UNIVERSITY of TECHNOLOGY, MAURITIUS

School of Innovative Technologies and Engineering

BSc (Hons) Software Engineering
Academic Year 3(Semester 1)

Module: Advance Mobile Application & Development

Assignment2b " Hotel Room GRUD + Firebase"

Name (Student ID): Maheswaren CHINNASAMY (2203_24229)

Cohort: BSE/22A/FT

Submitted To: Mr Aneerav Sukhoo

Submission Date: 8th August 2024

Table of Contents

Hotel Booking System with Firebase and Authentication Integration

Introduction

This Flutter application is designed for managing hotel room bookings. The application integrates Firebase Firestore for real-time data synchronization and SQLite for local database operations and with firebase authentication. Below, I provide a detailed report on the implementation, along with code snippets and screenshots to elucidate the workflow and integration of Firebase with the Flutter application.

Firebase Setup

1. Firebase Initialization:

Firebase is initialized in the main function using **Firebase.initializeApp** with **DefaultFirebaseOptions.currentPlatform.**

Main.dart :

```dart
import 'package:auth_firebase/firebase_options.dart';
import 'package:auth_firebase/pages/login/login.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'pages/signup/signup.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();

  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform
  );
```

These import statements bring in necessary packages for building a Flutter app, connecting to Firebase, and accessing the cloudFire database.

- **WidgetsFlutterBinding.ensureInitialized()**: Ensures that Flutter bindings are initialized before using any Flutter APIs.
- **Firebase.initializeApp**: Initializes Firebase with the platform-specific options provided.
- **Imports**: It imports necessary packages, including Firebase options, the login page, Firebase initialization, and Flutter's material design components.
- **main() Function**: The entry point of the application. It ensures that Flutter's bindings are initialized before anything else, then asynchronously initializes Firebase with platform-specific options.
- **MyApp Class**: A stateless widget that represents the root of the app. It disables the debug banner and sets the home screen to the Login page.

Flutter Application Setup

2. Flutter App Structure:

- The app consists of **MyApp** which sets up the Material theme and home screen.
- **ProductPage** is the main screen for CRUD operations on hotel rooms.

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ProductPage(),
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.blue,
        textTheme: TextTheme(
          headline1: TextStyle(fontSize: 32, fontWeight: FontWeight.bold, color: Colors.b
          bodyText1: TextStyle(fontSize: 18, color: Colors.black),
        ),
        inputDecorationTheme: InputDecorationTheme(
          border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(8),
            borderSide: BorderSide(color: Colors.blue),
          ),
          focusedBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(8),
            borderSide: BorderSide(color: Colors.blue),
          ),
          labelStyle: TextStyle(fontSize: 22, color: Colors.blue),
        ),
        elevatedButtonTheme: ElevatedButtonThemeData(
          style: ElevatedButton.styleFrom(
            foregroundColor: Colors.white,
            backgroundColor: Colors.blue,
            textStyle: TextStyle(fontSize: 20),
            padding: EdgeInsets.symmetric(horizontal: 20, vertical: 15),
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(8),
```

App Theme and UI Configuration:

MaterialApp: Sets up the main structure of the app, including themes and initial route (ProductPage).

```dart
class ProductPage extends StatefulWidget {
  @override
  _ProductPageState createState() => _ProductPageState();
}

class _ProductPageState extends State<ProductPage> {
  final FirebaseFirestore firestore = FirebaseFirestore.instance;
  final CollectionReference mUsers = FirebaseFirestore.instance.collection('products');
  final _formKey = GlobalKey<FormState>();
  TextEditingController _descriptionController = TextEditingController();
  TextEditingController _priceController = TextEditingController();
  TextEditingController _roomNumberController = TextEditingController();
  int? _selectedProductId;

  String? _selectedRoomCategory;
  final List<String> _roomCategories = [
    'Classic Room',
    'Family Room',
    'Family Suite',
    'VIP Suite',
  ];

  List<Map<String, dynamic>> _products = [];
```

State Management and Firebase Firestore:

- **FirebaseFirestore:** Instance of Firestore to interact with the database.
- **Form Controllers:** Manage input data for form fields.

```dart
@override
void initState() {
  super.initState();
  _fetchProducts();
}

Future<void> _fetchProducts() async {
  final products = await DatabaseHelper().getProducts();
  setState(() {
    _products = products;
  });
}
```

Authentification Operations

3. Authentication Services:

This code defines an AuthService class that handles user authentication tasks such as signing up, signing in, and signing out using Firebase Authentication in a Flutter app.

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

import '../pages/home/home.dart';
import '../pages/login/login.dart';

class AuthService {

  Future<void> signup({
    required String email,
    required String password,
    required BuildContext context
  }) async {

    try {

      await FirebaseAuth.instance.createUserWithEmailAndPassword(
        email: email,
        password: password
      );
```

Imports: The necessary packages for Firebase authentication, UI components, and toast notifications are imported.

SignIn (continue):

```
    await FirebaseAuth.instance.signInWithEmailAndPassword(
      email: email,
      password: password
    );

    await Future.delayed(const Duration(seconds: 1));
    Navigator.pushReplacement(
      context,
      MaterialPageRoute(
        builder: (BuildContext context) => const Home()
      )  // MaterialPageRoute
    );

  } on FirebaseAuthException catch(e) {
    String message = '';
    if (e.code == 'invalid-email') {
      message = 'No user found for that email.';
    } else if (e.code == 'invalid-credential') {
      message = 'Wrong password provided for that user.';
    }
     Fluttertoast.showToast(
      msg: message,
      toastLength: Toast.LENGTH_LONG,
      gravity: ToastGravity.SNACKBAR,
      backgroundColor: Colors.black54,
      textColor: Colors.white,
      fontSize: 14.0,
    );
  }
  catch(e){
```

Explanation: This method logs in an existing user with email and password. On success, it navigates the user to the Home page. If an error occurs, it shows a toast notification with the error message.

Similar to the signup method, this block represents the signin logic with a try-catch structure for success and error handling.

SignOut:

```
  Future<void> signout({
    required BuildContext context
  }) async {

    await FirebaseAuth.instance.signOut();
    await Future.delayed(const Duration(seconds: 1));
    Navigator.pushReplacement(
        context,
        MaterialPageRoute(
          builder: (BuildContext context) =>Login()
        )  // MaterialPageRoute
      );
  }
}
```

Fetching Data:

- **_fetchProducts:** Retrieves products from the local SQLite database and updates the state to display them.

```
Future<void> _addProduct() async {
  if (_formKey.currentState!.validate()) {
    _formKey.currentState!.save();

    final roomNumberExists = await DatabaseHelper().doesProductExist(_roomNumberControll
    if (roomNumberExists) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Product with this room number already exists!')),
      );
      return;
    }

    try {
      await DatabaseHelper().insertProduct({
        'room_category': _selectedRoomCategory,
        'description': _descriptionController.text,
        'price': double.parse(_priceController.text),
        'room_number': _roomNumberController.text,
      });
      _fetchProducts();
      _clearForm();
    } catch (e) {
      print('Error inserting product: $e');
```

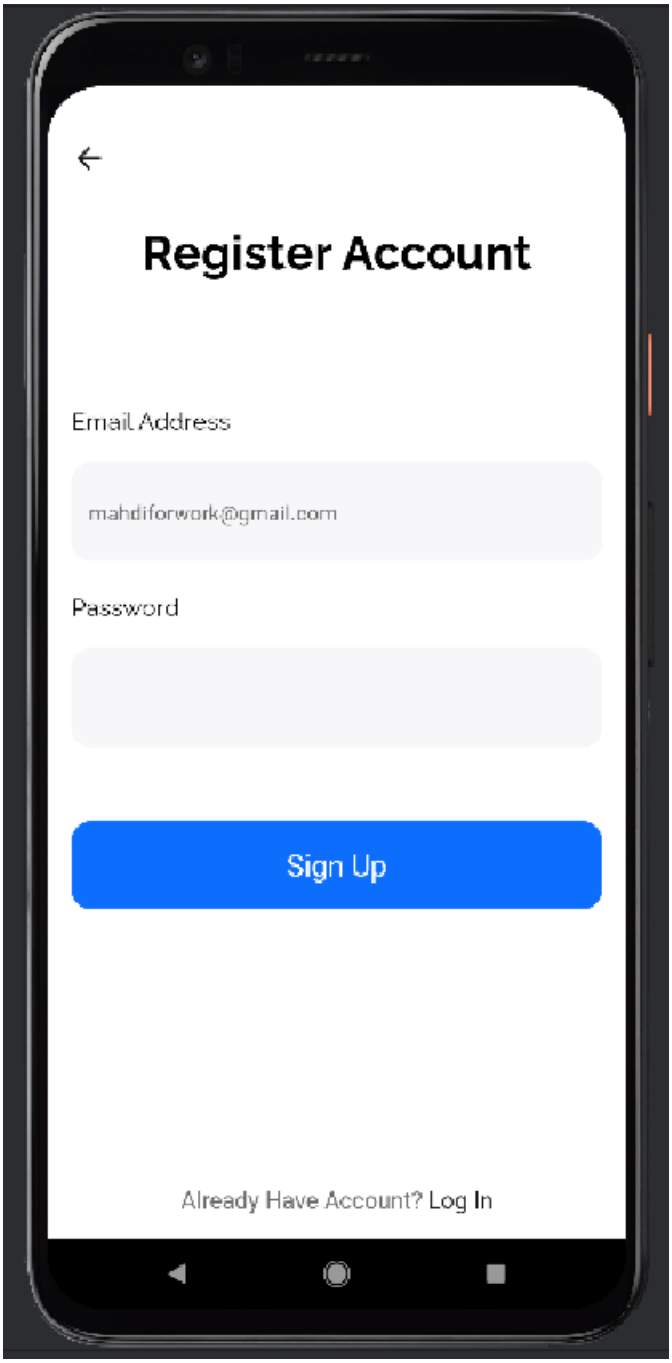
```
Future<void> _onCreate(Database db, int version) async {
  await db.execute('''
    CREATE TABLE products(
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      description TEXT,
      price REAL,
      room_category TEXT,
      room_number TEXT UNIQUE -- Added UNIQUE constraint here
    )
  ''');
}

Future<void> _onUpgrade(Database db, int oldVersion, int newVersion) async {
  if (oldVersion < 2) {
    await db.execute('ALTER TABLE products ADD COLUMN room_number TEXT');
  }
  if (oldVersion < 3) {
    await db.execute('ALTER TABLE products ADD COLUMN room_category TEXT');
  }
  if (oldVersion < 4) {
    // To add a unique constraint, you need to recreate the table
    await db.execute('''
      CREATE TABLE new_products(
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        description TEXT,
        price REAL,
        room_category TEXT,
        room_number TEXT UNIQUE -- Added UNIQUE constraint here
      )
    ''');
```

Create:

```
Future<void> insertProduct(Map<String, dynamic> product) async {
  final db = await database;
  try {
    // Insert into SQLite
    int id = await db.insert('products', product, conflictAlgorithm: ConflictAlgorithm.i
    product['id'] = id; // Ensure the product has the SQLite ID

    // Also insert into Firestore with the same ID
    await _firestore.collection('products').doc(id.toString()).set(product);
  } catch (e) {
    // Handle the error if necessary
    print('Error inserting product: $e');
  }
}

Future<List<Map<String, dynamic>>> getProducts() async {
  final db = await database;
  return await db.query('products');
}
```

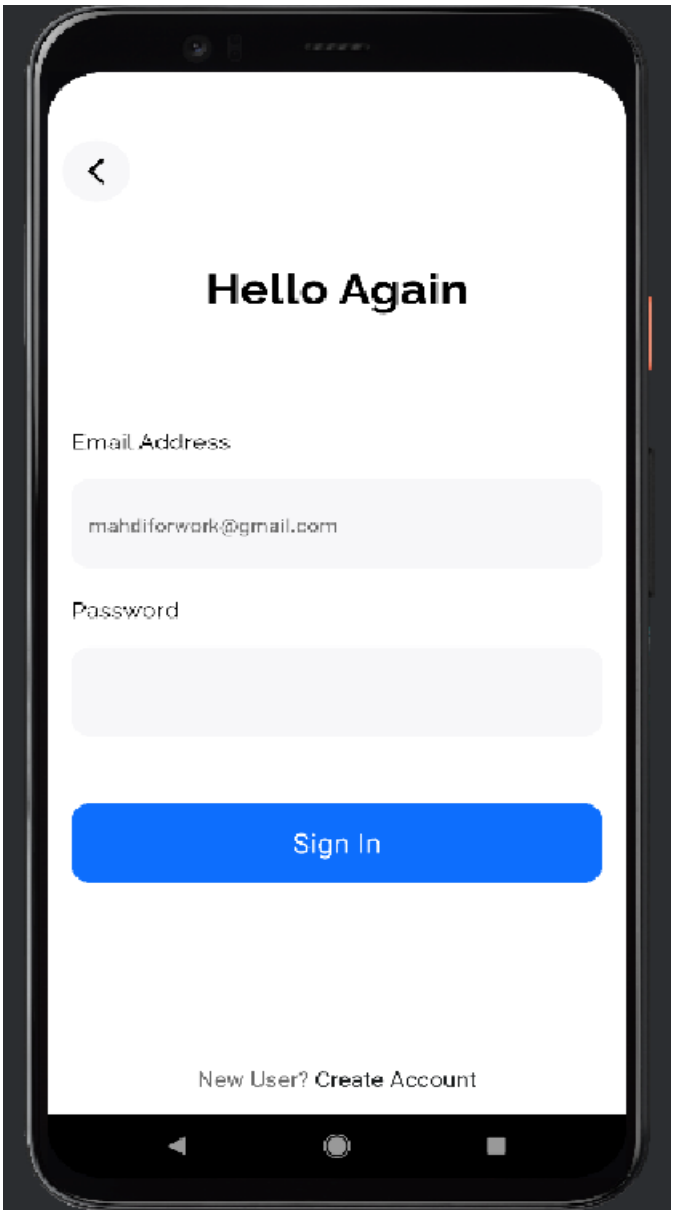
User Interface:

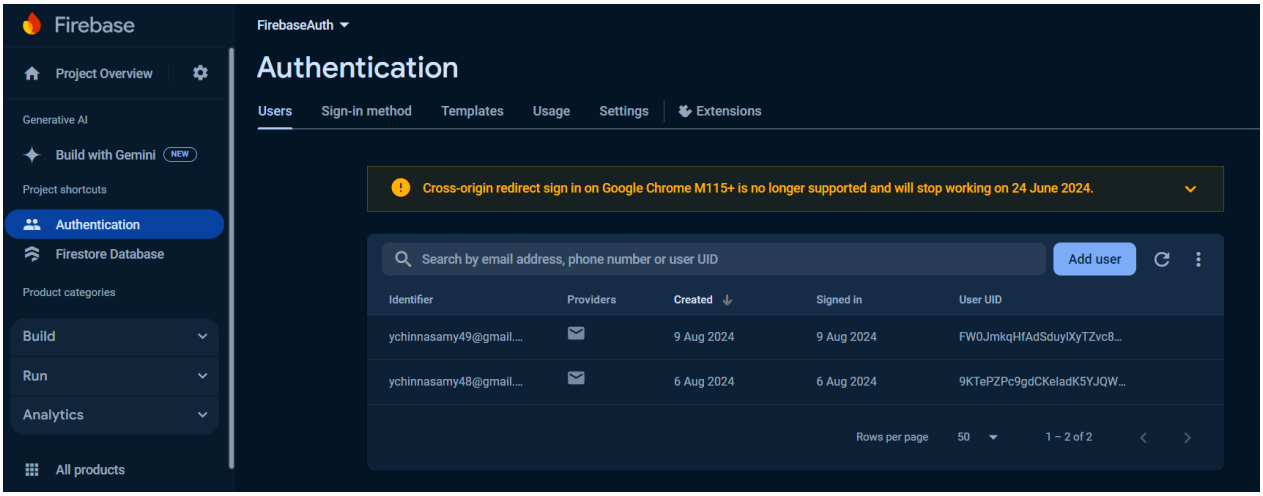
Registration page:
User have to register first if they have not done so, to be able to move to the home page.



Login:

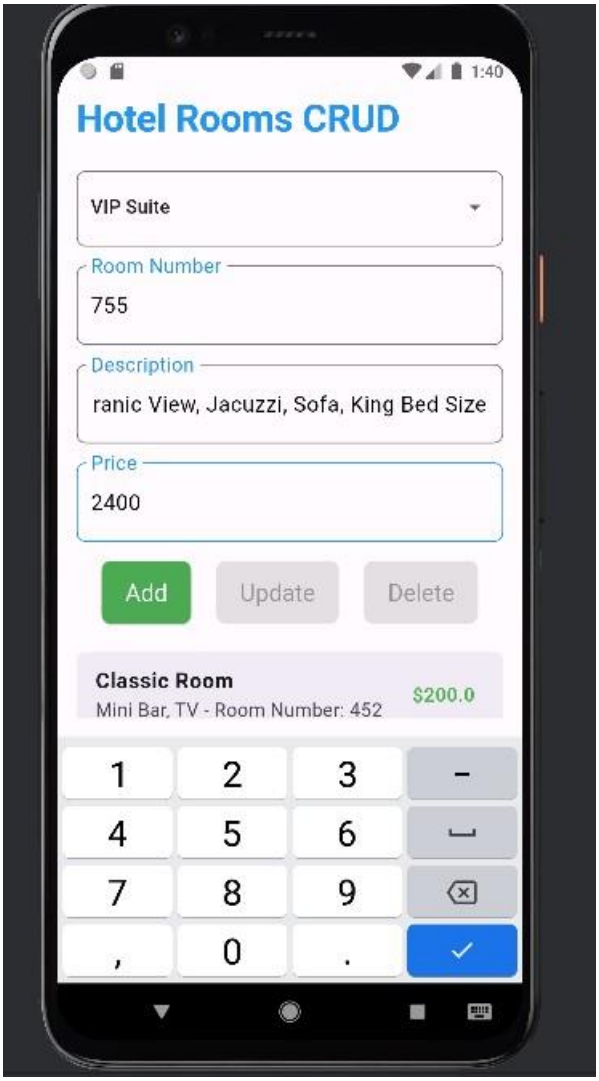
After the user have registered and their credential has been stored in the firebase authentication services, users will be able to access their account and register for their rooms.



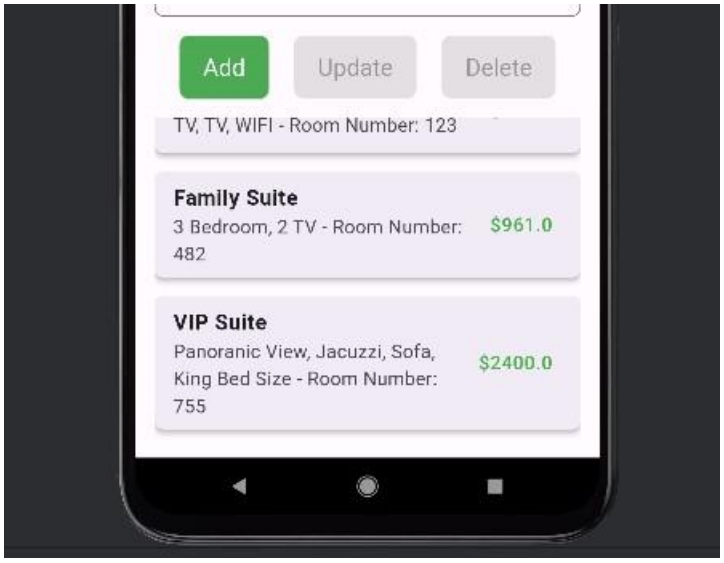


HomePage:

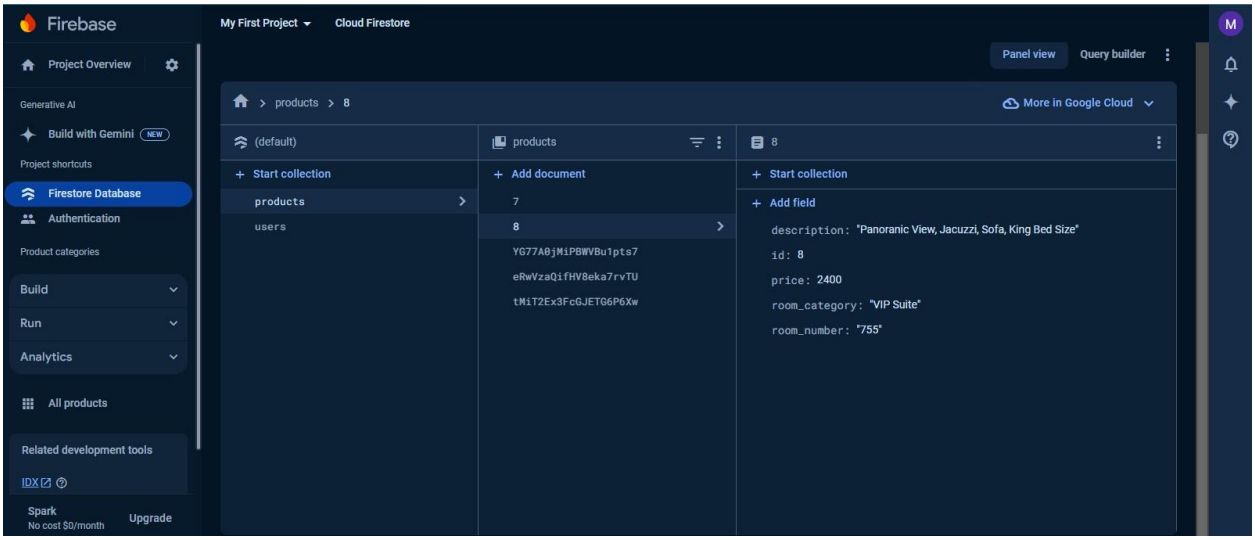
Here the user accesses this page and can add room details, update, and delete if they see its necessary.


1:40
Hotel Rooms CRUD
VIP Suite
Room Number
755
Description
ranic View, Jacuzzi, Sofa, King Bed Size
Price
2400
Add
Update
Delete
Classic Room
Mini Bar, TV - Room Number: 452
$200.0
1 2 3 -
4 5 6
7 8 9
, 0 .

In the List:



On firebase:

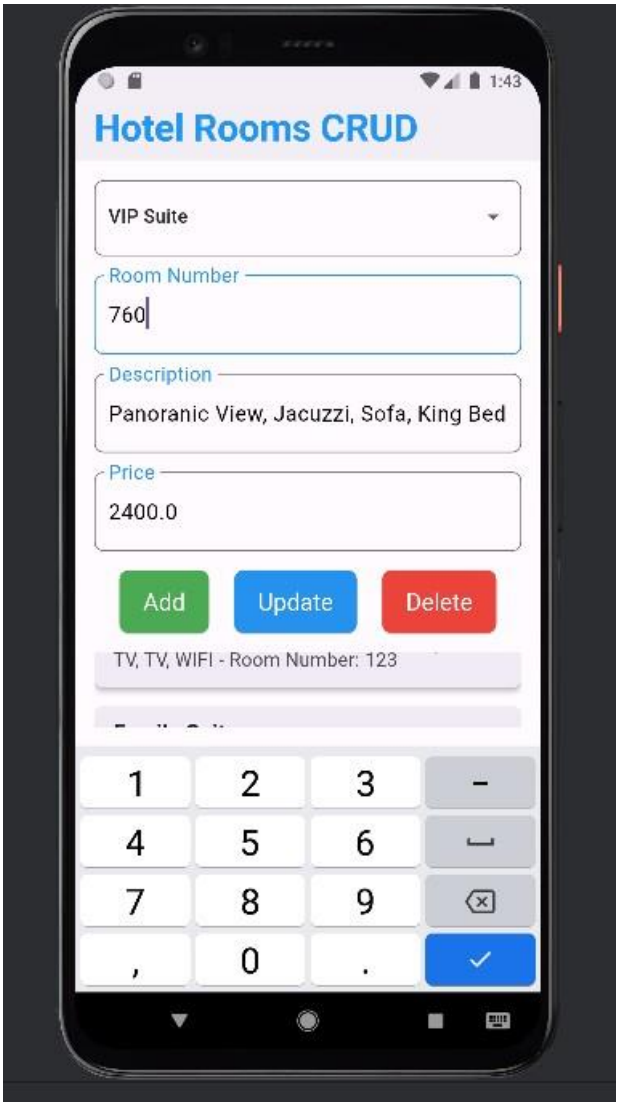


Update:

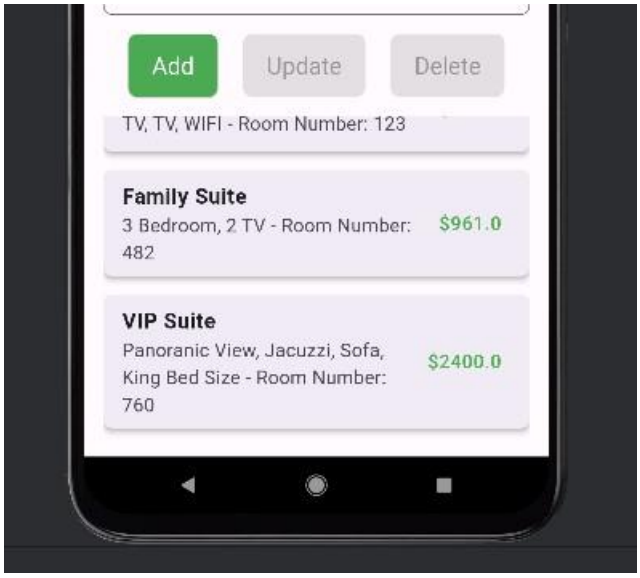
```
Future<void> updateProduct(int id, Map<String, dynamic> product) async {
  final db = await database;
  try {
    // Update in SQLite
    await db.update('products', product, where: 'id = ?', whereArgs: [id]);

    // Update in Firestore using the document ID
    await _firestore.collection('products').doc(id.toString()).update(product);
  } catch (e) {
    // Handle the error if necessary
    print('Error updating product: $e');
  }
}
```

- **_addProduct:** Validates and saves form data, checks for duplicate room numbers, and inserts new product data into both SQLite and Firestore.

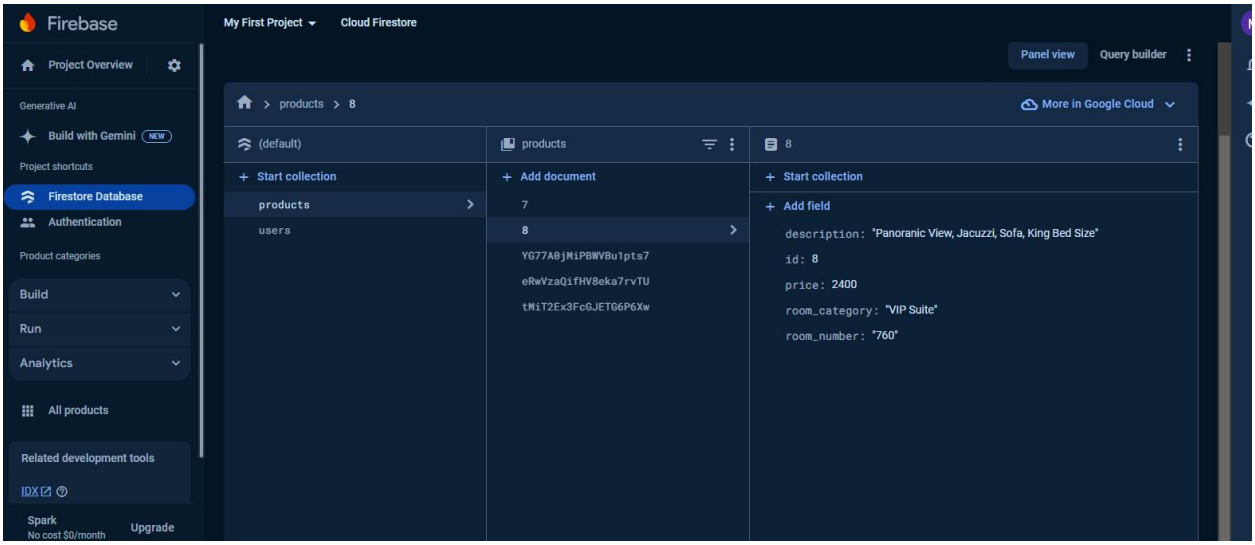
UI:


1:43
Hotel Rooms CRUD
VIP Suite
Room Number
760
Description
Panoranic View, Jacuzzi, Sofa, King Bed
Price
2400.0
Add
Update
Delete
TV, TV, WIFI - Room Number: 123
1
2
3
4
5
6
7
8
9
,
0
.

In the list:



On firebase:

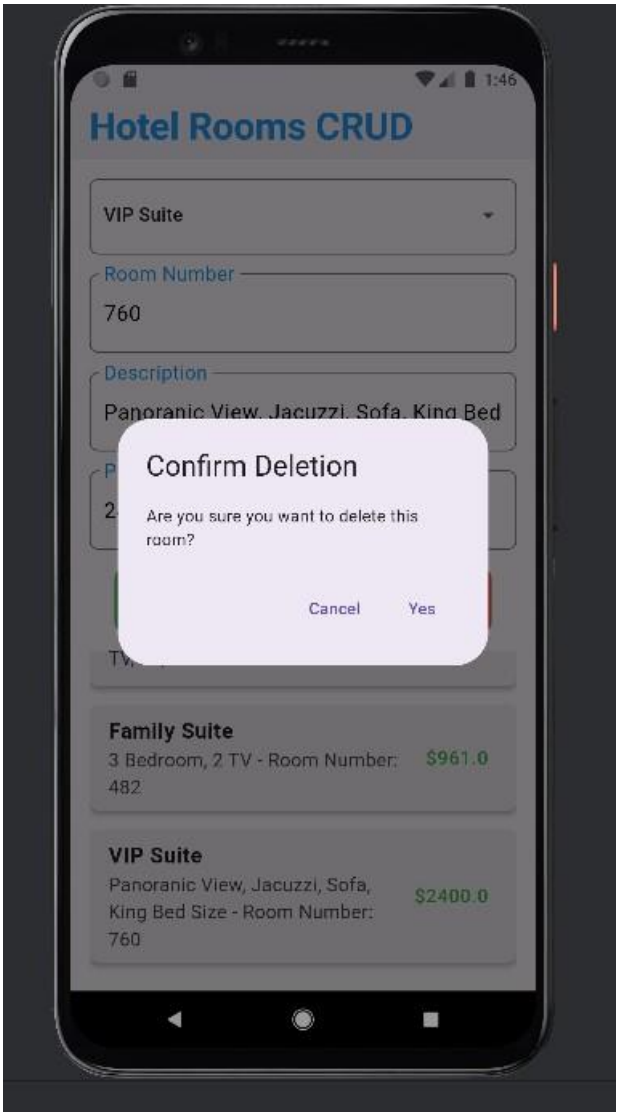


Delete:

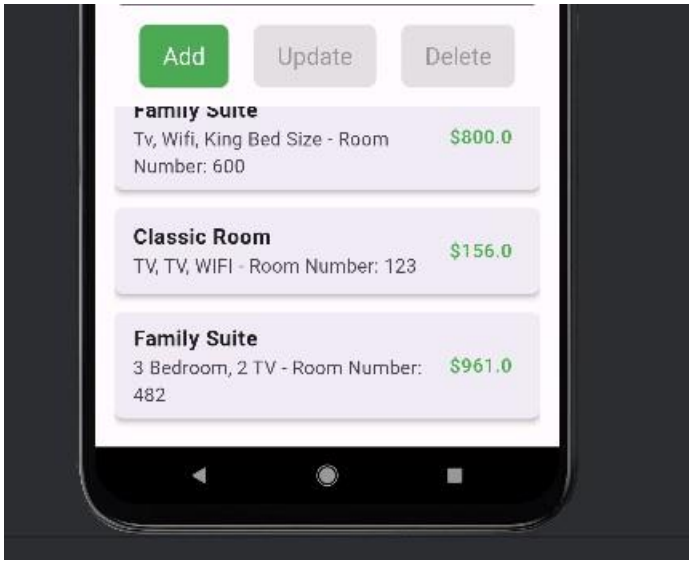
```
  Future<void> deleteProduct(int id) async {
    final db = await database;
    try {
      // Delete from SQLite
      await db.delete('products', where: 'id = ?', whereArgs: [id]);

      // Delete from Firestore using the document ID
      await _firestore.collection('products').doc(id.toString()).delete();
    } catch (e) {
      // Handle the error if necessary
      print('Error deleting product: $e');
    }
  }

  Future<bool> doesProductExist(String roomNumber) async {
    final db = await database;
    final result = await db.query(
      'products',
      where: 'room_number = ?',
      whereArgs: [roomNumber],
    );
    return result.isNotEmpty;
  }
}
```

- **updateProduct:** Validates and saves form data, then updates the product in both SQLite and Firestore.
- **_deleteProduct:** Deletes the product from both SQLite and Firestore.

UI:

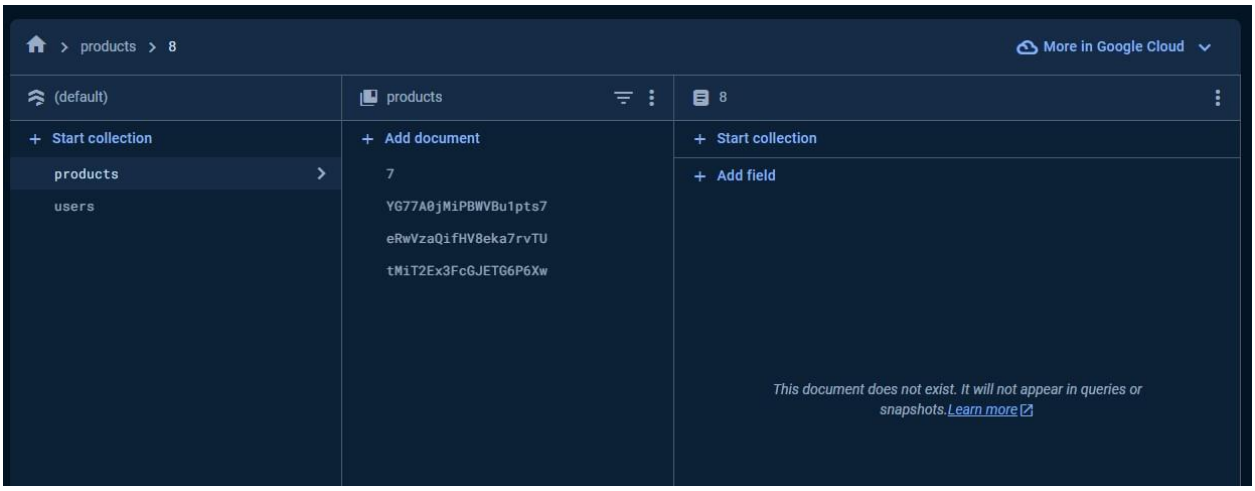

1:46
Hotel Rooms CRUD
VIP Suite
Room Number
760
Description
Panoranic View, Jacuzzi, Sofa, King Bed
Confirm Deletion
Are you sure you want to delete this room?
Cancel
Yes
Family Suite
3 Bedroom, 2 TV - Room Number: 482
$961.0
VIP Suite
Panoranic View, Jacuzzi, Sofa, King Bed Size - Room Number: 760
$2400.0

In the list:



- **It is no more present in the list.**

In firebase:



Delete Confirmation:

- **_showDeleteConfirmationDialog:** Displays a dialog to confirm the deletion of a product.

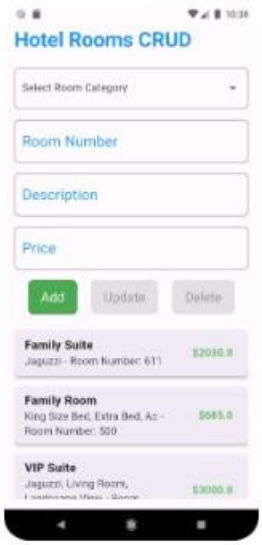
```
void _clearForm() {
  setState(() {
    _formKey.currentState!.reset();
    _selectedRoomCategory = null;
    _descriptionController.clear();
    _priceController.clear();
    _roomNumberController.clear();
    _selectedProductId = null;
  });
}
```

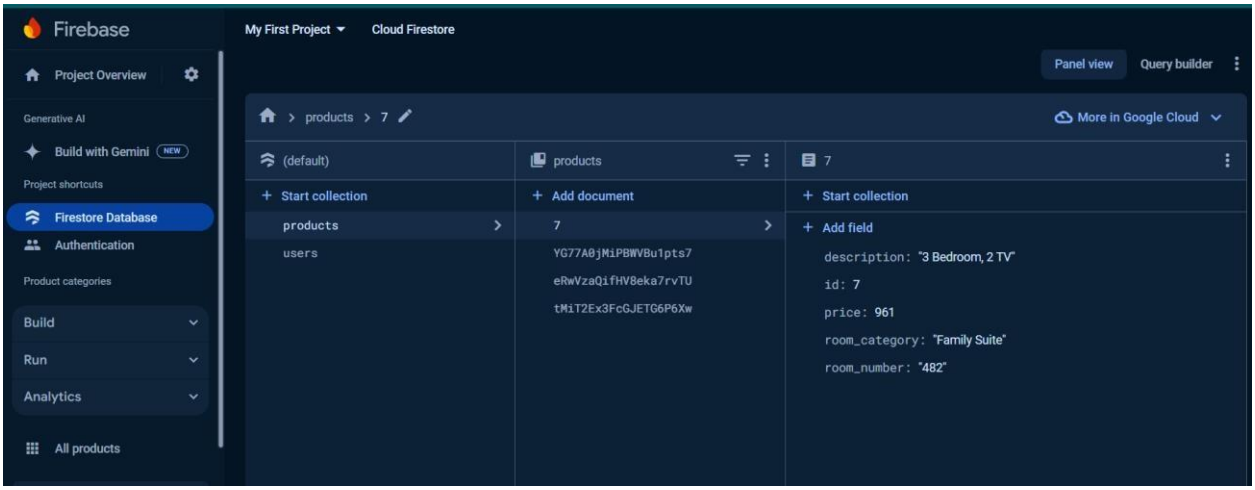
Firestore Database Interface

Product Page:

- The ProductPage allows users to add, update, and delete room bookings.
- Forms are used for input, and a ListView displays the products.

After adding data to the database of cloud_firebase on on the application.





Conclusion

This Flutter application effectively integrates Firebase and authentication services to manage hotel room bookings with a secure operation. The application features a robust UI for CRUD operations and ensures data consistency between local storage and cloud storage. For future improvement, we can consider adding more authentication properties to the app.