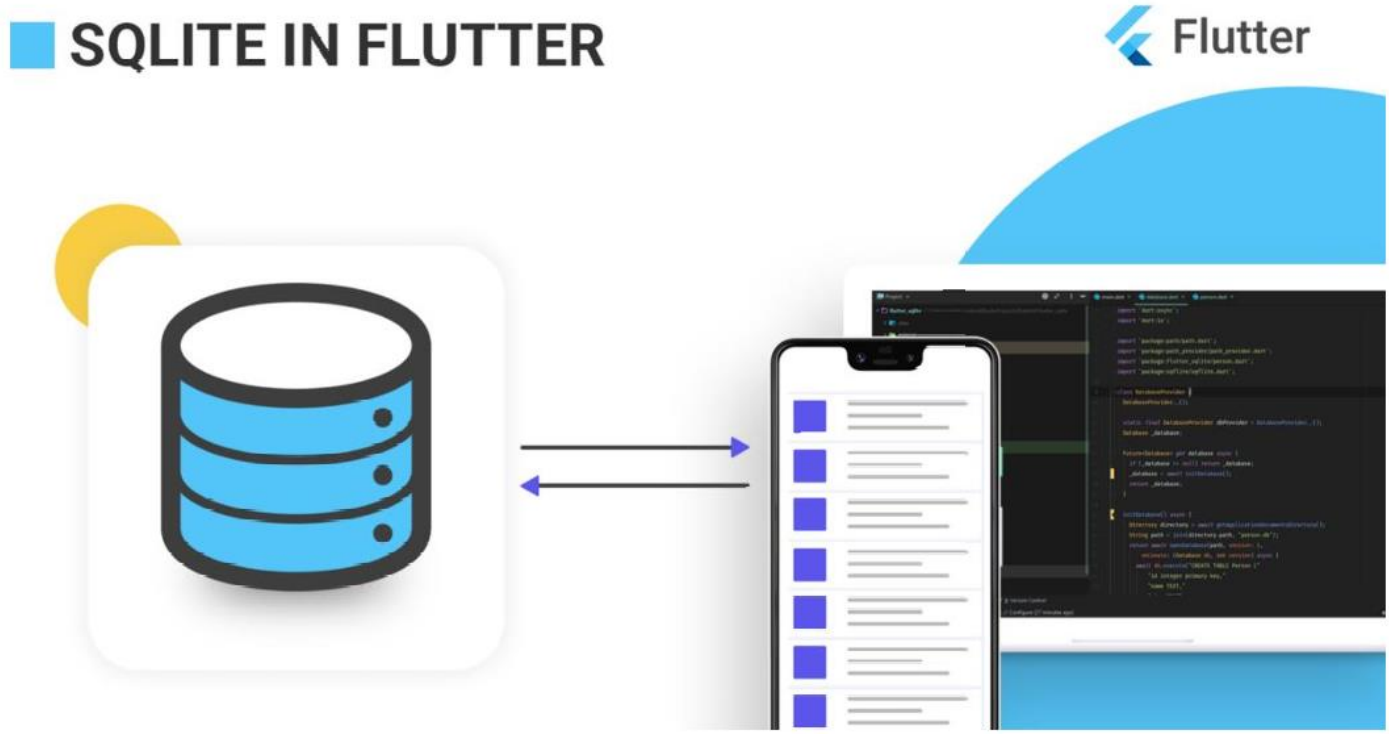
Flutter SQLite CRUD



Database in Flutter

• ใน Flutter การจัดเก็บข้อมูลจะมี 3 รูปแบบที่นิยมใช้คือ

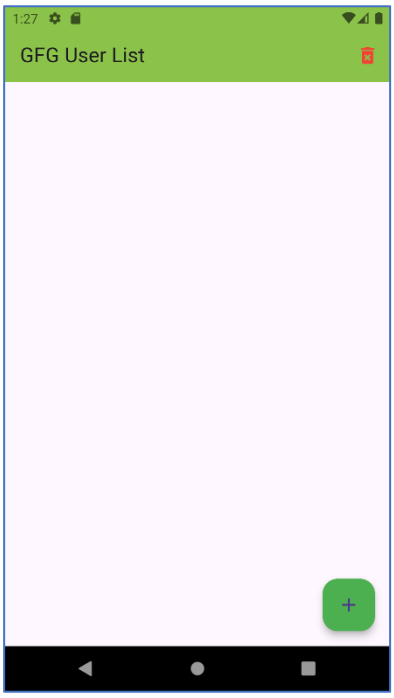
1. Relational เป็นฐานข้อมูลแบบเป็น Table มี Relational เป็น database มี primary key ทุก table จะสามารถกำหนด Relation เชื่อมกันได้ใน Flutter จะใช้ Sqflite เป็น ฐานข้อมูลที่พัฒนาต่อยอดจาก sqlite นั้นเอง

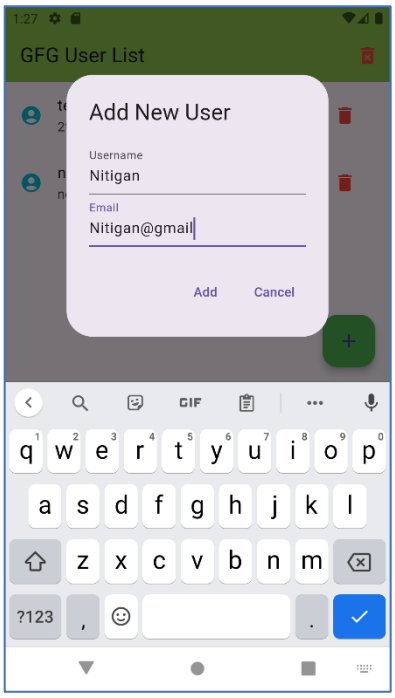
2. NoSQL ฐานข้อมูลที่ไม่อาศัย Relational เหมาะกับข้อมูลขนาดใหญ่ และสามารถขยายได้เรื่อย ๆ โดยไม่ต้องใช้หลักการ Table แบบ Relational ใน Flutter จะใช้ Firebase - online NoSQL storage โดยใช้ Service เชื่อมไปยังฐานข้อมูลเช่น MongoDB

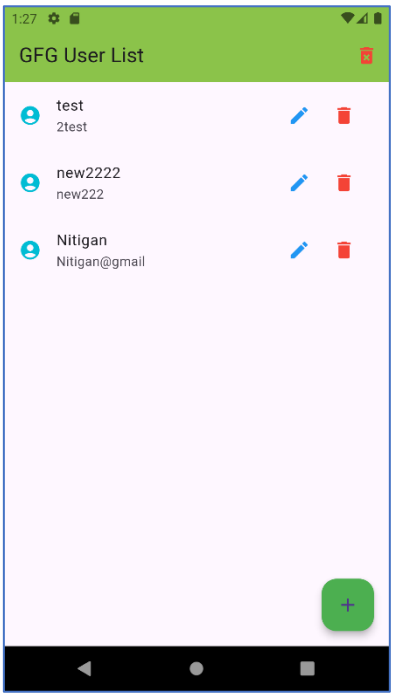
3. Individually tailored data storage เก็บเป็นไฟล์ไว้บน Device เช่นการบันทึกไฟล์ json เก็บไว้บนเครื่องเลย

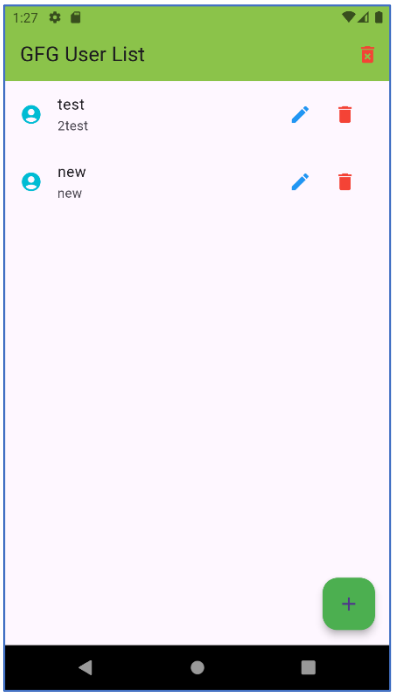
SQLite เป็นฐานข้อมูลเชิงสัมพันธ์ที่สามารถใช้เพื่อเก็บข้อมูลออฟไลน์สำหรับแอปพลิเคชันมือถือ Process
การทำงาน แบบ CRUD หมายถึง Create, Read, Update and Delete

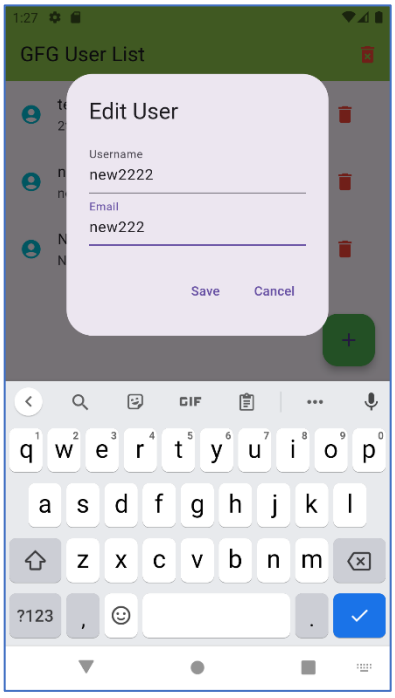
สร้าง Application Flutter

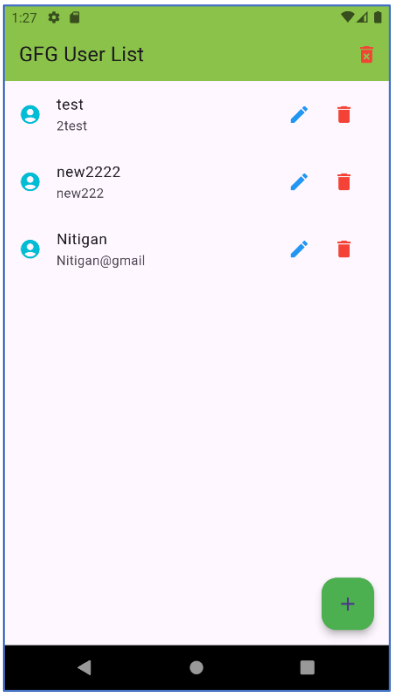












Database Structure

สร้าง database สำหรับเก็บข้อมูล กำหนดชื่อได้ตามความต้องการ ในที่นี้ ตั้งชื่อ databaseapp.db

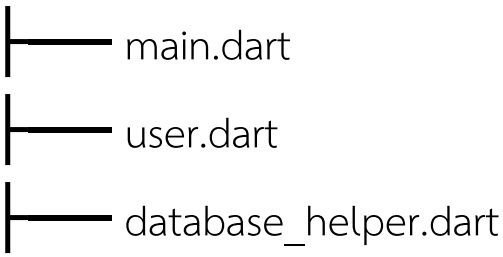
สร้างตาราง ชื่อ `tbUsers` สำหรับเก็บข้อมูล มีโครงสร้างดังนี้

Column	Type	Description
Id	INTEGER	id of user is Primary Key
username	TEXT	The username
email	TEXT	The email of user

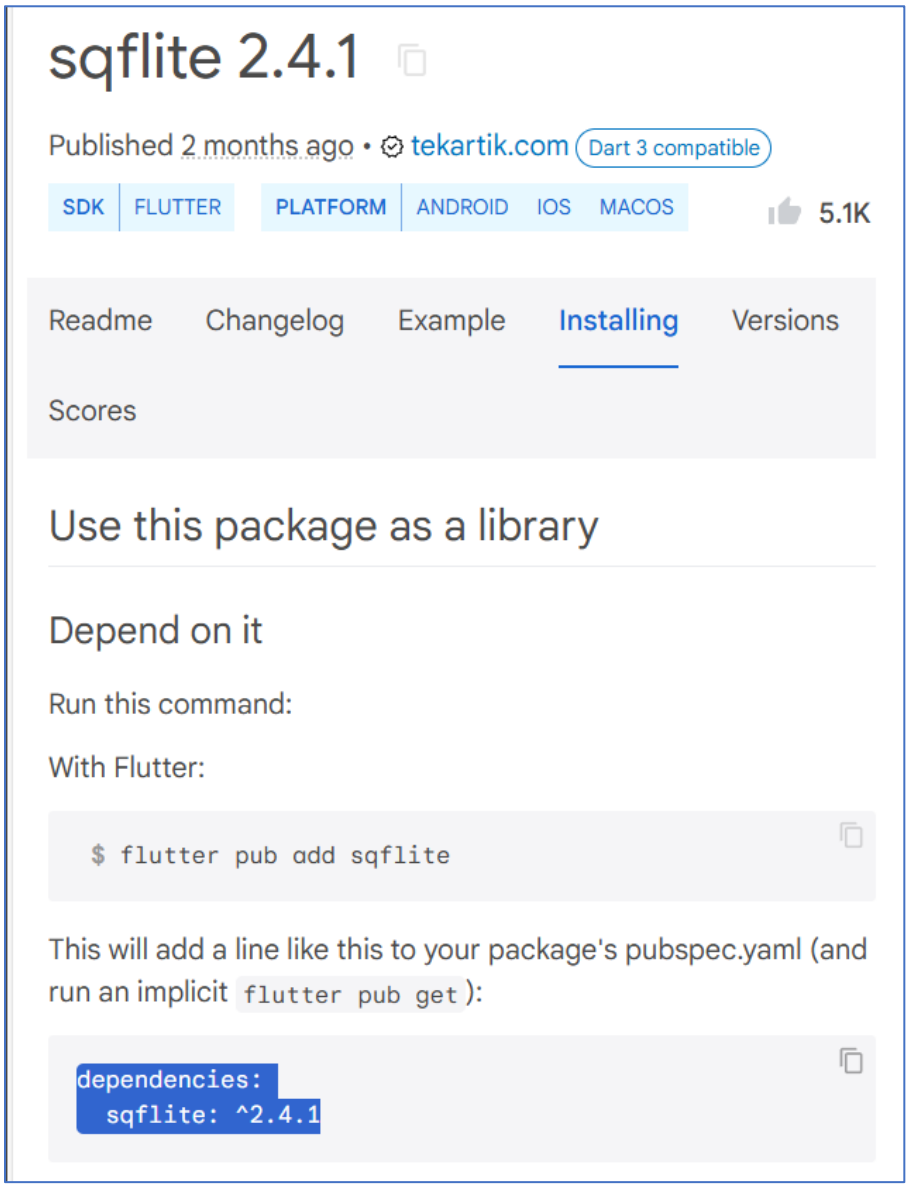
* หากต้องการเพิ่มวันเวลาที่บันทึก ให้ทดลองสร้าง attribute เพิ่มเติม

createdAt TIMESTAMP The time that the item was created. It will be automatically added by SQLite.

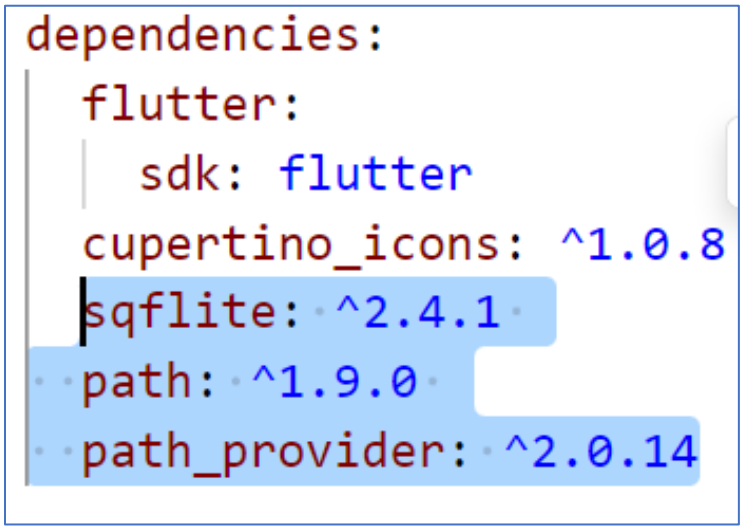
โครงสร้างของไฟล์ อยู่ในภายใต้ folder lib



Install sqflite



Pubspec.yaml



- Sqflite คือแพ็คเกจที่ให้การรวม SQLite สำหรับ Flutter
- pathเป็นแพ็คเกจที่ช่วยในการค้นหาเส้นทางของไฟล์ฐานข้อมูล

user.dart

สร้างไฟล์ใน ' **lib/user.dart**'เพื่อกำหนดคลาสโมเดลเพื่อแสดงข้อมูลผู้ใช้ นี่คือตัวอย่างของคลาสโมเดล

ที่มี attribute id , username , email พร้อมด้วยคอนสตรัคเตอร์ที่กำหนดค่าเริ่มต้น ของข้อมูล

```dart
class User {
  final int? id; // User's ID (nullable to handle new users)
  final String username; // User's username
  final String email; // User's email

  // Constructor
  User({this.id, required this.username, required this.email});

  // Convert User object to Map (for database insertion)
  Map<String, dynamic> toMap() {
    return {
      'id': id, // 'id' is nullable for new users (can be null)
      'username': username,
      'email': email,
    };
  }

  // Create User object from Map (for database queries)
  factory User.fromMap(Map<String, dynamic> map) {
    return User(
      id: map['id'], // 'id' can be null if not set yet
      username: map['username'],
      email: map['email'],
    );
  }
}
```

database_helper.dart

สร้างคลาสเพื่อจัดการฐานข้อมูล โดยมีฟังก์ชันทั้งหมดที่ใช้งานที่นี่ คลาสฐานข้อมูลมีวิธีการดังต่อไปนี้

initDb()	ใช้ในการเริ่มต้นฐานข้อมูล โดยจะตรวจสอบว่าตาราง 'tbUsers' มีอยู่หรือไม่ และถ้าไม่มีตาราง ฟังก์ชันจะสร้างตารางนั้นขึ้นมา
_onCreate()	เป็นฟังก์ชันคอลแบ็กที่เรียกใช้งานเมื่อสร้างฐานข้อมูล วัตถุประสงค์คือเพื่อกำหนดโครงร่างของตาราง 'tbUsers'
insertUser()	เพื่อแทรกผู้ใช้ใหม่เข้าไปในตาราง 'tbUsers'
queryAllUsers()	ดึงข้อมูลผู้ใช้ทั้งหมดจากตาราง 'tbUsers'
updateUser()	เพื่ออัปเดตผู้ใช้ที่มีอยู่ในตาราง 'tbUsers'
deleteUser()	ลบผู้ใช้ออกจากตาราง 'tbUsers' โดยใช้ ID ของ User นั้น
deleteAllUsers()	ลบผู้ใช้ทั้งหมด ออกจากตาราง 'tbUsers' โดยใช้ ID ของ User นั้น

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

class DatabaseHelper {
  static final DatabaseHelper instance = DatabaseHelper._instance();
  static Database? _database;

  DatabaseHelper._instance();

  // Get the database instance
  Future<Database> get db async {
    _database ??= await initDb();
    return _database!;
  }

  // Initialize the database
  Future<Database> initDb() async {
    String databasesPath = await getDatabasesPath();
    String path = join(databasesPath, 'appDB.db');
    return await openDatabase(path, version: 1, onCreate: _onCreate);
  }

```

```dart
  // Create the table
  Future _onCreate(Database db, int version) async {
    await db.execute('''
      CREATE TABLE tbUsers (
        id INTEGER PRIMARY KEY,
        username TEXT,
        email TEXT
      )
    ''');
  }

  // Insert a new user into the database
  Future<int> insertUser(User user) async {
    Database db = await instance.db;
    return await db.insert('tbUsers', user.toMap());
  }

  // Query all users from the database
  Future<List<Map<String, dynamic>>> queryAllUsers() async {
    Database db = await instance.db;
    return await db.query('tbUsers');
  }

  // Update user information in the database
  Future<int> updateUser(User user) async {
    Database db = await instance.db;
    return await db
        .update('tbUsers', user.toMap(), where: 'id = ?', whereArgs: [user.id]);
  }

  // Delete user from the database
  Future<int> deleteUser(int id) async {
    Database db = await instance.db;
    return await db.delete('tbUsers', where: 'id = ?', whereArgs: [id]);
  }

  // ฟังก์ชันในการลบข้อมูลทั้งหมด
  Future<void> deleteAllUsers() async {
    Database db = await instance.db;
    await db.delete('tbUsers'); // ลบข้อมูลทั้งหมดจากตาราง
  }

  // Initialize some default users in the database
  Future<void> initializeUsers() async {
    List<User> usersToAdd = [
      User(username: 'John', email: 'john@example.com'),
      User(username: 'Jane', email: 'jane@example.com'),
      User(username: 'Alice', email: 'alice@example.com'),
      User(username: 'Bob', email: 'bob@example.com'),
    ];

    for (User user in usersToAdd) {
      await insertUser(user);
    }
  }
}

```

main.dart

การเริ่มต้น : ต้องแน่ใจว่าได้เริ่มต้นฐานข้อมูลและระบุข้อมูลโครงสร้างของ user ก่อนที่จะรันวิดเจ็ต **MyApp**

การแสดงรายชื่อผู้ใช้: วิดเจ็ต UserList รับผิดชอบในการแสดงรายชื่อผู้ใช้ที่ดึงมาจากฐานข้อมูลโดยใช้คลาส DatabaseHelper

การจัดการสถานะ: เมธอด initState เพื่อดึงข้อมูลเมื่อมีการเริ่มต้นวิดเจ็ตเป็นครั้งแรก และอัปเดตอินเทอร์เฟซเมื่อมีข้อมูล user เปลี่ยนแปลง

การแสดงข้อมูล : แสดง username และ email โดยใช้ วิดเจ็ต **ListTile** ภายใน **ListView.builder**

```dart
import 'package:flutter/material.dart';
import 'database_helper.dart'; // Import the DatabaseHelper class
import 'user.dart'; // Import the User class

void main() async {
  // Initialize the database and insert users
  WidgetsFlutterBinding.ensureInitialized();
  await DatabaseHelper.instance.initDb();
  await DatabaseHelper.instance.initializeUsers();

  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'User Management',
      home: UserList(),
    );
  }
}

class UserList extends StatefulWidget {
  @override
  _UserListState createState() => _UserListState();
}

class _UserListState extends State<UserList> {
  List<User> _users = [];

```

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

  Future<void> _fetchUsers() async {
    final userMaps = await DatabaseHelper.instance.queryAllUsers();
    setState(() {
      _users = userMaps.map((userMap) => User.fromMap(userMap)).toList();
    });
  }

  // Function to handle delete user action
  Future<void> _deleteUser(int userId) async {
    await DatabaseHelper.instance.deleteUser(userId);
    _fetchUsers(); // Refresh the user list
  }

  // Function to handle edit user action
  void _editUser(User user) {
    TextEditingController usernameController =
        TextEditingController(text: user.username);
    TextEditingController emailController =
        TextEditingController(text: user.email);

    showDialog(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
          title: Text('Edit User'),
          content: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            mainAxisSize: MainAxisSize.min,
            children: [
              TextField(
                controller: usernameController,
                decoration: InputDecoration(labelText: 'Username'),
              ),
              TextField(
                controller: emailController,
                decoration: InputDecoration(labelText: 'Email'),
              ),
            ],
          ),
```

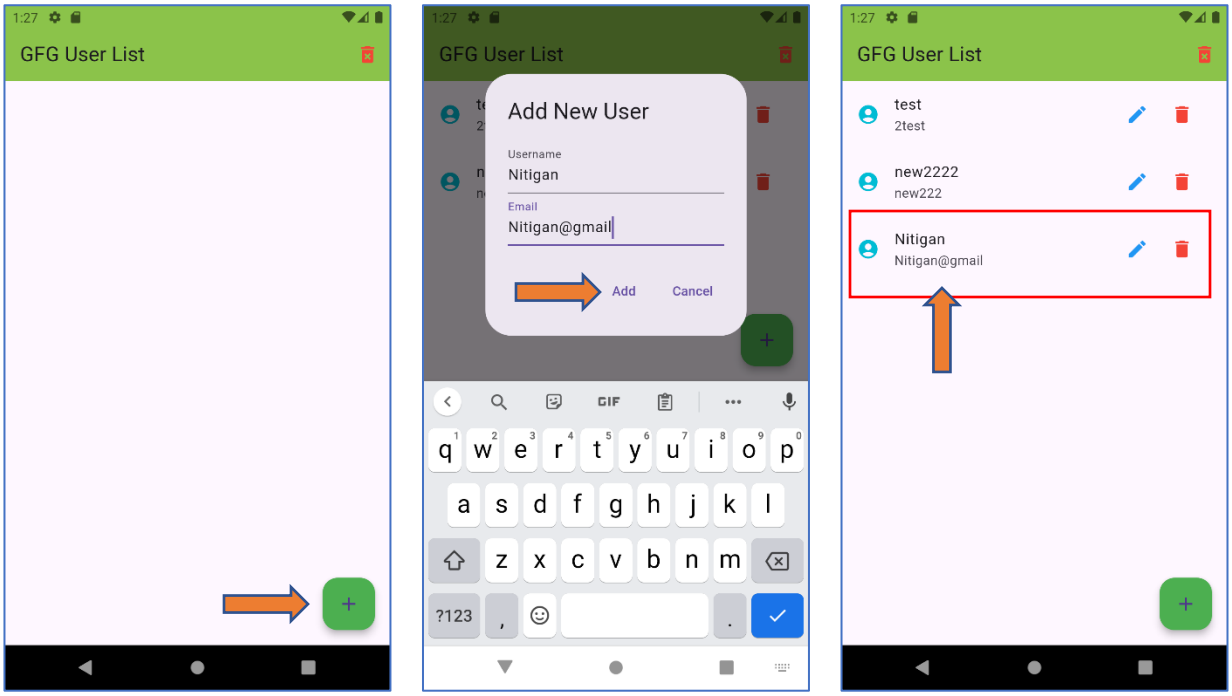
```
78            actions: [
79              TextButton(
80                onPressed: () {
81                  final updatedUser = User(
82                    id: user.id, // Keep the same id to update the correct record
83                    username: usernameController.text,
84                    email: emailController.text,
85                  );
86                  DatabaseHelper.instance.updateUser(updatedUser).then((value) {
87                    _fetchUsers(); // Refresh the user list
88                  });
89                  Navigator.pop(context); // Close the dialog
90                },
91                child: Text('Save'),
92              ),
93              TextButton(
94                onPressed: () =>
95                    Navigator.pop(context), // Close the dialog without saving
96                child: Text('Cancel'),
97              ),
98            ],
99          );
100       },
101     );
102   }
103
104   void _addUser() {
105     TextEditingController usernameController = TextEditingController();
106     TextEditingController emailController = TextEditingController();
107     showDialog(
108       context: context,
109       builder: (BuildContext context) {
110         return AlertDialog(
111           title: Text('Add New User'),
112           content: Column(
113             crossAxisAlignment: CrossAxisAlignment.start,
114             mainAxisSize: MainAxisSize.min,
115             children: [
116               TextField(
117                 controller: usernameController,
118                 decoration: InputDecoration(labelText: 'Username'),
119               ),
120               TextField(
121                 controller: emailController,
122                 decoration: InputDecoration(labelText: 'Email'),
123               ),
124             ],
125           ),
```

```
          actions: [
            TextButton(
              onPressed: () {
                final newUser = User(
                  username: usernameController.text,
                  email: emailController.text,
                );
                DatabaseHelper.instance.insertUser(newUser).then((value) {
                  _fetchUsers(); // Refresh the user list
                });
                Navigator.pop(context); // Close the dialog
              },
              child: Text('Add'),
            ),
            TextButton(
              onPressed: () =>
                  Navigator.pop(context), // Close the dialog without adding
              child: Text('Cancel'),
            ),
          ],
        );
      },
    );
  }

  Future<void> _deleteAllUsers() async {
    await DatabaseHelper.instance.deleteAllUsers(); // ลบข้อมูลผู้ใช้ทั้งหมด
    _fetchUsers(); // อัปเดตข้อมูลใหม่
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('GFG User List'),
        backgroundColor: const Color.fromARGB(255, 6, 207, 252),
        actions: [
          IconButton(
            icon: Icon(Icons.delete_forever),
            onPressed: _deleteAllUsers, // ลบข้อมูลทั้งหมดเมื่อกด
            color: Colors.red,
          ),
        ],
      ),
```

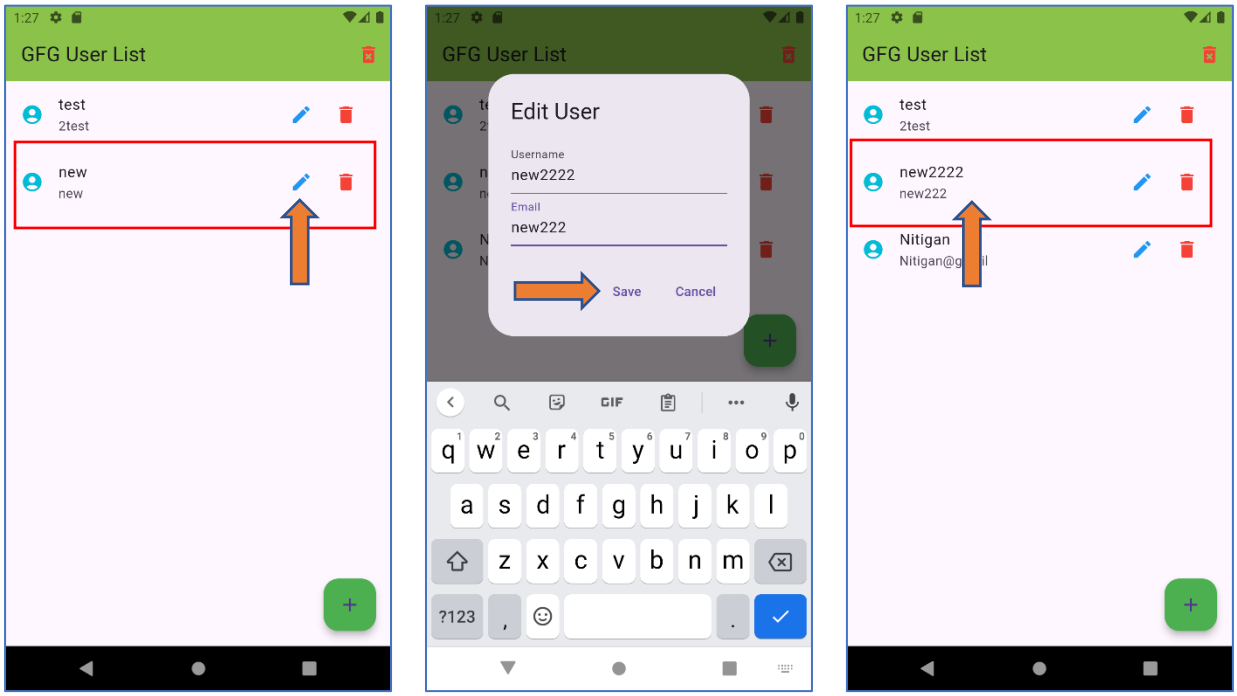
```
      body: ListView.builder(
        itemCount: _users.length,
        itemBuilder: (context, index) {
          final user = _users[index];
          return ListTile(
            leading: Icon(
              Icons.account_circle,
              color: Colors.cyan,
            ),
            title: Text(user.username),
            subtitle: Text(user.email),
            trailing: Row(
              mainAxisSize: MainAxisSize.min,
              children: [
                IconButton(
                  icon: Icon(Icons.edit),
                  onPressed: () => _editUser(user), // Edit action
                  color: Colors.blue,
                ),
                IconButton(
                  icon: Icon(Icons.delete),
                  onPressed: () => _deleteUser(user.id!), // Delete action
                  color: Colors.red,
                ),
              ],
            ),
          );
        },
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _addUser,
        child: Icon(Icons.add),
        backgroundColor: const Color.fromARGB(255, 7, 174, 221),
      ),
    );
  }
}

```

ทดสอบการทำงาน

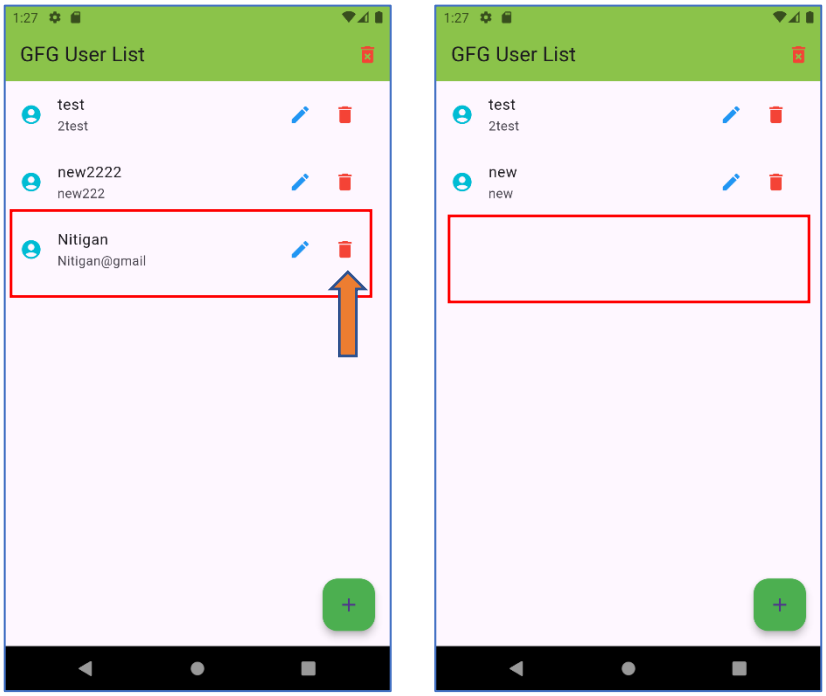
Add New User



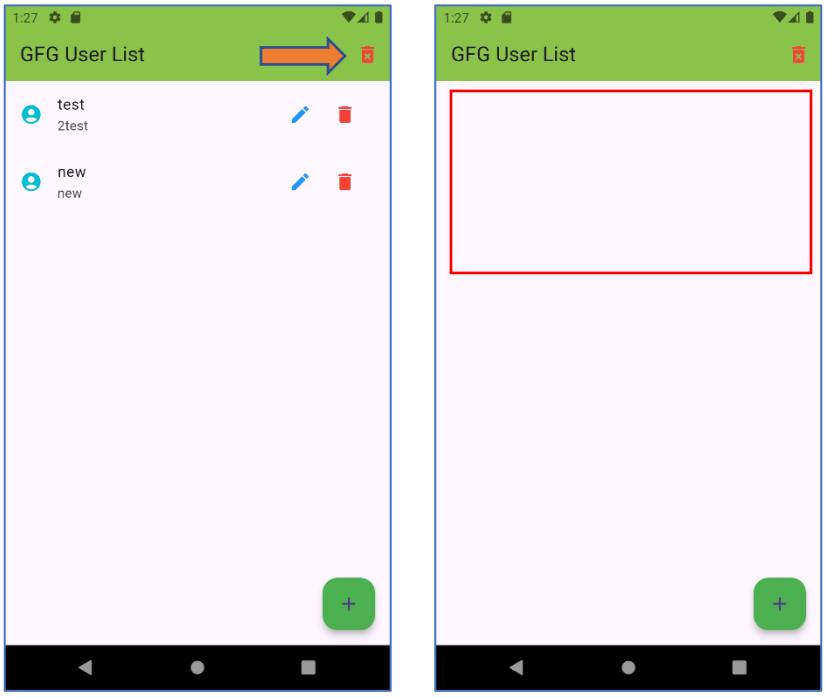
Edit User



Delete user



Delete All



** นศ.ทดลอง การเพิ่ม Attribute เพิ่ม คือ password และ createdAt

password เพื่อเก็บข้อมูล รหัสผ่าน

createdAt **เวลาที่มีการบันทึกข้อมูล โดยใช้กำหนดเป็น** TIMESTAMP เพื่อบันทึกเวลาอัตโนมัติโดยฐานข้อมูล