**Experiment: 5**

**AIM:** To learn data handling using APIs by integrating a mobile expense trackerapp with a backend API

**THEORY:** APIs (Application Programming Interfaces) allow mobile applications to interact with external services efficiently. In this experiment, we implement an API using Flask to manage cart operations while storing product data in Firebase Firestore.

### **Key Components:**

1. **Flutter (Frontend):** Used to create the user interface and send requests to the API.
2. **Firebase Firestore:** Stores product details but does not manage cart operations directly.
3. **Flask (Backend API):** Handles cart operations such as adding, retrieving, and removing items.
4. **Postman (API Testing):** Used to demonstrate API request handling.

### **Implementation Steps:**

#### **1. Setting Up Firebase Firestore for Product Data**

* Create a Firestore database.
* Add a products collection.
* Each product document contains fields like name ,price ,id .

#### **2. Developing the Flask API**

Flask serves as a simulated backend for cart management. The API includes:

* **POST /add-to-cart** – Adds an item to the cart.
* **GET /cart** – Retrieves all cart items.
* **DELETE /remove-from-cart** – Removes an item from the cart.

| **from flask import Flask, request, jsonify**  **app = Flask(\_\_name\_\_)**  **cart = [] # Temporary storage for cart items**  **@app.route('/add-to-cart', methods=['POST'])**  **def add\_to\_cart():**  **data = request.json**  **cart.append(data)**  **return jsonify({"message": "Item added to cart", "cart": cart})**  **@app.route('/cart', methods=['GET'])**  **def get\_cart():**  **return jsonify({"cart": cart})**  **@app.route('/remove-from-cart', methods=['DELETE'])**  **def remove\_from\_cart():**  **data = request.json**  **global cart**  **cart = [item for item in cart if item['id'] != data['id']]**  **return jsonify({"message": "Item removed", "cart": cart})**  **if \_\_name\_\_ == '\_\_main\_\_':**  **app.run(debug=True)** |
| --- |

#### **3. Integrating API with Flutter**

* Use the http package to send requests to the API.
* Implement functions to send API calls for adding, retrieving, and deleting cart items.
* Update the UI to reflect cart changes dynamically.

#### **4. Testing API Using Postman**

* Send a **POST** request to /add-to-cart with product details in JSON format.
* Retrieve cart items using a **GET** request to /cart.
* Remove an item using a **DELETE** request to /remove-from-cart

#### **1. Add Item to Cart (POST /add-to-cart)**

{

"id": "1",

"name": "Harmonic Serenity Necklace",

"price": "41999",

"quantity": 1

}

#### **2. Get Cart Items (GET /cart)**

* No JSON body required. Simply send a **GET** request to /cart.

#### **3. Remove Item from Cart (DELETE /remove-from-cart)**

{

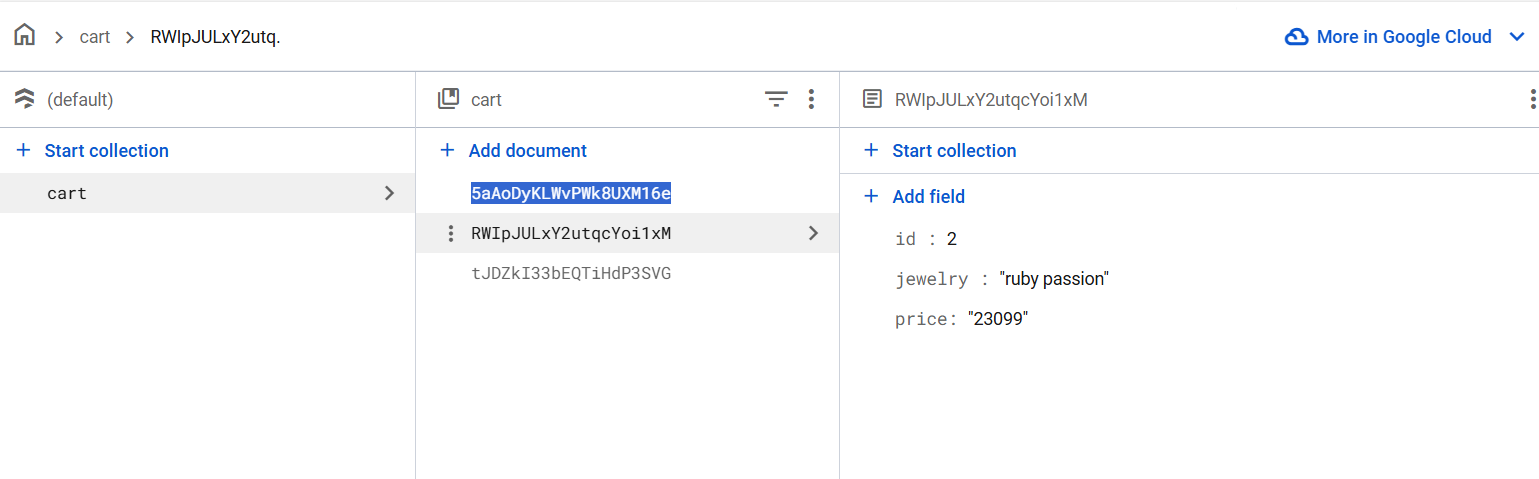
"id": "1"

}

**code:**

| import 'dart:convert';  import 'package:flutter/material.dart';  import 'package:http/http.dart' as http;  class CartItem {  final String id;  final String name;  final String imageUrl;  final double price;  int quantity;  final String caret;  final String size;  final String color;  CartItem({  required this.id,  required this.name,  required this.imageUrl,  required this.price,  this.quantity = 1,  required this.caret,  required this.size,  required this.color,  });  Map<String, dynamic> toMap() {  return {  'id': id,  'name': name,  'imageUrl': imageUrl,  'price': price,  'quantity': quantity,  'caret': caret,  'size': size,  'color': color,  };  }  factory CartItem.fromMap(Map<String, dynamic> map) {  return CartItem(  id: map['id'],  name: map['name'],  imageUrl: map['imageUrl'],  price: map['price'],  quantity: map['quantity'],  caret: map['caret'],  size: map['size'],  color: map['color'],  );  }  }  class CartProvider with ChangeNotifier {  List<CartItem> \_items = [];  final String apiUrl = "http://127.0.0.1:5000";  List<CartItem> get items => \_items;  double get totalAmount {  return \_items.fold(0, (sum, item) => sum + (item.price \* item.quantity));  }  Future<void> fetchCart() async {  try {  final response = await http.get(Uri.parse('$apiUrl/cart'));  if (response.statusCode == 200) {  final List<dynamic> data = json.decode(response.body);  \_items = data.map((item) => CartItem.fromMap(item)).toList();  notifyListeners();  }  } catch (e) {  print("Error: $e");  }  }  Future<void> addItem(CartItem item) async {  try {  final response = await http.post(  Uri.parse('$apiUrl/add-to-cart'), | return CartItem(  id: map['id'],  name: map['name'],  imageUrl: map['imageUrl'],  price: map['price'],  quantity: map['quantity'],  caret: map['caret'],  size: map['size'],  color: map['color'],  );  }  }  class CartProvider with ChangeNotifier {  List<CartItem> \_items = [];  final String apiUrl = "http://127.0.0.1:5000";  List<CartItem> get items => \_items;  double get totalAmount {  return \_items.fold(0, (sum, item) => sum + (item.price \* item.quantity));  }  Future<void> fetchCart() async {  try {  final response = await http.get(Uri.parse('$apiUrl/cart'));  if (response.statusCode == 200) {  final List<dynamic> data = json.decode(response.body);  \_items = data.map((item) => CartItem.fromMap(item)).toList();  notifyListeners();  }  } catch (e) {  print("Error: $e");  }  }  Future<void> decreaseQuantity(String id) async {  final item = \_items.firstWhere((i) => i.id == id);  if (item.quantity > 1) {  item.quantity--;  } else {  \_items.removeWhere((i) => i.id == id);  }  notifyListeners();  await updateCart();  }  Future<void> updateCart() async {  try {  final response = await http.post(  Uri.parse('$apiUrl/update-cart'),  headers: {'Content-Type': 'application/json'},  body: json.encode({'cart': \_items.map((item) => item.toMap()).toList()}),  );  if (response.statusCode == 200) {  notifyListeners();  }  } catch (e) {  print("Error: $e");  }  }  } |
| --- | --- |

### **OUTPUT:**



* Screenshots of Firestore product storage.
* API responses in Postman for add, fetch, and delete operations.
* Flutter UI demonstrating cart interactions.

### **CONCLUSION:**

In this experiment, we successfully implemented a cart management system using a REST API. The Flask API facilitated structured interaction between the mobile app and cart operations, while Firebase Firestore stored product details. This setup simulates how real-world e-commerce applications manage cart data through backend services.