Flutter Basics Course
Day 2

Course Overview (1/2)

1. Flutter Quick Theory
2. Setting up the Envt.
3. Dart Basics
4. Flutter App Structure
5. Flutter Widgets 101
6. Lists and Scroll Views
7. Basic State Management

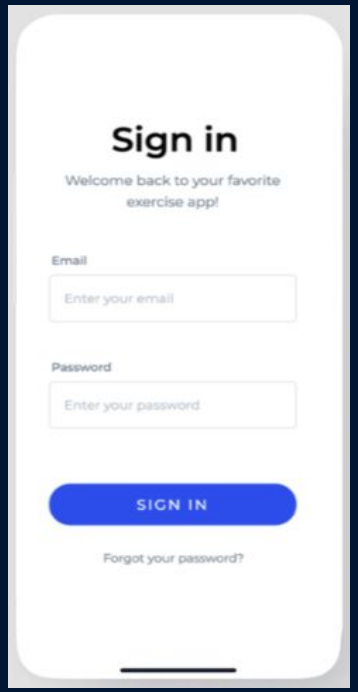
Course Overview (2/2)

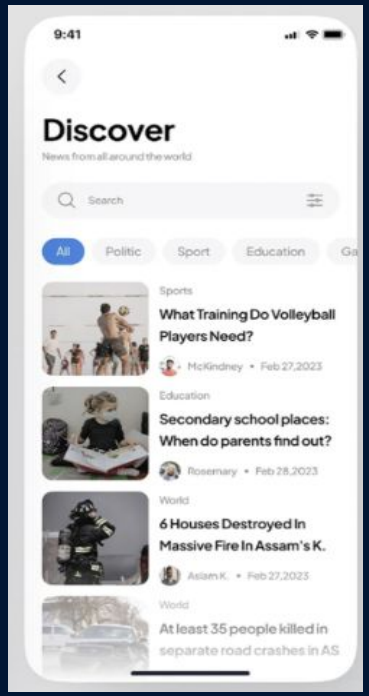
8. Navigation
9. Theming & Styling
10. Basic Form Handling
11. Network and Data
12. Advanced App State Management

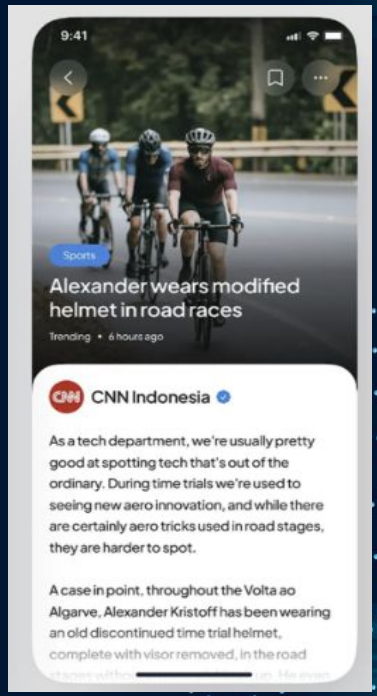
Day 2 - Agenda

Time	Agenda
8:00 am - 9:00 am	Coffee Break
9:00 am - 10:00 am	Flutter Widgets 101 - Hands-On
10:00 am - 10:45 am	Breakfast Break
10:45 am - 12:30 pm	Widgets 101 (Continuation), Basics,State Management & Navigation
12:30 pm - 2:00 pm	Lunch
2:00 pm - 3:00 pm	Lists and Scroll Views, Basic Form Handling
3:00 pm - 3:15 pm	FAQ & Break
3:15 pm - 4:00 pm	Theming and Styling

Your Challenge for this Course







Widgets 101:
Hands- On

Flutter Widgets 101 - Layout & Structure

Widget	Purpose
`Container`	Basic box with padding, margin, color, etc.
`Column`	Layout children vertically
`Row`	Layout children horizontally
`Stack`	Overlap widgets on top of each other
`Expanded`	Fills remaining space in `Row` / `Column`
`SizedBox`	Adds fixed space or dimensions
`Padding`	Adds padding around child
`Align`	Aligns a child within a parent
`Center`	Centers a child widget

Flutter Widgets 101 - Layout & Structure

Widget	Purpose
`Container`	Basic box with padding, margin, color, etc.
`Column`	Layout children vertically
`Row`	Layout children horizontally
`Stack`	Overlap widgets on top of each other
`Expanded`	Fills remaining space in `Row` / `Column`
`SizedBox`	Adds fixed space or dimensions
`Padding`	Adds padding around child
`Align`	Aligns a child within a parent
`Center`	Centers a child widget

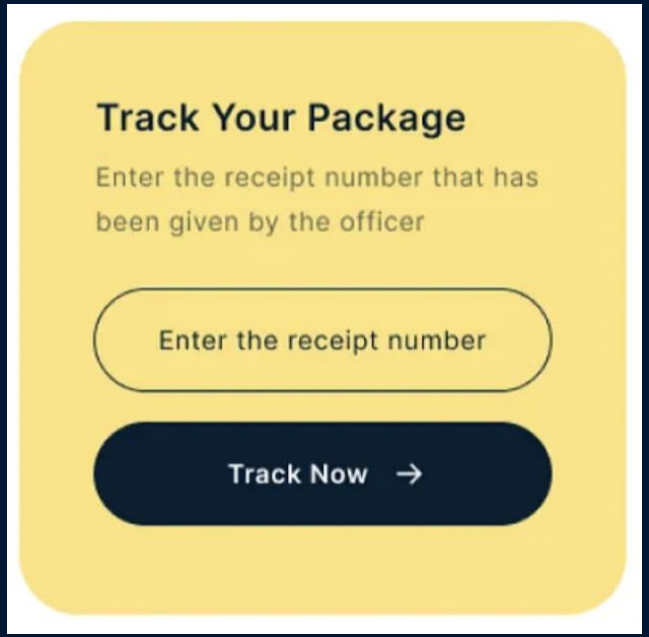
Flutter Widgets 101 - Visual & Styling

Widget	Purpose
`Text`	Displays a string of text
`Image`	Displays images from assets or network
`Icon`	Displays a material design icon
`Card`	Material-style panel
`Container`	(Again) Also used for decoration
`DecoratedBox`	Applies decorations like borders and shapes

Flutter Widgets 101 - Navigation & Routing

Widget	Purpose
`Navigator`	Manages a stack of routes (pages/screens)
`MaterialApp`	Top-level app wrapper with routing config
`Scaffold`	Standard screen structure (app bar, body)
`AppBar`	Top app bar with title and actions
`Drawer`	Side navigation panel
`BottomNavigationBar`	Tabs at bottom of screen

Flutter Widgets 101 - Practice UIs



Lists and Scroll Views

Lists and Scroll Views

1. **ListView Types:**

   ListView is the most commonly used scrolling widget to display a linear array of widgets.

   - **ListView() :** Good for small lists.
   - **ListView.builder() :** Builds items on demand (lazily). Good for long or infinite lists.
   - **ListView.separated() :** Similar to **builder,** but allows a separator widget between items.

Basic ListView:

```
ListView(
  children: <Widget>[
    Text('Item 1'),
    Text('Item 2'),
    Text('Item 3'),
  ],
);
```

Lazily ListView:

```
ListView.builder(
  itemCount: 100,
  itemBuilder: (context, index) {
    return ListTile(title: Text('Item $index'));
  },
);
```

Separated ListView:

```
ListView.separated(
  itemCount: 10,
  itemBuilder: (context, index) =>
ListTile(title: Text('Item $index')),
  separatorBuilder: (context, index) =>
Divider(),
);
```

Lists and Scroll Views

2. **ScrollView Types:**

Scroll View is a widget that enables its child (or children) to be scrollable when the content exceeds the available screen space.

- **SingleChildScrollView()** : For scrollable content with a single child vertically or horizontally.
- **CustomScrollView()** : Allows combining multiple slivers (scrollable areas) like SliverList, **SliverAppBar.**
- **GridView()** : It displays widgets in a scrollable grid format. You can control the number of columns, spacing, and layout behavior.

SingleChildScrollView:

```
SingleChildScrollView(
  child: Column(
    children: List.generate(20, (index) => Text('Item
$index')),
  ),
);
```

GridView:

Variants include:

- `GridView.count()`
- `GridView.builder()`
- `GridView.custom()`

CustomScrollView:

```
CustomScrollView(
  slivers: [
    SliverAppBar(title: Text('Header')),
    SliverList(
      delegate: SliverChildBuilderDelegate(
        (context, index) => ListTile(title: Text('Item
$index')),
        childCount: 30,
      ),
```

Lists and Scroll Views

Use Case	Widget
Small static list	`ListView()`
Long/infinite list	`ListView.builder()`
List with separators	`ListView.separated()`
Scrollable single child	`SingleChildScrollView`
Mixed scrollable content	`CustomScrollView`
Grid layout	`GridView`

Basic State Management

Basic App State Management

It refers to the techniques used to handle and update UI in response to user interactions or data changes. Flutter offers several ways to manage state, the most common methods are:

1. **Ephemeral state:** The (sometimes called *UI state* or *local state*) is the state you can neatly contain in a single widget.

   setState() (Built-in, Local State Management)

2. **App State: (**Shall be discussed in the Advanced State Management Topic).

Learn more at: **https://docs.flutter.dev/data-and-backend/state-mgmt/ephemeral-vs-app**

Navigation

Navigation

Navigation in Flutter allows to move between different screens (called "routes" in Flutter). There are several ways to handle navigation, depending on the complexity of an app:

1. **Basic Navigation** Using **Navigator**:

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

Navigator.pop(context); // to go back/close current screen.
```

Navigation

2. Named Routes: Useful for managing multiple screens more cleanly.

```
// Define routes in MaterialApp

MaterialApp(

 initialRoute: '/', routes: {'/': (context) => HomeScreen(), '/second': (context) => SecondScreen(),},

);

// Navigate using route name

Navigator.pushNamed(context, '/second');
```

Navigation

3. Passing Data Between Screens

```
Navigator.push(
 context,
 MaterialPageRoute(
   builder: (context) => PassingScreen(data: 'Hello'),
 ),
);
```

Navigation

4. Return Data from a Screen

```
// returning screen
Navigator.pop(context, 'Returned Data');
// first screen
final result = await Navigator.push(
 context,
 MaterialPageRoute(builder: (context) => SecondScreen()),
);
print(result);
```

Basic Form Handling

Basic Form Handling

Form handling in Flutter involves managing user input using Form, TextFormField, and GlobalKey<FormState>. Here's a complete breakdown of basic form handling in Flutter:

1. **Form Widget Basics:** The Form widget groups and manages multiple form fields. You need a GlobalKey<FormState> to validate and save the form.
2. **TextFormField Widget:** A TextFormField combines a TextField with built-in validation.
3. **Handling Submission:** Use _formKey.currentState!.validate() to check if the form is valid.

```
class MyForm extends StatefulWidget {
  @override
  _MyFormState createState() => _MyFormState();
}

class _MyFormState extends State<MyForm> {
  final _formKey = GlobalKey<FormState>();
  String _email = '';
  String _password = '';

  void _submitForm() {
    if (_formKey.currentState!.validate()) {
      _formKey.currentState!.save();
      print('Email: $_email, Password: $_password');
    }
  }
```

```
  @override
  Widget build(BuildContext context) {
    return Form(
      key: _formKey,
      child: Column(
        children: [
          TextFormField(
            decoration: InputDecoration(labelText:
'Email'),
            validator: (value) =>
                value == null || value.isEmpty ? 'Enter
email' : null,
            onSaved: (value) => _email = value ?? '',
          ),
          TextFormField(
            decoration: InputDecoration(labelText:
'Password'),
            obscureText: true,
            validator: (value) =>
                value == null || value.length < 6 ? 'Min 6
chars' : null,
            onSaved: (value) => _password = value ?? '',
          ),
          ElevatedButton(
            onPressed: _submitForm,
            child: Text('Submit'),
          ),
        ],
      ),
    );
  }
}
```

Basic Form Handling

Concept	Description
`Form`	Groups form fields and manages validation/saving
`TextFormField`	Form field with built-in validation and saving
`GlobalKey<FormState>`	Controls form state for validation/saving
`validator`	Validates input
`onSaved`	Called when form is saved
`TextEditingController`	Optional: for manual control of field values

Theming and Styling

Theming & Styling

Theming allows you to define **global styles** for your app. This includes fonts, colors, button styles, etc., and applies them consistently.

1. Applying Styling to Text.
2. Application of Theme.
3. Light & Dark Themes.

Learn more at: **https://docs.flutter.dev/data-and-backend/state-mgmt/ephemeral-vs-app**

Install Google Fonts Package:
https://pub.dev/packages/google_fonts

Reference:
https://docs.flutter.dev/cookbook/design/themes

Day 3 - Agenda

Time	Agenda
8:00 am - 9:00 am	Coffee Break
9:00 am - 10:00 am	Networking and Data
10:00 am - 11:00 am	Breakfast Break
11:00 am - 12:30 pm	Advanced State Management
12:30 pm - 2:00 pm	Lunch
2:00 pm - 3:00 pm	Building a Basic News App - (Challenge)
3:00 pm - 3:15 pm	FAQ & Break
3:15 pm - 4:00 pm	Building a Basic News App - (Challenge)

Networking & Data

Networking & Data

1. API Call (Use of http package): Retrieving data from the internet through API calls.
2. Parsing JSON Data also called Serialization - Converting Dart objects in JSON format
3. Use Local Data Storage options below to save objects (Optional):
    a. shared_preferences – for simple key-value storage
    b. sqflite – for full SQLite database support
    c. hive – fast, lightweight, NoSQL-like solution

Install Http Package:

https://pub.dev/packages/http

To convert JSON to dart objects:

https://javiercbk.github.io/json_to_dart/

Install Shared Preferences Package:

https://pub.dev/packages/shared_preferences

Example Code Snippet for API (GET):

```
Future<void> fetchData() async {

        final response = await
       http.get(Uri.parse('https://jsonplaceholder.typicode.com/po
       sts/1'));

        if (response.statusCode == 200) {

          var data = jsonDecode(response.body);

          print(data);

        } else {

          throw Exception('Failed to load data');

        }

}
```

Advance App Statement Management

Networking & Data

App State: Shared across multiple widgets or the entire app.Handled using more robust state management solutions such as.

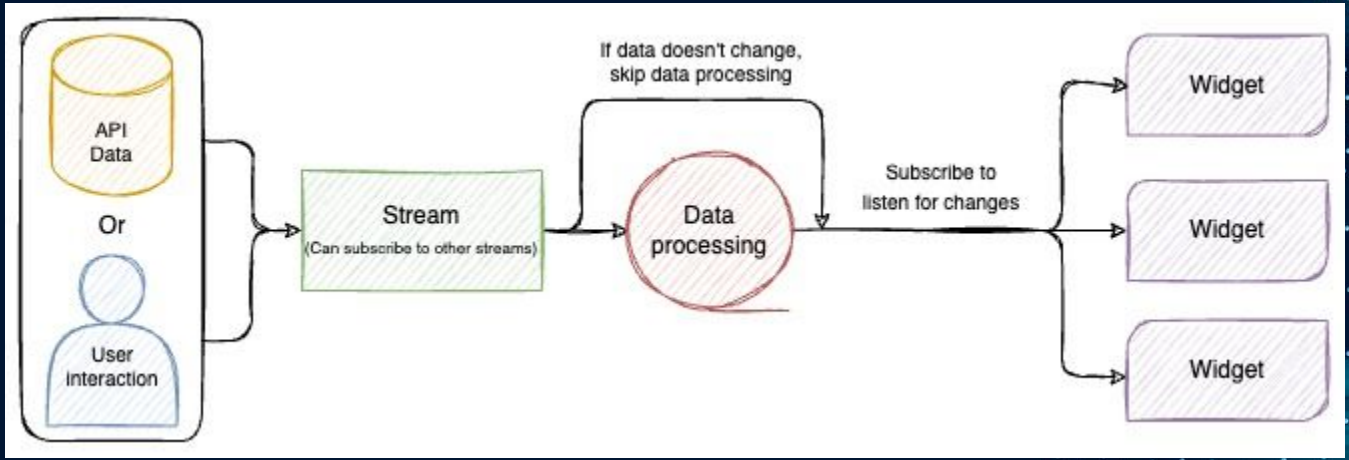
- **Provider**
- **Riverpod**
- **Bloc**
- **GetX**
- **Redux, etc.**

Learn more at: **https://docs.flutter.dev/data-and-backend/state-mgmt/ephemeral-vs-app**

Networking & Data

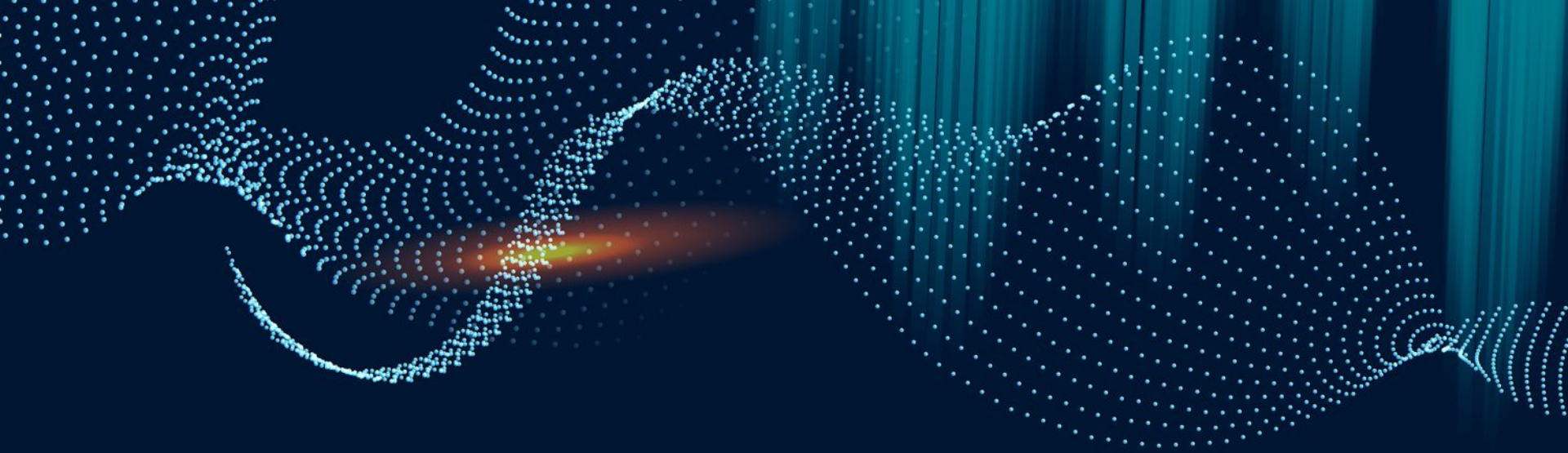
- **GetX is Reactive**

  In GetX, you define **reactive variables** (using .obs) that automatically update the UI whenever their values change. You don't need to manually call anything to trigger the rebuild—**GetX listens and reacts**.



Install Getx Package:

https://pub.dev/packages/get



THANK YOU!