dialog.'),
              Text('Would you like to approve of this message?'),
            ], // <Widget>[]
          ), // ListBody
        ), // SingleChildScrollView
        actions: <Widget>[
          TextButton(
            child: const Text('Approve'),
            onPressed: () {
              Navigator.of(context).pop();
            },
          ), // TextButton
        ], // <Widget>[]
      ); // AlertDialog
    },
  );
}
```

```
Future<void> _showMyDialog(BuildContext context) async {
  return showDialog<void>(
    context: context,
    barrierDismissible: false, // user must tap button!
    builder: (BuildContext context) {
      return AlertDialog(
        title: const Text('AlertDialog Title'),
        content: SingleChildScrollView(
          child: ListBody(
            children: const <Widget>[
              Text('This is a demo alert dialog.'),
              Text('Would you like to approve of this message?'),
            ],
          ),
        ),
        actions: <Widget>[
          TextButton(
            child: const Text('Approve'),
            onPressed: () {
              Navigator.of(context).pop();
            },
          ),
        ],
      );
    },
  );
}
```

Long-running tasks are common in mobile apps. The way this is handled in Flutter / Dart is by using a Future. A Future allows you to run work asynchronously to free up any other threads that should not be blocked. Like the UI thread.

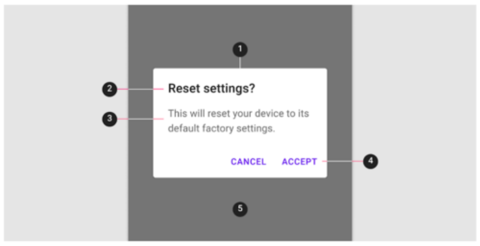
Define a Future

A future is defined exactly like a function in dart, but instead of void you use Future. If you want to return a value from the Future then you pass it a type



Alert dialog anatomy and key properties

1. Container
2. Title (optional)
3. Supporting text
4. Buttons
5. Scrim



Reset settings?

This will reset your device to its default factory settings.

Container attributes
PropertiesColor : backgroundColor
Shape : shape
Elevation : elevation

Title attributes
PropertiesText label : title
Color : style on title when using a Text
Typography : style on title when using a Text

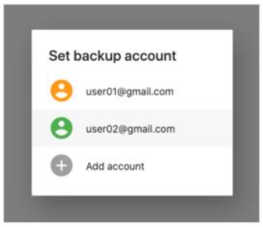
Supporting text attributes
PropertiesText label : content
Color : style on content when using a Text
Typography : style on content when using a Text

Buttons attributes
PropertiesButtons : actions

Simple dialog

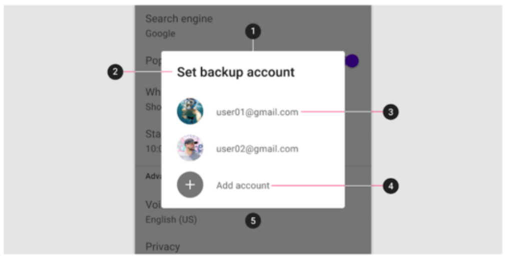
- Simple dialogs can display items that are immediately actionable when selected. They don’t have text buttons

```
SimpleDialog(
  title: Text('Set backup account'),
  children: [
    SimpleDialogItem(
      icon: Icons.account_circle,
      color: Colors.orange,
      text: 'user01@gmail.com',
      onPressed: () {
        Navigator.pop(context, 'user01@gmail.com');
      },
    ),
    SimpleDialogItem(
      icon: Icons.account_circle,
      color: Colors.green,
      text: 'user02@gmail.com',
      onPressed: () {
        Navigator.pop(context, 'user02@gmail.com');
      },
    ),
    SimpleDialogItem(
      icon: Icons.add_circle,
      color: Colors.grey,
      text: 'Add account',
      onPressed: () {
        Navigator.pop(context, 'user02@gmail.com');
      },
    ),
  ],
)
```



Simple dialog anatomy and key properties

- Container
- Title
- List item
    - Supporting visual
    - Primary text
- Button
- Scrim



Container attributes
PropertiesColor : backgroundColor
Shape : shape
Elevation : elevation

Title attributes
PropertiesText label : title
Color : style on title when using a Text
Typography : style on title when using a Text

List item supporting visual attributes
PropertiesOptions : children (Use SimpleDialogOption and customize its child parameter as needed.)

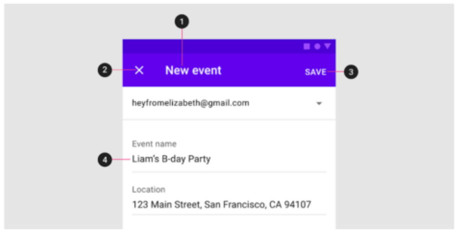
Confirmation dialog

- Confirmation dialogs give users the ability to provide final confirmation of a choice before committing to it, so they have a chance to change their minds if necessary.
- If the user confirms a choice, it’s carried out. Otherwise, the user can dismiss the dialog. For example, users can listen to multiple ringtones but only make a final selection upon tapping “OK.”

NOTE: There is no explicit confirmation dialog in Flutter but this can be built using the `Dialog` widget as a blank slate and providing your own custom `child`.

Full-screen dialog

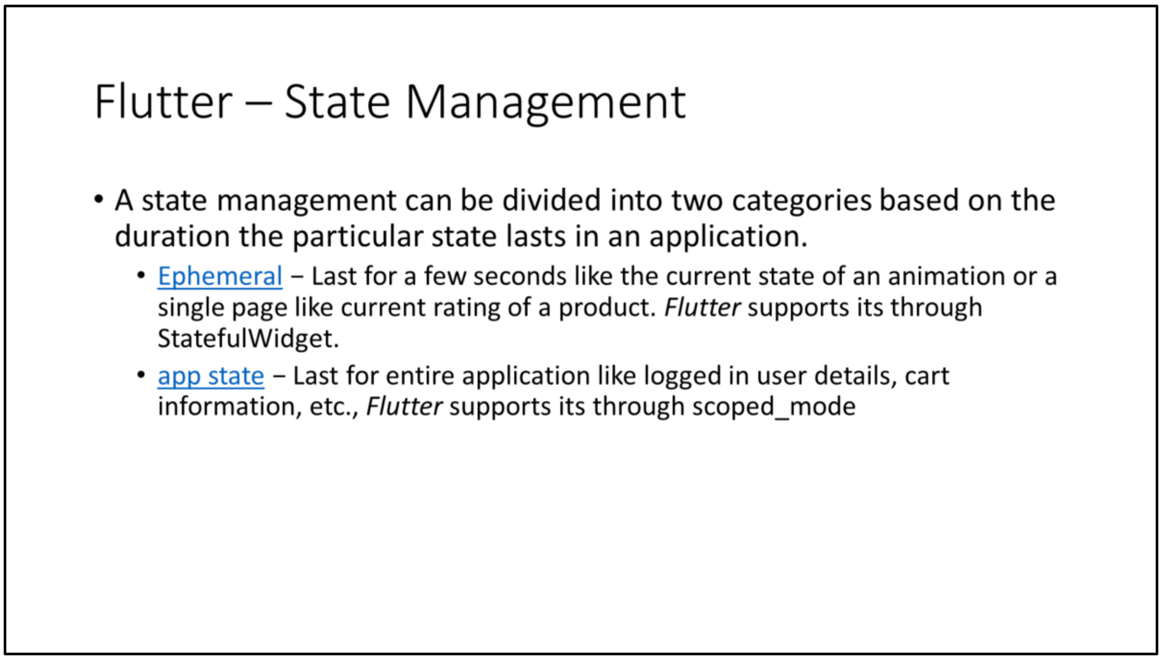
- To use a full-screen dialog, simply set the **fullscreenDialog** to true when pushing a new **MaterialPageRoute**
- **Full-screen dialog anatomy**
  - Title
  - Icon Button
  - Buttons
  - Scrim


1
2
New event
SAVE
3
heyfromelizabeth@gmail.com
Event name
4
Liam's B-day Party
Location
123 Main Street, San Francisco, CA 94107

Flutter – State Management

Flutter – State Management

- Managing state in an application is one of the most important and necessary process in the
- Let us consider a simple shopping cart application.
  - User will login using their credentials into the application.
  - Once user is logged in, the application should persist the logged in user detail in all the screen.
  - Again, when the user selects a product and saved into a cart, the cart information should persist between the pages until the user checked out the cart.
  - User and their cart information at any instance is called the state of the application at that instance. life cycle of an application.

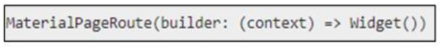


Flutter – State Management

- A state management can be divided into two categories based on the duration the particular state lasts in an application.
    - Ephemeral – Last for a few seconds like the current state of an animation or a single page like current rating of a product. *Flutter* supports its through StatefulWidget.
    - app state – Last for entire application like logged in user details, cart information, etc., *Flutter* supports its through scoped_mode

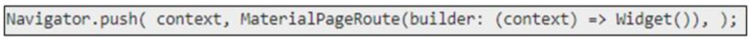
Navigation and Routing

- In any application, navigating from one page / screen to another defines the work flow of the application.
- MaterialPageRoute is a widget used to render its UI by replacing the entire screen with a platform specific animation
- Syntax



```
MaterialPageRoute(builder: (context) => Widget())
```

- Navigator.push is used to navigate to new screen using MaterialPageRoute widget



```
Navigator.push( context, MaterialPageRoute(builder: (context) => Widget()), );
```

Ephemeral State Management

- Since Flutter application is composed of widgets, the state management is also done by widgets
- The entry point of the state management is Statefulwidget.

```
class RatingBox extends StatefulWidget {
}
```

- Create a state for RatingBox, _RatingBoxState by inheriting State

```
class _RatingBoxState extends State<RatingBox> {
 }
```

- Override the createState of StatefulWidget method to create the state, _RatingBoxState

```
class RatingBox extends StatefulWidget {
  @override
  _RatingBoxState createState() => _RatingBoxState();
}
```