UNIT – 4

Flutter Routing & Working with SQLite Database

Navigation & Routing

➢ Navigation and routing are some of the core concepts of all mobile application, which allows the user to move between different pages. We know that every mobile application contains several screens for displaying different types of information. For example, an app can have a screen that contains various products. When the user taps on that product, immediately it will display detailed information about that product

Step 1: First, you need to create two routes.

Step 2: Then, navigate to one route from another route by using the Navigator.push() method.

Step 3: Finally, navigate to the first route by using the Navigator.pop() method.

Flutter Packages

➢ A package is a namespace that contains a group of similar types of classes, interfaces, and sub-packages. We can think of packages as similar to different folders on our computers where we might keep movies in one folder, images in another folder, software in another folder, etc. In Flutter, Dart organizes and shares a set of functionality through a package. Flutter always supports shared packages, which is contributed by other developers to the Flutter and Dart ecosystem. The packages allow us to build the app without having to develop everything from scratch.

Types of Packages

According to the functionality, we can categorize the package into two types:

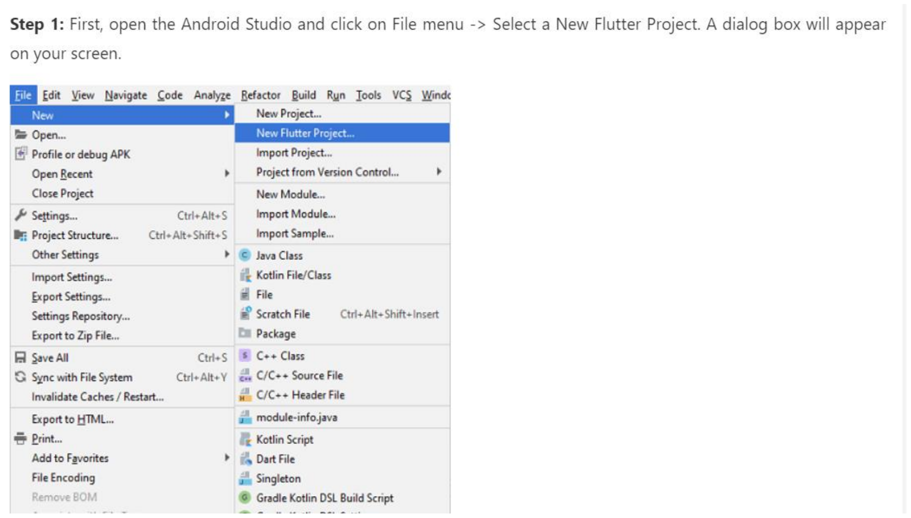
- Dart Package
- Plugin Package

Dart Package: It is a general package, which is written in the dart language, such as a path package. This package can be used in both the environment, either it is a web or mobile platform. It also contains some Flutter specific functionality and thus has a dependency on the Flutter framework, such as fluro package.

Plugin Package: It is a specialized Dart package that includes an API written in Dart code and depends on the Flutter framework. It can be combined with a platform-specific implementation for an underlying platform such as Android (using Java or Kotlin), and iOS (using Objective C or Swift). The example of this package is the battery and image picker plugin package.

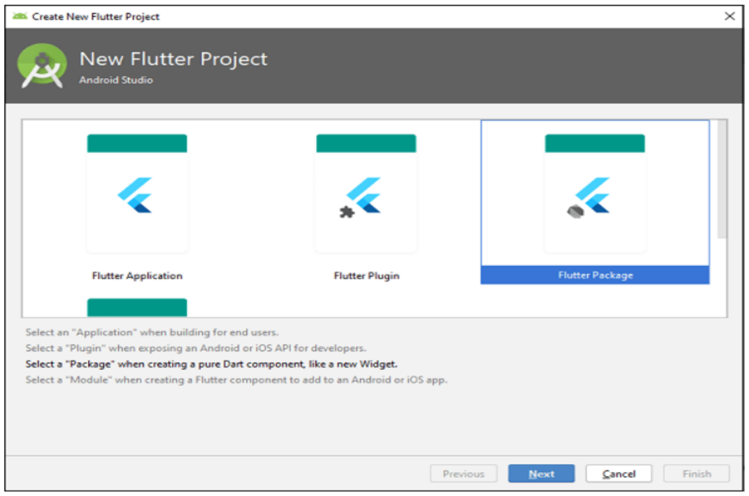
Develop a Flutter Package or Plugin

- Developing a Flutter plugin or package is similar to creating a Dart application or Dart package. But, it has some exceptions means plugin always uses system API specific to a platform such as Android or iOS to get the required functionality. Now, let us see step by step how to develop a package in Flutter.



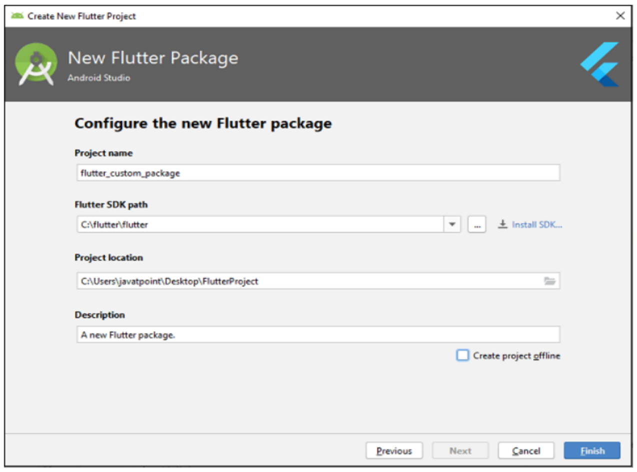
Step 1: First, open the Android Studio and click on File menu -> Select a New Flutter Project. A dialog box will appear on your screen.

Step 2: In this dialog box, you need to select a New Flutter Project option, as shown in the image below, and click on **Next**.



Select an "Application" when building for end users.
Select a "Plugin" when exposing an Android or iOS API for developers.
Select a "Package" when creating a pure Dart component, like a new Widget.
Select a "Module" when creating a Flutter component to add to an Android or iOS app.

Step 3: In the next dialog box, enter all the details of your package, such as project name, location of your project, and project description. After filling all the details, click on Finish.



Configure the new Flutter package

Project name: flutter_custom_package

Flutter SDK path: C:\flutter\flutter

Project location: C:\Users\javatpoint\Desktop\FlutterProject

Description: A new Flutter package.

Create project offline

__Step 4:__ Finally, your project is created. Now, open the flutter_custom_package.dart file and delete the default code created at the time of project creation. Then insert the following code. This code snippet creates an alert box package.

```
library flutter_custom_package;

import 'package:flutter/material.dart';

class CustomPackageAlertBox {
  static Future showCustomAlertBox({
    @required BuildContext context,
    @required Widget willDisplayWidget,
  }) {
    assert(context != null, "If context is null!!");
    assert(willDisplayWidget != null, "If willDisplayWidget is null!!");
    return showDialog(
        context: context,
        builder: (context) {
          return AlertDialog(
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.all(Radius.circular(20)),
            ),
            content: Column(
              mainAxisSize: MainAxisSize.min,
              children: <Widget>[
                willDisplayWidget,
                MaterialButton(
                  color: Colors.white70,
                  child: Text('Close Alert'),
                  onPressed: () {
                    Navigator.of(context).pop();
                  },
                )
              ],
            ),
            elevation: 12,
          );
        });
  }
}
```

Now, you need to test your newly created package. To test the package, create a new project. In this project, first, open the pubspec.yaml file and the following code in the dependency section.

```
dependencies:
   flutter:
      sdk: flutter
   flutter_custom_package:
      path: ../
```

```
import 'package: flutter_custom_package/flutter_custom_package.dart';
```

How to Publish a Package

When you have successfully implemented a package, you can publish it on the pub.dev, so that anyone can use it easily in the project.

Before publishing the package, make sure that the content of pubspec.yaml, README.md, and CHANGELOG.md file is complete and correct.

Next, run the following command in the terminal window to analyze every phase of the package.

```
$ flutter pub publish --dry-run
```

Finally, you need to run the following command to publish the package.

```
$ flutter pub publish
```

Flutter Google Maps

- A map is used to get information about the world simply and visually. It presents the world places by showing its shape and sizes, locations and distance between them. We can add a map in our application with the use of the Google Maps Flutter plugin. This plugin can automatically access the Google Maps servers, map display, and respond to user gestures. It also allows us to add markers to our map.

Why use Google Maps with Flutter?

- Flutter developers prefer Google Maps for their application because they provide native performance for android and iOS both. It allows us to implement the code one time and permit them to run the code for both devices (android and iOS). Google Maps Flutter plugin is provided in the Google Map widget that supports initialCameraPosition, maptype and onMapCreated. We can set the position of the camera and marker in any place on the earth. We can design the marker according to our choice. It also comes with a zoom property in a cameraposition to provide the zooming in google map view on the initial page.

  Let us see step by step to how to add Google Maps in Flutter application.

Step 1:

- Create a new project. Open this project in the IDE, navigate to the lib folder, and then open the pubspec.yaml file for setting the map.

Step 2:

- In pubspec.yaml file, we need to add the Google Maps Flutter plugin in the dependency section, which is available as google_maps_flutter on pub.dartlang.org. After adding a dependency, click on the get package link to import the library in the main.dart file.

```
  dependencies:
 flutter:
sdk: flutter
cupertino_icons: ^0.1.2
google_maps_flutter: ^0.5.21
```

It ensures that we have left two spaces from the left side of a google_maps_flutter dependency while adding the dependencies.

Step 3:

- The next step is to get an API key for your project. If we are using an Android platform, then follow the instructions given on Maps SDK for Android: Get API Key. After creating the API key, add it to the application manifest file. We can find this file by navigating to android/app/src/main/AndroidManifest.xml as follows:

```
<manifest ...
  <application ...
    <meta-data android:name="com.google.android.geo.API_KEY"
               android:value="YOUR ANDROID API KEY HERE"/>
```

Step 4: Next, import the package in the dart file as below:

```
import 'package:google_maps_flutter/google_maps_flutter.dart';
```

Step 5: Now, we are ready to add a GoogleMap widget to start creating a UI to display the map.

```
import 'package:flutter/material.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';

void main() => runApp(MyApp());

class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  GoogleMapController myController;

  final LatLng _center = const LatLng(45.521563, -122.677433);

  void _onMapCreated(GoogleMapController controller) {
    myController = controller;
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Flutter Maps Demo'),
          backgroundColor: Colors.green,
        ),
        body: Stack(
          children: <Widget>[
            GoogleMap(
              onMapCreated: _onMapCreated,
              initialCameraPosition: CameraPosition(
                target: _center,
                zoom: 10.0,
              ),
            ),
            Padding(
              padding: const EdgeInsets.all(14.0),
              child: Align(
                alignment: Alignment.topRight,
                child: FloatingActionButton(
                  onPressed: () => print('You have pressed the button'),
                  materialTapTargetSize: MaterialTapTargetSize.padded,
                  backgroundColor: Colors.green,
                  child: const Icon(Icons.map, size: 30.0),
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

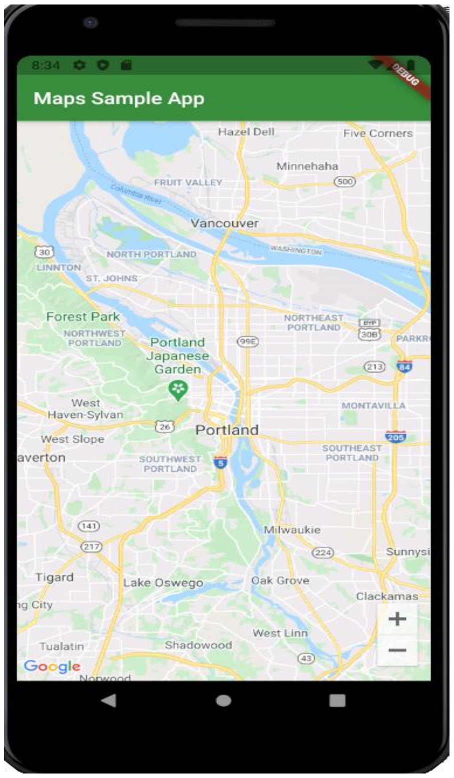
In the above code, we have noticed these terms:

mapController: It is similar to other controllers that we had seen in Flutter. It controls all activities on the GoogleMap class. Here, it manages the camera function, such as position, animation, zooming, etc.

onMapCreated: It is a method called for creating a map and takes a MapController as an argument.

initialCameraPosition: It is a required parameter that sets the camera position from where we want to start. It allows us to set which part of the world we want to point on the map.

stack: It is used to place other Flutter widgets on top of the map widget.



Working with SQLite Database

SQLite

- is another data storage available in Android where we can store data in the user's device and can use it any time when required. In this article, we will take a look at creating an SQLite database in Android app and adding data to that database in the Android app. This is a series of 4 articles in which we are going to perform the basic CRUD (Create, Read, Update, and Delete) operation with SQLite Database in Android.

What is SQLite Database?

- Android SQLite Database is an open-source database provided in Android that is used to store data inside the user's device in the form of a Text file. We can perform many operations on this data such as adding new data, updating, reading, and deleting this data. SQLite is an offline database that is locally stored in the user's device and we do not have to create any connection to connect to this database.

- **here are several reasons why you might choose to use SQLite in your project:**

  Ease of use:
  - SQLite is very easy to get started with, as it requires no setup or configuration. You can simply include the library in your project and start using it.

  Embeddability:
  - SQLite is designed to be embedded into other applications. It is a self-contained, serverless database engine, which means you can include it in your application without the need for a separate database server.

  Lightweight:
  - SQLite is a very lightweight database engine, with a small library size (typically less than 1MB). This makes it well-suited for use in applications where the database is embedded directly into the application binary, such as mobile apps.

  Serverless:
  - As mentioned earlier, SQLite is a serverless database engine, which means there is no need to set up and maintain a separate database server process. This makes it easy to deploy and manage, as there are no additional dependencies to worry about.

  Cross-platform:
  - SQLite is available on many platforms, including Linux, macOS, and Windows, making it a good choice for cross-platform development.

Standalone:

- SQLite stores all of the data in a single file on the filesystem, which makes it easy to copy or backup the database.

High reliability:

- SQLite has been widely tested and used in production systems for many years, and has a reputation for being a reliable and robust database engine.

How Data is Being Stored in the SQLite Database?

- Data is stored in the Android SQLite database in the form of tables. When we store this data in our SQLite database it is arranged in the form of tables that are similar to that of an Excel sheet. Below is the representation of our SQLite database which we are storing in our SQLite database.

Cursors and Content Values

- ContentValues objects are used to insert new rows into database tables (and Content Providers). Each Content Values object represents a single row, as a map of column names to values.
- Queries in Android are returned as Cursor objects. Rather than extracting and returning a copy of the result values, Cursors acts pointers to a subset of the underlying data. Cursors are a managed way of controlling your position (row) in the result set of databased query.

The Cursor class includes several functions to navigate query results including, but not limited to, the following:

- **moveToFirst** Moves the cursor to the first row in the query result.
- **moveToNext** Moves the cursor to the next row.
- **moveToPrevious** Moves the cursor to the previous row.
- **getCount** Returns the number of rows in the result set.
- **getColumnIndexOrThrow** Returns an index for the column with the specifi ed name (throwing an exception if no column exists with that name).
- **getColumnName** Returns the name of the specifi ed column index.
- GetColumnNames Returns a String array of all the column names in the current cursor.
- **moveToPosition** Moves the cursor to the specifi ed row.
- **getPosition** Returns the current cursor position.

Creating content provider

- In Flutter, a "provider" refers to a design pattern and a package that helps manage state in your application. It's commonly used to efficiently share and update data between different parts of your app, such as widgets, without the need for prop drilling (passing data through multiple widget layers).The "provider" package in Flutter provides a way to manage and consume data using a provider class, which acts as a source of truth for your app's state

❖ Step 1: Setting up the Project

- Assuming you have Flutter and Dart installed, start by creating a new Flutter project:

```
flutter create provider_example
cd provider_example
```

- Open the **pubspec.yaml** file of your project and add the following dependency under the dependencies section:

```
dependencies:
  flutter:
    sdk: flutter
  provider: ^5.0.0
```

Then, run flutter pub get to install the package.

❖ Step 2: Creating a Data Model

- Create a file named user.dart inside the lib folder of your project. This file will define a simple User data model:

```
class User {
  final String name;
  final int age;

  User(this.name, this.age);
}
```

❖ Step 3: Creating a Provider

- Create a file named user_provider.dart inside the lib folder. This file will define a provider class using the ChangeNotifier class from the provider package:

```
import 'package:flutter/material.dart';
import 'user.dart';

class UserProvider extends ChangeNotifier {
  User _user;

  User get user => _user;

  void updateUser(String name, int age) {
    _user = User(name, age);
    notifyListeners(); // Notify listeners that the data has changed
  }
}
```

❖ Step 4: Using the Provider in Widgets

➢ Create a file named main.dart inside the lib folder. This file will demonstrate how to use the UserProvider in your app:

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'user_provider.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider(
      create: (context) => UserProvider(),
      child: MaterialApp(
        title: 'Provider Example',
        home: HomeScreen(),
      ),
    );
  }
}
```

```
class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final userProvider = Provider.of<UserProvider>(context);

    return Scaffold(
      appBar: AppBar(
        title: Text('Provider Example'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text('Name: ${userProvider.user?.name ?? 'N/A'}'),
            Text('Age: ${userProvider.user?.age ?? 'N/A'}'),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () {
                userProvider.updateUser('Rahul', 25);
              },
              child: Text('Update User'),
            ),
          ],
        ),
      ),
    );
  }
}
```

❖ <u>Step 5: Running the App</u>

➢ Run the app using flutter run in your terminal. You should see the app with the "Name" and "Age" fields displayed as "N/A" initially. When you click the "Update User" button, the user's data should update accordingly.