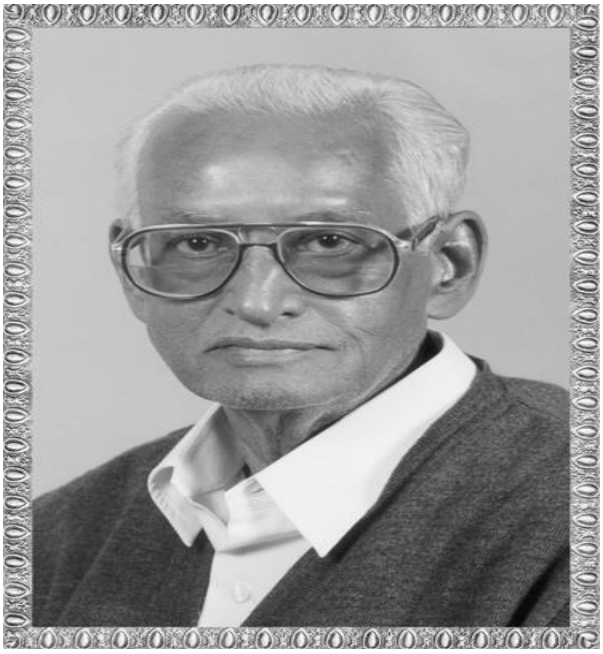
# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

**(AFFILIATED TO SAURASHTRA UNIVERSITY)**


**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**

# Lt. Shree Chimanbhai Shukla

# MSCIT SEM-3 – FLUTTER APP DEVELOPMENT


**Shree H.N.Shukla College**
**Street No. 2, Vaishali Nagar,**
**Nr. Amrapali Under Bridge,**
**Raiya Road, Rajkot.**
**Ph. (0281)2440478, 2472590**

**Shree H.N.Shukla College**
**Street No. 2, Vaishali Nagar,**
**Nr. Amrapali Under Bridge,**
**Raiya Road, Rajkot.**
**Ph. (0281)2440478, 2472590**

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

**(AFFILIATED TO SAURASHTRA UNIVERSITY)**

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


| CS – 13 : Flutter App Development | | |
|---|---|---|
| **SR.NO.** | **TOPIC** | **DETAIL** |
| **1** | **Introduction to Flutter and Dart** | • Overview of Flutter and Dart<br>• Overview of the Flutter architecture and how it works<br>• Setting up the development environment<br>• Dart syntax and data types<br>• Basic Flutter widgets and layout<br>• Basic Flutter widget properties and methods<br>• Dart libraries and packages<br>• Control flow and loops in Dart |
| **2** | **Building User Interfaces** | • Stateful vs. Stateless widgets<br>• Layout and widgets hierarchy<br>• Navigation and routing<br>• Responsive design and media queries<br>• Advanced Flutter widgets (e.g.,<br>• SliverAppBar, AnimatedContainer)<br>• Custom widget creation in Flutter<br>• Debugging UI issues in Flutter<br>• Advanced layout techniques (e.g., Flexbox, GridView) |
| **3** | **Managing App State** | • State management in Flutter<br>• InheritedWidget and InheritedModel<br>• ScopedModel and Provider<br>• BLoC pattern for state management<br>• Stream-based state management with RxDart |

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

| | | |
|---|---|---|
| | | • Redux architecture for Flutter<br>• Firebase integration for state management<br>• Using the Flutter DevTools for debugging |
| **4** | **Networking and Persistence** | • RESTful APIs and HTTP requests<br>• JSON serialization and deserialization<br>• SQLite and local storage<br>• Shared preferences and secure storage<br>• WebSockets for real-time<br>• communication in Flutter<br>• Firebase integration for data storage<br>• Caching data in Flutter<br>• Using third-party libraries for networking and data storage (e.g., Dio, Hive) |
| **5** | **Animation, Integration, Testing, Debugging & Accessibility** | • Animations and motion<br>• Advanced animation techniques (e.g.,<br>• Flare, Lottie)<br>• Internationalisation and localization<br>• Native platform integration<br>• Push notifications in Flutter (Firebase)<br>• Integration with other native features<br>• (e.g., camera, location)<br>• Testing and debugging<br>• Accessibility in Flutter apps |

**SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.**
(AFFILIATED TO SAURASHTRA UNIVERSITY)

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

# CHAPTER-4

## Networking and Persistence

- RESTful APIs and HTTP requests
- JSON serialization and deserialization
- SQLite and local storage
- Shared preferences and secure storage
- WebSockets for real-time
- communication in Flutter
- Firebase integration for data storage
- Caching data in Flutter
- Using third-party libraries for networking and data storage (e.g. Dio, Hive)

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


**Q-1 Explain RESTful APIs and HTTP requests in Flutter.**

**Answer :**

- Flutter provides http package to consume HTTP resources. http is a Future-based library and uses await and async features. It provides many high level methods and simplifies the development of REST based mobile applications.
- **R**epresentational **S**tate **T**ransfer (REST) is an architectural style that defines a set of constraints to be used for creating web services. **REST API** is a way of accessing web services in a simple and flexible way without having any processing.
- 
- REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST uses less bandwidth, simple and flexible making it more suitable for internet usage. It's used to fetch or give some information from a web service. All communication done via REST API uses only HTTP request.
- **Working:** A request is sent from client to server in the form of a web URL as HTTP GET or POST or PUT or DELETE request. After that, a response comes back from the server in the form of a resource which can be anything like HTML, XML, Image, or JSON. But now JSON is the most popular format being used in Web Services.


http package provides a high level class and http to do web requests.

- http class provides functionality to perform all types of HTTP requests.
- http methods accept a url, and additional information through Dart Map (post data, additional headers, etc.,). It requests the server and collects the response back in

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

async/await pattern. For example, the below code reads the data from the specified url and print it in the console.

```
print(await http.read('https://flutter.dev/'));
```

Some of the core methods are as follows −

- **read** − Request the specified url through GET method and return back the response as Future<String>
- **get** − Request the specified url through GET method and return back the response as Future<Response>. Response is a class holding the response information.
- **post** − Request the specified url through POST method by posting the supplied data and return back the response as Future<Response>
- **put** − Request the specified url through PUT method and return back the response as Future <Response>
- **head** − Request the specified url through HEAD method and return back the response as Future<Response>
- **delete** − Request the specified url through DELETE method and return back the response as Future<Response>

http also provides a more standard HTTP client class, client. client supports persistent connection. It will be useful when a lot of request to be made to a particular server. It needs to be closed properly using close method. Otherwise, it is similar to http class. The sample code is as follows −

```
var client = new http.Client();
try {
  print(await client.get('https://flutter.dev/'));
}
finally {
  client.close();
}
```

## Request and Response

Now we will see how request and response work for different **HTTP** methods. Let's assume we have an **API(**https://www.geeksforgeeks.org/api/students**)** for all students data of gfg.

- **GET:** Request for all Students.

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

| Request |
| --- |
| GET:/api/students |

- **POST:** Request for Posting/Creating/Inserting Data

| Request |
| --- |
| POST:/api/students<br>{“name”:”Raj”} |

- **PUT or PATCH:** Request for Updating Data at id=1

| Request |
| --- |
| PUT or PATCH:/api/students/1<br>{“name”:”Raj”} |

- **DELETE:** Request for Deleting Data of id=1

| Request |
| --- |
| DELETE:/api/students/1 |

*RESTful web services are very popular because they are light weight, highly scalable and maintainable and are very commonly used to create APIs for web-based applications.*

## Create a List of fruits

Create a *FruitList* class in *fruitList.dart* as shown below:

- Dart

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

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

```
import 'fruit.dart';
import 'fruitItem.dart';

class FruitList extends StatelessWidget {
  final List<Fruit> items;

  FruitList({Key key, this.items});
  @override
  Widget build(BuildContext context) {
    return ListView.builder(
      itemCount: items.length,
      itemBuilder: (context, index) {
        return FruitItem(item: items[index]);
      },
    );
  }
}
```

**Step 6: Displaying the Responses**

Display the response on the screen from *main.dart file* as shown below:

Dart

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

class MyHomePage extends StatelessWidget {
  final String title;
  final Future<List<Fruit>> products;
```

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

```
MyHomePage({Key key, this.title, this.products}) : super(key: key);

@override
Widget build(BuildContext context)
{
  return Scaffold(
      appBar: AppBar(
        backgroundColor: Color(0xFF4CAF50),
        title: Text("GeeksForGeeks"),
      ),
      body: Center(
        child: FutureBuilder<List<Fruit>>(
          future: products,
          builder: (context, snapshot) {
            if (snapshot.hasError) print(snapshot.error);
            return snapshot.hasData
                ? FruitList(items: snapshot.data)
                : Center(child: CircularProgressIndicator());
          },
        ),
      ));
  }
}
```

Following is the **JSON Data** upon running the application:

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

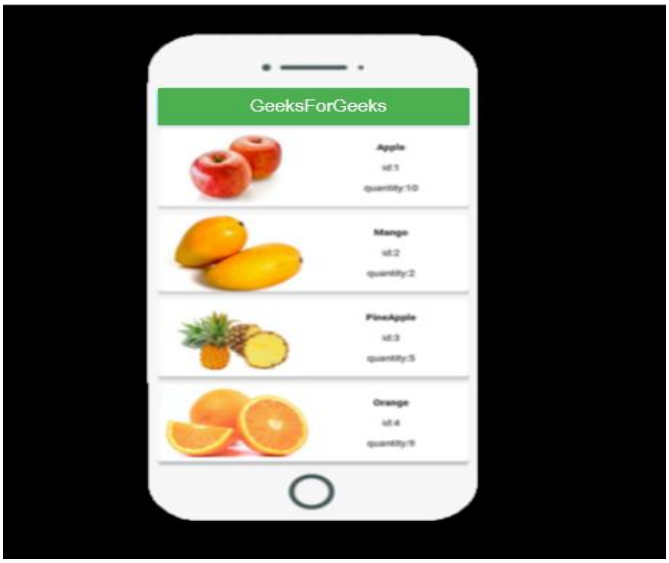
**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**



```
{
  "id" : 1,
  "title" : "Apple",
  "imgUrl" : "https://www.eho.eu/wp-content/uploads/2018/10/jabolko_novo.jpg",
  "quantity" : 10
},
{
  "id" : 2,
  "title" : "Mango",
  "imgUrl" : "https://m.media-amazon.com/images/I/31KSJlUe16L._AC_SS350_.jpg",
  "quantity" : 2
},
{
  "id" : 3,
  "title" : "PineApple",
  "imgUrl" : "https://www.healthifyme.com/blog/wp-content/uploads/2015/12/shutterstock_1670265607-1.jpg",
  "quantity" : 5
},
{
  "id" : 4,
  "title" : "Orange",
  "imgUrl" : "https://static9.depositphotos.com/1642482/1149/i/450/depositphotos_11490801-stock-photo-oranges-fruit.jpg",
  "quantity" : 9
}
```


Following is the UI screen of the homepage of Flutter application with decoded JSON data.

**Output:**



**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


**Q-2 Explain JSON serialization and deserialization in Flutter.**

**Answer:**

- JSON, which stands for **JavaScript Object Notation**, is an open-standard format used on the web and in mobile clients. It's the most widely used format for Representational State Transfer (REST)-based APIs that servers provide. If you talk to a server that has a REST API, it will most likely return data in a JSON format.

This article covers two general strategies for working with JSON:

- Manual serialization
- Automated serialization using code generation

Different projects come with different complexities and use cases. For smaller proof-of-concept projects or quick prototypes, using code generators might be overkill. For apps with several JSON models with more complexity, encoding by hand can quickly become tedious, repetitive, and lend itself to many small errors.

Use manual serialization for smaller projects

Manual JSON decoding refers to using the built-in JSON decoder in dart:convert. It involves passing the raw JSON string to the jsonDecode() function, and then looking up the values you need in the resulting Map<String, dynamic>. It has no external dependencies or particular setup process, and it's good for a quick proof of concept.

erializing JSON manually using dart:convert

Basic JSON serialization in Flutter is very simple. Flutter has a built-in dart:convert library that includes a straightforward JSON encoder and decoder.

The following sample JSON implements a simple user model.

*content_copy*

```
{
  "name": "John Smith",
  "email": "john@example.com"
}
```


**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


With dart:convert, you can serialize this JSON model in two ways.

Serializing JSON inside model classes

Combat the previously mentioned problems by introducing a plain model class, called User in this example. Inside the User class, you'll find:

- A User.fromJson() constructor, for constructing a new User instance from a map structure.
- A toJson() method, which converts a User instance into a map.

With this approach, the *calling code* can have type safety, autocompletion for the name and email fields, and compile-time exceptions. If you make typos or treat the fields as ints instead of Strings, the app won't compile, instead of crashing at runtime.

**user.dart**

*content_copy*

```
class User {
  final String name;
  final String email;

  User(this.name, this.email);

  User.fromJson(Map<String, dynamic> json)
      : name = json['name'],
        email = json['email'];

  Map<String, dynamic> toJson() => {
        'name': name,
        'email': email,
      };
}
```

The responsibility of the decoding logic is now moved inside the model itself. With this new approach, you can decode a user easily.

*content_copy*

```
Map<String, dynamic> userMap = jsonDecode(jsonString);
var user = User.fromJson(userMap);

print('Howdy, ${user.name}!');
print('We sent the verification link to ${user.email}.');
```

To encode a user, pass the User object to the jsonEncode() function. You don't need to call the toJson() method, since jsonEncode() already does it for you.

*content_copy*

```
String json = jsonEncode(user);
```

With this approach, the calling code doesn't have to worry about JSON serialization at all. However, the model class still definitely has to. In a production app, you would want to ensure that the serialization works properly.

## How to Deserialize a list of objects from JSON in Flutter?

When using the dart package **json_serializable** for **JSON** serialization. To deserialize a list of objects from JSON in **flutter** follows the below steps:
**1**-> create a model class and click here to convert JSON to dart.
**2**-> create a response like
loginResponce=**LoginResponce.fromJson**(json.decode(response.body));
**3** -> Now you get your data in instence of model (as loginResponce ).

Another example on **JSON Parsing** for further clarification.

```
factory YoutubeResponse.fromJSON(Map<string, dynamic=""> youtubeResponseJson) {
var list = youtubeResponseJson['items'] as List;
List itemsList = list.map((i) => Item.fromJSON(i)).toList();
return new YoutubeResponse(
kind: youtubeResponseJson['kind'],
etag: youtubeResponseJson['etag'],
nextPageToken: youtubeResponseJson['nextPageToken'],
regionCode: youtubeResponseJson['regionCode'],
mPageInfo: pageInfo.fromJSON(youtubeResponseJson['pageInfo']),
items: itemsList);
```

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

}
</string,>

You can also do it like below:

List itemsList =
List.from(parsedListJson.map((i) => Item.fromJson(i)));

If the response body is iterable, then you need to parse and walk accordingly, if I am understanding your question correctly.

Example:

Iterable l = json.decode(response.body);
List<Post> posts = List<Post>.from(l.map((model)=> Post.fromJson(model)));

**Q-3 Explain SQLite and local storage in Flutter.**

**Answer:**

- we load data from server once and save data in local storage and use that to perform next actions.
- To store data in device's local storage, Android provides built in open source SQL database which stores data in it. It is a C-language library and is serverless and lightweight solution. SQLite supports all the relational database operations such as store, manipulate and retrieving data from database.

**SQLite in Flutter**

Flutter apps can also use SQLite data base using sqflite plugin which is available on pub dev. It supports almost all SQL standards. This plugin contains helpers for common CRUD operations, but it also provides ability to write your own SQL queries in String. In flutter, all the database operations are handled in background thread.

**Add dependency to Flutter project**

To use SQLite in flutter apps. You need to add sqflite plugin in your project. First create a new Flutter project, then open pubspec.yml file, and add the following code.

```
dependencies:
  flutter:
    sdk: flutter
  sqflite: ^2.0.2
  path: 1.8.0
  cupertino_icons: ^1.0.2
```

Now you can use SQLite in your Flutter app.

**Initialize Database in Flutter**

Now create DatabaseHelper class. This class will contain the methods to create database and perform all database related operations. Like create database, create table, insert data in tables, fetch data from tables, update and delete data etc. Now first create a method initializeDB().

```
Future<Database> initializeDB() async {
  String path = await getDatabasesPath();
  return openDatabase(
    join(path, "users.db"),
    onCreate: (database, version) async {
      await database.execute(
          "CREATE TABLE users (id INTEGER PRIMARY KEY AUTOINCREMENT,
first_name TEXT NOT NULL, last_name TEXT NOT NULL )");
    },
    version: 1,
  );
}
```

Here we initialize database and create a table in it. The openDatabase method is in sqflite package and is used to open the database connection. It accepts the path of database and a version number and a callback named onCreate. This onCreate method executes once when database is created for first time. Therefore we have created users table in this method. The users table contains id, first_name and second_name columns.

**Model Class**

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


The data In SQLite will be save in Dart Maps data type. So you need to create model class with toMap and fromMap mothods. Here we will do simple example of database operations. So we just create User model class with only three data members that are id, first name and last name.

```
class User {
  int id = 0;
  String firstName = '';
  String lastName = '';

  User.empty();

  User({
    required this.id,
    required this.firstName,
    required this.lastName,
  });

  factory User.fromMap(Map<String, dynamic> json) {
    return User(
      id: json['id'],
      firstName: json['first_name'],
      lastName: json['last_name'],
    );
  }

  Map<String, dynamic> toMap() => {
        'first_name': firstName,
        'last_name': lastName,
      };
}
```

**SQLite CRUD Operations**

**Insert Data in Database**

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


As we have discussed that all the db operations will be performed in database helper class. So to insert data in database we will write method in this class. We are working on users so we will create inserUser(User user) method with user as a parameter.

```
Future<int> insertUser(User user) async {
  final Database db = await initializedDB();
  return await db.insert('users', user.toMap());
}
```

**Retrieve Data from Database**

To fetch all data from database, create a method getAllUser() method in database helper class. Which will return all rows of that table.

```
Future<List<User>> getAllUsers() async {
  final Database db = await initializedDB();
  List<Map<String, dynamic>> result = await db.query('users');
  return result.map((e) => User.fromMap(e)).toList();
}
```

**Delete Data from Database**

To delete a single user form database we will use delete method. Delete method requires three parameters. First one in table name, Second one is for where clause and third one is for arguments.

```
Future<void> deleteUser(int id) async {
  final Database db = await initializedDB();
  db.delete('users', where: 'id= ?', whereArgs: [id]);
}
```

**Update Data in Database**

To update data in database we will use update method. This method requires 4 parameters.


**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


```
Future<void> updateUsingHelper(User user) async {
  final Database db = await initializedDB();
  await db.update('users', user.toMap(), where: 'id= ?', whereArgs: [user.id]);
}
```

**Complete Code of Sqflite CRUD example in Flutter**

**main.dart**

```
import 'dart:io';

import 'package:flutter/material.dart';
import 'package:my_demo/data_base_handler.dart';
import 'package:my_demo/user.dart';
import 'package:my_demo/user_form.dart';
import 'package:my_demo/users_list.dart';
import 'package:sqflite/sqflite.dart';

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

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

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const MyHomePage(),
    );
  }
```

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

```
}

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

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  late Future<List<User>> users;
  TextEditingController firstNameTextController = TextEditingController();
  TextEditingController lastNameTextController = TextEditingController();
  DatabaseHandler dbHandler = DatabaseHandler();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Users'),

      ),
      body: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Column(
          children: [
            TextField(
              controller: firstNameTextController,
              decoration: const InputDecoration(hintText: 'First name'),
            ),
            TextField(
              controller: lastNameTextController,
              decoration: const InputDecoration(hintText: 'Last name'),
            ),
            Container(
```

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

```
  width: double.infinity,
  margin: const EdgeInsets.symmetric(horizontal: 10, vertical: 10),
  child: ElevatedButton(
   onPressed: () async {
     User user = User.empty();
     user.firstName = firstNameTextController.text.toString();
     user.lastName = lastNameTextController.text.toString();

     await dbHandler.insertUser(user);
     initState();
     setState(() {});
   },
   child: const Text('Save'),
  ),
 ),
 Row(children: const <Widget>[
  Expanded(
     child: Divider(
   color: Colors.black,
  )),
 ]),
 Expanded(
  child: FutureBuilder(
     future: users,
     builder: (context, snapshot) {
       if (snapshot.hasData) {
         var usersList = snapshot.data as List<User>;
         return ListView.builder(
            itemCount: usersList.length,
            itemBuilder: (BuildContext context, int index) {
              User user = usersList[index];
              return ListTile(
                title: Text(user.firstName + ' ' + user.lastName),
                trailing: IconButton(
                  onPressed: () {
```

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

```
                        dbHandler.deleteUser(user.id);
                        initState();
                        setState(() {});
                      },
                      icon: const Icon(Icons.delete),
                    ),
                  );
                });
          } else {
            return const CircularProgressIndicator();
          }
        }),
      ),
    ],
   ),
  ),
 );
}

@override
void initState() {
 users = this.getUsersLIst();
}

Future<List<User>> getUsersLIst() async {
 return await DatabaseHandler().getAllUsers();
}
}
```

data_base_handler.dart

```
import 'package:my_demo/user.dart';
import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';
```

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**

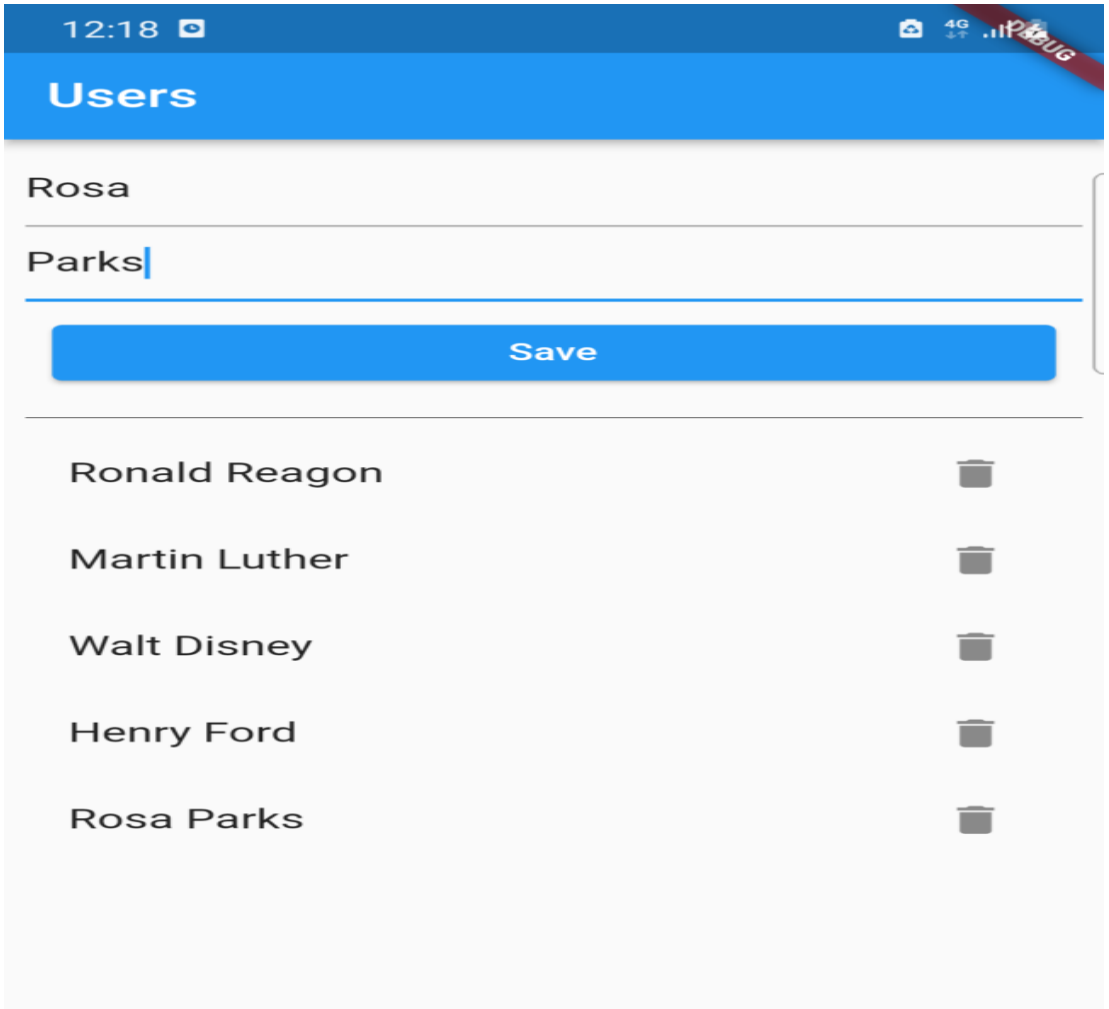

```
class DatabaseHandler {
  Future<Database> initializedDB() async {
    String path = await getDatabasesPath();
    return openDatabase(
      join(path, "users.db"),
      onCreate: (database, version) async {
        await database.execute(
            "CREATE TABLE users (id INTEGER PRIMARY KEY AUTOINCREMENT,
first_name TEXT NOT NULL, last_name TEXT NOT NULL )");
      },
      version: 1,
    );
  }

  Future<int> insertUser(User user) async {
    final Database db = await initializedDB();
    return await db.insert('users', user.toMap());
  }

  Future<List<User>> getAllUsers() async {
    final Database db = await initializedDB();
    List<Map<String, dynamic>> result = await db.query('users');
    return result.map((e) => User.fromMap(e)).toList();
  }

  Future<void> deleteUser(int id) async {
    final Database db = await initializedDB();
    db.delete('users', where: 'id= ?', whereArgs: [id]);
  }

  Future<void> updateUsingHelper(User user) async {
    final Database db = await initializedDB();
    await db.update('users', user.toMap(), where: 'id= ?', whereArgs: [user.id]);
  }
```

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

```
}
```

**Output**




**Sha**

**Q-4 Explain Shared preferences and secure storage in Flutter.**

**Answer:**

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


- Storage is a very important aspect of any app. You will need to store some kind of data in any app that you make. Unless and until it is just a 1 or 2-page apps like a calculator or weather app.
- There are 2 major ways to store data. One is to use databases and call them via APIs or fetch data directly from them. The second is to make use of the local storage of your android devices.
- SQLite
- SQLite is a SQL database that can be used to store data locally. It is very useful if you want to store a large amount of data in an ordered manner. One example would be a to-do list.
- Go read these official docs to get an idea of how you can use SQLite to persist data locally.
- There are some cases where we don't need SQLite. For example, if we just need to store something like dark mode or light mode of the app, why use a table like SQL database? We have different options to store such kinds of data. One of them is shared preferences.

- Shared Preferences
- Shared Preferences is one of the most popular (if not the most popular) ways to store data locally. It uses the concept of key-value pairs to store data locally. We can store integers, doubles, floats, and strings with Shared Preferences. It is a great tool if we want to store some user preferences locally. For example, app theme.
- Go read this article to know how to use Shared Preferences to store your data locally.

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


- What if I wanted to store the login status of a user? Shouldn't Shared Preferences be the best way to do it? Umm, I would say no. I am saying no because shared preferences don't encrypt the data it is storing.
- Suppose I want to save some kind of access token locally which verifies if the user is logged in or not, it can be unsafe to store it unencrypted. So this is where Secure Storage comes in. Let's see what it is.

- Secure Storage
- The concept of Secure Storage is similar to shared preferences, i.e store data as key-value pairs. The only difference is that in secure storage, the stored data is encrypted.


- Here is the documentation that will guide you on how to use secure storage.
- Hive
- Hive is a local data storage that is very popular among Flutter developers. The two main reasons for its popularity are that it is a NoSQL database and it can not only store primitive data types, but it can also store Dart classes.
- So yeah, that is a big deal. You can create a class and store it as a NoSQL document in the hive database. It can store almost the same amount of data compared to SQLite. Hive is used for caching text messages of a particular chat, for caching the profile details of the user, and much more complex data like this.
- Hive is definitely an upgrade over Shared Preferences and Secure Storage in terms of data storing size and if you prefer NoSQL over SQL, then Hive is the way to go.
- Hive has amazing documentation. Read it if you want to set up Hive in your app and use it as your local database.

**Q-5 Explain WebSockets for real-time in Flutter.**

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


**Answer:**

- **Mobile app development** nowadays requires real-time data to offer rapid responses to users, whether it is a chat application that displays a person typing in real time or a distant application that plots data directly from a hardware sensor.
- We try to fix these concerns with REST, but we run into a tricky problem: to get near-instant input, we must ping the server multiple times per minute, which can be tough to do architecturally and overload the server.
- When utilizing solutions such as Firebase Realtime Database, you will notice that whenever a new record is added to the database, the Flutter application gets it as a **Stream** and displays the data to the user.

## What is WebSocket API?

The WebSocket API, according to Mozilla, is "a sophisticated technology that allows for the establishment of a two-way interactive communication session between the user's browser and a server… You may send messages to a server and obtain event-driven answers without polling the service."

WebSockets are made up of the following components:

- A server that broadcasts data
- A **client** in the application that is prepared to receive the new data stream
- A communication **channel** between the **client** and the server.
- **Messages** transmitted back and forth between the **client** and the server

Unlike REST, we do not wait for a response from the server after sending a message to it with WebSockets. We can send a single message and receive hundreds of responses from the server.

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**

In some ways, it's similar to subscribing to notifications; we subscribe to a certain topic, such as the USD-EUR exchange rate, and then we receive a new message from the server each time the USD-EUR exchange rate changes.

WebSockets' real-time communication stream makes it the ideal solution for stock exchange apps, chat apps, IoT apps, and any other app that requires an incoming stream of data.

## Using WebSockets with Flutter in Apps

Thankfully, Dart, Flutter's language, includes an out-of-the-box solution for dealing with WebSockets: **The WebSocket class.**

We can take use of **WebSocket** securely if we design apps for only one target (desktop, web, or mobile).

Nevertheless, if we choose to utilize our app on several platforms, we must keep in mind that the **WebSocket** class is dependent on dart:io and dart:html, which means we cannot build for both mobile and web at the same time.

Fortunately, the Dart team produced the **web_socket_channel**, an official library that encapsulates the **dart:io** and **dart:html** functionality and allows us to construct a multiplatform application with a single class.

We must take three easy actions to use the **web_spclet_channel**:

- Build a new **WebSocketChannel** client and **connect** to a channel using the connect method.
- With the **stream** getter, you may listen in on incoming messages.
- Send messages to the server using the **sink** getter.

Here are the steps for building real-time apps with Flutter and WebSockets.

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


## Steps to build a real-time app using Flutter and WebSocket

Flutter is a powerful and popular framework for building mobile and web applications. Real-time apps require real-time data synchronization, which can be achieved using WebSocket, a protocol for real-time communication between a client and a server.

Here are the steps to build a real-time app with **Flutter app development** and WebSocket:

Create a new Flutter project:

To create a new project using Flutter, follow the command mentioned below:

```
1 flutter create my_realtime_app
```

Add dependencies:

Add the web_socket_channel package to the pubspec.yaml file. This package provides a high-level API for working with WebSocket.

```
1 dependencies:
2   flutter:
3     sdk: flutter
4   web_socket_channel: ^2.1.0
```

Create a WebSocket connection:

To create a WebSocket connection, use the WebSocketChannel.connect method, which returns a WebSocketChannel object. You can create the connection in the initState method of your widget:

```
1  class MyRealtimeApp extends StatefulWidget {
2
3    @override
```

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

```
4
5   _MyRealtimeAppState createState() => _MyRealtimeAppState();
6
7  }
8
9  class _MyRealtimeAppState extends State<MyRealtimeApp> {
10
11   WebSocketChannel channel;
12 @override
13   void initState() {
14
15     super.initState();
16
17     channel = WebSocketChannel.connect(Uri.parse('ws://localhost:8080'));
18   }
19
20 @override
21   Widget build(BuildContext context) {
22
23     // ...
24   }
25 @override
26   void dispose() {
27     channel.sink.close();
28     super.dispose();
29   }
30 }
```

In this example, we create a WebSocket connection to ws://localhost:8080. You can replace this with the URL of your WebSocket server.

Send and receive messages:

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


Once you have a WebSocket connection, you can use the sink and stream properties of the WebSocketChannel object to send and receive messages.

```
class _MyRealtimeAppState extends State<MyRealtimeApp> {
  WebSocketChannel channel;
  TextEditingController _controller = TextEditingController();
  @override

  void initState() {
    super.initState();
    channel = WebSocketChannel.connect(Uri.parse('ws://localhost:8080'));
  }
@override

  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
       title: Text('Realtime App'),
      ),
      body: Column(
        children: <Widget>[
          Form(
            child: TextFormField(
     controller: _controller,
              decoration: InputDecoration(labelText: 'Send a message'),
            ),
          ),
          StreamBuilder(
            stream: channel.stream,
            builder: (context, snapshot) {
              if (snapshot.hasData) {
             return Text(snapshot.data);
              } else {
                return Text('No data');
              }
```

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

```
33            },
34          ),
35        ],
36      ),
37
38      floatingActionButton: FloatingActionButton(
39        onPressed: _sendMessage,
40
41        tooltip: 'Send message',
42
43        child: Icon(Icons.send),
44    ),
45    );
46   }
47 void _sendMessage() {
48     if (_controller.text.isNotEmpty) {
49       channel.sink.add(_controller.text);
50     }
51   }
52
53 @override
54   void dispose() {
55     channel.sink.close();
56     super.dispose();
57   }
58 }
```

In this example, we create a simple UI with a text input and a text view to display the received messages. We use the StreamBuilder widget to listen to incoming messages and update the UI accordingly.

The _sendMessage method sends the message entered in the text input to the WebSocket server.

Run the app:

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


To run the app, use the following command:

1 flutter run

This will launch the app on an emulator or a connected device.

After running a Flutter app using the command flutter run, the next step would be to test and interact with the app to verify that it is working as intended.

**Q-6 Write note on communication in Flutter.**

**Answer:**

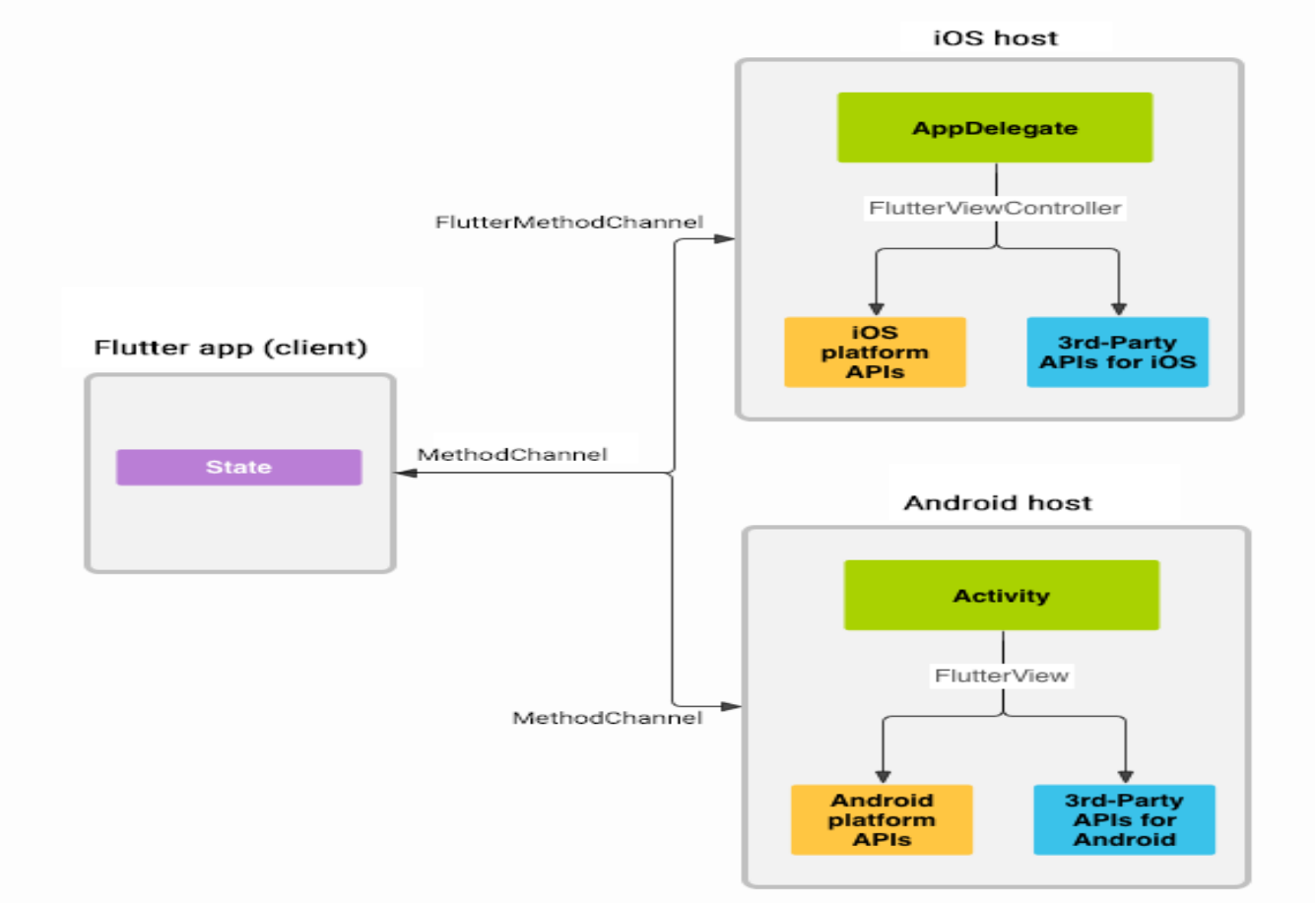
- An Android Plugin for Serial Communication which allow you to read and write the data through the available ports The supported features are:

- Listing the available serial ports on the device, including USB to serial adapters
- Configuring serial ports (baud rate, stop bits, permission, ...)
- Providing standard InputStream and OutputStream

**communicate between Flutter and Native**

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645


Sometimes we need to communicate with native in Flutter. We need to use Platform Channels APIs for communication, mainly including the following three types:

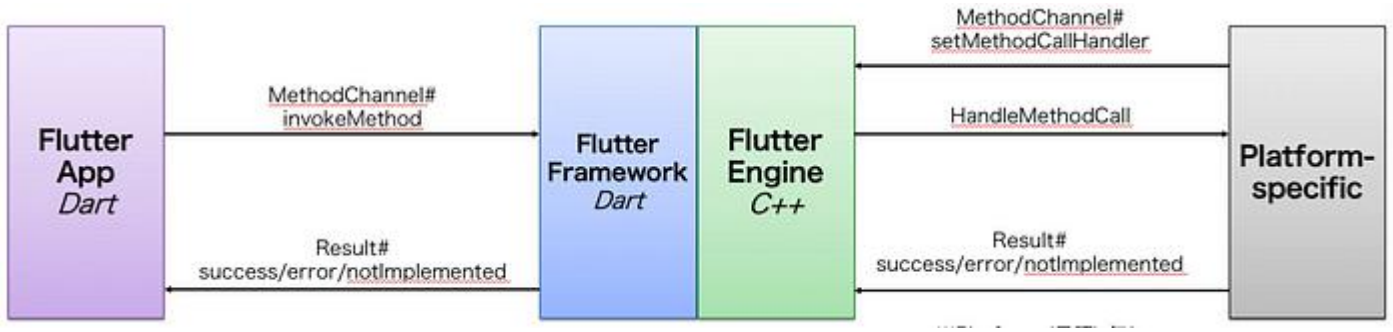
- MethodChanel：used for method invocation.
- EventChannel：used for communication of event streams.
- BasicMessageChannel：used for communication of message.

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


## MethodChannel

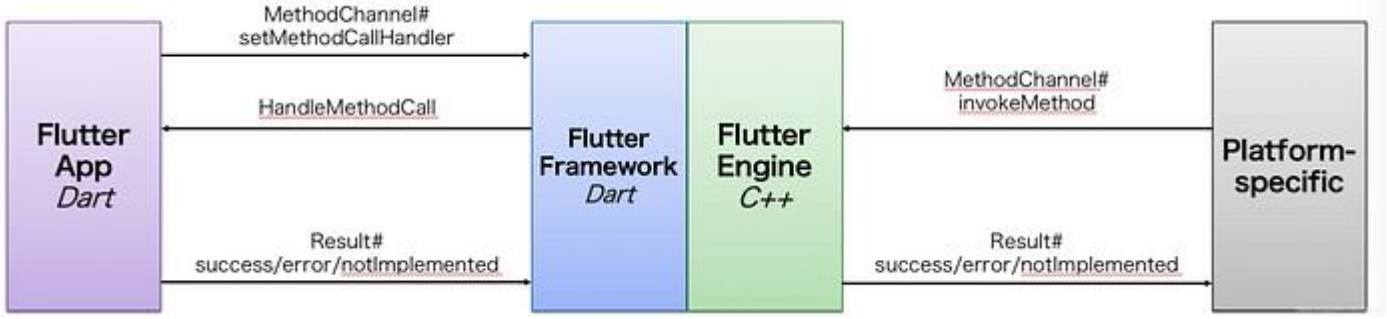
*Flutter calls native*




First, the following functions are implemented in the flutter:

- Create a MethodChannel and register the channel name, generally using ***"package name/identity"*** as the channel name.
- An asynchronous call is initiated through ***invokeMethod***.

*Native calls flutter*



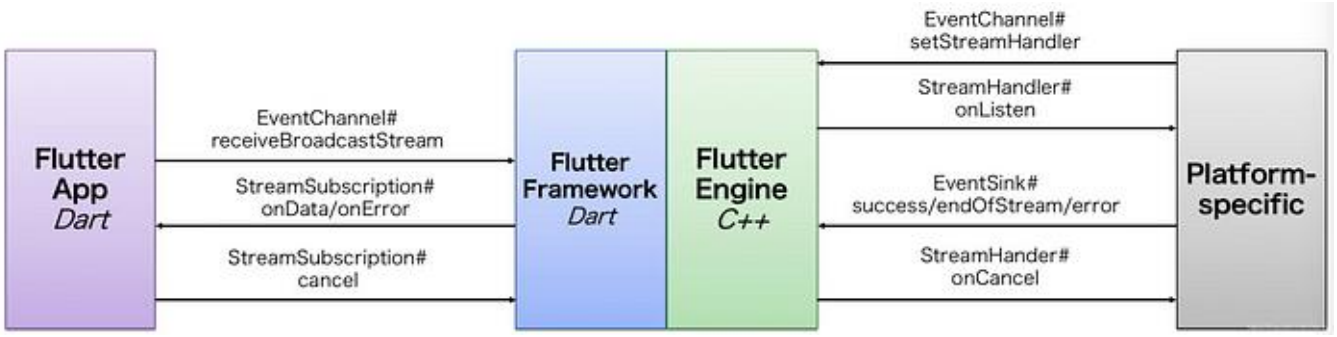
**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


The code implementation of android calling flutter is similar to that of flutter calling native (android) which via ***invokeMethod***.

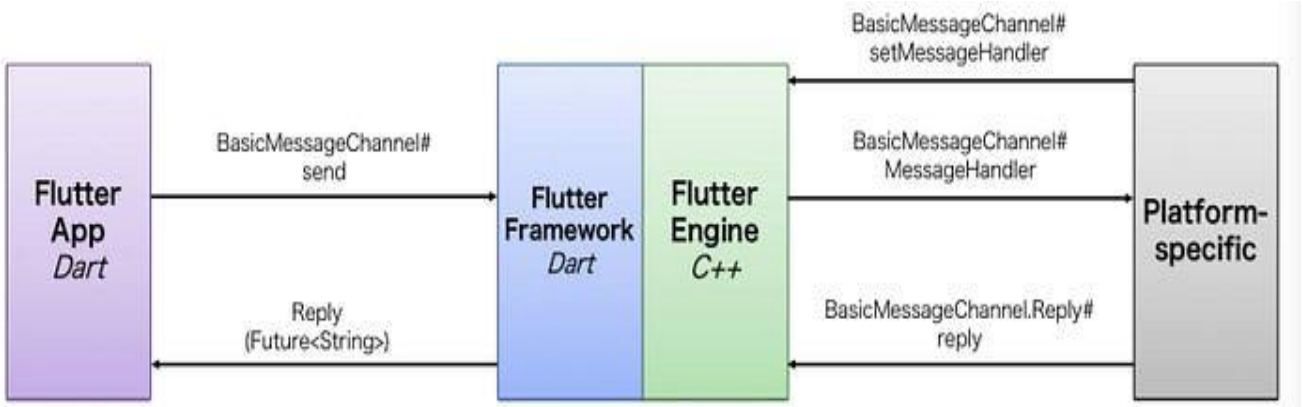
## EventChannel

It is used to send notification events from native to flutter. Unlike MethodChannel, EventChannel is a one-way call from native to flutter. The call is multicast (one-to-many).




## BasicMessageChannel

It is used to send messages between flutter and native, one party sends a message to the other, and gives a reply after receiving the message.*Flutter sends message to native*



# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

**(AFFILIATED TO SAURASHTRA UNIVERSITY)**

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


## Flutter Live Streaming and Real-time Communication packages

Live streaming or Realtime communication is a mode of communication in which one or more users can exchange information instantly or with negligible latency or transmission delays. This means that there is no delay between when someone speaks and when the other person hears what they said. It is useful for voice over IP (VoIP), video streaming, teleconferencing, video calling, file sharing, screen sharing, multiplayer gaming, etc.

Flutter ecosystem provides various packages that can help you add the following features in your Flutter app:

1. **Real-Time Video Streaming**: Seamlessly integrate live video streaming capabilities into apps.

2. **Interactive Chat and Messaging**: Real-time chat functionality for users to communicate during live streams with support for text, emoji, and multimedia messages.

3. **Multi-Platform Support**: 3rd party SDKs and APIs for integrating the real-time communication across web, mobile (iOS and Android), and desktop applications.

4. **Customizable UI Components**: Pre-built UI components for video player, chat interface, and user controls. Customizable themes and styling options to match the app's design.

5. **Audience Engagement**: Feeds, Polls, Q&A sessions, and interactive features to engage the audience during live streams. Integration with social media sharing for broader reach.

6. **Multi-Host Collaboration**: Support for multiple hosts or presenters in live streams, enhancing panel discussions or collaborative events.

7. **Recording and Playback**: Ability to record live streams for later playback and on-demand viewing with the option to store recordings locally or on cloud storage.

8. **Real-Time Notifications**: Push notifications to alert users about upcoming live streams or when their favorite creators go live.

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

9. **Low Latency Streaming**: Ultra-low latency streaming options for real-time interaction between hosts and viewers.

10. **Screen Sharing and Presentation**: Screen sharing functionality for presentations, demonstrations, and collaborative work.

11. **Third-Party Integration**: Integration with popular platforms, APIs, and services for enhanced functionality (e.g., integrating with social media platforms, analytics tools, payment gateways).

**Q-7 Write note on Firebase integration for data storage in Flutter.**

**Answer:**

- **Flutter Firebase** is a powerful combination of **Flutter(Google's open-source UI toolkit)** and **Firebase (a backend-as-a-service platform)**. By integrating Flutter's expressive UI framework with Firebase's backend services, developers can efficiently build cross-platform applications.

- **Firebase** can be integrated and used in synergy with Flutter to essentially empower Flutter. Firebase offers features like authentication, real-time database, cloud storage, and hosting, enabling seamless integration with Flutter for creating visually appealing, scalable, and real-time applications.

- **Firebase Realtime Database** offers a cloud-hosted solution for storing data in JSON format, providing developers with a secure and efficient backend database. With Firebase Realtime Database, developers can build collaborative and responsive applications directly from the client side.

**Firebase Storage**

**Firebase Storage** offers secure and efficient cloud storage for your Flutter application. It enables you to store and serve user-generated content such as images, videos, and audio

files. With Firebase Storage, you can easily upload and download files, manage permissions, and generate secure URLs for sharing files. It integrates seamlessly with other Firebase services, making it simple to combine storage capabilities with other functionalities like Firebase Authentication or Firebase Cloud Firestore. Firebase Storage ensures scalability, durability, and high performance, allowing you to handle file storage efficiently in your Flutter application.

## Integrating Firebase Services with Flutter

### Adding Firebase Packages to Flutter

To integrate Firebase services into your Flutter project, you need to add the necessary Firebase packages as dependencies. These packages include firebase_core (required for all Firebase services) and additional packages based on the specific services you intend to use, such as firebase_auth for authentication or cloud_firestore for the Firestore database. Open your project's pubspec.yaml file and add the desired packages under the dependencies section. Then, run flutter pub get to fetch the packages and make them available in your project.

**pubspec.yaml file:**

```
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^2.14.0
  firebase_auth: ^4.6.3
  cloud_firestore: ^4.8.2
  firebase_messaging: ^14.6.4
```

### Initializing Firebase Services

Before using any Firebase service, you need to initialize Firebase in your Flutter application. This initialization process ensures that your application establishes a connection with Firebase backend services. In your app's entry point (usually main.dart), import the firebase_core package and call Firebase.initializeApp() inside the main() function. This method returns a Future which indicates when the initialization

is complete. You can use FutureBuilder to handle the initialization process and ensure that Firebase services are ready to be used.

**main.dart file:**

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

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();

  runApp(MyApp());
}
```

## Implementing Firebase Services in Flutter

Once Firebase is initialized, you can implement various Firebase services in your Flutter application.

- **Firebase Authentication:**
  To implement Firebase Authentication, you can utilize the firebase_auth package. This package provides APIs to handle user authentication and authorization. You can integrate features such as email/password authentication, social sign-in, and phone number authentication. By following the Firebase Authentication documentation, you can implement the desired authentication flows and handle user sessions in your Flutter app.

- **Firebase Cloud Firestore:**
  To use Firebase Cloud Firestore, add the cloud_firestore package to your project. Firestore provides real-time, scalable, and cloud-based NoSQL database functionality. You can perform CRUD operations, query data, and listen for real-time updates. By using the Firestore API, you can store and retrieve data from Firestore collections and documents, implement data synchronization, and leverage Firestore's security rules to control data access.

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

- **Firebase Cloud Messaging:**
  For integrating **Firebase Cloud Messaging (FCM)**, include the firebase_messaging package. You can configure the necessary Firebase console settings and handle incoming notifications in your app. Implement the required logic to receive and handle FCM messages, and customize the notification display to provide a rich notification experience to your users.

- **Firebase Realtime Database:**
  To include Firebase Realtime Database in your Flutter app, add the firebase_database package to your project. The **Realtime Database** is a cloud-hosted NoSQL database that allows you to synchronize data between clients in real time. You can store and retrieve data as JSON and listen for changes using event-based callbacks. The Realtime Database is suitable for building real-time collaborative applications, chat features, and more.

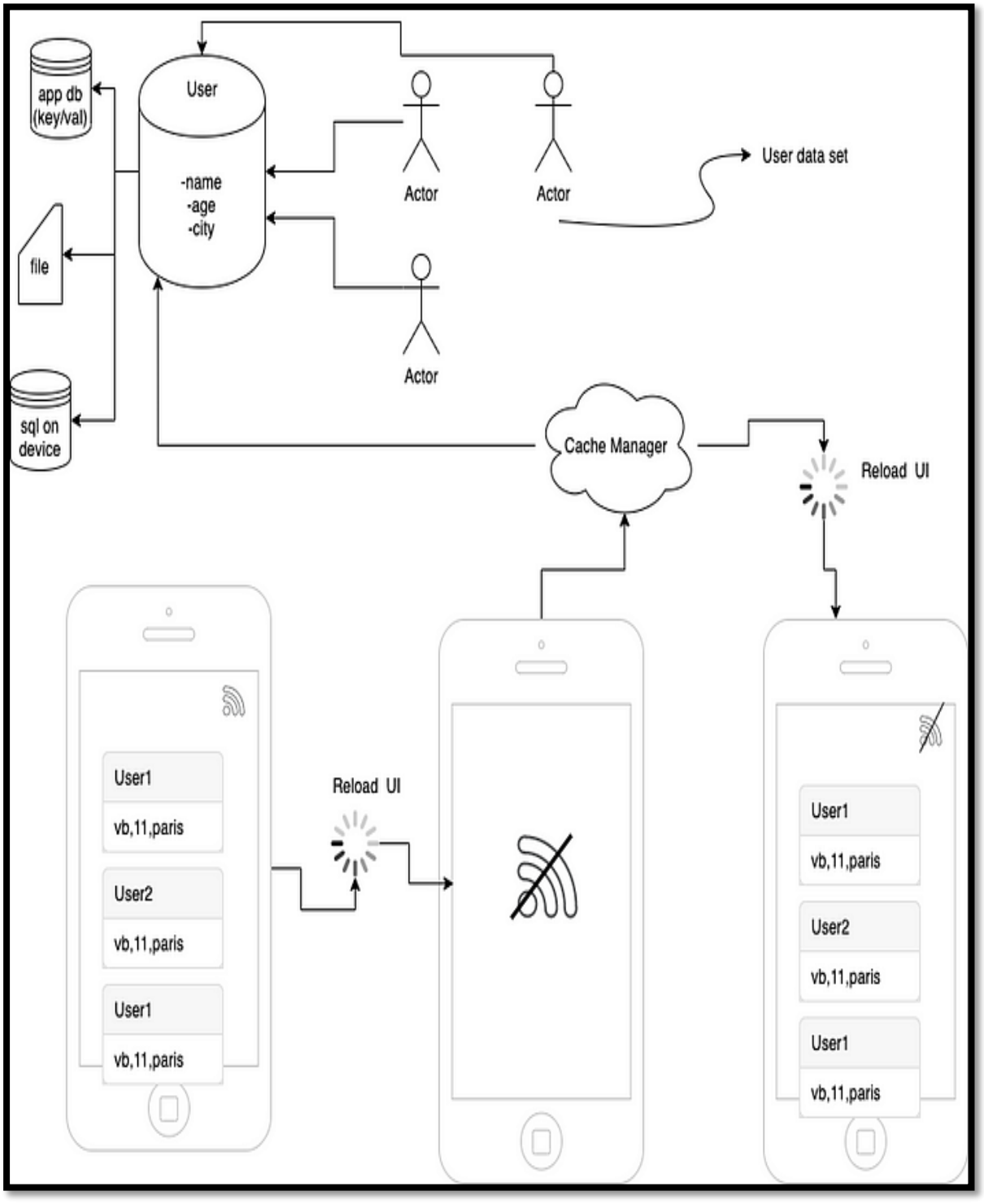
**Q-8 Write note on Caching data in Flutter.**

**Answer:**

- In Flutter, caching refers to storing data on disk or in memory and retrieving it from there when needed. This improves performance by avoiding repetitive network operations. Caching Strategies: a. Memory Cache: Flutter offers the MemoryCache class for storing data temporarily in memory.

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

- Caching is a mechanism implemented by a system or a system creator that temporarily stores data in available memory. It makes it easier to access and retrieve that information stored. Cache memory is made available for system use in a computer system, and data currently being used by the user while operating the system are sometimes cached.
- In a situation where there is an HTTP request made to a server for a stale resource (data that does not change) when a user switches between screens in your application, caching is vital in such situations, which improves the user experience.

Users do not have to wait for data to be fetched every time they switch between the screens. For data that may not be stable, you can give the user the option to refresh the data fetched and update the cached records if new data is present.

Benefits of caching

- Caching saves resources.
- Caching gives users a better experience.
- Caching makes your application work faster and better.
- It helps you create efficient applications.

Caching implementation

To show a sample of this, we will perform an HTTP request in our flutter application to fetch data when the user navigates to a new screen where its contents are fetched.

***Step one: Setting up our application***

In your main.dart file set up your flutter application to display a button that navigates to a new screen when clicked.

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


```
import 'package:flutter/material.dart';
void main() {
 runApp(const MyApp());
}
class MyApp extends StatelessWidget {
 const MyApp({Key? key}) : super(key: key);
  @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Caching',
     theme: ThemeData(

       primarySwatch: Colors.blue,
     ),
     Home: Scaffold(
 appBar: AppBar(

   title: Text('Caching'),
 ),
 body: CountriesSelect(),
       );
   );
 }
}
```

We have a basic flutter application setup with an app bar with the title caching in the above snippet. It has a body and a container that has nothing within it.

- **Q-9 Explain How to use third-party libraries for networking and data storage (e.g. Dio, Hive) in Flutter.**

  **Answer:**

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

The storage of the data locally is a requirement for almost every app. The storage and manipulation of information is a crucial part of app development, and the same holds for Flutter apps. Perhaps you want to cache REST API responses, build an application that runs offline or store customer information for a food delivery app.

Several options are available for developers to persist local data in Flutter.shared_preferences: Provides a good way to store small pairs of keys and values .sqflite : It's a good choice when your database must handle complex relationships between relational data.

## What is Hive in Flutter?

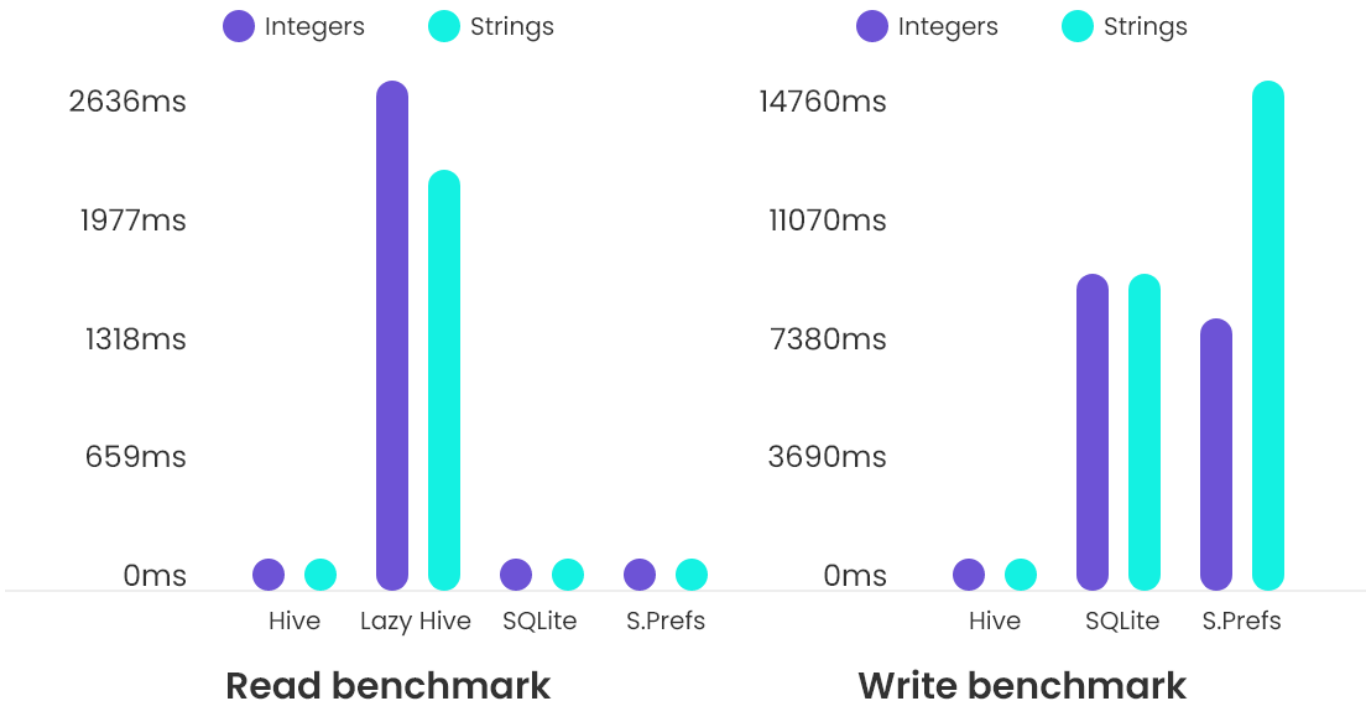
Hive is a lightweight and blazing fast key-value database written in pure Dart, which allows you to store and sync application data offline.

As a key-value data store written in Dart, Hive supports primitive and complex data structures while providing the highest level of performance.

As an example, here is a graph that compares Flutter Hive to other similar databases:

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

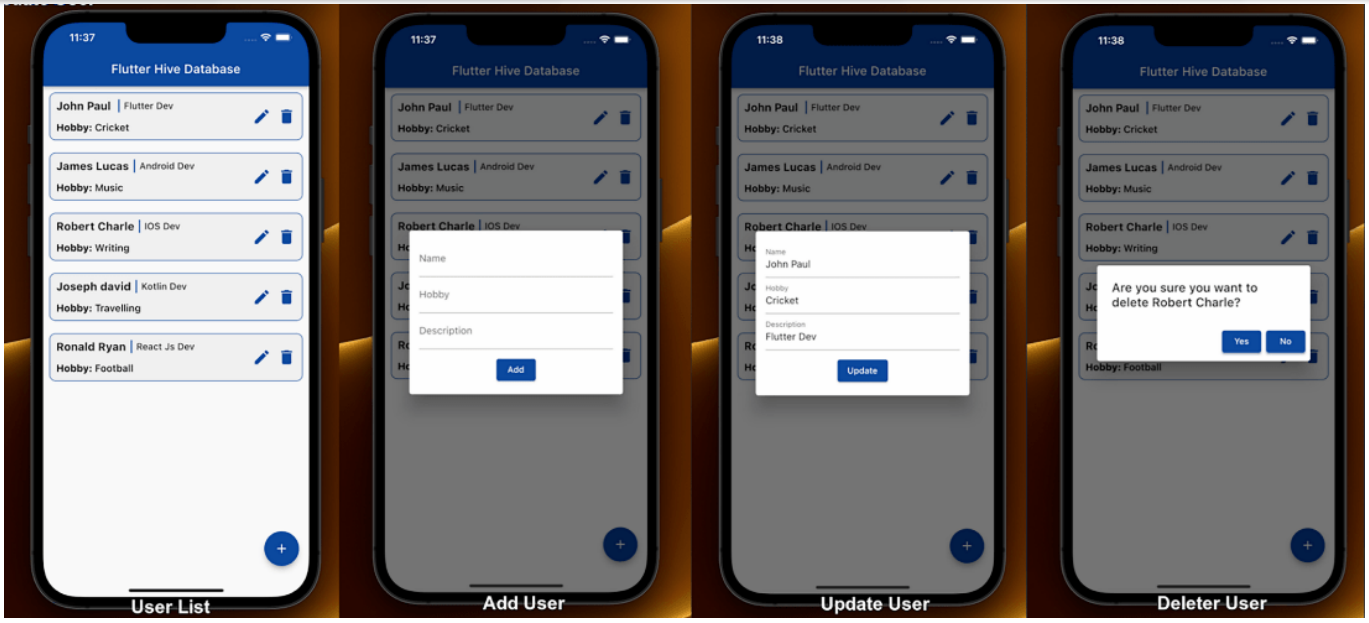



## Getting Started with Handle Offline Data Storage with Flutter Hive

In this blog, we will explore the Hive DataBase With TypeAdapter In Flutter. We will also create a simple app with only one page that shows a list of users, add new users, update existing users, and delete users.

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645


## How to use Flutter Hive to Handle Offline Data Storage?

### Step 1: Dependency installation

- Two dependencies are required before we can use Hive.

hive and hive_flutter

- You need to add the Hive and hive_flutter packages to pubspec.yaml as follows:

```
dependencies:
 Flutter:
      sdk: flutter
 hive: ^2.2.3
 hive_flutter: ^1.1.0
```

Add the dev dependencies

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

```
dev_dependencies:
 flutter_test:
   sdk: flutter
hive_generator: ^1.1.3
build_runner: ^2.2.0
```

**Step 2: Initialization Hive Database**

First, we must initialize Hive before calling runApp in the flutter app.

```
void main() async{
 WidgetsFlutterBinding.ensureInitialized();
   // Initializes Hive with a valid directory in your app files
 await Hive.initFlutter();
 runApp(const MyApp());
}
```

The initFlutter() function is provided by Hive.Basically, it initializes Hive by using the path returned by getApplicationDocumentsDirectory

**Box in Hive**

Here's how to *handle offline data storage with Flutter Hive*.

**2 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot – 360001**
**Ph.No–(0281)2440478**

**3 – Vaishalinagar**
**Nr. Amrapali Under Bridge**
**Raiya Road**
**Rajkot - 360001**
**Ph.No–(0281)2471645**


The data stored in Flutter Hive is organized into boxes. The box is similar to a table in SQL, but it has no structure and can hold anything. As I mentioned in the introduction, Hive encrypts data.Additionally, encrypted boxes can be used to store sensitive information.

Using key-value sets, Hive stores its data. First of all, you need to open your box.

```
void main() async{
 WidgetsFlutterBinding.ensureInitialized();
// Initializes Hive with a valid directory in your app files
 await Hive.initFlutter();
// open box
await Hive.openBox("userBox");
runApp(const MyApp());
}
```

## CRUD operations

### Creating Data in Hive

You can use the reference to the Hive box to add data by calling add() function.A key-value pair is accepted by this method.

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

```
/// Add new user
Future addUser({required UserModel userModel}) async {
 await box.add(userModel);
}
```

**Retrieving Data in Hive**

Box objects can be read by using the get() method. To retrieve its value, you simply need to provide the key, like this

```
var userHobby = box.get('hobby');
```

**Updating Data in Hive**

The put() method can update the data you originally stored for a key.In this way, the newly provided value will be updated at that key.

```
/// update user data
Future updateUser({required int index,required UserModel userModel}) async {
 await box.putAt(index,userModel);
}
```

Here we have used the auto-incrementing values, you can use the putAt(index) method of the box object to update using the index.

2 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot – 360001
Ph.No–(0281)2440478

3 – Vaishalinagar
Nr. Amrapali Under Bridge
Raiya Road
Rajkot - 360001
Ph.No–(0281)2471645

## Deleting Data in Hive

In order to delete data, you can pass the key to the delete() method.

```
/// delete user
Future deleteUser({required int index}) async {
 await box.deleteAt(index);
}
```

Here we have used the auto-incrementing values, you can use the deleteAt(index) method of the box object to delete using the index.