

120A3051

Shreya Idate

Batch: E3

Experiment No: 4**AIM:** To create Flutter charts with charts_flutter**THEORY:****Flutter Charts:**

Charts within applications provide graphical displays or pictorial representations of data, spanning industries and apps. Mobile applications like Mint use pie charts to monitor spending habits and fitness apps like Strava use line charts and bar charts for analyzing pacing, heart rate, and elevation gain.

When building Flutter apps, developers can use the official charts_flutter library, maintained by Google, to create these types of charts.

Create and set up a Flutter project with charts_flutter:

1. To create a new Flutter project, run the following command: flutter create **projectName**
2. To import **chart_flutter** into your project, open the **pubspec.yaml** file and add it under dependencies:

```
dependencies:  
  flutter:  
    sdk: flutter  
  
  charts_flutter: ^0.12.0
```

Scaffolding the app

Now we have the basic code that comes with new Flutter apps: a counter that keeps a record of how many times a button is pushed.

Since we don't need that in our bar chart app, go ahead and delete that code found in the **main.dart** page. Delete everything except the following:

```
import 'package:flutter/material.dart';  
  
Run | Debug | Profile  
void main() {  
  runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
  const MyApp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp();  
  }  
}
```

Now, return the **MaterialApp** class in our build widget so we can use Material Design.

Department of IT, SIES GST

Creating a homepage

To create a homepage for our app, navigate into the `lib` folder and create a new page named `home.dart`:

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

class HomePage extends StatelessWidget{

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: , // Center
      ), // Scaffold
    );
  }
}
```

With `import 'package:flutter/material.dart'`, we can then import Material Design.

Then, the `HomePage` class extends the `statelessWidget`, as no states change on this page.

Inside the `BuildContext` widget, we return the `Scaffold` class to give us a basic Material Design layout structure. Our bar chart will go where the child parameter is, and we will center it on the body of our screen. All of this now serves as the scaffold for our app.

With the homepage complete, we can specify `HomePage` in our `main.dart` file since `main.dart` brings all the features in our app together. With this code, `main.dart` knows which page to show first whenever the app loads.

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

Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: HomePage(),
    ); // MaterialApp
  }
}
```

Creating a Flutter chart app

Series and models

The two terms commonly used with Flutter charts: series and models.

A series is a group (or series) of information that we can use to plot our chart. A model is the format our information comes in that specifies the attributes every data item using the model must have.

Creating a bar chart : Creating a model for bar chart data

To begin, we'll create a bar chart to show the number of new fictional Flutter chart developers that were added over the past five years. In other words, we want to track the growth the fictional Flutter chart community.

Our model, which defines the format of our data, consists of the year we are looking at, the number of developers that joined the Flutter chart community that year, and the color of the corresponding bar.

Inside the `lib` folder, create a file named `developer_series.dart`. Below, implement the code for our model:

```
import 'package:charts_flutter/flutter.dart' as charts;

class DeveloperSeries {
  final String year;
  final int developers;
  final charts.Color barColor;

  DeveloperSeries(
    {
      required this.year,
      required this.developers,
      required this.barColor
    }
  );
}
```

We named the model `DeveloperSeries` and specified the properties that each series item must have (`year`, `developers`, and `barColor`).

To prevent the parameter of a class from being null when creating an object of the class, we use the `@required` annotation, as seen in the code block above.

To use the `@required` keyword, we must import the `foundation.dart` package.

Creating data for a bar chart

Now that we have a model for our bar chart data, let's proceed to actually create some data. In the homepage, generate data for the bar chart by adding the following:

```
import 'package:flutter/material.dart';
import 'package:charts_flutter/flutter.dart' as charts;
import 'package:prj/developer_series.dart';
import 'package:prj/developer_chart.dart';

class HomePage extends StatelessWidget {
  final List<DeveloperSeries> data = [
    DeveloperSeries(
      year: "2017",
      developers: 40000,
      barColor: charts.ColorUtil.fromDartColor(Colors.indigo),
```

```
),
DeveloperSeries(
  year: "2018",
  developers: 5000,
  barColor: charts.ColorUtil.fromDartColor(Colors.indigo),
),
DeveloperSeries(
  year: "2019",
  developers: 40000,
  barColor: charts.ColorUtil.fromDartColor(Colors.indigo),
),
DeveloperSeries(
  year: "2020",
  developers: 35000,
  barColor: charts.ColorUtil.fromDartColor(Colors.indigo),
),
DeveloperSeries(
  year: "2021",
  developers: 45000,
  barColor: charts.ColorUtil.fromDartColor(Colors.indigo),
),
];

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(
      child: DeveloperChart(
        data: data,
      ),
    ),
  );
}
```

This is a simple list named `data`. Each item in the list is modeled after the `DeveloperSeries` model, meaning each item has a year (`year`), number of developers (`developers`), and bar color (`barColor`) property.

Building the bar chart

We've successfully created the data for our bar chart. Now, let's create the bar chart itself. To make our project organized, we'll put the code for our bar chart in a separate file.

Inside `lib`, create a `developer_chart.dart` file:

```
import 'package:flutter/material.dart';
import 'package:charts_flutter/flutter.dart' as charts;
import './developer_series.dart';

class DeveloperChart extends StatelessWidget {
  final List<DeveloperSeries> data;

  DeveloperChart({required this.data});
  @override
```

```
Widget build(BuildContext context) {  
  List<charts.Series<DeveloperSeries, String>> series = [  
    charts.Series(  
      id: "developers",  
      data: data,  
      domainFn: (DeveloperSeries series, _) => series.year,  
      measureFn: (DeveloperSeries series, _) => series.developers,  
      colorFn: (DeveloperSeries series, _) => series.barColor)  
    ];  
  
  return charts.BarChart(series, animate: true);  
}
```

With `final List<DeveloperSeries> data`, we defined a list called `data`, which is a `List` of data items in the form of our `DeveloperSeries` model we created earlier.

Every data item on the list comes with a corresponding year, number of developers, and bar color.

The `DeveloperChart` constructor inside the class ensures that everywhere the bar chart class is used, the data it requires is always provided; this is done using the `@required` keyword.

The actual bar chart is created inside our build widget. As you know, all bar charts have groups of data plotted against each other (in our case, the last five years and the number of developers the Flutter chart community gained).

Together, these groups of data are known as a series. The series tells us Flutter which group of data to put on the horizontal side and which group to put on the vertical side of our bar chart.

The list of data we created earlier then inserts into our series and used appropriately by Flutter.

With `List<charts.Series<DeveloperSeries, String>> series`, we created a list named `series`. This list has a type of `charts.Series`; `charts` imports Flutter into our project and the `Series` function creates series for a bar chart in Flutter.

The series we just created is modeled after our `DeveloperSeries` model.

The parameters we'll be specifying inside our series include `id`, `data`, `domainFn`, `measureFn`, and `colorFN`:

- `id` identifies the chart
- `data` points to the list of items to plot on the bar chart
- `domainFn` points to the values that will be on the horizontal side of the bar chart
- `measureFn` points to the quantity of the values on the vertical side
- `colorFN` refers to the color of the bars

With the `domainFn`, `measureFn`, and _____ functions, we create functions that take the _____ series

`colorFN``Subscriber`

as an argument, create instances of it, and then use the instances to access its different properties.

The underscores in the `developer_chart.dart` file signify that the second arguments are not required.

After pointing our series to all the data it requires, we then use it to create our bar chart using Flutter's `BarChart` function.

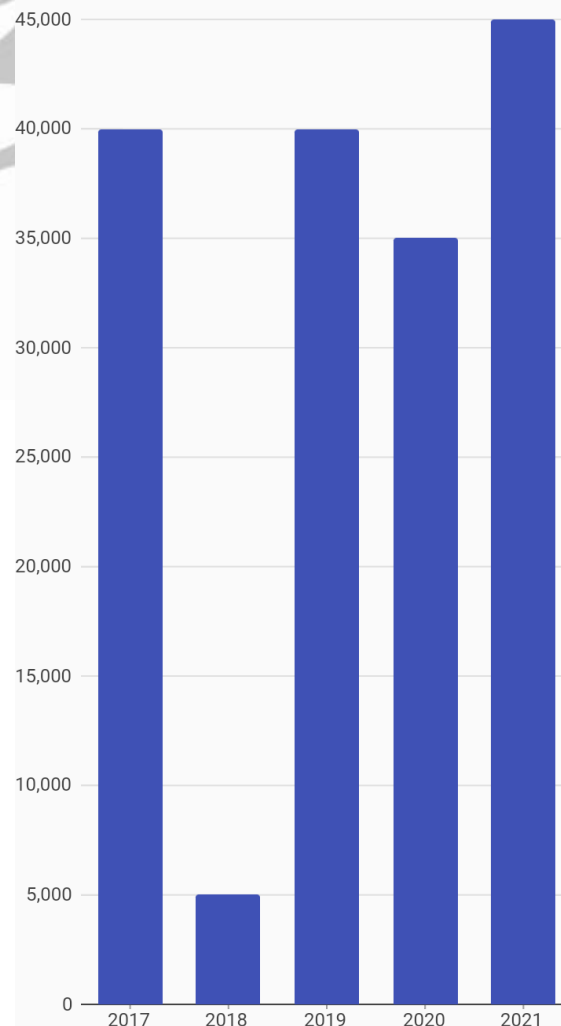
We can also add an animation for visual appeal by simply setting `animate` to `true`, which renders the chart with a nice animation.

Adding the bar chart to the homepage

Now, we can add our newly created bar chart to our homepage and display it:

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(
      child: DeveloperChart(
        data: data,
      ), // DeveloperChart
    ), // Center
  ); // Scaffold
}
```

Here, we simply call the `DeveloperChart` class inside the body of our page and point it to the data we want to use. To ensure our chart fits well on a screen, we'll put it in a `Card`, wrap a container around it, and give it a set height and some padding:



```

...

class DeveloperChart extends StatelessWidget {
  final List<DeveloperSeries> data;

  DeveloperChart({required this.data});
  @override
  Widget build(BuildContext context) {
    List<charts.Series<DeveloperSeries, String>> series = [
      charts.Series(
        id: "developers",
        data: data,
        domainFn: (DeveloperSeries series, _) => series.year,
        measureFn: (DeveloperSeries series, _) => series.developers,
        colorFn: (DeveloperSeries series, _) => series.barColor)
    ];

    return Container(
      height: 300,
      padding: EdgeInsets.all(25),
      child: Card(
        child: Padding(
          padding: const EdgeInsets.all(9.0),
          child: Column(
            children: <Widget>[
              Text(
                "Yearly Growth in the Flutter Community",
                style: Theme.of(context).textTheme.bodyMedium,
              ),
              Expanded(
                child: charts.BarChart(series, animate: true),
              )
            ],
          ),
        ),
      ),
    );
  }
}

```

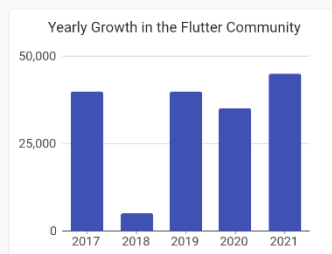
By using the **expanded** widget, we expanded our bar chart nicely into the **Card**. The **Text** widget above it gives our bar chart a title to let people know what it's about.

And, with **Theme.of(context).textTheme.body2**, we applied Material Design's default style for the body text to our title.

With **padding: const EdgeInsets.all(9.0)**, we gave the card holding our bar chart padding of 9 px on all sides. Lastly, we wrapped the **Card** in a container and gave the container a height of 300 px and padding of 25 px on all sides.

Now, our bar chart should render nicely on our screen.

Department of IT, SIES GST



Conclusion: - Hence we have successfully added Flutter charts.