EX.NO:01	Study and installation of Flutter/Kotlin multi-platform environment.
DATE:	

AIM:

To Study the Installation of Flutter/Kotlin multi-platform environment.

STUDY:

Setting Up Flutter Environment:

1.Install Flutter:
1.1: Download Flutter from the official Flutter website.
1.2:Follow the installation instructions for your operating system.

2.Set Up Flutter SDK Path:
2.1:Add the Flutter SDK path to your system's PATH variable.
2.2:This step allows you to run Flutter commands from the command line.

3.Install Android Studio:
3.1:Install Android Studio with the Flutter plugin.
3.2:This IDE provides a comfortable environment for Flutter.

4.Create a New Flutter Project:
4.1:Use the command flutter create <project_name> to create a new Flutter project.

Setting Up Kotlin Multiplatform Environment:

1.Install Kotlin:
1.1:If you haven't already, install Kotlin by following the instructions on the official Kotlin website.

2.Set Up Kotlin Multiplatform Project:
2.1:Create a new Kotlin Multiplatform project using IntelliJ IDEA or Android Studio.
2.2:Follow the IDE's instructions for creating a new Kotlin Multiplatform project.

3.Configure Shared Code:
3.1:Define the shared code modules that you want to use across platforms.
3.2:This could include business logic, data models, or utilities.

4.Add Platform-Specific Code:
4.1:Implement platform-specific code for Android and iOS.
4.2:Use Kotlin Multiplatform Mobile (KMM) plugins or libraries to facilitate communication between Kotlin and platform-specific code.

RESULT:

Therefore an complete study on the installation of Flutter/Kotlin multi-Platform environment.

EX.NO:02	Develop an application that uses Widgets, GUI components, Fonts, and Colors.
DATE:	

AIM:

To develop an application that uses Widgets,GUI components, Fonts, and Colors.

ALGORITHM:

1.Import the necessary package for Flutter material design.

2.Define the main function.
- 2.1: Inside the main function, call runApp() with an instance of the MyApp widget.

3.Define the MyApp class which extends StatelessWidget.
- 3.1: Override the build method to return a MaterialApp widget.
- 3.2: Set the title of the app and define the theme for the app.
- 3.3: Set the home property to MyHomePage.

4.Define the MyHomePage class which extends StatelessWidget.
- 4.1: Override the build method to return a Scaffold widget.
- 4.2: Set the appBar property to an AppBar widget with a title and set the body property to a Center widget.
- 4.3: Inside the Center widget, use a Column widget to arrange multiple widgets vertically.
- 4.4: Add a Text widget for the welcome message.
- 4.5: Add a SizedBox widget to create space between widgets, add another Text widget for a description, add another SizedBox widget for spacing and finally an ElevatedButton widget.
- 4.6: Set the onPressed property to a function that prints a message.
- 4.7: Customize the button's appearance using the style property.

5.Run the Flutter application, displaying the UI created by MyApp.

PROGRAM:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Sample App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
```

```
      fontFamily: 'Roboto', // Setting default font family
    ),
    home: MyHomePage(),
  );
 }
}

class MyHomePage extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Sample App'),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text(
            'Welcome to Flutter Sample App',
            style: TextStyle(
              fontSize: 24,
              fontWeight: FontWeight.bold,
              color: Colors.blue,
            ),
          ),
          SizedBox(height: 20),
          Text(
            'This is a sample application using Flutter.',
            style: TextStyle(
              fontSize: 16,
              color: Colors.grey,
            ),
            textAlign: TextAlign.center,
          ),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: () {
              // Button clicked
              print('Button clicked!');
            },
            child: Text(
              'Click Me',
              style: TextStyle(
                fontSize: 18,
                color: Colors.white,
              ),
            ),
            style: ElevatedButton.styleFrom(
              primary: Colors.green,
              padding: EdgeInsets.symmetric(horizontal: 20, vertical: 10),
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

OUTPUT:

RESULT:

Therefore an an application that uses Widgets, GUI components, Fonts, and Colors has been created successfully .

EX.NO:03	DEVELOP A NATIVE CALCULATOR APPLICATION
DATE:	

AIM:

To develop a native Calculator application using fluttter.

ALOGRITHM:

1: Set up Flutter Project.
   1.1: Open Android Studio.
   1.2:Create a new Flutter project using the New project wizard.

2: Design User Interface
   2.1: Create a new Dart file (e.g., calculator_screen.dart) for the calculator screen.
   2.2: Define a StatefulWidget named CalculatorScreen.
   2.3: Design the UI with buttons for digits, operators, clear, and equals.

3: Initialize state variables output, num1, num2, and operand in the CalculatorScreenState class.

4: Implement the buttonPressed function to handle button presses.

5: If the button pressed is 'C', reset output, num1, num2, and operand.

6: If the button pressed is an operator (+, -, *, /), store the current input as num1 and set operand.

7: If the button pressed is '=', perform the calculation based on the stored operands and operator.

8:If the button pressed is a number, append it to the current input.

9: Implement the buildButton function to create a styled ElevatedButton for each button.

10: Implement the build method to structure the UI using a Column for the display and rows of buttons.

11:Set up the app's theme, including dark mode and custom button styles.

12: Thoroughly test the calculator app for different input scenarios, ensuring it handles edge cases and behaves correctly.

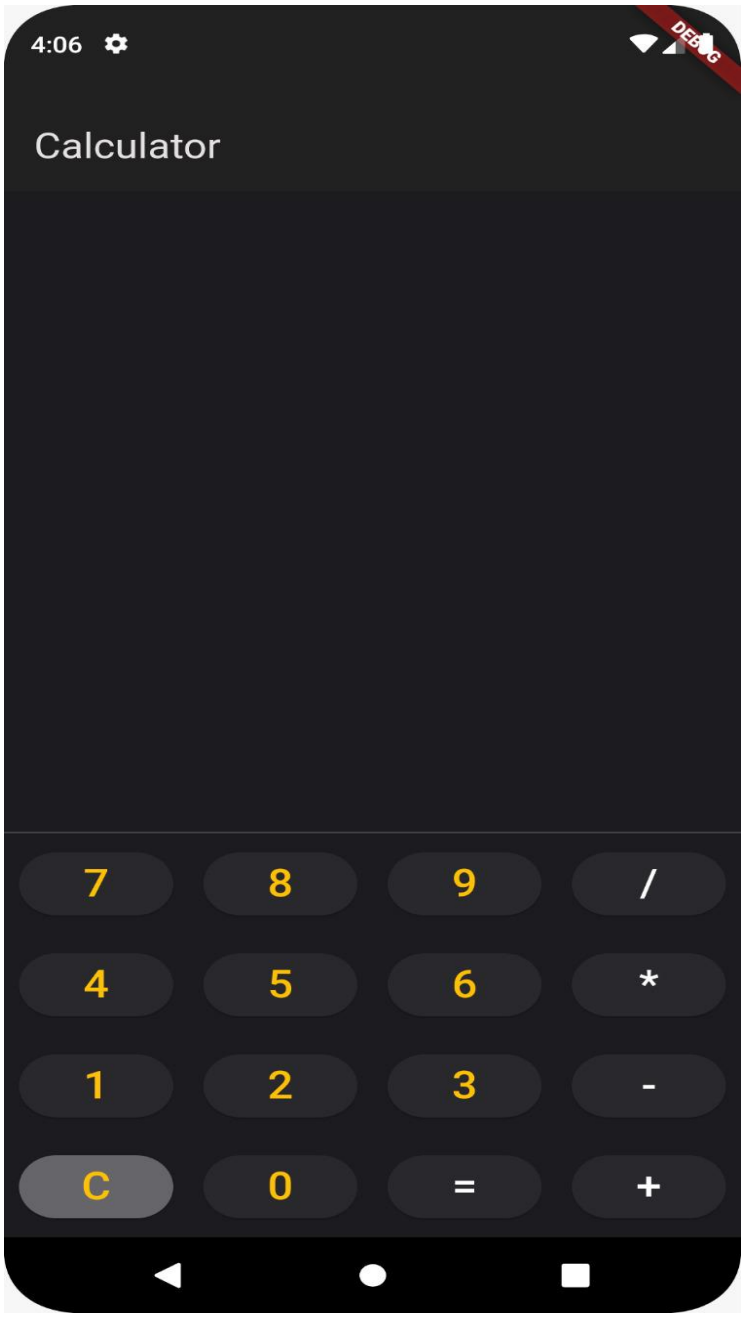
PROGRAM:

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

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

class CalculatorApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Calculator',
      theme: ThemeData.dark().copyWith(
        colorScheme: ThemeData.dark().colorScheme.copyWith(
              surface: Colors.grey[900],
            ),
        elevatedButtonTheme: ElevatedButtonThemeData(
          style: ElevatedButton.styleFrom(
            primary: Colors.grey[900],
            onPrimary: Colors.white,
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(24.0),
            ),
          ),
        ),
        textButtonTheme: TextButtonThemeData(
          style: TextButton.styleFrom(
            primary: Colors.blue,
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(24.0),
            ),
          ),
        ),
      ),
      home: CalculatorScreen(),
    );
  }
}

class CalculatorScreen extends StatefulWidget {
  @override
  _CalculatorScreenState createState() => _CalculatorScreenState();
}

class _CalculatorScreenState extends State<CalculatorScreen> {
  String _output = '';
  double _num1 = 0;
```

```
double _num2 = 0;
 String _operand = '';

 void _buttonPressed(String buttonText) {
   setState(() {
     if (buttonText == 'C') {
       // Clear button pressed
       _output = '';
       _num1 = 0;
       _num2 = 0;
       _operand = '';
     } else if (buttonText == '+' ||
         buttonText == '-' ||
         buttonText == '*' ||
         buttonText == '/') {
       // Operator button pressed
       _num1 = double.parse(_output);
       _operand = buttonText;
       _output = '';
     } else if (buttonText == '=') {
       // Equals button pressed
       _num2 = double.parse(_output);
       if (_operand == '+') {
         _output = (_num1 + _num2).toString();
       }
       if (_operand == '-') {
         _output = (_num1 - _num2).toString();
       }
       if (_operand == '*') {
         _output = (_num1 * _num2).toString();
       }
       if (_operand == '/') {
         _output = (_num1 / _num2).toString();
       }
       _num1 = 0;
       _num2 = 0;
       _operand = '';
     } else {
       // Number button pressed
       _output += buttonText;
     }
   });
 }

 Widget _buildButton(
   String buttonText,
   Color buttonColor,
   Color textColor,
 ) {
   return Expanded(
     child: Container(
       margin: EdgeInsets.all(8.0),
       child: ElevatedButton(
         style: ElevatedButton.styleFrom(
           primary: buttonColor,
           onPrimary: textColor,
         ),
         onPressed: () {
           _buttonPressed(buttonText);
         },
```

```
        child: Text(
          buttonText,
          style: TextStyle(fontSize: 24.0),
        ),
      ),
    ),
  );
}

Widget _buildButtonRow(List<String> buttons) {
  return Row(
    children: buttons
        .map((button) => _buildButton(
              button,
              button == 'C' ? Colors.grey[700]! : Colors.grey[900]!,
              button == '=' ||
                      button == '/' ||
                      button == '*' ||
                      button == '-' ||
                      button == '+'
                  ? Colors.white
                  : Colors.amber,
            ))
        .toList(),
  );
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Calculator'),
    ),
    body: Column(
      children: [
        Expanded(
          child: Container(
            alignment: Alignment.bottomRight,
            padding: EdgeInsets.all(24.0),
            child: Text(
              _output,
              style: TextStyle(fontSize: 48.0, fontWeight: FontWeight.bold),
            ),
          ),
        ),
        Divider(height: 0.0),
        Column(
          children: [
            // Rows of buttons
            _buildButtonRow(['7', '8', '9', '/']),
            _buildButtonRow(['4', '5', '6', '*']),
            _buildButtonRow(['1', '2', '3', '-']),
            _buildButtonRow(['C', '0', '=', '+']),
          ],
        ),
      ],
    ),
  );
}
}
```

OUTPUT:



RESULT:

Thus the calculator application was developed successfully using flutter.

EX.NO: 04	DEVELOPAGAMING APPLICATION THAT USES 2-D ANIMATIONS AND GESTURES.
DATE:	

AIM:

To develop a gaming application that uses 2-D Animations and Gestures using flutter.

ALGORITHM:

1:Set up Flutter Project.

1.1: Open Android Studio.

1.2:Create a new Flutter project using the New project wizard.

2 .Initialization:

2.1Initialize the character's X position (_characterXPosition) to 0.0.

2.2Set the character's movement speed (characterSpeed) to 5.0.

3.User Interaction:

3.1Get the X position of the tap (tapPosition).

3.2 Get the total screen width (screenWidth) using MediaQuery.

3.3 If the tap is on the left half of the screen , Move the character left by subtracting characterSpeed from _characterXPosition.

3.4 If the tap is on the right half of the screen Move the character right by adding characterSpeed to _characterXPosition.

4. The Flutter framework will automatically re-render the screen when the state changes due to the character's position update (triggered by setState).

5. The app UI is structured using the Flutter framework with a Scaffold, AppBar, and GestureDetector to capture tap events.

6. Visual Representation:

6.1 A blue Container represents the game background.

6.2 A red square Container represents the character, positioned at the top center of the screen with its left edge aligned with _characterXPosition.

7. The algorithm takes into account the screen width and divides it into two halves for left and right movement.

8. The setState function is used to update the state, triggering a re-render of the UI.

9. The app continuously runs, waiting for user interactions. The UI updates dynamically based on the user's taps.

10. The GestureDetector is configured with the onTapDown callback to respond to tap events.

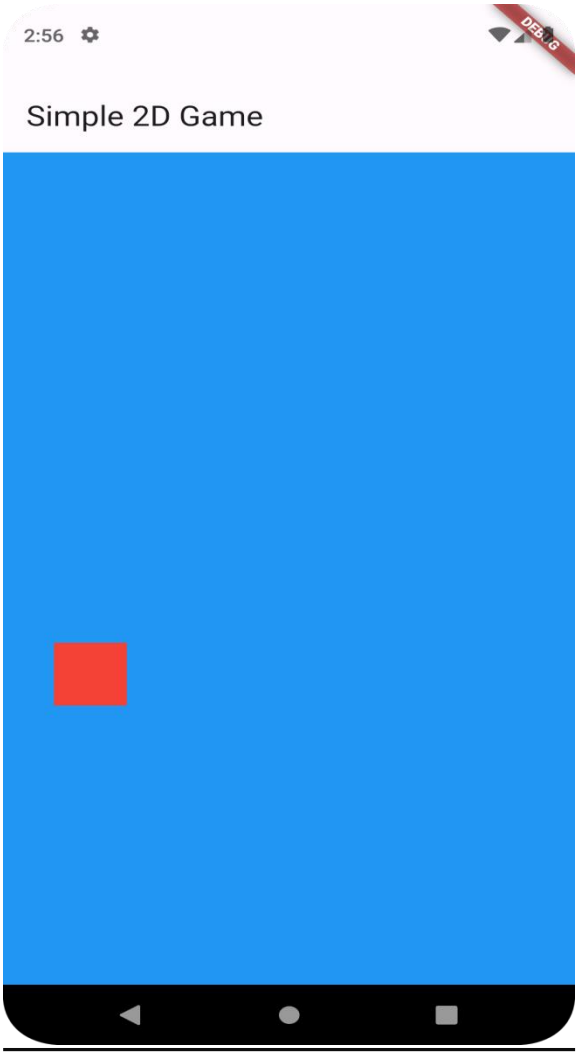
11. The game elements are styled with basic colors, such as a blue background and a red character.

PROGRAM:

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

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

class GameApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: GameScreen(),
    );
  }
}

class GameScreen extends StatefulWidget {
  @override
  _GameScreenState createState() => _GameScreenState();
}

class _GameScreenState extends State<GameScreen> {
  double _characterXPosition = 0.0;
  final double characterSpeed = 5.0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Simple 2D Game'),
      ),
      body: GestureDetector(
        behavior: HitTestBehavior.opaque,
        onTapDown: (TapDownDetails details) {
          // Move character left or right based on tap position
          double tapPosition = details.localPosition.dx;
          double screenWidth = MediaQuery.of(context).size.width;
          if (tapPosition < screenWidth / 2) {
            // Move left
            setState(() {
              _characterXPosition -= characterSpeed;
            });
          } else {
            // Move right
            setState(() {
              _characterXPosition += characterSpeed;
            });
          }
        },
        child: Stack(
          children: [
            // Game background



              Container(
              decoration: BoxDecoration(
```

```
        color: Colors.blue,
            ),
          ),


        // Character
          Positioned(
            left: _characterXPosition,
            top: MediaQuery.of(context).size.height / 2,
            child: Container(
              width: 50,
              height: 50,
              color: Colors.red,
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

OUTPUT:



RESULT:

Thus a gaming application that uses 2-D animations and Gestures has been developed successfully using flutter.

EX.NO: 05	DEVELOPAMOVIE RATING APPLICATION.
DATE:	

AIM:

To develop a movie rating Application using Flutter.

ALGORITHM:

1. Initialize the App:
   1.1: Set up the Flutter project with necessary dependencies.
   1.2: Define the main entry point of the app.

2.Define Movie Data Structure:
   2.1:Decide on the data structure to represent movies.In this case, a map with keys for 'title' and 'rating' is used.

3.Create MyApp Widget:
   3.1:Implement a stateless widget MyApp that returns a MaterialApp.
   3.2:Configure the app's title and theme.
   3.3:Set the initial screen to MovieListScreen.

4.Create MovieListScreen Widget:
   4.1:Implement a stateless widget MovieListScreen that displays a list of movies.
   4.2:Use a ListView.builder to dynamically create list items for each movie.
   4.3:Each list item (represented by ListTile) contains the movie's title and rating.
   4.4:Add onTap functionality to navigate to the MovieDetailScreen when a movie is tapped.

5.Create MovieDetailScreen Widget:
   5.1:Implement a stateless widget MovieDetailScreen that displays detailed information about a selected movie.
   5.2:Receive movie data through the constructor.
   5.3:Display the movie's title and rating in a Column widget.
   5.4:Additional movie details can be added here.

6.Handle Navigation:
   6.1:Use the Navigator class to handle navigation between screens.
   6.2:When a movie is tapped in MovieListScreen, navigate to MovieDetailScreen and pass the selected movie's data.

7.Run the App:
   7.1:Run the app using the runApp() function, passing an instance of MyApp.

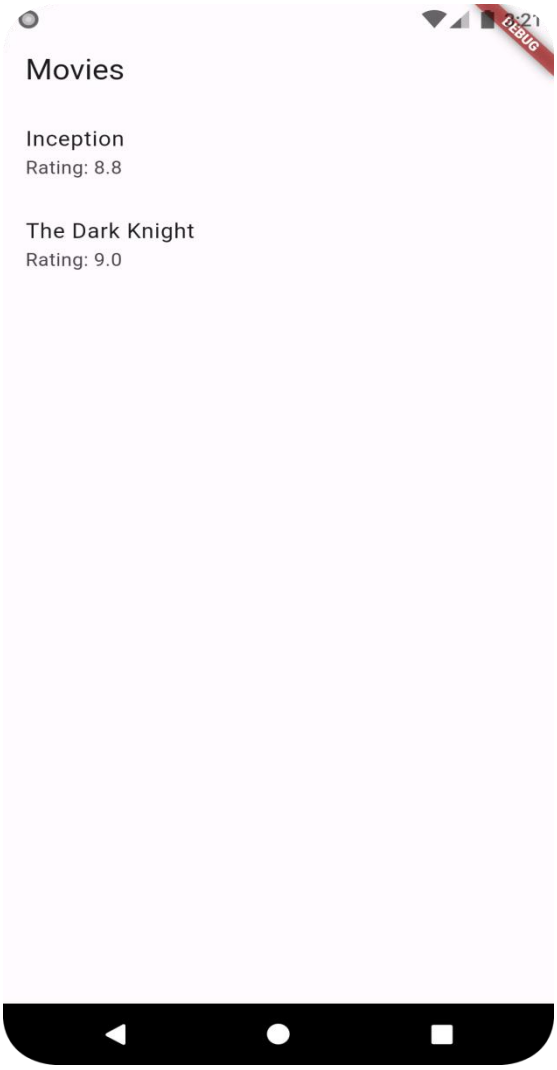
8. VIew the Output:
   8.1: The output of the given flutter code can be viewd in the emulator for movie rating Application.

PROGRAM:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Movie Rating App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MovieListScreen(),
    );
  }
}

class MovieListScreen extends StatelessWidget {
  final List<Map<String, dynamic>> movies = [
    {'title': 'Inception', 'rating': 8.8},
    {'title': 'The Dark Knight', 'rating': 9.0},
// Add more movies here
];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Movies'),
      ),
      body: ListView.builder(
        itemCount: movies.length,
        itemBuilder: (BuildContext context, int index) {
          return ListTile(
            title: Text(movies[index]['title']),
            subtitle: Text('Rating: ${movies[index]['rating']}'),
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => MovieDetailScreen(movie: movies[index]),
                ),
              );
            },
          );
        },
      ),
    );
  }
}

class MovieDetailScreen extends StatelessWidget {
  final Map<String, dynamic> movie;
```

```
 MovieDetailScreen({required this.movie});

 @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(movie['title']),
      ),
      body: Padding(
        padding: EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text('Rating: ${movie['rating']}'),
// Add more details here
],
        ),
      ),
    );
  }
}
```

OUTPUT:



RESULT:

Thus the Movie Rating application using Flutter has been created and executed successfully.

Ex.No:06	Develop an application to connect to a web service and to retrieve data with HTTP.
Date:	

Aim:

To Develop an application that connect's to a web service and to retrieve data with HTTP.

Algorithm:

1. Start

2. Import necessary packages: flutter/material.dart, http/http.dart.

3. Define main() function to run the app.

4. Create a MyApp StatelessWidget.
    - 4.1. Set the app title and theme.
    - 4.2. Set the home screen as LaunchListScreen.
5. Create a LaunchListScreen StatefulWidget.
    - 5.1. Define _LaunchListScreenState.
    - 5.2. Implement fetchData() async function:
    - 5.3. Implement build() method:
        - i. Display a loading indicator if _isLoading is true.
        - ii. If _isLoading is false and _launches list is empty:
        - iii. If _isLoading is false and _launches list is not empty:

6. End

Program:

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

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

class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'SpaceX Launches',
theme: ThemeData(
primarySwatch: Colors.blue,
),
home: LaunchListScreen(),
);
}
```

```
}

class LaunchListScreen extends StatefulWidget {
@override
_LaunchListScreenState createState() => _LaunchListScreenState();
}

class _LaunchListScreenState extends State<LaunchListScreen> {
List<dynamic> _launches = [];
bool _isLoading = true;

@override
void initState() {
super.initState();
fetchData();
}

Future<void> fetchData() async {
final response = await http.get(Uri.parse('https://api.spacexdata.com/v4/launches/upcoming'));

if (response.statusCode == 200) {
setState(() {
_launches = jsonDecode(response.body);
_isLoading = false;
});
} else {
setState(() {
_isLoading = false;
_launches = [];
});
showDialog(
context: context,
builder: (context) => AlertDialog(
title: Text('Error'),
content: Text('Failed to load launches.'),
actions: [
TextButton(
onPressed: () => Navigator.pop(context),
child: Text('OK'),
),
],
),
);
}
}

@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
title: Text('SpaceX Launches'),
),
body: _isLoading
? Center(
child: CircularProgressIndicator(),
)
: _launches.isEmpty
? Center(
child: Text('No launches available.'),
)
```

```
: ListView.builder(
itemCount: _launches.length,
itemBuilder: (context, index) {
final launch = _launches[index];
return ListTile(
title: Text(launch['name']),
subtitle: Text('Date: ${launch['date_local']}'),
onTap: () {
// Add onTap functionality if needed
},
);
},
),
);
}
}
```

Output:

Result:

Thus an an application to connect to a web service and to retrieve data with HTTP. Was done Successfully.

Ex.No:07	Develop a simple shopping application
Date:	

Aim :

To develop a simple shopping application

Algorithm:

1.Initialize Products and Cart Lists:
- 1.1.Create a list of products, each with a name and price.
- 1.2.Create an empty list to represent the cart.

2.Display Products Screen (ProductListScreen):
- 2.1.Create a screen to display a list of products.
- 2.2.Each product should have a name, price, and an "Add to Cart" button

3.Display Cart Screen (CartScreen):
- 3.1.Create a screen to display the items in the cart.
- 3.2.Each item in the cart should display the product name, price, and a "Remove from Cart" button..

4.Calculate Total Price:
- 4.1.Create a function to calculate the total price of all items in the cart.
- 4.2.Iterate through the products in the cart, summing up their prices.
- 4.3.Return the total price.

5.Navigation:
- 5.1.Implement navigation between the Products Screen and the Cart
- 5.2.Screen using Flutter's Navigator class.
- 5.3.Use MaterialPageRoute to navigate between screens.

Program:

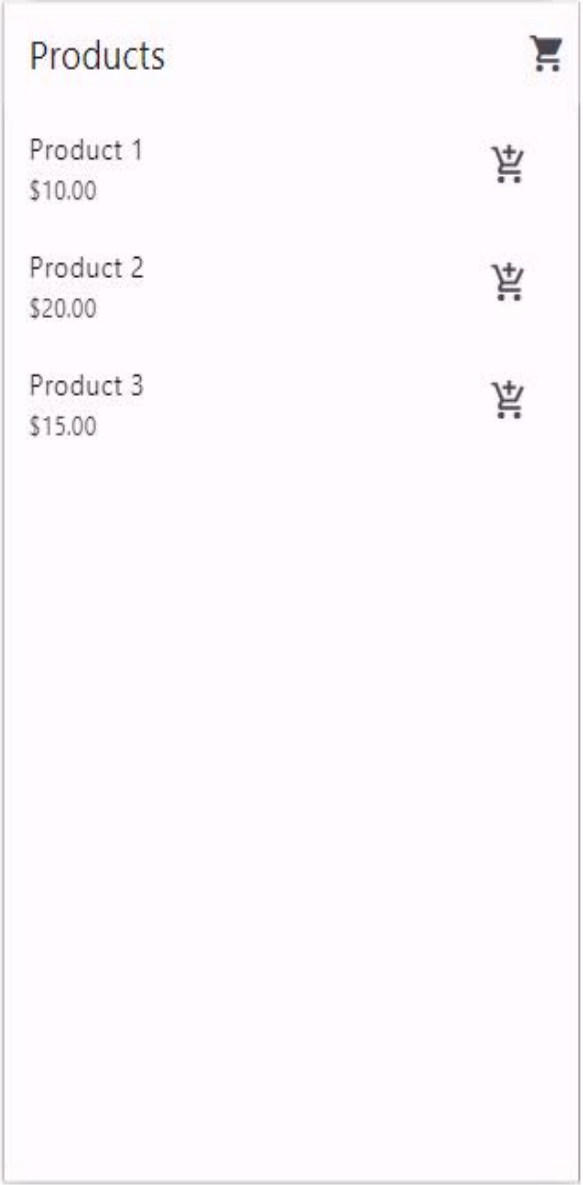
```
import 'package:flutter/material.dart';

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

class ShoppingApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Shopping App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: ProductListScreen(),
    );
  }
}

class Product {
  final String name;
```

```
  final double price;

  Product({required this.name, required this.price});
}

class ProductListScreen extends StatefulWidget {
  @override
  _ProductListScreenState createState() => _ProductListScreenState();
}

class _ProductListScreenState extends State<ProductListScreen> {
  final List<Product> products = [
    Product(name: 'Product 1', price: 10.0),
    Product(name: 'Product 2', price: 20.0),
    Product(name: 'Product 3', price: 15.0),
  ];

  final List<Product> cart = [];

  @override

 Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Products'),
        actions: [
          IconButton(
            icon: Icon(Icons.shopping_cart),
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => CartScreen(cart: cart)),
              );
            },
          ),
        ],
      ),
      body: ListView.builder(
        itemCount: products.length,
        itemBuilder: (context, index) {
          final product = products[index];
          return ListTile(
            title: Text(product.name),
            subtitle: Text('\$${product.price.toStringAsFixed(2)}'),
            trailing: IconButton(
              icon: Icon(Icons.add_shopping_cart),
              onPressed: () {
                setState(() {
                  cart.add(product);
                });
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(
                    content: Text('${product.name} added to cart'),
                    duration: Duration(seconds: 1),
                  ),
                );
              },
            ),
          );
        },
```

```
      ),
    );
  }
}

class CartScreen extends StatefulWidget {
  final List<Product> cart;

  CartScreen({required this.cart});

  @override
  _CartScreenState createState() => _CartScreenState();
}

class _CartScreenState extends State<CartScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Shopping Cart'),
      ),
      body: ListView.builder(
        itemCount: widget.cart.length,
        itemBuilder: (context, index) {
          final product = widget.cart[index];
          return ListTile(
            title: Text(product.name),
            subtitle: Text('\$${product.price.toStringAsFixed(2)}'),
            trailing: IconButton(
              icon: Icon(Icons.remove_shopping_cart),
              onPressed: () {
                setState(() {
                  widget.cart.remove(product);
                });
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(
                    content: Text('${product.name} removed from cart'),
                    duration: Duration(seconds: 1),
                  ),
                );
              },
            ),
          );
        },
      ),
      bottomNavigationBar: BottomAppBar(
        child: Padding(
          padding: EdgeInsets.all(16.0),
          child: Text(
            'Total: \$${calculateTotal().toStringAsFixed(2)}',
            style: TextStyle(fontSize: 18.0, fontWeight: FontWeight.bold),
          ),
        ),
      ),
    );
  }

  double calculateTotal() {
    double total = 0;
    for (var product in widget.cart) {
```

```
      total += product.price;
    }
    return total;
  }
}
```

Output:



Result:

Therefore a simple shopping application has been created successfully.

Ex.No:08	Design a web server supporting push notifications
Date:	

Aim:

To develop a web server supported notifications

Algorithm:

1.Initialize WebSocket Connection:

1.1.Establish a WebSocket connection to the echo server wss://echo.websocket.org using the HtmlWebSocketChannel.connect method.

2.Build User Interface:

2.1.Create a MaterialApp with the title 'WebSocket Demo'.
2.2.Set the home screen to a StatefulWidget called WebSocketDemo.
2.3.Scaffold the app with an AppBar displaying the title 'WebSocket Demo'.
2.4.Center the body of the Scaffold and use a StreamBuilder to display Incoming WebSocket data.
2.5.Show a CircularProgressIndicator while waiting for data and Add a FloatingActionButton to send messages over the WebSocket connection.

3.Handle WebSocket Data:

3.1.Use a StreamBuilder to listen to the WebSocket channel's stream for incoming data.
3.2.When data is received, update the UI to display the received message using a Text widget.

4.Send Messages:

4.1.Implement a FloatingActionButton to send messages over the WebSocket connection.
4.2.When the FloatingActionButton is pressed, send the message 'Hello, WebSocket!' through the WebSocket channel's sink using channel.sink.add.

5.Cleanup on Dispose:

5.1.Override the dispose method of the StatefulWidget to close the WebSocket channel's sink when the widget is disposed, ensuring proper cleanup and resource release.

6.Run the Application:

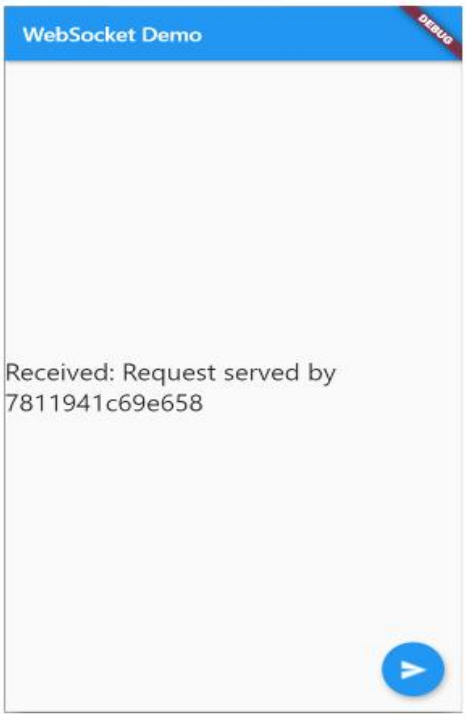
6.1.Start the Flutter application by running the main function, which will run the MyApp widget.

Program:

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

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

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'WebSocket Demo',
      home: WebSocketDemo(),
    );
  }
}

class WebSocketDemo extends StatefulWidget {
  @override
  _WebSocketDemoState createState() => _WebSocketDemoState();
}

class _WebSocketDemoState extends State<WebSocketDemo> {
  final channel = HtmlWebSocketChannel.connect('wss://echo.websocket.org');

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('WebSocket Demo'),
      ),
      body: Center(
        child: StreamBuilder(
          stream: channel.stream,
          builder: (context, snapshot) {
            return snapshot.hasData
                ? Text(
                    'Received: ${snapshot.data}',
                    style: TextStyle(fontSize: 24),
                  )
                : CircularProgressIndicator();
          },
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          channel.sink.add('Hello, WebSocket!');
        },
        child: Icon(Icons.send),
      ),
    );
  }

  @override
  void dispose() {
    channel.sink.close();
    super.dispose();
  }
}
```

Output:


WebSocket Demo
DEBUG
Received: Request served by
7811941c69e658


WebSocket Demo
DEBUG
Received: Hello, WebSocket!

Result:

Therefore a web server supporting push notification has been created successfully.

Ex.No:09	Develop an application by integrating Google maps
Date:	

Aim:

To develop an application integrating Google maps

Algorithm:

1.Initialize the Flutter Application:
    1.1.Define a main function to start the Flutter application.
    1.2.Instantiate MyApp and run it using runApp.

2.Create MyApp Widget:
    2.1.Define a MyApp StatelessWidget.
    2.2.Return a MaterialApp with a title and set the home property to MapSample.

3.Create MapSample Widget:
    3.1.Define a MapSample StatefulWidget.
    3.2.Implement createState method to create an instance of MapSampleState.

4.Create MapSampleState Widget:
    4.1Define a MapSampleState class that extends State<MapSample>.
    4.2.Declare a Completer to handle the asynchronous loading of the GoogleMapController.
    4.3.Define a static constant CameraPosition representing the initial camera position centered on Vellore, India.
    4.4.Implement a method _onMapType() to toggle between normal and satellite map types.
    4.5.Override the build method to construct the UI:
        4.5.1.Scaffold with an AppBar displaying the title and a blue background.
        4.5.2.Stack widget to overlay the GoogleMap and a
        4.5.3.FloatingActionButton.Override the build method to construct the UI:

5.Initialize GoogleMap Widget:
    5.1.Use the GoogleMap widget to display the map.
    5.2.Set the initialCameraPosition to the predefined Vellore location.

6.Implement Map Type Toggle:
    6.1.Create a FloatingActionButton to toggle between map types.
    6.2.Implement onPressed callback _onMapType() to switch between normal and satellite map types.
    6.3.Update the state to reflect the current map type.

7.Run the Application:
    7.1.Execute the Flutter application, which will display the Google Map with the specified initial settings.

8.API Key Requirement:
    8.1.Note that the Google Maps API key is required for the map to function properly. Make sure to include it in the code before running the application.

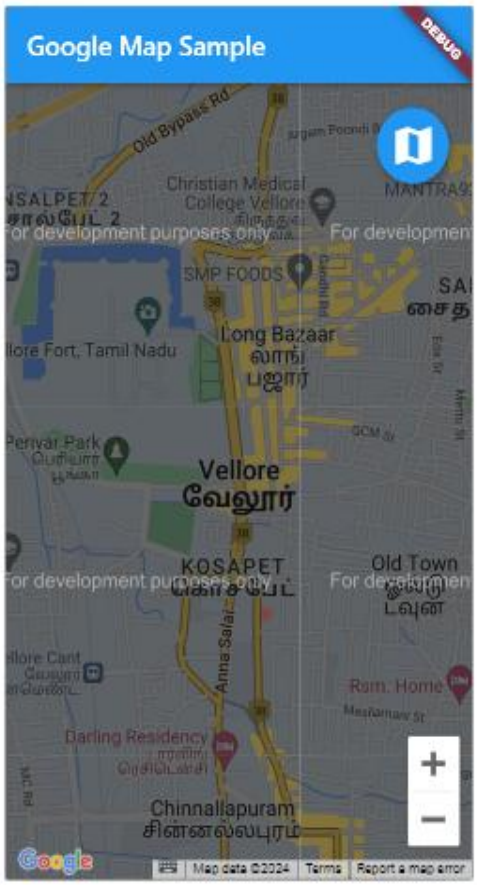
Program:

API-Key: AIzaSyCDBIv3Y6HKOE4ZEIO-j_6LD62Wsj6JKqA

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'Flutter Google Maps Demo',
      home: MapSample(),
    );
  }
}

class MapSample extends StatefulWidget {
  const MapSample({Key? key}) : super(key: key);

  @override
  State<MapSample> createState() => MapSampleState();
}

class MapSampleState extends State<MapSample> {
  final Completer<GoogleMapController> _controller = Completer();
  static const CameraPosition _velloreLocation = CameraPosition(
    target: LatLng(12.9166, 79.1325), // Vellore coordinates
    zoom: 15,
  );
  MapType _currentMapType = MapType.normal;

  void _onMapType() {
    setState(() {
      _currentMapType = _currentMapType == MapType.normal ? MapType.satellite :
MapType.normal;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Google Map Sample'),
        backgroundColor: Colors.blue,
      ),
      body: Stack(
        children: [
          GoogleMap(
            mapType: _currentMapType,
            initialCameraPosition: _velloreLocation,
            onMapCreated: (GoogleMapController controller) {
              _controller.complete(controller);
```

```
          },
        ),
        Padding(
          padding: const EdgeInsets.all(18),
          child: Align(
            alignment: Alignment.topRight,
            child: FloatingActionButton(
              onPressed: _onMapType,
              child: const Icon(Icons.map, size: 36),
            ),
          ),
        ),
      ],
    ),
  );
 }
}
```

Output:



Result:

Therefore an application by integrating Google maps has been created successfully.