1. แนะนำ Firebase

1.1) Firebase คืออะไร

จีราวุธ วารินทร์. (2564) ได้อธิบายว่า Firebase เป็นบริการของ Google ที่เกี่ยวกับแบ็คเอนด์ (Backend Service) เช่น บริการด้านฐานข้อมูล (Cloud Firestore และ Realtime Database) บริการส่ง การแจ้งเตือนไปยังผู้ใช้ (Cloud Message) บริการวิเคราะห์ประสิทธิภาพการทำงาน ของแอปพลิเคชัน (Performance Monitoring และ Google Analytic) บริการจัดเก็บ และการเข้าใช้งานไฟล์ (Cloud Storage) และบริการอื่นๆ

สำหรับในบทแรกนี้จะแนะนำวิธีใช้งาน Firestore ซึ่งเป็นบริการสำหรับจัดการ ฐานข้อมูลในแบบเรียลไทม์ เป็นเครื่องมือที่ช่วยให้นักพัฒนาสามารถสร้างแอปพลิเคชัน ได้อย่างง่าย รวดเร็ว และมีเสถียรภาพยิ่งขึ้น

ศุภชัย สมพานิช. (2563) ได้อธิบายว่า Firebase เป็นบริการประเภทหนึ่งที่ทำงานอยู่ฝั่งหลังบ้าน (Back End) รองรับการพัฒนา ทั้ง Mobile Apps และ Web Apps ภายในประกอบไปด้วยบริการต่างๆ มากมาย แต่เราจะ ขอให้บริการหลักเพียงตัวเดียวที่เรียกว่า Firebase เป็นระบบฐานข้อมูลที่เรียกว่า Realtime Database ส่งผลให้เป็นการแบ่งการทำงานออกเป็น 2 ฝั่ง คือ

- Front End (หน้าบ้าน) คือ โปรเจ็กต์ Android Apps ที่คุณสร้างขึ้นมา

- Back End (หลังบ้าน) คือ การขอใช้บริการ Firebase เข้ามาทำหน้าที่จัดเก็บข้อมูลกล่าวได้อีกนัยหนึ่งว่าพื้นที่ส่วนนี้คือ API ที่ทำหน้าที่คอยให้บริการ Android Apps ของ เรานั่นเอง

สรุปได้ว่า Firebase คือชุดเครื่องมือและบริการที่ครอบคลุมซึ่งนำเสนอเป็นแพลตฟอร์ม Backend-as-aService (BaaS) ช่วยให้นักพัฒนาสร้าง เปิดใช้ และขยายทั้งแอปพลิเคชันมือถือและเว็บได้อย่างง่ายดาย มีฐานข้อมูลเรียลไทม์ การพิสูจน์ตัวตน พื้นที่เก็บข้อมูล โฮสติ้ง และคุณลักษณะอื่นๆ อีกมากมาย และจัดการทั้งหมดได้จากแพลตฟอร์มเดียว

1.2) บริการของ Firebase

การใช้งาน Firebase มีบริการหลายอย่างมาก ๆ โดยแบ่งหัวข้อของ Firebase ดังนี้

- Cloud Firestore คือ บริการทางด้าน Database ที่เป็นลักษณะเป็น NoSQL โดยนำข้อดีของ Realtime Database ของ Firebase

- Authentication ชื่อก็บอกอยู่แล้ว ใช่ครับ คือบริการที่จัดการ Auth ให้เรา ซึ่งคลอบคลุมมาก ๆ ทั้ง email-password, phone ไปจนถึง Facebook, twitter, GitHub สำหรับการ Login อีกด้วย
- Hosting คือ hosting สำหรับ single-page web app, landing page website ซึ่งจัดการการ Deploy ให้ และในส่วนของ Custom Domain (ไม่ฟรี) ก็มีการติดตั้ง SSL ให้ด้วย

Improve app quality

- Crashlytics ช่วยจัดการ Issue ต่าง ๆ และสามารถตรวจจับ Crash ได้ว่าเกิดขึ้นที่การทำงานไหนใน Mobile App แต่เดิมเริ่มต้นพัฒนาจากทีมงานของ Fabric ซึ่งมีผู้ใช้จำนวนมาก
- Performance Monitoring สรรพคุณตามชื่อเช่นกัน โดยผู้พัฒนาสามารถทราบถึง Performance ของ Code และ Network

Grow your business

- Google Analytics คือ ตัวที่เก็บข้อมูลสถิติ พฤติกรรมของ User ที่ใช้งาน Mobile App (Web ก็ใช้ได้นะ) โดยสามารถแบ่งพฤติกรรมให้เราดูได้อย่างละเอียด
- Remote Config คือ ส่วนที่จัดการรูปแบบของ Mobile App ในเรื่องของหน้าตา เช่น หากเราต้องการเปลี่ยนภาพ Background ในหน้า Main เราก็สามารถเปลี่ยนได้ที่ Remote Config นี้ได้เลย ไม่ต้องไปแก้ที่ Code ของ Mobile App
- Cloud Messaging คือ ตัวที่จะทำให้ Mobile App ของเรารับ Notification ได้โดยส่ง Message ไปหาได้ทุก Platform ทั้ง iOS และ Android รวมไปถึง Web (www.http://predictive.co.th., ออนไลน์)

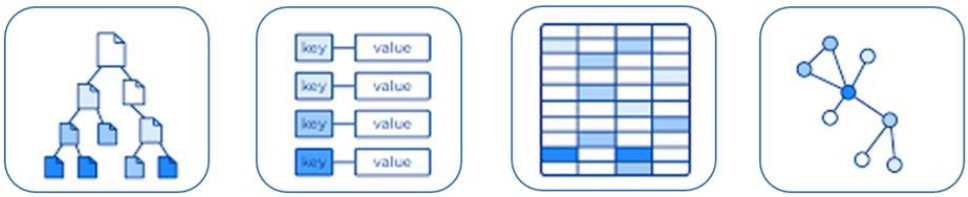
1.3) ฐานข้อมูล SQL และ NoSQL คืออะไร

ฐานข้อมูลที่นิยมใช้งานในปัจจุบันสามารถแบ่งออกได้ 2 แบบ คือ แบบ SQL และแบบ NoSQL

ฐานข้อมูล SQL ย่อมาจาก Structured query Language เป็นภาษาโปรแกรมสำหรับจัดเก็บและประมวลผลข้อมูลในฐานข้อมูลแบบเชิงสัมพันธ์ ฐานข้อมูลแบบเชิงสัมพันธ์เก็บข้อมูลในรูปแบบตารางที่มีแถวและคอลัมน์ที่เป็นตัวแทนของหมวดข้อมูลที่แตกต่างกันและความสัมพันธ์ต่างๆ ระหว่างค่าข้อมูล สามารถใช้คำสั่ง SQL ในการจัดเก็บ ปรับปรุง ลบ ค้นหา และดึงข้อมูลจากฐานข้อมูล นอกจากนี้ยังสามารถใช้ SQL ในการรักษาและเพิ่มประสิทธิภาพการทำงานของฐานข้อมูล

ฐานข้อมูล NoSQL ย่อมาจาก Non-relational database บางท่านอาจจะเรียกว่า Not only SQL ก็ไม่ได้ผิดแต่อย่างใด เป็น Database อื่น ๆ ที่ไม่ได้เป็นแบบ Relational หรือมีความสัมพันธ์กันชัดเจนแบบ

Pattern เหมาะสำหรับการใช้งานจำพวก Big Data และ Real-time Web Application ว่ากันง่าย ๆ ก็คือ เกิดมาเพื่อแก้ไขปัญหาของ RDBMS เลยล่ะครับ แต่ก็มีข้อเสียอยู่เช่นกัน ข้อดีข้อเสียเดี๋ยวเปรียบเทียบให้ตอนท้ายนะครับ สำหรับประเภทของ NoSQL จะแบ่งออกเป็น 4 แบบหลัก ๆ ได้แก่



หมายเหตุ. จาก https://blog.cloudhm.co.th/sql-vs-nosql/

ภาพประกอบ 8.1 ภาพแสดงประเภทของข้อมูลแบบ NoSQL

1) Document ข้อมูลและ Metadata จะเก็บเป็นลำดับชั้นในรูปแบบ Semi-structure data เช่น JSON หรือ XML ใน Database ตัวอย่าง Database Software ที่ใช้งานลักษณะนี้ ได้แก่ Cosmos DB, IBM Domino, MongoDB, Couchbase, ArangoDB

2) Key-Value เป็นการเก็บ Record ที่ไม่มีอะไรซับซ้อน มีแค่ Key และ Value ทำให้สามารถเข้าถึงข้อมูลได้รวดเร็ว โดยการเข้าถึงข้อมูลก็ให้ใช้ Key ก็จะได้ Value ที่ต้องการ ตัวอย่าง Database Software ที่ใช้งานลักษณะนี้ ได้แก่ Redis, Memcached, Apache Ignite, Couchbase, Dynamo

3) Graph ข้อมูลจะเก็บอยู่ในรูปแบบกราฟแผนภูมิ มี Node และ Edge ที่เชื่อมต่อกัน ทำให้ไม่ต้องนำข้อมูลมา JOINS กันเหมือนของ RDBMS ตัวอย่าง Database Softwareที่ใช้งานลักษณะนี้ ได้แก่ ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J, OrientDB, Virtuoso

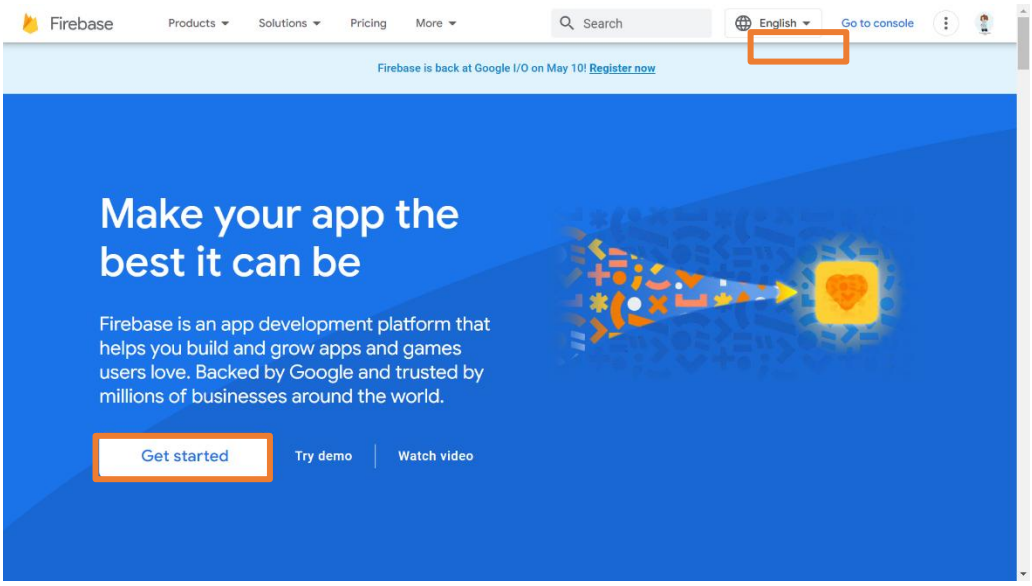
4) Wide-Column รูปแบบของ Wide-Column จะบันทึกข้อมูลในรูปแบบ Tables (Rows และ Columns) แต่จะต่างจาก RDBMS ตรงที่ แต่ละ Rows จะไม่ Fix Column (ถ้าเป็น RDBMS จะ Fix มาเป็น Pattern เดียวกัน) ตัวอย่าง Database Software ที่ใช้งานลักษณะนี้ ได้แก่ Amazon DynamoDB, Cassandra, Azure Tables, Accumulo, HBase

(www.blog.cloudhm.co.th., ออนไลน์)

2. การใช้งาน Flutter ร่วมกับ Firebase

2.1) ให้ไปที่เว็ปไซต์ https://firebase.google.com/

2.2) ทำการล็อกอินเข้าใช้งาน โดยใช้ e-mail ของ google เมื่อทำการล็อกอินสำเร็จแล้ว ให้ทำการเริ่มต้นใช้โดยกดที่ปุ่ม Get started หรือปุ่ม Go to console ดังภาพประกอบ 8.2

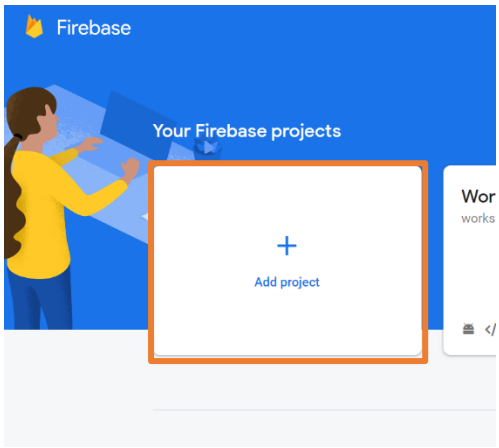


ภาพประกอบ 8.2 ภาพแสดงการเริ่มต้นใช้งาน Firebase

2.3) เริ่มต้นการสร้างโปรเจคในการใช้งาน Firebase ดังภาพประกอบ 8.3

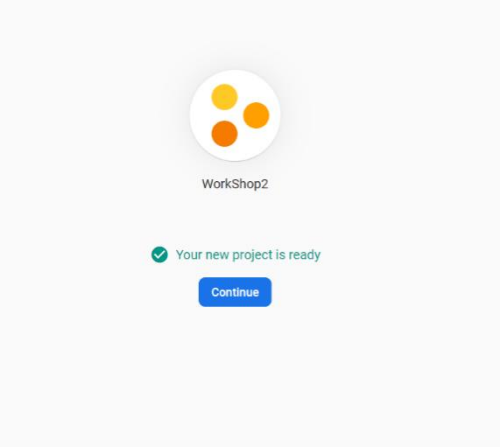
1) กดที่ Add project หลังจากนั้นตั้งชื่อโปรเจคของ Firebase

2) สามารถเลือกเปิดหรือปิดการใช้งาน Google Analytics ของโปรเจค Firebase



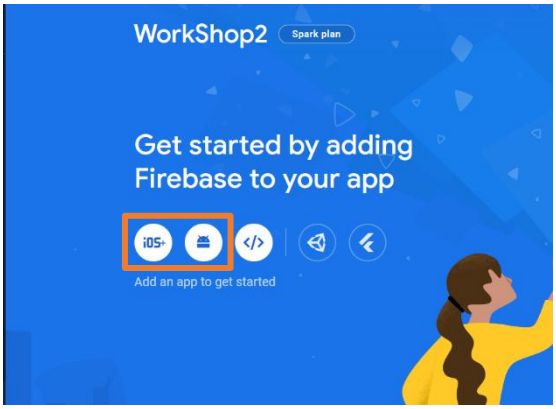
ภาพประกอบ 8.3 ภาพแสดงการเริ่มสร้างโปรเจค Firebase

3) เมื่อตั้งชื่อ และตั้งค่าการใช้งาน Google Analytics เรียบร้อยแล้วรอเวลาสร้างโปรเจคสักครู่ หากสร้างสำเร็จจะขึ้นดังภาพประกอบ 8.4



ภาพประกอบ 8.4 ภาพแสดงการสร้างโปรเจค Firebase สำเร็จ

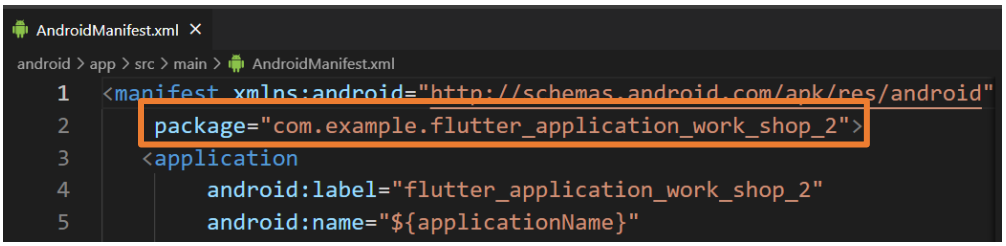
2.4) เริ่มต้นการเพิ่มแอปพลิเคชันของเข้าไปในโปรเจคของ firebase เนื่องจาก Flutter เป็น hybrid app เราจึงต้องเพิ่ม ทั้ง Android และ iOS ดังภาพประกอบ 8.5



ภาพประกอบ 8.5 ภาพแสดงการสร้างโปรเจค Firebase

โดยจะแสดงขั้นตอนการเพิ่มแอปพลิเคชันในฝั่ง Android การเพิ่มสามารถเลือกคลิกที่ไอคอนของแต่ระบบปฏิบัติการที่เราต้องการใช้งานดังขั้นตอนต่อไปนี้

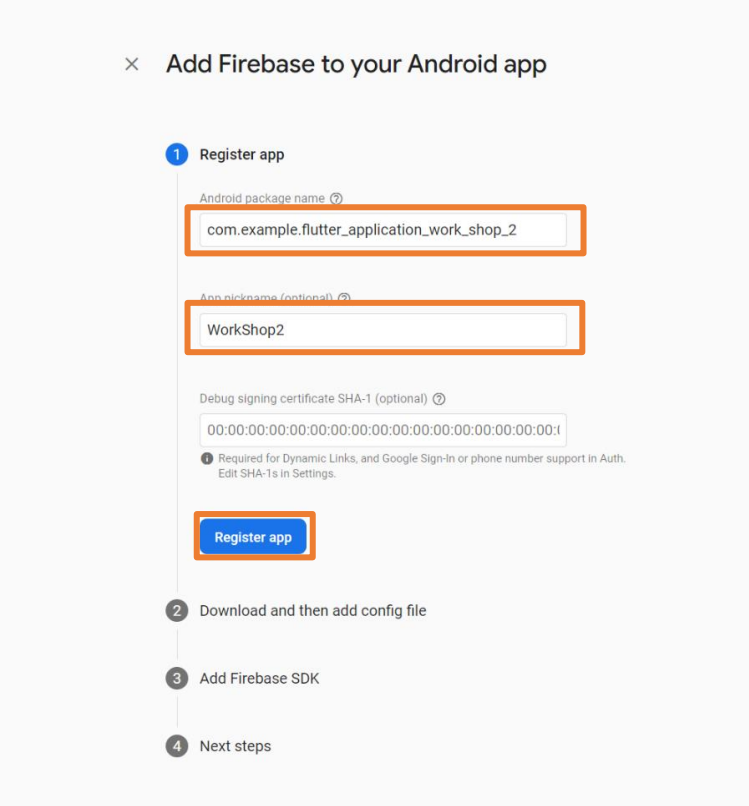
2.4.1) หา Package Name เพื่อนำไปลงทะเบียนไว้บน Firebase ได้ที่ Android->app->src->main->AndroidManifest.xml ดังภาพประกอบ 8.6



ภาพประกอบ 8.6 Android->app->src->main->AndroidManifest.xml

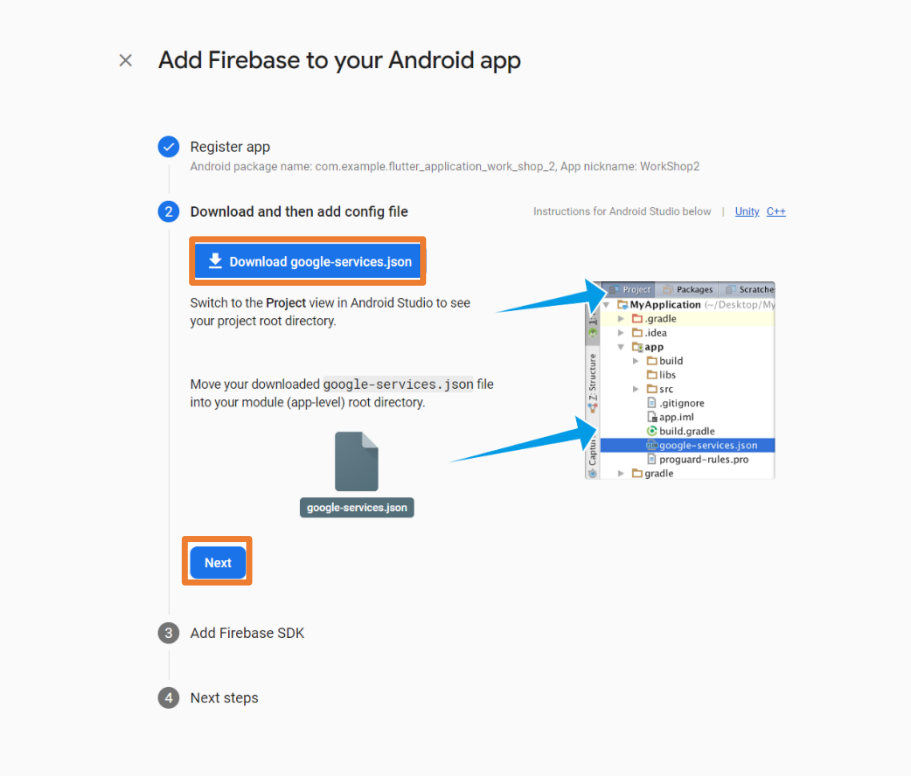
เมื่อได้ package name ของโปรเจคให้นำไปใส่ดังภาพประกอบ 8.6

1. ใส่ package name ที่ได้จากโปรเจค Flutter
2. ตั้งชื่อ
3. กด Register App



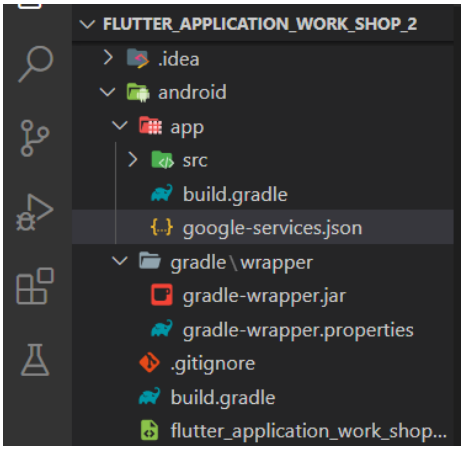
ภาพประกอบ 8.7 ภาพแสดงการเพิ่แอปพลิเคชันฝั่ง Android

2.4.2) เมื่อลงทะเบียนแอปเสร็จ ให้ดาวโหลดไฟล์ google-services.json เอาไว้ และกดถัดไป ดังภาพประกอบ 8.8



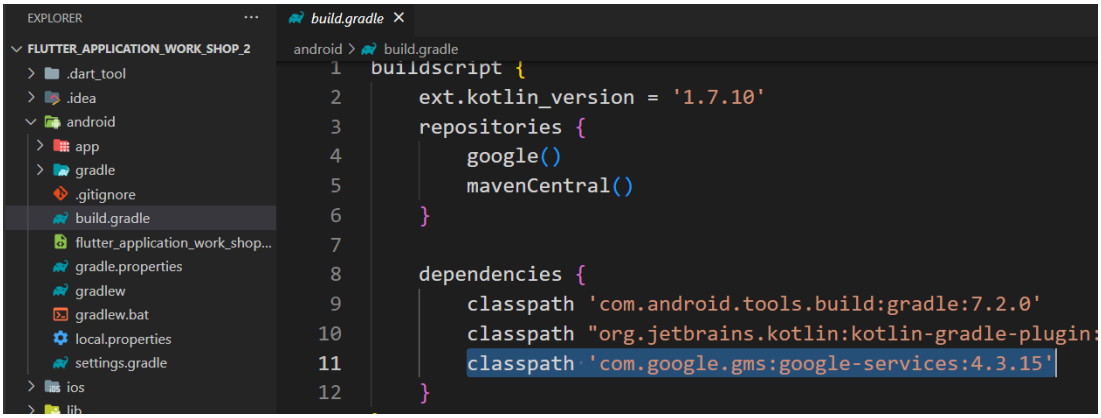
ภาพประกอบ 8.8 ภาพแสดงการดาวโหลดไฟล์ google-services.json

1. กด ดาวน์โหลด google-service.json
2. กด ถัดไป และลากไฟล์ google-services.json ไปไว้ใน Android->app



ภาพประกอบ 8.9 ภาพแสดงการวางไฟล์ google-services.json ใน Android->app

2.4.3) ไปที่ android\build.gradle หาแท็ก dependencies และวาง classpath 'com.google.gms:google-services:4.3.15' ลงไป ดังภาพประกอบ 8.10



ภาพประกอบ 8.10 ภาพแสดงการวาง classpath 'com.google.gms:google-services: 4.3.15'

จากนั้นเราต้อง apply plugin โดยไปที่ android\app\build.gradle เพิ่ม apply plugin: ‘com.google.gms.google-services’ ดังภาพประกอบ 8.11

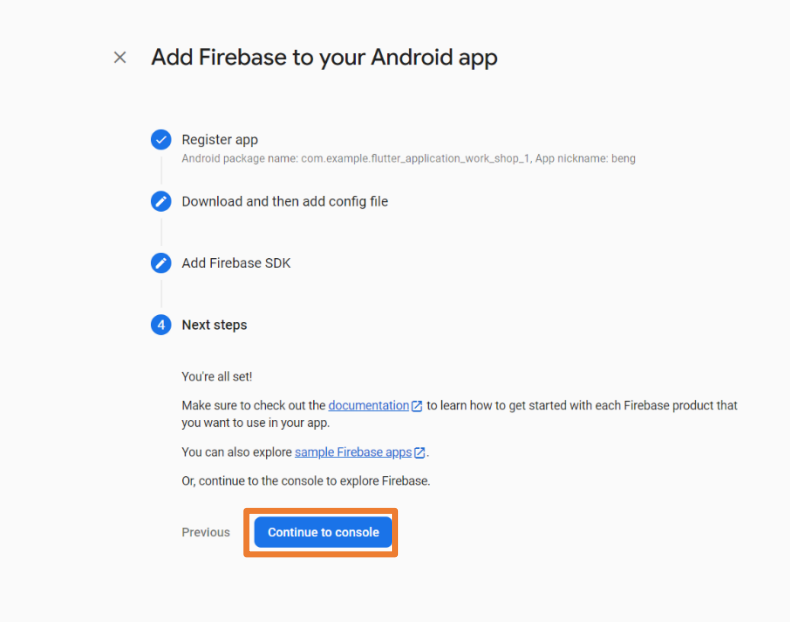
ภาพประกอบ 8.11 ภาพแสดงการ apply plugin

ไปที่ android\app\build.gradle หาแท็ก dependencies และวางimplementation platform('com.google.firebase:firebase-bom:31.2.3') ลงไป ดังภาพประกอบ 8.12

```
65
66  flutter {
67      source '../..'
68  }
69
70  dependencies {
71      implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version
72      implementation platform('com.google.firebase:firebase-bom:31.2.3')
73  }
```

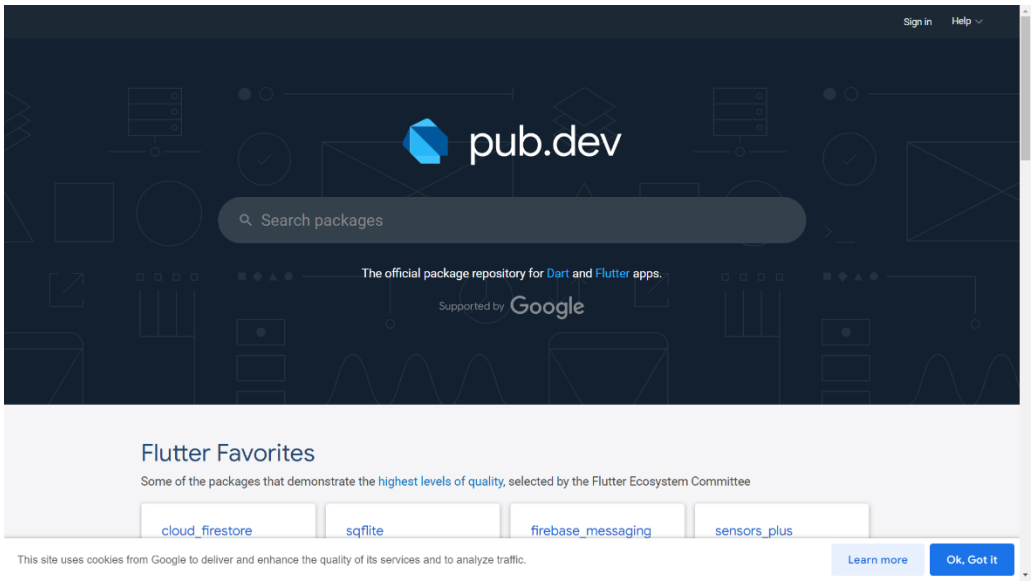
ภาพประกอบ 8.12 ภาพแสดงการวาง implementation platform

เมื่อใส่ข้อมูลครบแล้ว ให้กด Continue to console การเพิ่ม Firebase ในแอป Android ก็เสร็จเรียบร้อยแล้ว ดังภาพประกอบ 8.13



ภาพประกอบ 8.13 ภาพแสดงการเพิ่ม Firebase ในแอป Android เมื่อเพิ่มข้อมูลครบถ้วน

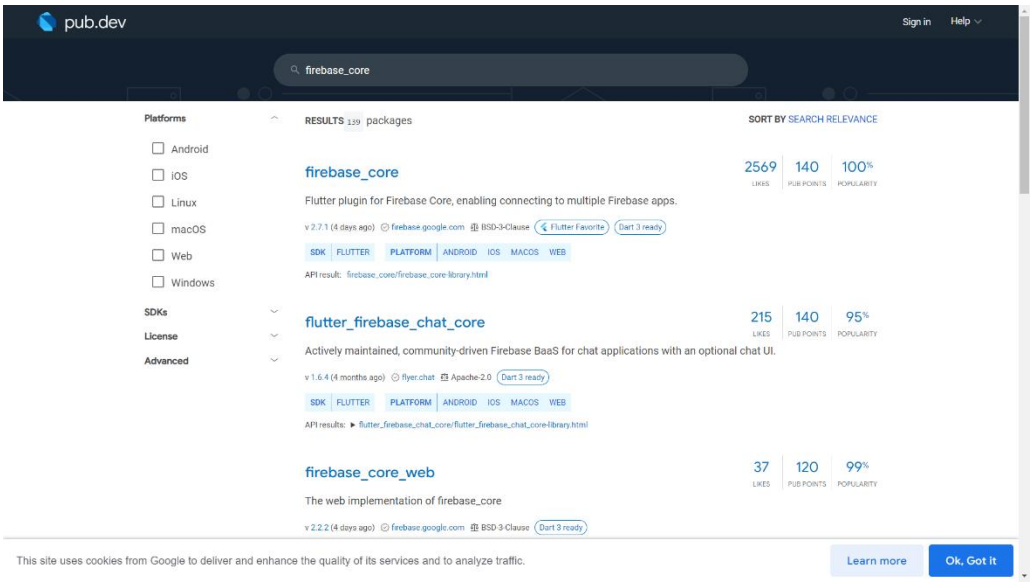
2.5) การใช้งาน Firebase มีบริการหลายอย่างมาก ๆ ซึ่งการใช้งานในบริการต่างๆ ของ Firebase จำเป็นต้องติดตั้ง library ในการใช้งาน หา library ได้จาก https://pub.dev/ ดังภาพประกอบ 8.14



ภาพประกอบ 8.14 ภาพแสดงเว็ปไซต์ pub.dev(https://pub.dev/)

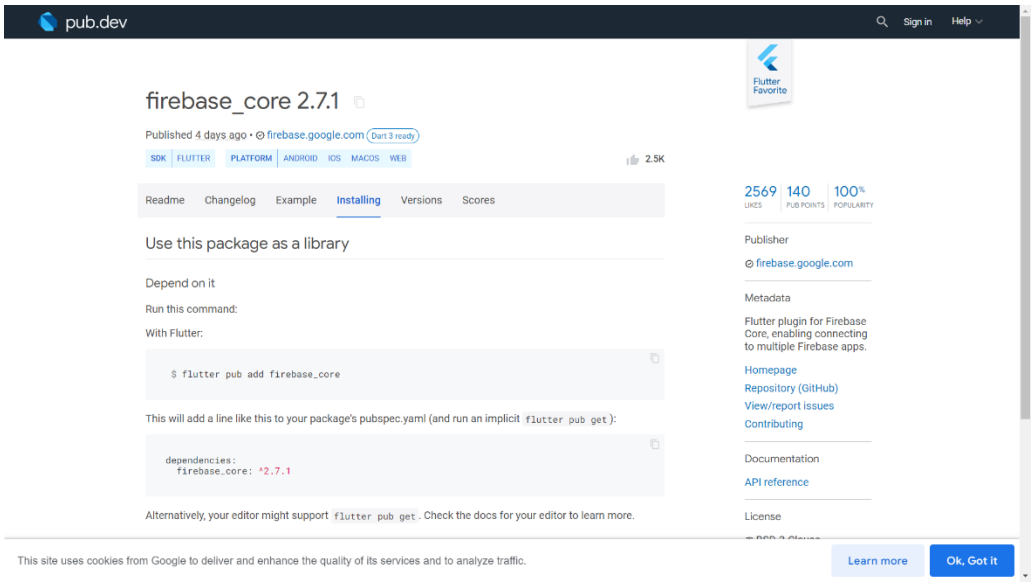
3. การติดตั้ง library ในโปรเจค Flutter

3.1) กรอกชื่อ library ที่เราต้องการใช้งานแล้วกดค้นหา ผลการค้นหาจะแสดงดังภาพประกอบ 8.15 ซึ่งจะมีให้เลือกใช้งานมากมาย



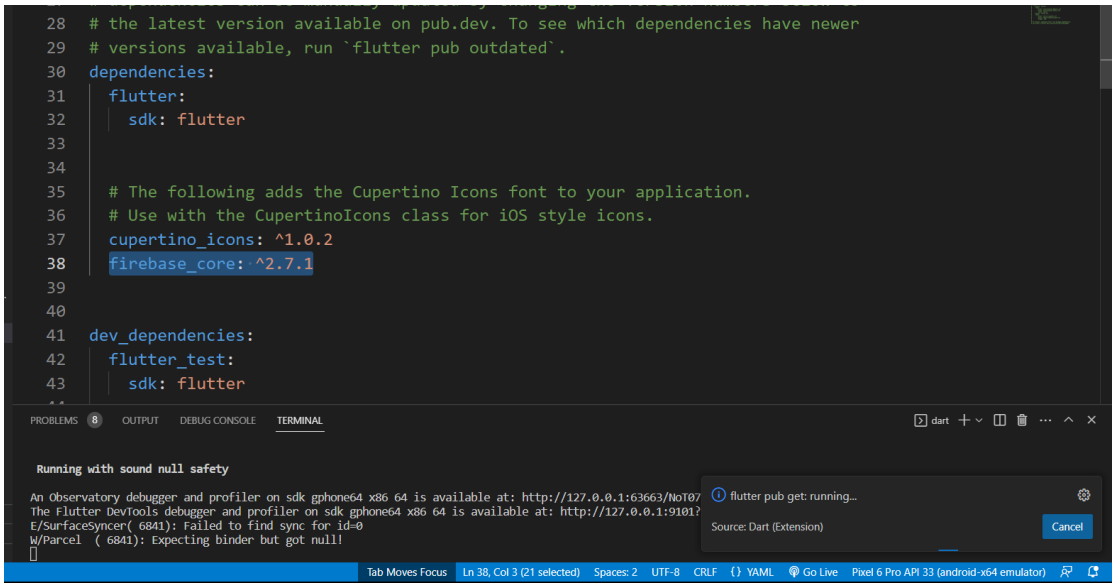
ภาพประกอบ 8.15 ภาพแสดงผลการค้นหา library

3.2) จากนั้นให้ทำการคลิกที่ library ที่เราต้องการ ไปที่แท็บเมนู installing แล้วเลื่อนหา firebase_core: ^2.7.1 แล้วทำการ Copy ไว้ ดังภาพประกอบ 8.16



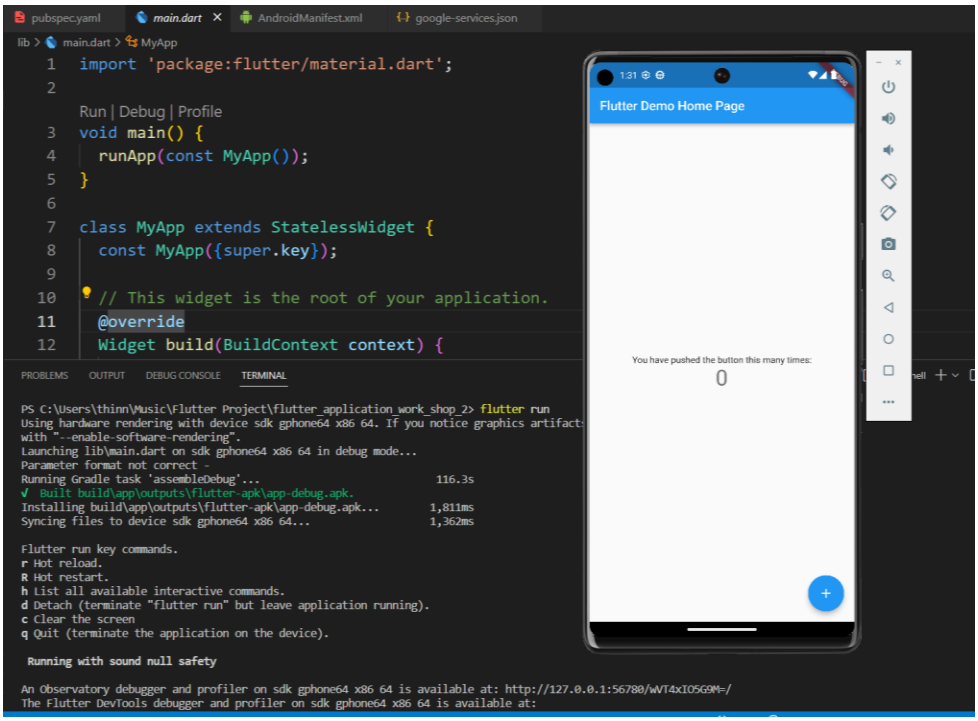
ภาพประกอบ 8.16 ภาพแสดง library ที่เราต้องการใช้งาน

3.3) ไปที่ไฟล์ pubspec.yaml แล้วนำ firebase_core: ^2.7.1 ไปวางไว้ แล้วทำการกดบันทึก จะขึ้น flutter pub get :running ที่มุมขาวล่างของ VS Code แล้วทำการเปิด imurater หรือโปรแกรมจำลอง android ทำการสั่ง run หาก run สำเร็จ การติดตั้ง library ก็เสร็จสิ้น ดังภาพประกอบ 8.17



ภาพประกอบ 8.17 ภาพแสดงการติดตั้ง library

หาก run และสามารถเปิดหน้าแอปพลิเคชันได้ตามปกติ ไม่ error สามารถเขียน code เรียกใช้งาน library ได้เลย ดังภาพประกอบ 8.18



ภาพประกอบ 8.18 ภาพแสดงการ run หลังการติดตั้ง library สำเร็จ

3.4) หากมีการติดตั้ง library เพิ่มเติมให้เราสั่ง run แอปพลิเคชันใหม่ทุกครั้ง

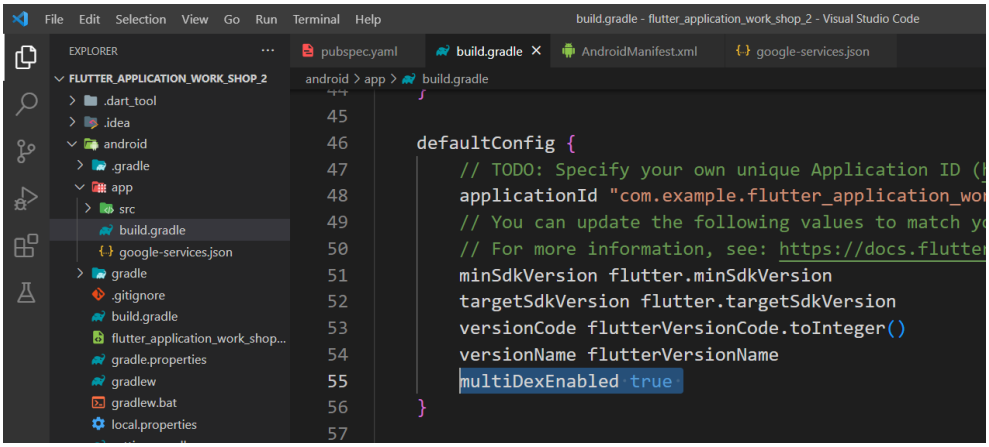
4. การใช้งาน Cloud Firestore

4.1 เริ่มต้นด้วยการติดตั้ง library ดังต่อไปนี้

- firebase_core

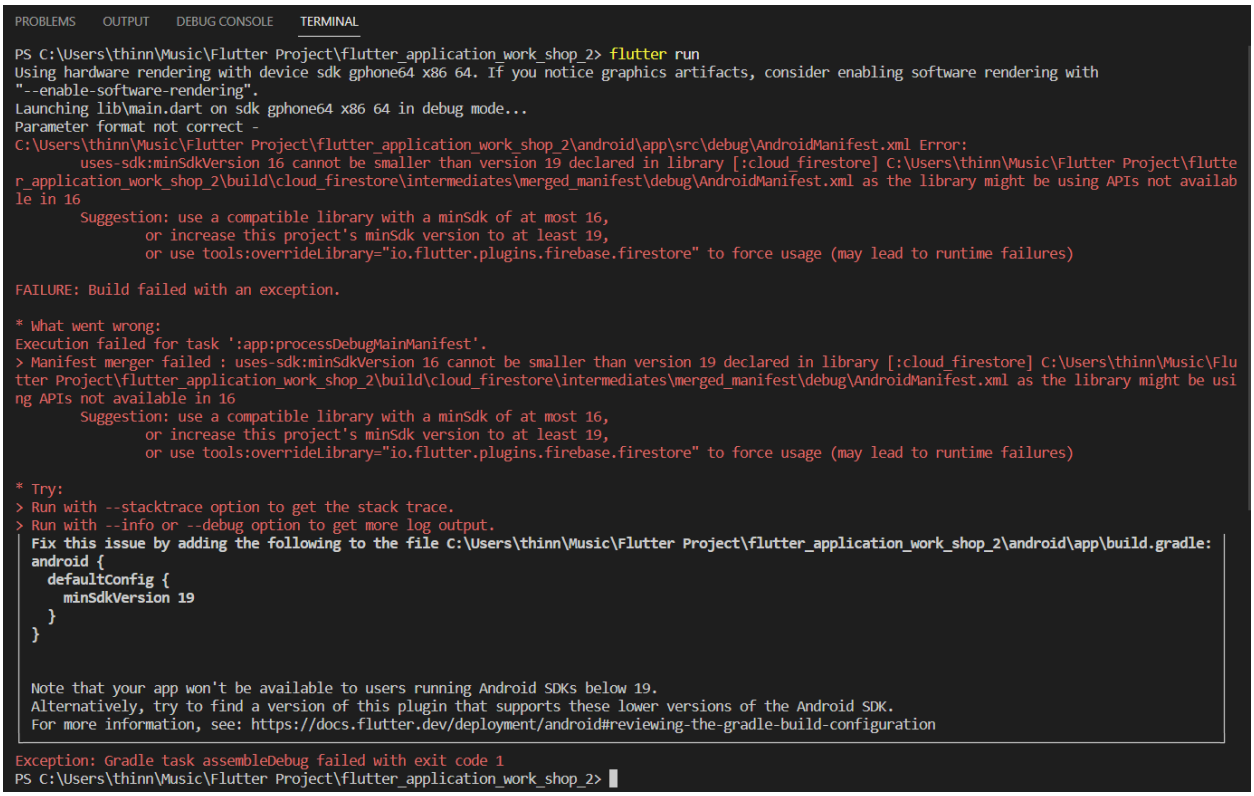
- cloud firestore

การใช้งาน cloud firestore ถ้าเจอ ERROR: Cannot fit requested classed in a single dex file แก้ด้วยการเพิ่ม multiDexEnabled true ใน android -> app -> build.gradle ในแท็ก defaultConfig



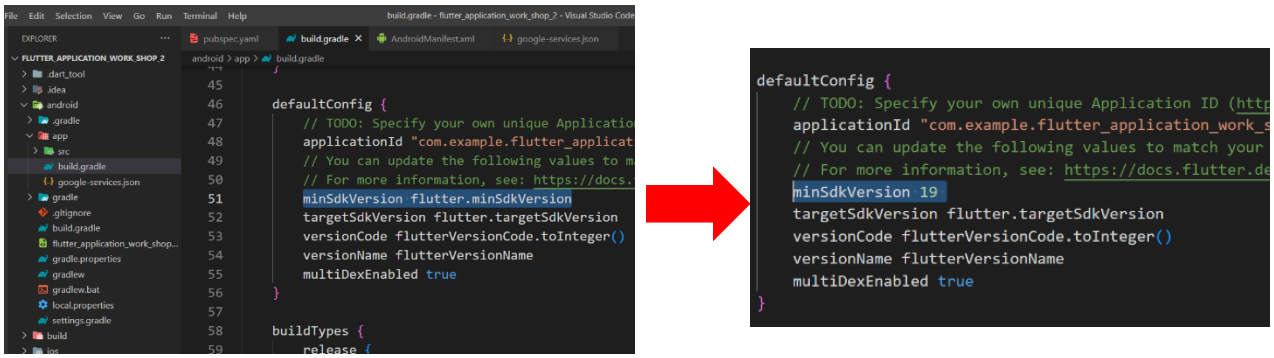
ภาพประกอบ 8.19 ภาพแสดงการเพิ่ม multiDexEnabled true

ถ้าเจอ Error:uses-sdk:minSdkVersion 16 cannot be smaller than version 19 declared in library [:cloud_firestore] ดังภาพประกอบ 8.20



ภาพประกอบ 8.20 ภาพแสดง Error:uses-sdk:minSdkVersion

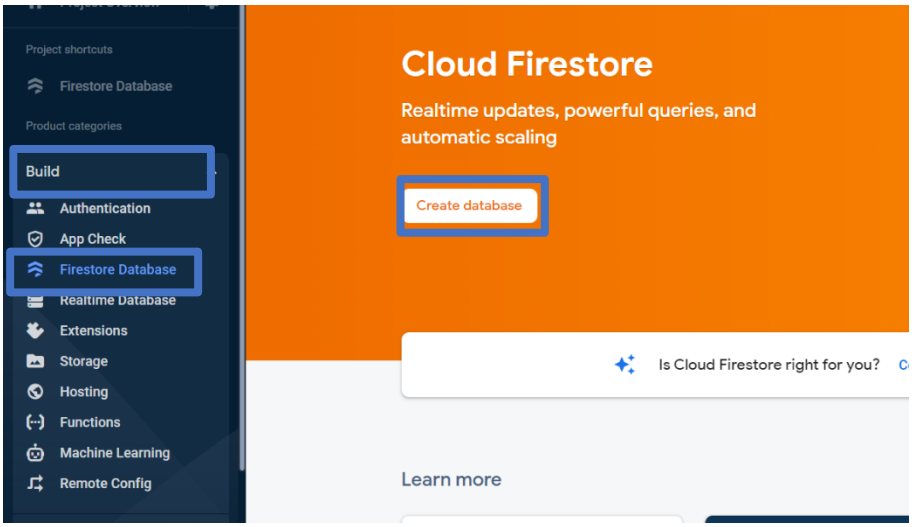
ให้ทำการไปแก้ไป minSdkVersion flutter.minSdkVersion ที่แท็ก defaultConfig ให้เป็น minSdkVersion 19 ดังภาพประกอบ 8.21



ภาพประกอบ 8.21 ภาพแสดงการเพิ่ม minSdkVersion

4.2 เมื่อติดตั้ง library สำเร็จแล้ว ให้ไปที่ Firebase โปรเจค เพื่อทำการสร้างฐานข้อมูล Firestore Database ดังภาพประกอบ 8.22

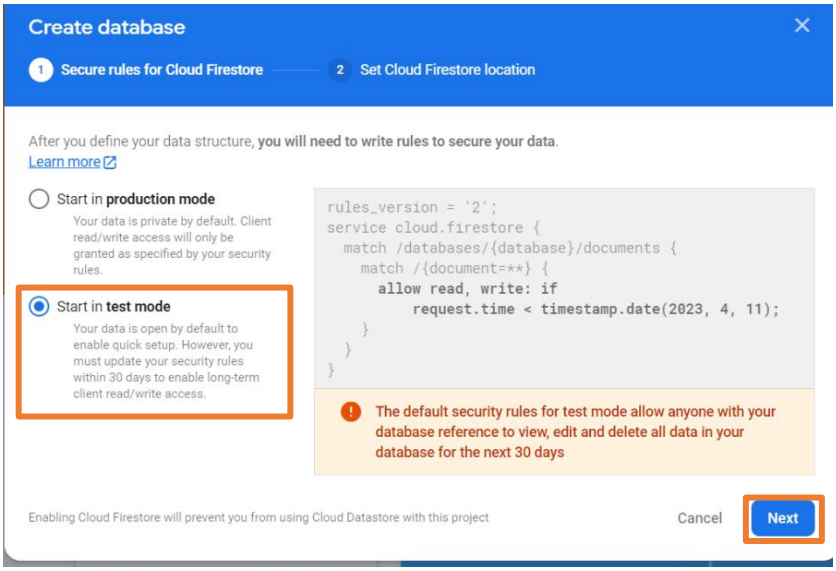
1. ไปที่ build
2. เลือก Firestore Database
3. กด Create database



ภาพประกอบ 8.22 ภาพแสดงการเริ่มต้นสร้าง Firestore Database

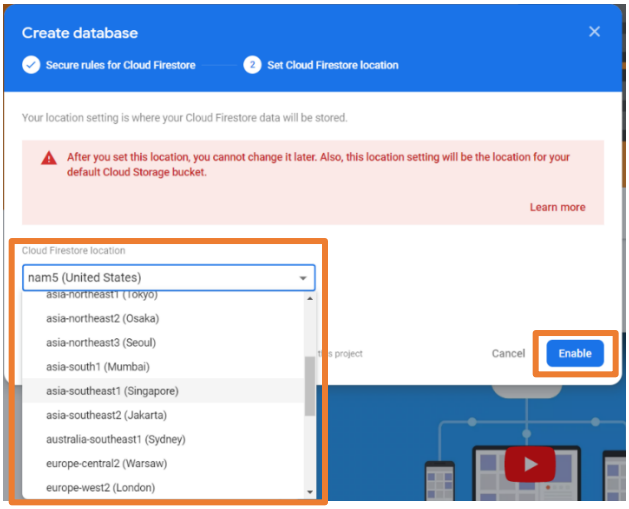
4.3 หลังจากที่ทำการ กดสร้าง Firestore Database จะแสดงดังภาพประกอบ 8.23 – 7.24 ให้ทำตามขั้นตอนดังต่อไปนี้

1. เลือก test โหมด
2. กด Next ดังภาพประกอบ 8.23



ภาพประกอบ 8.23 ภาพแสดงการตั้งค่าเริ่มต้นให้กับ Firestore Database

3. เลือกที่ตั้งในการเก็บข้อมูล ของ Firestore database ของโปรเจค
4. กด Enable ดังภาพประกอบ 8.24



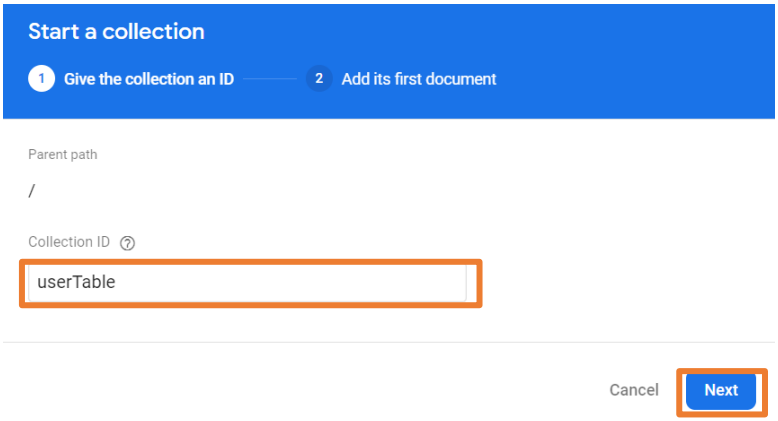
ภาพประกอบ 8.24 ภาพแสดงการเลือกที่ตั้งในการเก็บข้อมูล Firestore Database

5. ทำการเพิ่ม collection เพิ่มข้อมูลลงไป

   1 Start collection

   2 ใส่ชื่อ collection ลงไป

   3 กดถัดไป ดังภาพประกอบ 8.25



ภาพประกอบ 8.25 ภาพแสดงการเพิ่ม collection

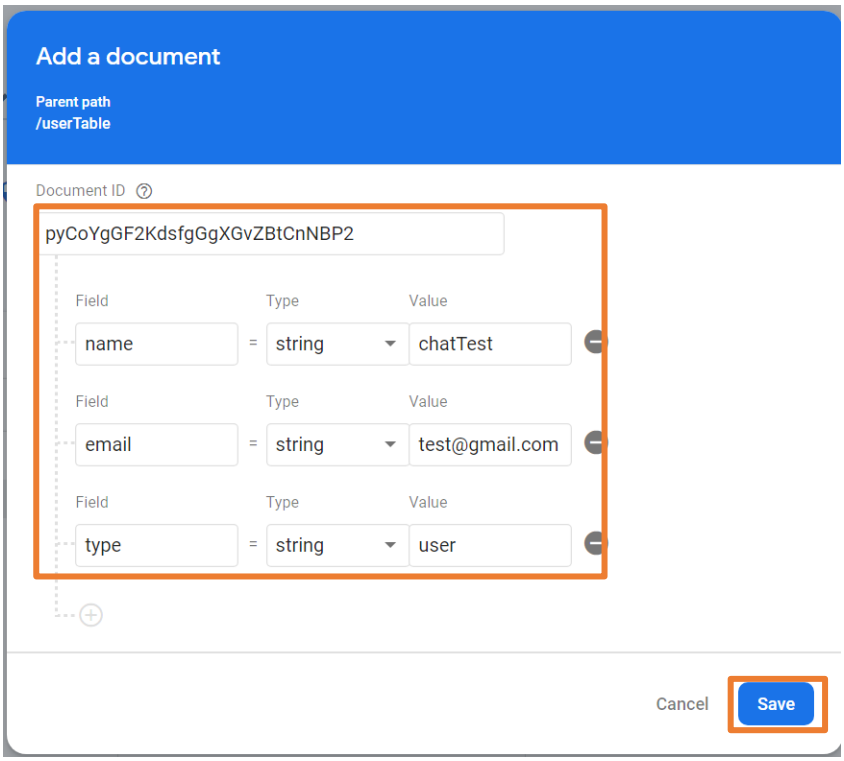
6. ทำการเพิ่ม ข้อมูลลงไปใน document

   1.ใส่ชื่อ Field ลงไป

   2. ระบุประเภทข้อมูลที่ต้องการเก็บ

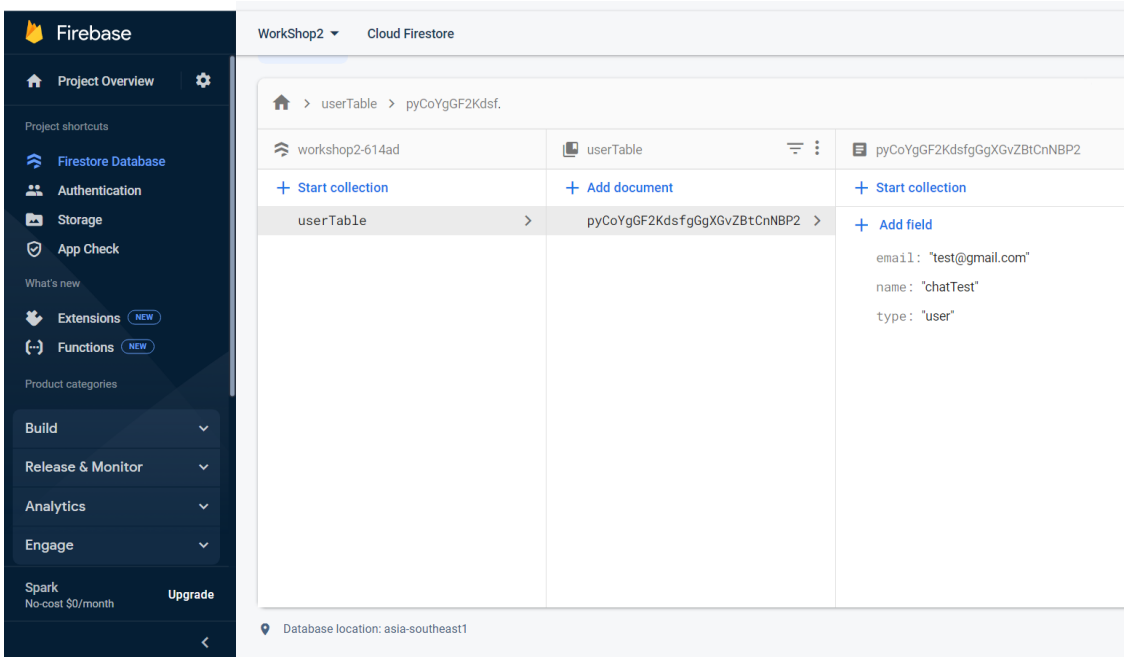
   3. ระบุข้อมูลที่ต้องการเก็บ

   4. กดบันทึก ดังภาพประกอบ 8.26



ภาพประกอบ 8.26 ภาพแสดงการเพิ่มข้อมูลลงไปใน collection

7. เมื่อกรอก Field , Type และ value เสร็จแล้วจะได้ดังภาพ และ กดถัดไป จะได้ข้อมูลดังภาพประกอบ 8.27

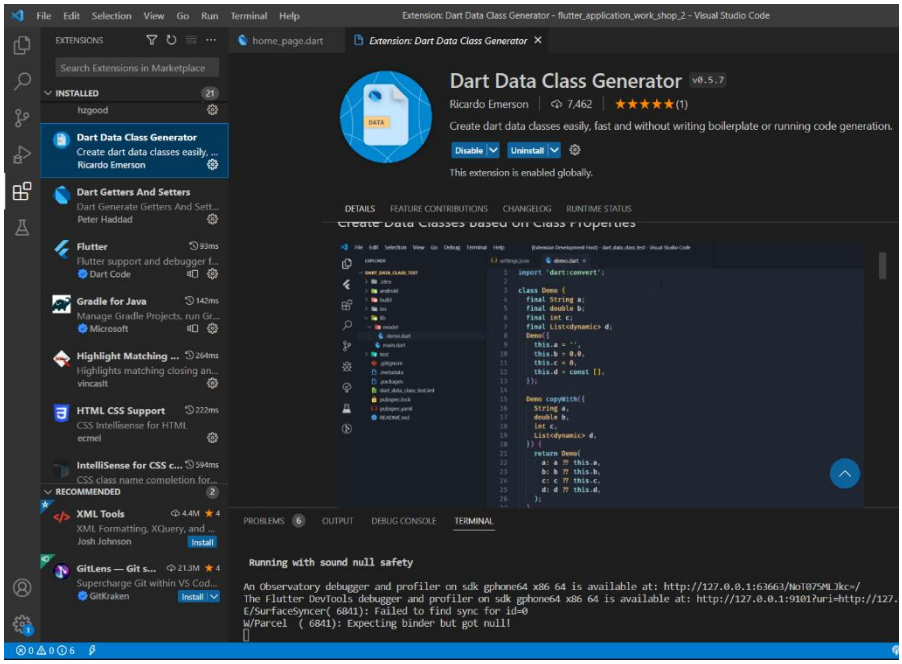


ภาพประกอบ 8.27 ภาพแสดงผลที่เพิ่มลงไปใน collection

5. การเขียนแสดงผลข้อมูลจาก Firestorre database ไปบน imurater

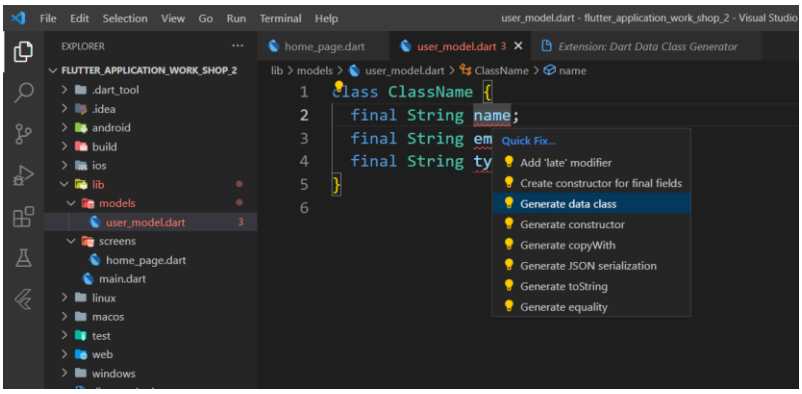
เราจะต้องทำการสร้าง model ในการอ่านค่าข้อมูลที่ได้จาก Firestore database มาเก็บไว้ใน model ที่เราสร้างไว้

5.1 สิ่งแรกที่ต้องมีในการสร้าง model คือ Extension Dart Data Class Generator



ภาพประกอบ 8.28 ภาพแสดง Extension Dart Data Class Generator

5.2 เมื่อลง Extension เรียบร้อยแล้ว ปิด/เปิด Vs Code ใหม่ ทำการสร้างโฟลเดอร์ models ของในโฟลเดอร์ lib เสร็จเรียบร้อยให้ทำการสร้างไฟล์ข้างในโฟลเดอร์ models ชื่อไฟล์ user_model.dart ทำการสร้าง Class ประกาศตัวแปรที่เราการอ่านข้อมูลมาเก็บไว้ แล้วทำการกดที่หลอดไฟ หรือนำเมาส์คลิกตัวแปรที่ประกาศที่ error กด Ctrl+จุด แล้วเลือก Generate data class



ภาพประกอบ 8.29 ภาพแสดงการสร้าง model

เมื่อทำการ Generate data class เรียบร้อยแล้วจะแสดงดังภาพประกอบ 8.30

```dart
import 'dart:convert';

class ClassName {
  final String name;
  final String email;
  final String type;
  ClassName({
    required this.name,
    required this.email,
    required this.type,
  });

  ClassName copyWith({
    String? name,
    String? email,
    String? type,
  }) {
    return ClassName(
      name: name ?? this.name,
      email: email ?? this.email,
      type: type ?? this.type,
    );
  }

  Map<String, dynamic> toMap() {
    final result = <String, dynamic>{};

    result.addAll({'name': name});
    result.addAll({'email': email});
    result.addAll({'type': type});

    return result;
  }
```

ภาพประกอบ 8.30 ภาพแสดง model ที่ Generate แล้ว

5.3 หลังจาที่เราสร้าง model เสร็จเรียบร้อยแล้ว เรามาเริ่มต้นเขียน code ในการอ่านข้อมูลจาก Firestore database มาเก็บไว้ใน model ที่เราสร้างไว้ มีขั้นตอนการเขียน code ดังนี้

1. เริ่มต้นด้วยการประการตัวแปรในรูปแบ List เพราะ model ที่เราสร้างไว้ ในการอ่านข้อมูลออกมาจาก Firestore database มีข้อมูลมากว่า 1 ตำแหน่ง

2. สร้าง initState method เป็น method ที่จะทำงานก่อนทุกครั้งที่เราเข้ามายังหน้าแอปพลิเคชันที่เราทำการสร้าง method นี้ไว้

3. สร้าง Future function คือ function app asynchronous จะดำเนินการรันโปรแกรมทีละชุดคำสั่ง และจะรันชุดคำสั่งถัดไปทันทีโดยไม่จำเป็นต้องรอชุดคำสั่งก่อนหน้าทำงานเสร็จ ตัวอย่างเช่น ถ้าโปรแกรมเรียกฟังก์ชัน A(); และ B(); ตามลำดับ โปรแกรมจะรันฟังก์ชัน A(); และ B(); ตามลำดับโดยไม่สนใจว่าฟังก์ชัน A(); จะทำงานเสร็จรึยัง จะไปเรียกฟังก์ชัน B(); ต่อเลยทันที

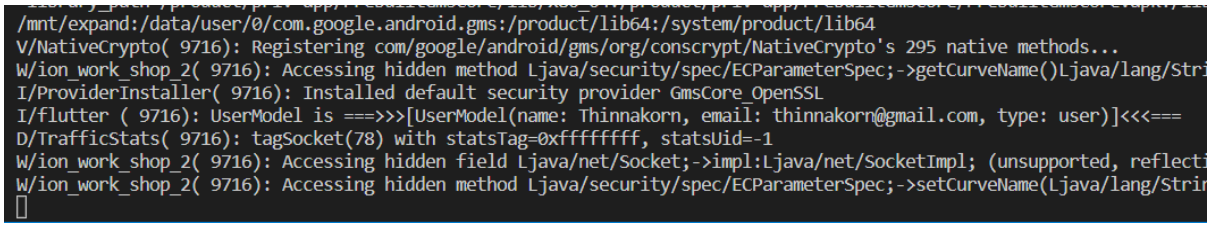
4. เขียน Code เรียกใช้งาน library ของ FirebaseFirestore ในการอ่านค่ามาเก็บไว้ใน model ที่เราสร้างไว้ ดังภาพประกอบ 8.31

```dart
class _HomePageState extends State<HomePage> {
  List<UserModel> userModel = [];

  @override
  void initState() {
    // TODO: implement initState
    super.initState();
    readDataUserModel();
  }

  Future<Null> readDataUserModel() async {
    Firebase.initializeApp().then((value) async {
      FirebaseFirestore.instance
          .collection('userModel')
          .snapshots()
          .listen((event) {
        for (var element in event.docs) {
          UserModel model = UserModel.fromMap(element.data());
          setState(() {
            userModel.add(model);
          });
          print('UserModel is ===>>>$userModel<<<===');
        }
      });
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        leading: Icon(Icons.home),
        title: Text('Home Page'),
      ),
    );
  }
}
```

ภาพประกอบ 8.31 ภาพแสดงการอ่านค่าข้อมูลจาก Firestore database มาเก็บไว้ใน model

5. นำ Future function ที่สร้างไปใส่ไว้ใน initState method แล้วทำการสั่ง run โปรเจค Flutter

6. เมื่อ run โปรเจคสำเร็จ ให้ไปดู console log terminal บน Vs code หาคำว่า UserModel is ===>>>$userModel<<<=== ที่เราสั่งปริ้นไว้ใน Future function จะได้ผลลัพธ์ดังภาพประกอบ 8.31



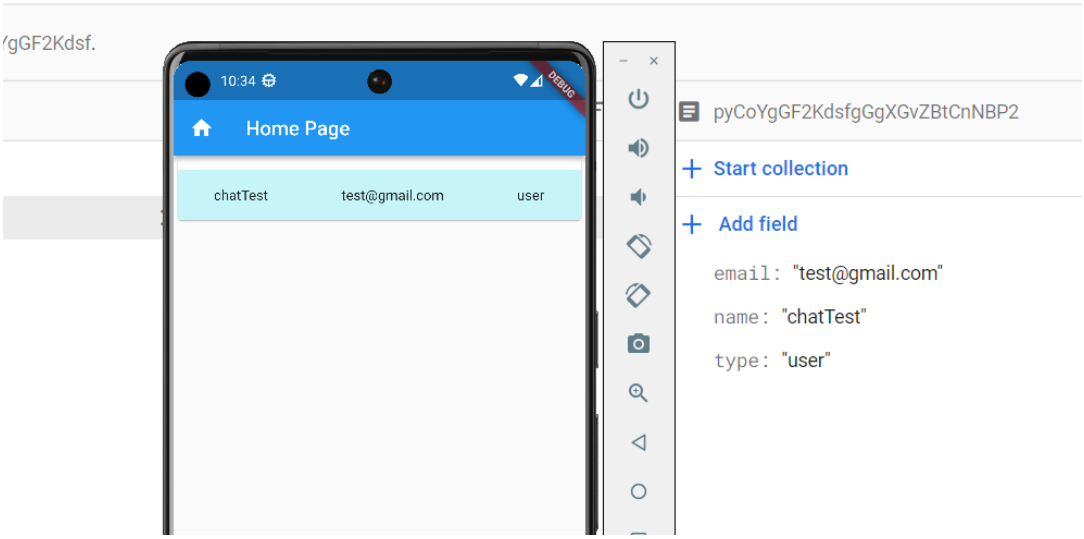
ภาพประกอบ 8.32 ภาพแสดงการข้อมูลของ userModel ที่ได้บน terminal ใน Vs code

7. เมื่อเราได้ข้อมูลดังภาพประกอบ 8.32 แล้ว เรามาเริ่มต้นการเขียน code แสดงข้อมูลที่ได้บน imurater หรือตัวจำลอง android โดยเราจะเขียนในส่วนของ body ดังภาพประกอบ 8.33

```
body: userModel.isEmpty
          ? Text('ไม่พบข้อมูลผู้ใช้งาน')
          : ListView.builder(
              itemCount: userModel.length,
              itemBuilder: (context, index) => Card(
                child: Container(margin: EdgeInsets.only(top: 10),
                  color: Color.fromARGB(255, 198, 244, 248),
                  height: 50,
                  child: Row(
                    mainAxisAlignment: MainAxisAlignment.spaceAround,
                    children: [
                      Text(userModel[index].name),
                      Text(userModel[index].email),
                      Text(userModel[index].type),
                    ],
                  ),
                ),
              ),
            ),
```

ภาพประกอบ 8.33 ภาพแสดงการเขียน code ในการแสดงผลข้อมูล บน imurater

8. เมื่อทำการเขียน Code เสร็จเรียบร้อยแล้วให้ทำการสั่ง run โปรเจคและดูผลลัพธ์ หากเขียนถูกต้องจะแสดงดังภาพประกอบ 8.34



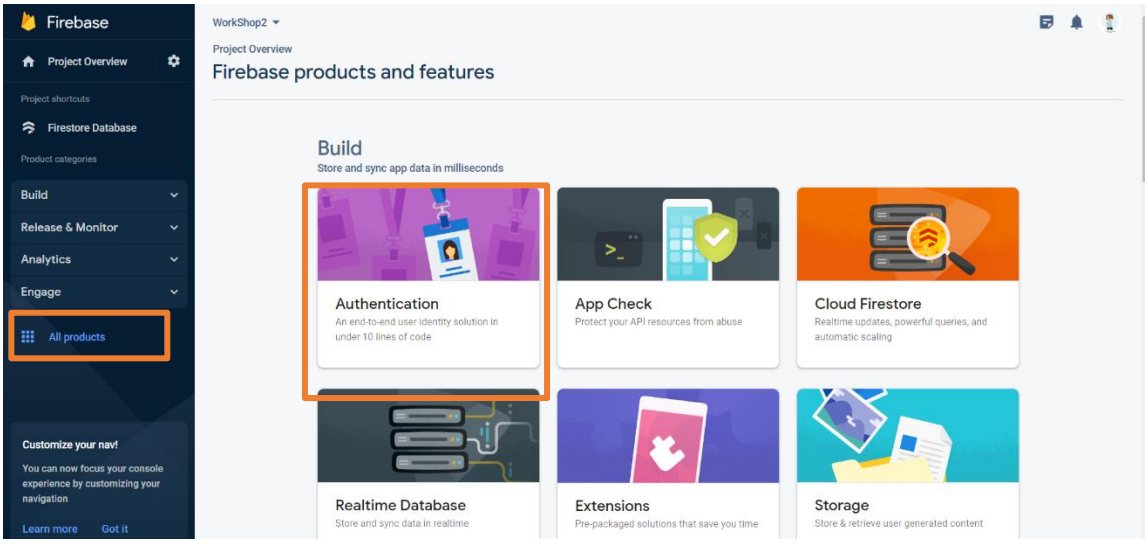
ภาพประกอบ 8.34 ภาพแสดงผลข้อมูล บน imurater และนำข้อมูลไปเปรียบเทียบกับ Firestore database

6. การใช้งาน Firebase Authenication

เริ่มต้นด้วยการติดตั้ง library ดังต่อไปนี้

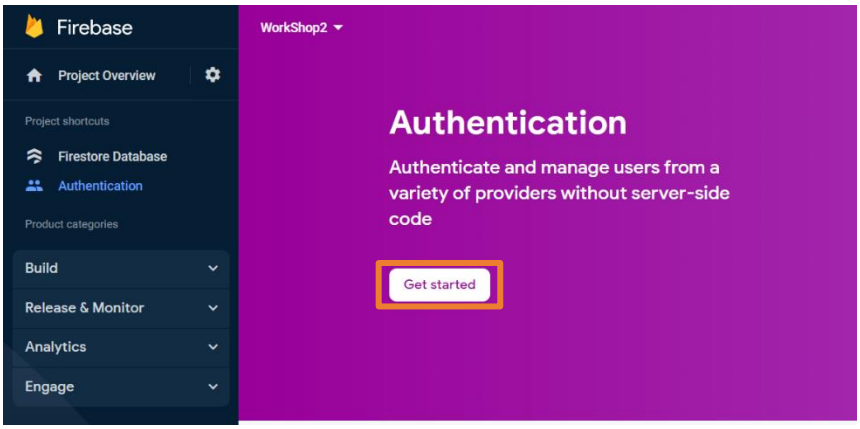
- firebase_auth

6.1 ไปที่ All products เลือกที่ Authentication ดังภาพประกอบ 8.35



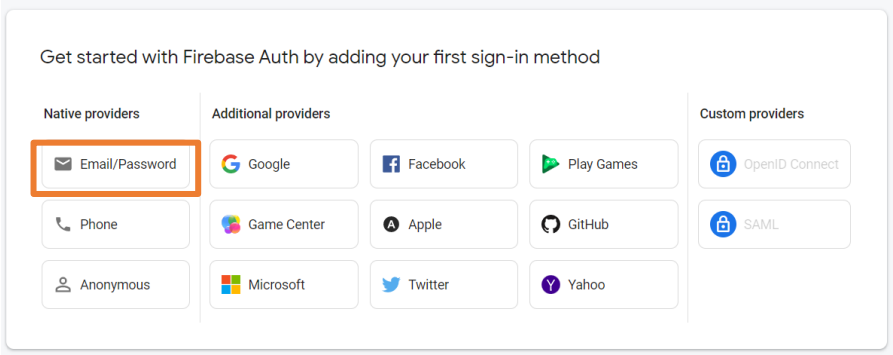
ภาพประกอบ 8.35 ภาพแสดงการเริ่มใช้งาน Firebase Authentication

6.2 ทำการกดที่ Get Started ดังภาพประกอบ 8.36



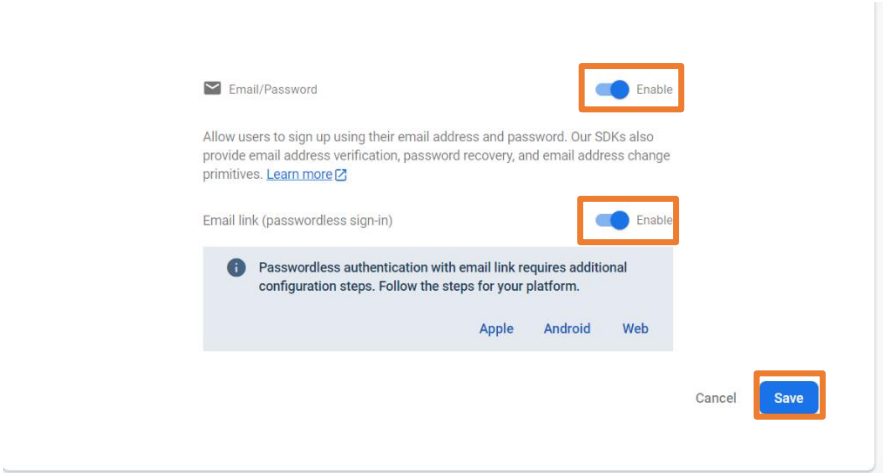
ภาพประกอบ 8.36 ภาพแสดงการเปิดการใช้งาน Firebase Authentication

6.3 เลือกแพลตฟอร์มที่ต้องการใช้งาน ในการล็อกอิน ดังภาพประกอบ 8.37



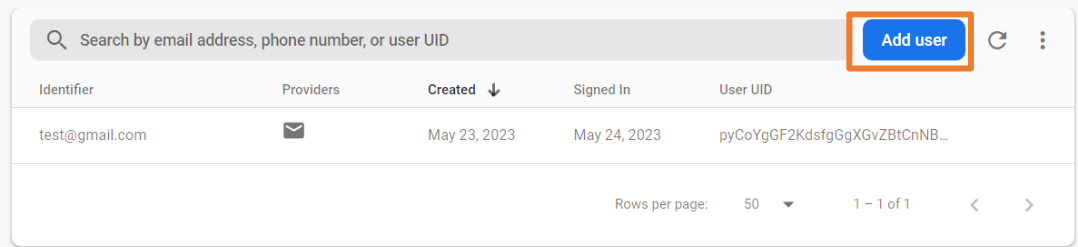
ภาพประกอบ 8.37 ภาพแสดงแพลตฟอร์มที่สามารถล็อดอินผ่าน Firebase Authentication

6.4 เลือกการล็อกอินด้วย Email/Password แล้วทำการเปิดการใช้งานและกด Save ให้เรียบร้อย การเปิดการใช้งาน Authentication ก็เป็นอันเสร็จสิ้น ดังภาพประกอบ 8.38



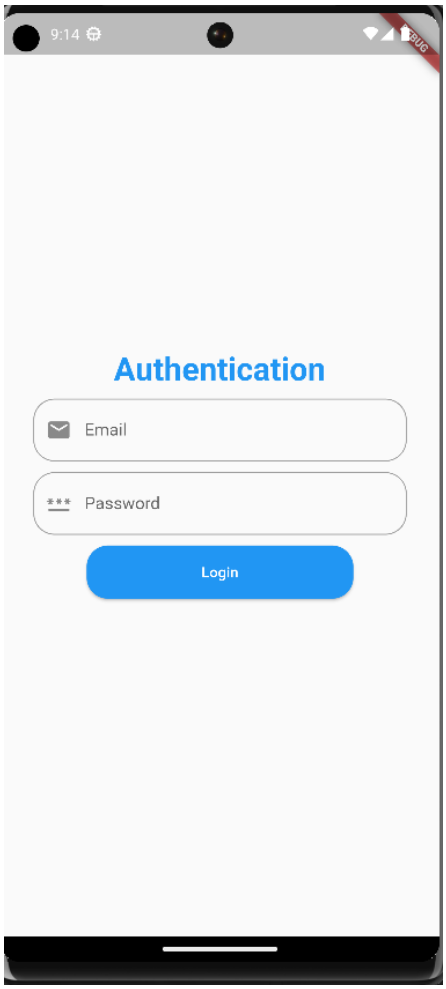
ภาพประกอบ 8.38 ภาพแสดงเปิดการใช้งาน Email/Password

6.5 ทำการเพิ่ม Users โดยกดที่ปุ่ม Add user ใส่ Email/Password ให้ครบถ้วนทำการกดบันทึกให้เรียบร้อย หากเพิ่มข้อมูลสำเร็จจะแสดงดังภาพประกอบ 8.39



ภาพประกอบ 8.39 ภาพแสดง User ที่ทำการเพิ่มข้อมูลเข้ามา

6.6 ออกแบบหน้าแอปพลิเคชันในการล็อกอินให้เรียบร้อย ดังภาพประกอบ 8.40



ภาพประกอบ 8.40 ภาพแสดงตัวอย่างการออกแบบหน้าล็อกอิน

จากภาพประกอบ 8.40 จะมี widget ที่ใช้งานในหน้าแอปพลิเคชัน ดังนี้ Text, TextField, ElevatedButton ดังภาพประกอบ 8.41- 8.42

```dart
Widget emailTextField() {
    return Container(
      margin: EdgeInsets.only(top: 10),
      width: 350,
      child: TextField(
        decoration: InputDecoration(
          prefixIcon: Icon(Icons.email_rounded),
          label: Text('Email'),
          enabledBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(20),
            borderSide: BorderSide(color: Colors.grey),
          ),
          focusedBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(20),
            borderSide: BorderSide(color: Colors.blue),
          ),
        ),
      ),
    );
  }
```

ภาพประกอบ 8.41 ภาพแสดงตัวอย่างการใช้งาน TextField

```dart
Widget loginElevatedButton() {
    return Container(
      margin: EdgeInsets.only(top: 10),
      width: 250,
      height: 50,
      child: ElevatedButton(
        style: ElevatedButton.styleFrom(
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(20),
          ),
        ),
        onPressed: () {},
        child: Text('Login'),
      ),
    );
  }
```

ภาพประกอบ 8.42 ภาพแสดงตัวอย่างการใช้งาน ElevatedButton

6.7 เมื่อออกแบบหน้าแอปพลิเคชันเรียบร้อยแล้ว ให้ไปทำการประการตัวแปรที่ใช้ในการอ่านค่าจากฐานข้อมูลมาเก็บไว้ ดังภาพประกอบ 8.43

```dart
class _AuthenticationState extends State<Authentication> {

  String? email, password, type; // ใช้ในการรับข้อมูลจาก TextField และเช็คประเภทการล็อกอิน
  List<UserModel> userModel = []; // model ในการรับข้อมูลจาก Firestore database

  @override
  Widget build(BuildContext context) {
    return Scaffold();
```

ภาพประกอบ 8.43 ภาพแสดงตัวอย่างประกาศตัวแปร

6.8 นำตัวแปรที่ประกาศในการใช้งานรับค่าจาก TextField ไปเพิ่มในแต่ละ TextField ที่ต้องการรับค่าและนำข้อมูลที่ได้มาใช้งานต่อ ดังภาพประกอบ 8.44

```dart
Widget emailTextField() {
    return Container(
      margin: EdgeInsets.only(top: 10),
      width: 350,
      child: TextField(
        onChanged: (value) => email = value.trim(),
        decoration: InputDecoration(
          prefixIcon: Icon(Icons.email_rounded),
          label: Text('Email'),
          enabledBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(20),
            borderSide: BorderSide(color: Colors.grey),
          ),
          focusedBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(20),
            borderSide: BorderSide(color: Colors.blue),
          ),
        ),
      ),
    );
  }
```

ภาพประกอบ 8.44 ภาพแสดงตัวอย่างการเพิ่มตัวแปรเพื่อรับค่าข้อมูลจากการกรอก บน TextField

6.9 เขียนเช็คข้อมูลที่ได้รับจากการกรอกบน TextField กับ Firebase Authentication และ Firestore database โดยสร้าง Future function ในการเช็คข้อมูลดังภาพประกอบ 8.45

```
Future<Null> checkAuthen() async {
   await Firebase.initializeApp().then((value) async {
     await FirebaseAuth.instance
         .signInWithEmailAndPassword(email: email!, password: password!)
         .then((value) async {
       String? uid = value.user!.uid;
       await FirebaseFirestore.instance
           .collection('userTable')
           .doc(uid)
           .snapshots()
           .listen((event) {
         UserModel model = UserModel.fromMap(event.data()!);
         switch (model.type) {
           case 'user':
             Navigator.pushAndRemoveUntil(
                 context,
                 MaterialPageRoute(
                   builder: (context) => HomePage(),
                 ),
                 (route) => false);
             break;
           default:
         }
       });
     });
   });
 }
```

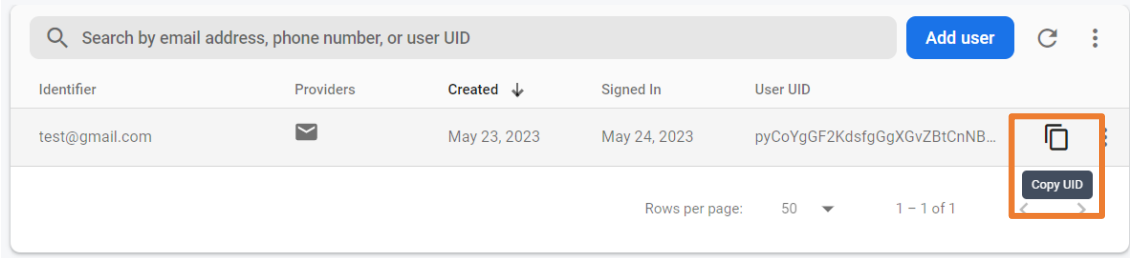
ภาพประกอบ 8.45 ภาพแสดงการเช็กการล็อกอิน Firebase

6.10 นำ Future function ที่สร้างขึ้นไปใส่ในปุ่ม Login ในส่วนของ onPressed ดังภาพประกอบ 8.46

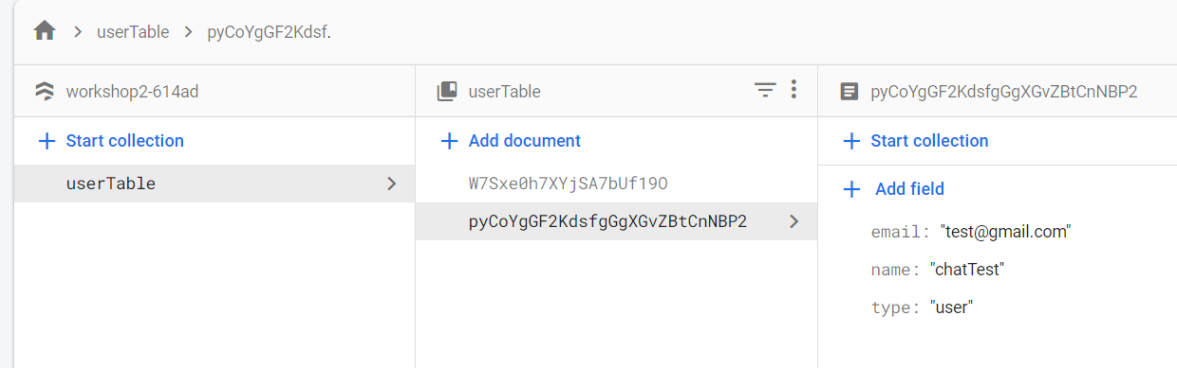
```dart
Widget loginElevatedButton() {
  return Container(
    margin: EdgeInsets.only(top: 10),
    width: 250,
    height: 50,
    child: ElevatedButton(
      style: ElevatedButton.styleFrom(
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(20),
        ),
      ),
      onPressed: () {
        checkAuthen();
      },
      child: Text('Login'),
    ),
  );
}
```

ภาพประกอบ 8.46 ภาพแสดงการเพิ่ม Future function checkAuthen ในปุ่ม Login

6.11 เนื่องจาก UID ของ User ที่สมัครไว้ใน Authentication ไม่เหมือนกันกับ UID ของ Collection Firestore database ให้ไปที่ Firebase Authentication ทำการ Copy UID ของ User ไว้ นำ UID ที่ได้ไปเพิ่ม Document ใหม่ ใน Cloud Firestore ดังภาพประกอบ 8.47

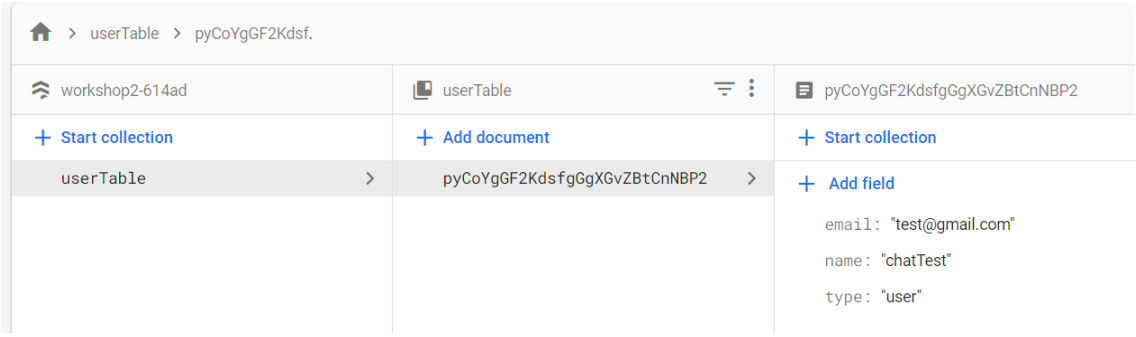


ภาพประกอบ 8.47 ภาพแสดงการ Copy UID User



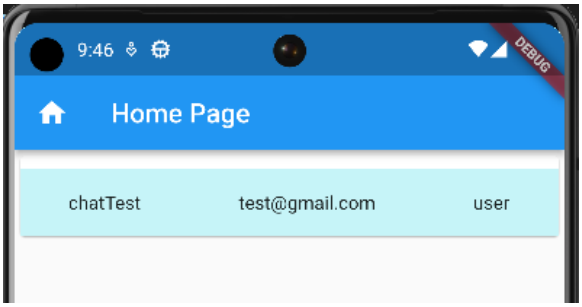
ภาพประกอบ 8.48 ภาพแสดงข้อมูลหลังการเพิ่ม Document

เมื่อได้ดังภาพประกอบ 8.48 แล้วให้ทำการลบ Document ตัวเดิมออกเนื่องจาก UID ของ Document ดังกล่าวไม่ตรงกับ UID User ที่เราทำการสมัครไว้บน Firebase Authentication ลบข้อมูลเรียบร้อยแล้วจะได้ดังภาพประกอบ 8.49



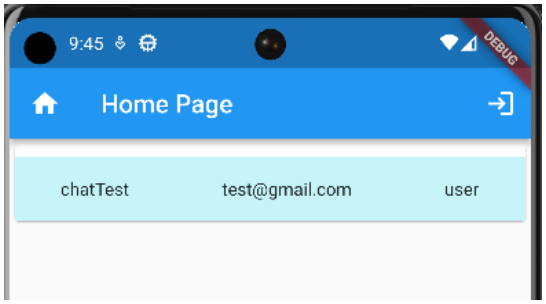
ภาพประกอบ 8.49 ภาพแสดงข้อมูลหลังการลบ Document

6.12 สั่ง Run แอปพลิเคชัน ทำการล็อกอินผ่านหน้าแอปพลิเคชัน โดยใส่ email และ password ที่ได้ทำการสมัครไว้บน Firebase Authentication หาก email password และ type ถูกต้องจะล็อกอินไปยังหน้า Home Page ดังภาพประกอบ 8.50



ภาพประกอบ 8.50 ภาพแสดงหน้า Home Page

6.13 เมื่อเราสามารถล็อกอินเข้ามาใช้งานแอปพลิเคชันได้แล้ว ให้ทำการสร้างปุ่มสำหรับใช้ในการล็อกเอาท์ และทำการเขียน code ในการล็อกเอาท์ ดังภาพประกอบ 8.51 – 7.53



ภาพประกอบ 8.51 ภาพแสดงปุ่มสำหรับการล็อกเอาท์

```
appBar: AppBar(
        leading: Icon(Icons.home),
        title: Text('Home Page'),
        actions: [
          IconButton(
              onPressed: () {
                signOut();
              },
              icon: Icon(Icons.login))
        ],
      ),
```

ภาพประกอบ 8.52 ภาพแสดงการเขียน Code เพิ่ม IconButton ไว้บน AppBar

```dart
Future<Null> signOut() async {
  await Firebase.initializeApp().then((value) async {
    await FirebaseAuth.instance.signOut().then((value) {
      Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => Authentication(),
          ));
    });
  });
}
```

ภาพประกอบ 8.53 ภาพแสดงการเขียน Code ล็อกเอาท์ หรือลงชื่อออกจากระบบ

7. การใช้งาน image_picker ร่วมกับ Sotorage

image_picker เป็น library ที่ใช้สำหรับ การเพิ่มรูปจากกล้อง แกลลอรี่ และวีดีโอ

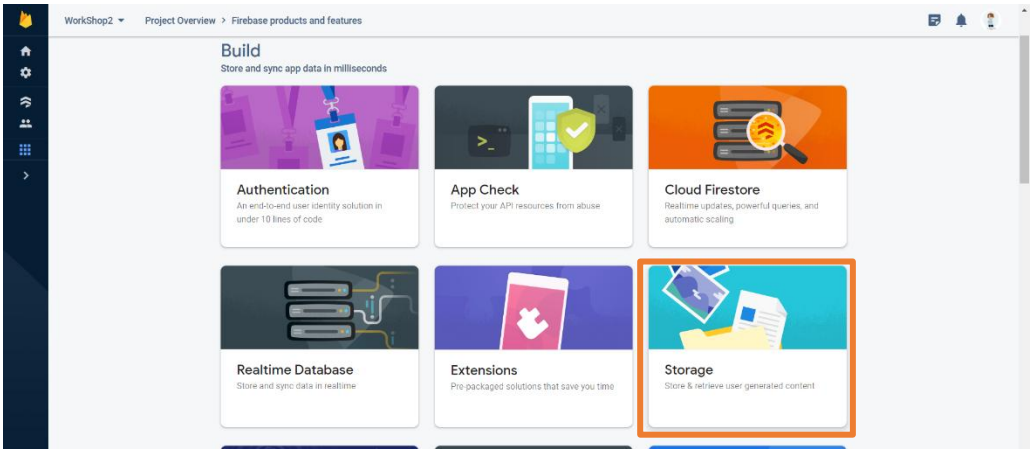
Sotorage เป็นบริการที่ใช้สำหรับการเก็บข้อมูลรูปภาพของ Firebase

เริ่มต้นด้วยการติดตั้ง library ดังต่อไปนี้

- image_picker

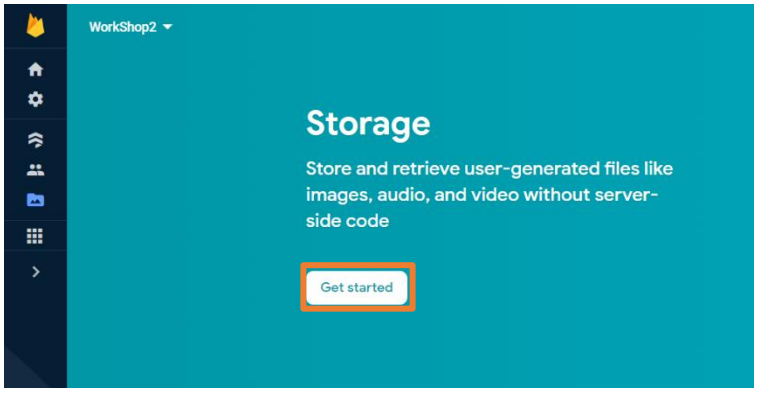
- firebase_storage

7.1 ไปที่ All products เลือกที่ Storage ดังภาพประกอบ 8.54



ภาพประกอบ 8.54 ภาพแสดงเริ่มต้นใช้งาน Storage

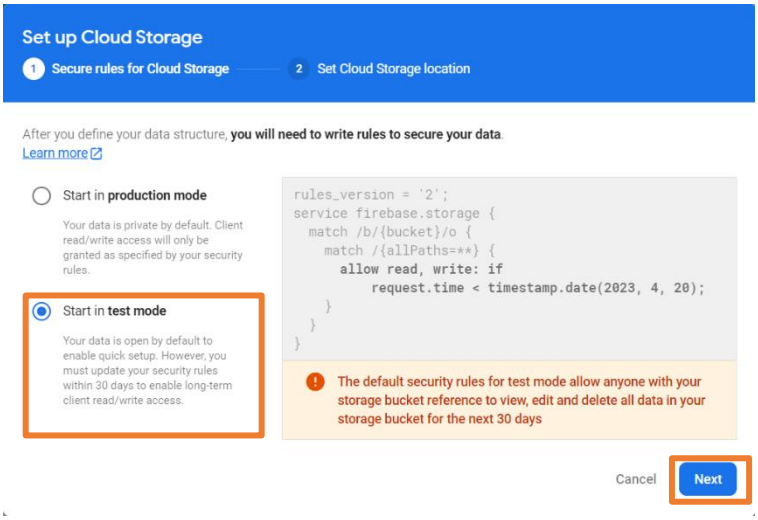
7.2 ทำการกดที่ Get Started ดังภาพประกอบ 8.55



ภาพประกอบ 8.55 ภาพแสดงการเปิดการใช้งาน Storage

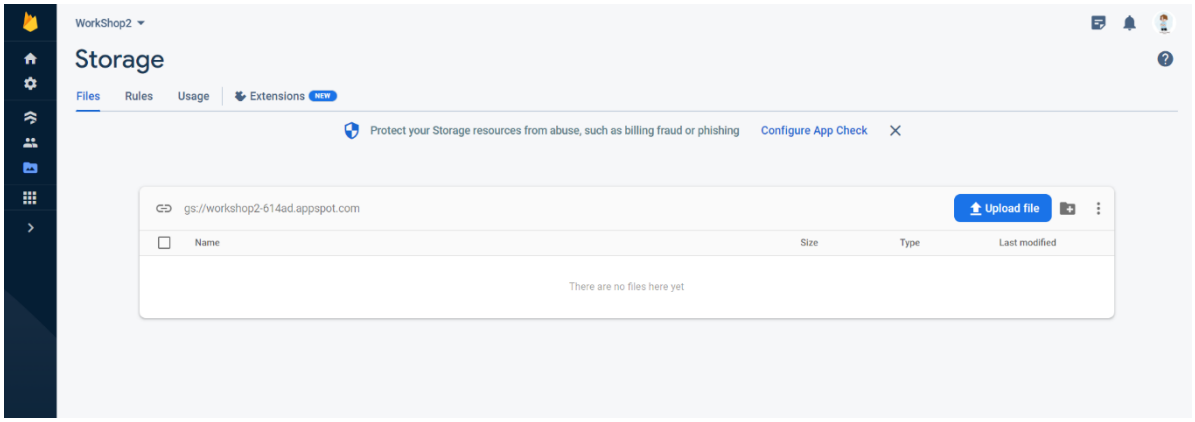
7.3 หลังจากที่ทำการ กดสร้าง Firestore Database จะแสดงดังภาพประกอบ 8.56

1. เลือก test โหมด
2. กด Next
3. เลือกที่ตั้งในการเก็บข้อมูล ของ Storage ของโปรเจค หากทำการเลือกที่เก็บข้อมูลตังแต่ใช้งารงาน FireStore database ไม่ต้องเลือกที่เก็บข้อมูลใหม่สามารถถกดสร้าง Storage ได้เลย
4. กด Done ดังภาพประกอบ 8.56



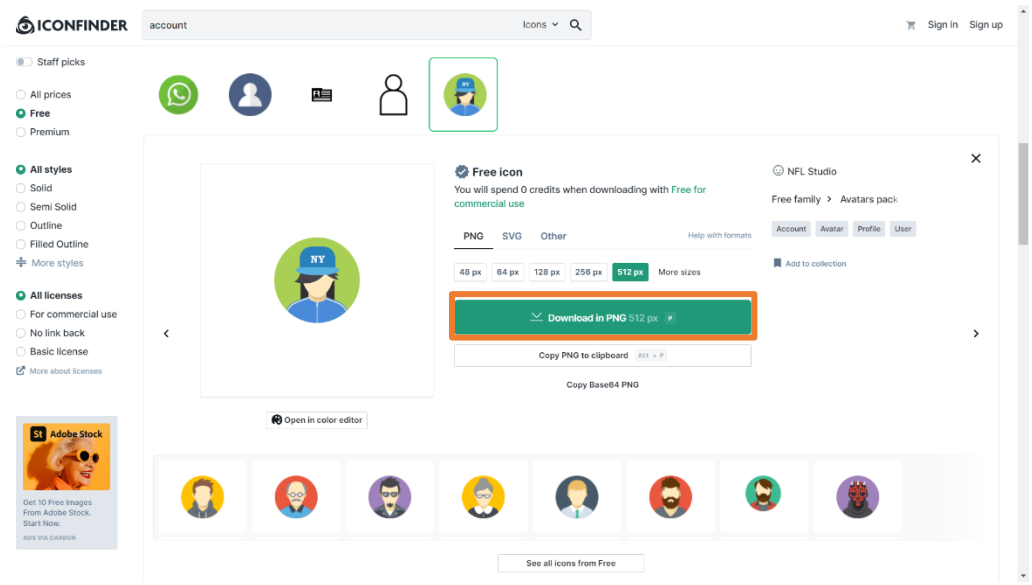
ภาพประกอบ 8.56 ภาพแสดงการตั้งค่าเริ่มต้นให้กับ Storage

หลังจากที่ทำการสร้างเรียบร้อยแล้ว จะได้ดังรูปที่ 7.57



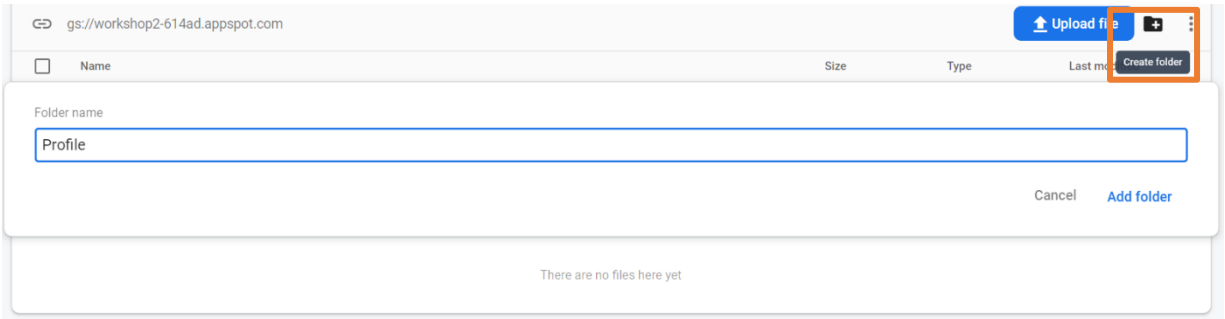
ภาพประกอบ 8.57 ภาพแสดง Storage สำหรับการใช้เก็บข้อมูลรูปภาพ Firebase

7.4 ไปที่เว็ปไซต์ https://www.iconfinder.com แล้วใส่คำค้นหาว่า account จากนั้นให้เลือกรูปที่ชอบแล้ว ดาวโหลดไว้ ดังภาพประกอบ 8.58



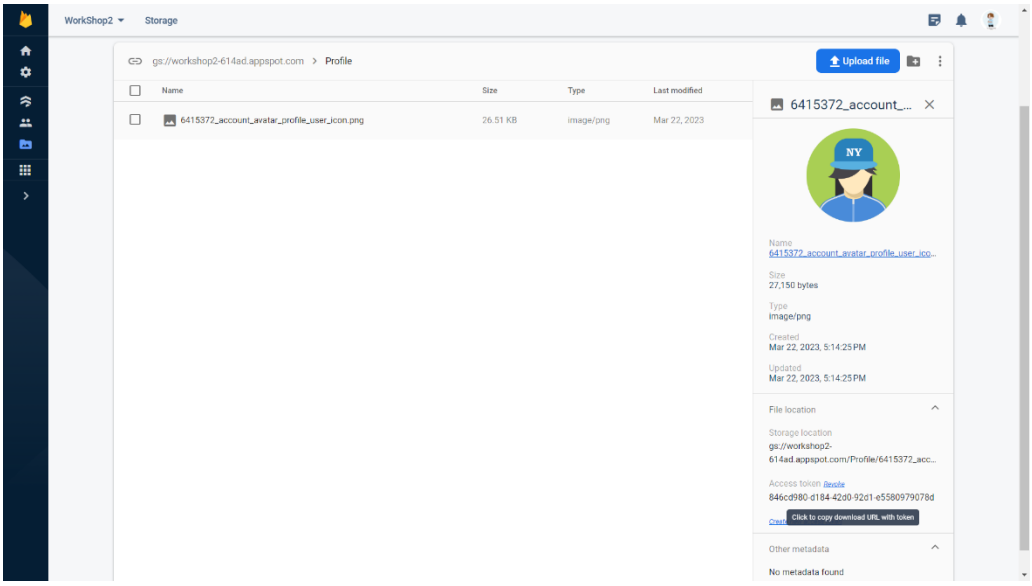
ภาพประกอบ 8.58 ภาพแสดงการดาวโหลดรูปภาพผ่านเว็ปไซต์ Iconfinder

7.5 สร้างโฟลเดอร์ Profile ดังภาพประกอบ 8.59



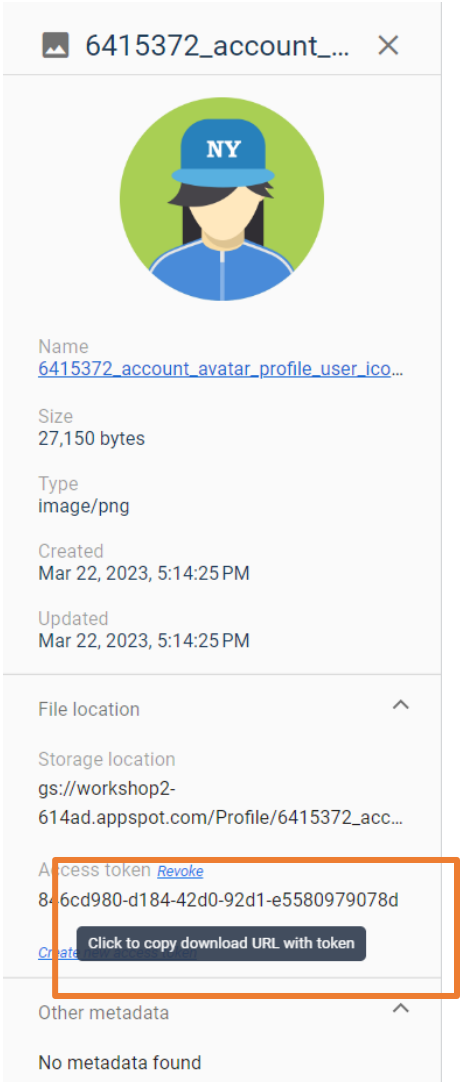
ภาพประกอบ 8.59 ภาพแสดงการสร้างโฟลเดอร์บน Storage

7.6 เมื่อสร้างโฟลเดอร์เสร็จเรียบร้อย ให้กดเข้าไปที่โฟลเดอร์ Prifile เพิ่มไฟล์รูปภาพที่ดาวโหลดไว้ ดังภาพประกอบ 8.60



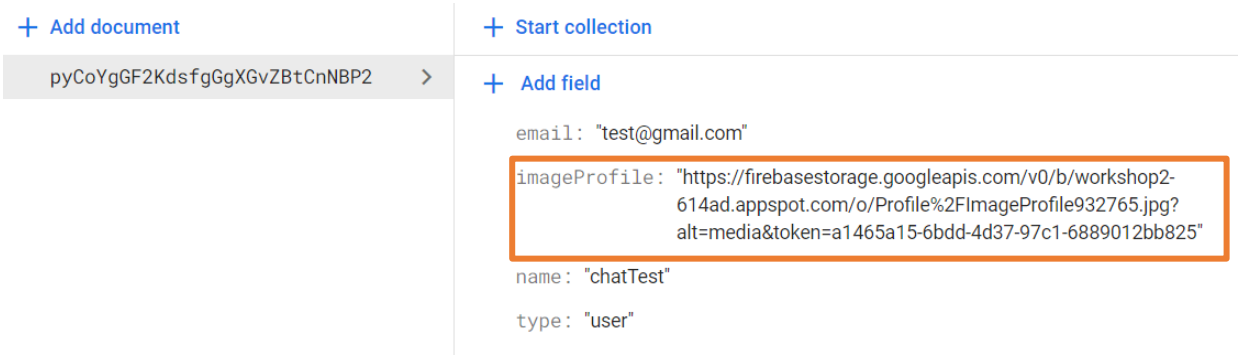
ภาพประกอบ 8.60 ภาพแสดงไฟล์รูปภาพที่ทำการอัพโหลดไปยัง Storage

7.7 ไปที่ File location หา Access token แล้ว Copy ไว้ดังภาพประกอบ 8.61



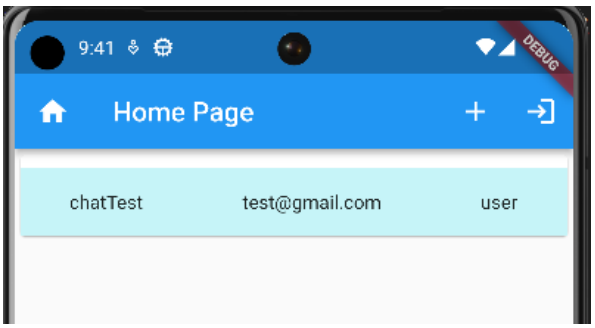
ภาพประกอบ 8.61 ภาพแสดงการ Copy Access token

8 นำ Storage location ไปทำการสร้าง Field ใน Ducment ของ User โดยมีชื่อ Field ว่า imageProfile จะได้ดังภาพประกอบ 8.62



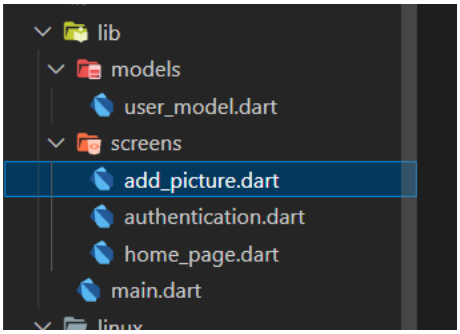
ภาพประกอบ 8.62 ภาพแสดงการเพิ่ม Field imageProfile

7.9 เพิ่ม Icon Button เพื่อกดไปยังหน้า AddPicturePage ใน AppBar ไว้ข้างหน้าปุ่มออกจากระบบ ดังภาพประกอบ 8.63



ภาพประกอบ 8.63 ภาพแสดงการเพิ่ม Icon Button

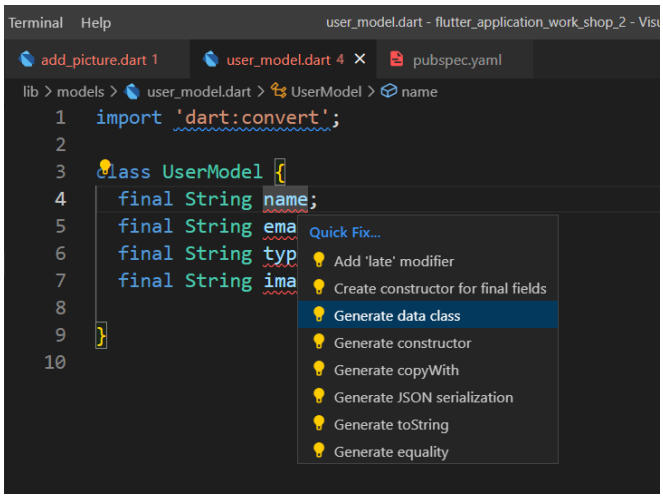
7.10 เพิ่มไฟล์ใหม่ ทำการเขียน Code ในไฟล์ให้เรียบร้อยโดยมีชื่อ Class ว่า AddPicturePage ดังภาพประกอบ 8.64



ภาพประกอบ 8.64 ภาพแสดงการเพิ่มไฟล์ add_picture.dart

7.11 เขียนโค้ดเมื่อทำการกดไอคอนที่เพิ่มขึ้นมาใหม่สามารถไปยังหน้า AddPicturePage

7.12 แก้ไข UserModel เพิ่ม final String imageProfile; ให้ Generate data Class ใหม่ดังภาพประกอบ 7.65



ภาพประกอบ 8.65 ภาพแสดงการ Generate data Class ใหม่

7.13 เขียนโค้ดแสดงผลข้อมูลที่เก็บไว้บน Firestore database และ Storage เริ่มด้วยการประกาศตัวแปร เขียน Future ฟังก์ชัน นำ Future ที่สร้างขึ้นไปไว้ใน initState(){} ให้เรียบร้อย ดังภาพประกอบ 8.66

```dart
String? urlImageProfile, uidUser, newUrlImageProfile;
  UserModel? userModel;

  @override
  void initState() {
    // TODO: implement initState
    super.initState();
    readDataUserLogin();
  }

  Future<Null> readDataUserLogin() async {
    FirebaseAuth.instance.authStateChanges().listen(
      (event) async {
        uidUser = event!.uid;
        FirebaseFirestore.instance
            .collection('userTable')
            .doc(uidUser)
            .snapshots()
            .listen(
          (event) async {
            setState(
              () {
                userModel = UserModel.fromMap(
                  event.data()!,
                );
              },
            );
            urlImageProfile = userModel!.imageProfile;
          },
        );
      },
    );
  }
```

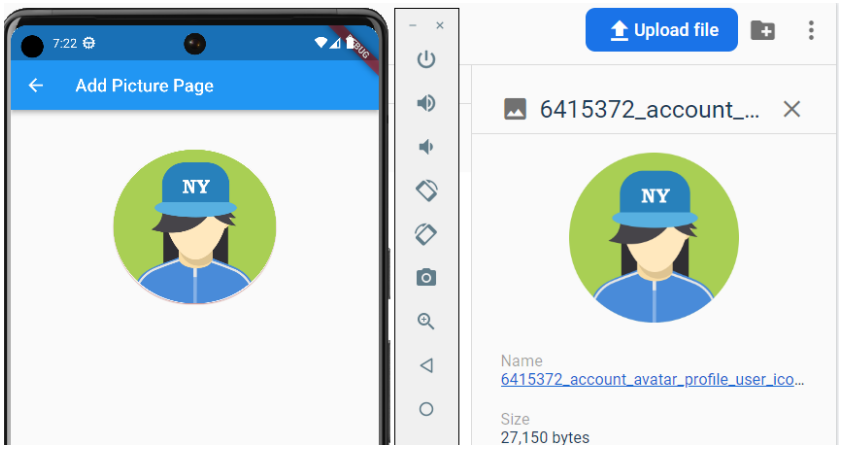
ภาพประกอบ 8.66 ภาพแสดงตัวอย่าง Code หาข้อมูลผู้ใช้งานมาเก็บไว้ใน userModel

7.14 สร้าง Widget showImage แล้วนำไปใส่ในส่วนของ body ดังภาพประกอบ 8.67

```
Widget showImage() {
    return Container(
      margin: EdgeInsets.only(top: 40),
      width: 190,
      height: 180,
      child: Card(
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.all(
            Radius.circular(100),
          ),
        ),
        shadowColor: Colors.red,
        child: Center(
          child: ClipOval(
            child: file == null
                ? Image.network(
                    urlImageProfile!,
                    width: 200,
                    height: 200,
                    fit: BoxFit.cover,
                  )
                : Image.file(
                    file!,
                    width: 200,
                    height: 200,
                    fit: BoxFit.cover,
                  ),
          ),
        ),
      ),
    );
  }

```

ภาพประกอบ 8.67 ภาพแสดงตัวอย่าง Code Widget shoeImage

เมื่อนำ Widget shoeImage ไปใส่ในส่วนของ body แล้ว ทำการสั่ง Run โปรเจค แล้วดูผลลัพธ์ดังภาพประกอบ 8.68



ภาพประกอบ 8.68 ภาพแสดงผลรูปภาพบน emulator

7.15 สร้าง Future ในการเรียกใช้งาน image_picker ดังภาพประกอบ 8.69

```dart
Future<Null> choooseImage(ImageSource imageSource) async {
    try {
      var object = await ImagePicker().pickImage(
        source: imageSource,
        maxHeight: 500,
        maxWidth: 500,
      );
      setState(() {
        file = File(object!.path);
      });
    } catch (e) {}
  }
```

ภาพประกอบ 8.69 ภาพแสดงตัวอย่าง code Future ในการเรียกใช้งาน image_picker

7.16 สร้างปุ่มในการกดเลือก gallery หรือ camera เมื่อทำการกดปุ่มให้เรียงใช้งานFuture chooseImage ดังภาพประกอบ 8.70 - 7.71

```dart
Widget choooseImageCamera() {
    return Container(
      padding: EdgeInsets.all(10),
      child: GestureDetector(
        onTap: () => choooseImage(ImageSource.camera),
        child: Container(
          width: 200,
          child: Card(
            color: Colors.blue,
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.all(
                Radius.circular(20),
              ),
            ),
            child: Row(
              children: [
                Container(
                    padding: EdgeInsets.all(10),
                    child: Icon(
                      Icons.camera_alt_rounded,
                      size: 30,
                      color: Colors.white,
                    )),
                Text(
                  'เลือกจากกล้อง',
                  style: TextStyle(
                    fontSize: 18,
                    color: Colors.white,
                  ),
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }
```

ภาพประกอบ 8.70 ภาพแสดงตัวอย่าง code เมื่อทำการกดจะเลือกจาก camera

```dart
Widget choooseImageGallery() {
  return Container(
    padding: EdgeInsets.all(10),
    child: GestureDetector(
      onTap: () => choooseImage(ImageSource.gallery),
      child: Container(
        width: 200,
        child: Card(
          color: Colors.blue,
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.all(
              Radius.circular(20),
            ),
          ),
          child: Row(
            children: [
              Container(
                  padding: EdgeInsets.all(10),
                  child: Icon(
                    Icons.image,
                    size: 30,
                    color: Colors.white,
                  )),
              Text(
                'เลือกจากแกลลอรี่',
                style: TextStyle(
                  fontSize: 18,
                  color: Colors.white,
                ),
              ),
            ],
          ),
        ),
      ),
    ),
  );
}
```

ภาพประกอบ 8.71 ภาพแสดงตัวอย่าง code เมื่อทำการกดจะเลือกจาก gallery

7.17 ทำการเขียน Future ในการบันทึกข้อมูลไปยัง Storage และ ClouFirestore ดังภาพประกอบ 8.72

```dart
Future<Null> upLoadPictureToStoage() async {
   Random random = Random();
   int i = random.nextInt(1000000);

   Firebase.initializeApp().then((value) async {
     FirebaseStorage storage = FirebaseStorage.instance;
     Reference reference = storage.ref().child('Profile/ImageProfile$i.jpg');
     UploadTask uploadTask = reference.putFile(file!);

     newUrlImageProfile = await (await uploadTask).ref.getDownloadURL();
     print('UrlImage ===>>> $newUrlImageProfile');

     upDatePictureToCloudFriestore();
   });
 }

 Future<Null> upDatePictureToCloudFriestore() async {
   Firebase.initializeApp().then((value) async {
     FirebaseFirestore.instance
         .collection('userTable')
         .doc(uidUser)
         .update({"imageProfile": newUrlImageProfile}).then(
       (value) async {},
     );
   });
 }
```

ภาพประกอบ 8.72 ภาพแสดงตัวอย่าง code บันทึกข้อมูลไปยัง Storage และ ClouFirestore

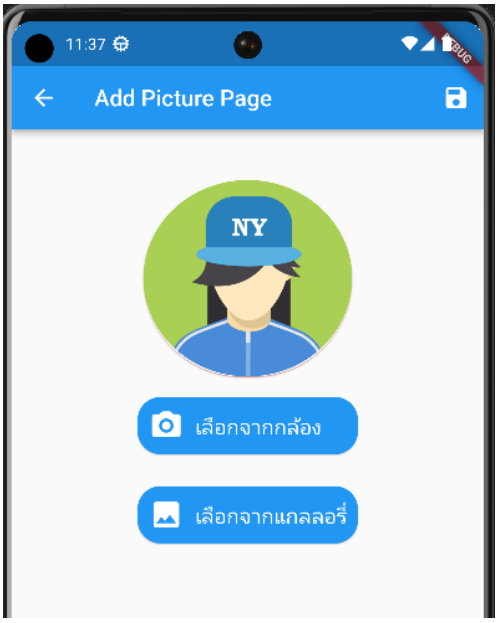
จากภาพประกอบ 8.72 การทำงานของ Future จะเริ่มต้นทำงานที่ Future upLoadPictureTostoage ให้เสร็จเรียบร้อยก่อน แล้วถึงจะไปเรียกใช้งาน Future upDatePictureToColundFirestore

7.18 สร้างปุ่มในการบันทึกข้อมูล เมื่อกดให้เรียกใช้งาน FutureupLoadPictureTostoage ดังภาพประกอบ 8.56

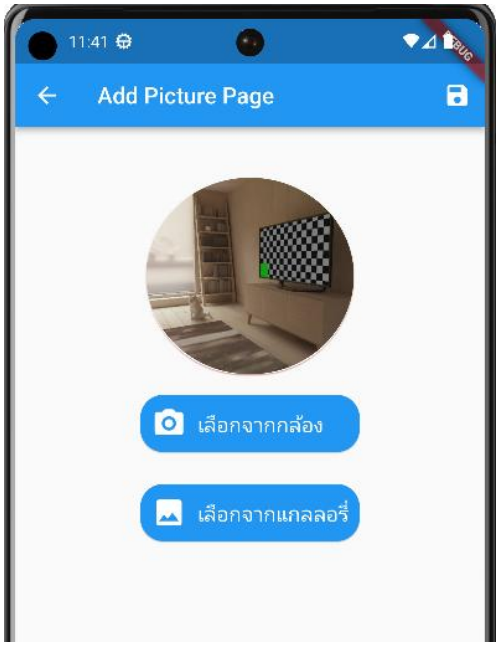
```
appBar: AppBar(
        actions: [
          IconButton(
              onPressed: () {
                upLoadPictureToStoage();
              },
              icon: Icon(Icons.save))
        ],
        title: Text('Add Picture Page'),
      ),
```

ภาพประกอบ 8.73 ภาพแสดงตัวอย่าง code ปุ่มในการกดบันทึกข้อมูล

7.19 สั่ง run โปรเจค แล้วไปยังหน้าที่เราจะทำการทดสอบอับโหลดรูปภาพไปยัง Storage และ ClouFirestore ดังภาพประกอบ 8.74 – 8.75

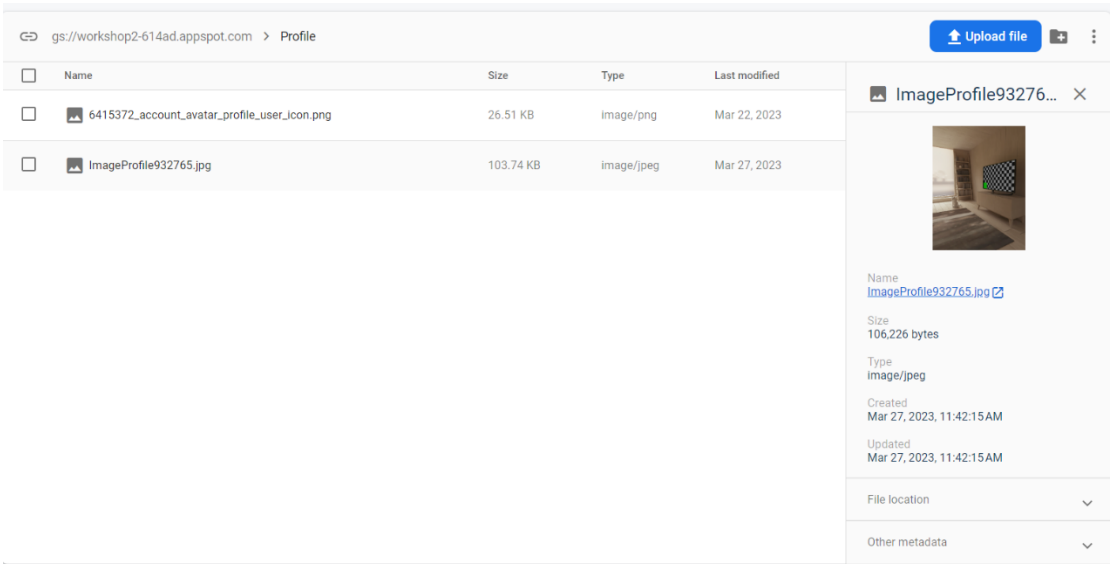


ภาพประกอบ 8.74 ภาพแสดงก่อนการเพิ่มรูปภาพ



ภาพประกอบ 8.75 ภาพแสดงหลังการเพิ่มรูปภาพ

หลังจากที่เราเพิ่มรูปภาพเสร็จแล้ว ทำการบันทึกรูปภาพไปยัง Storage และ CloudFirestore ไปเช็ครูปภาพที่ Storage และนำ Access token ของรูปภาพที่อัพขึ้นไปเช็คกับ CloudFirestore ที่ Field : imageProfile ว่าตรงกันหรือมั้ย



ภาพประกอบ 8.76 ภาพแสดงรูปภาพที่อัพขึ้นไปใหม่

8. Chat Application whit Firebase

1. เพิ่มไฟล์ chat.dart ในโปรเจคเดิม แล้วตั้งชื่อ StatefulWidget class ให้เรียบร้อย

2. ไปที่ไฟล์ authentication.dart แก้ไข้เมื่อล็อกอินสำเร็จให้ไปที่ ไฟล์ chat.dart ดังภาพประกอบ 8.77

```dart
Future<Null> checkAuthen() async {
    await Firebase.initializeApp().then((value) async {
      await FirebaseAuth.instance
          .signInWithEmailAndPassword(email: email!, password: password!)
          .then((value) async {
        String? uid = value.user!.uid;
        await FirebaseFirestore.instance
            .collection('userTable')
            .doc(uid)
            .snapshots()
            .listen((event) {
          UserModel model = UserModel.fromMap(event.data()!);
          switch (model.type) {
            case 'user':
              Navigator.pushAndRemoveUntil(
                  context,
                  MaterialPageRoute(
                    builder: (context) => ChatPage(),
                  ),
                  (route) => false);
              break;
            default:
          }
        });
      });
    });
  }
```

ภาพประกอบ 8.77 ภาพแสดง Code แก้ไขการ Route ไปยัง ChatPage

3. ประการตัวแปรที่ใช้ในหน้า ChatPage ดังภาพประกอบ 8.78

```dart
TextEditingController msg = new TextEditingController();
String? title, body, nameUser;
FirebaseFirestore _firestore = FirebaseFirestore.instance;
UserModel? userModel;
```

ภาพประกอบ 8.78 ภาพแสดงการประกาศตัวแปรที่ใช้ในหน้า ChatPage

4. อ่านข้อมูลผู้ใช้งานจาก Firestore database ดังภาพประกอบ 8.79

```dart
@override
void initState() {
  // TODO: implement initState
  super.initState();
  readDataUserLogin();
}

Future<Null> readDataUserLogin() async {
  FirebaseAuth.instance.authStateChanges().listen(
    (event) async {
      String uidUser = event!.uid;
      FirebaseFirestore.instance
          .collection('userTable')
          .doc(uidUser)
          .snapshots()
          .listen(
        (event) async {
          setState(
            () {
              userModel = UserModel.fromMap(
                event.data()!,
              );
            },
          );
          nameUser = userModel!.name;
        },
      );
    },
  );
}
```

ภาพประกอบ 8.79 ภาพแสดง Code การอ่านข้อมูลจาก CloudFireStore database

5. ออกแบบในส่วนของ AppBar โดยจะแสดงชื่อแล้วปุ่มออกจากระบบ ดังภาพประกอบ 8.80

```
return Scaffold(
  appBar: AppBar(
    actions: [
      IconButton(
          onPressed: () {
            signOut();
          },
          icon: Icon(Icons.login)),
    ],
    title: Text('คุณ $nameUser'),
  ),
```

ภาพประกอบ 8.80 ภาพแสดง Code การออกแบบปุ่มออกจากระบบ

จากภาพประกอบ 8.80 จะเห็นได้ว่าเมื่อมีการกดปุ่มทุกครั้งจะเรียกใช้งาน Future singOut เป็น Future ที่เขียนขึ้นมาใช้สำหรับการออกจากระบบการใช้งานของผู้ใช้งาน ดังภาพประกอบ 8.81

```
Future<Null> signOut() async {
  await Firebase.initializeApp().then((value) async {
    await FirebaseAuth.instance.signOut().then((value) {
      Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => Authentication(),
          ));
    });
  });
}
```

ภาพประกอบ 8.81 ภาพแสดง Code การออกจาระบบของ Fiebase

6. ออกแบบในส่วนของ Body จะ query snapshot จาก CloudFirestore database มาแสดงในรูปแบบ ListView และ Return ค่ากลับในรูปแบบของ widget ที่มีชื่อว่า message ดังภาพประกอบ 8.82

```
body: Container(
  height: size.height / 1.25,
  width: size.width,
  child: StreamBuilder<QuerySnapshot>(
    stream: _firestore
        .collection('chatTable')
        .orderBy('time', descending: false)
        .snapshots(),
    builder: (context, snapshot) {
      if (snapshot.data != null) {
        return ListView.builder(
          itemCount: snapshot.data!.docs.length,
          itemBuilder: (context, index) {
            var map = snapshot.data!.docs[index].data();
            return message(size, map);
          },
        );
      } else {
        return Container();
      }
    },
  ),
),
```

ภาพประกอบ 8.82 ภาพแสดงตัวอย่าง code ในส่วนของ body

```dart
Widget message(Size size, var map) {
  return Container(
    width: size.width,
    alignment: map['sendby'] == nameUser
        ? Alignment.centerRight
        : Alignment.centerLeft,
    child: Container(
      padding: EdgeInsets.symmetric(vertical: 10, horizontal: 15),
      margin: EdgeInsets.symmetric(vertical: 10, horizontal: 15),
      decoration: BoxDecoration(
        color: Colors.blue,
        borderRadius: BorderRadius.circular(20),
      ),
      child: Text(
        map['message'],
        style: TextStyle(color: Colors.white, fontSize: 16),
      ),
    ),
  );
}
```

ภาพประกอบ 8.83 ภาพแสดงตัวอย่าง code Widget message

จากภาพประกอบ 8.83 เป็น widget ที่เขียนเพื่อแสดงข้อความในหน้าแอปพลิเคชันโดยจะแยกข้อความจากชื่อผู้ใช้งานที่ทำการส่งข้อความ โดยใน Widget นี้จะมีการส่งค่ากลับไปยัง ListView ที่อยู่ในส่วนของ body

7. ส่วนด้านล่างสุดเราจะออกแบบสำหรับการกรอกข้อความ และปุ่มในการกดส่งข้อความ เมื่อทำการกดส่งข้อความจะเรียกใช้ void function ที่มีชื่อว่า onSentMessage() ดังภาพประกอบ 8.84

```
floatingActionButton: Container(
      margin: EdgeInsets.only(left: 40, top: 100),
      height: size.height / 8,
      width: size.height,
      alignment: Alignment.center,
      child: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Container(
          height: size.height / 10,
          width: size.height,
          child: TextField(
            controller: msg,
            decoration: InputDecoration(
              suffixIcon: IconButton(
                  onPressed: () {
                    onSentMessage();
                  },
                  icon: Icon(
                    Icons.send,
                    color: Colors.blue,
                  )),
              hintText: 'กรุณากรอกข้อความที่ต้องการส่ง...',
              border: InputBorder.none,
              enabledBorder: OutlineInputBorder(
                borderRadius: BorderRadius.circular(10),
                borderSide: BorderSide(color: Colors.blue),
              ),
              focusedBorder: OutlineInputBorder(
                borderRadius: BorderRadius.circular(10),
                borderSide: BorderSide(color: Colors.blue),
              ),
            ),
          ),
        ),
      ),
    ),
```

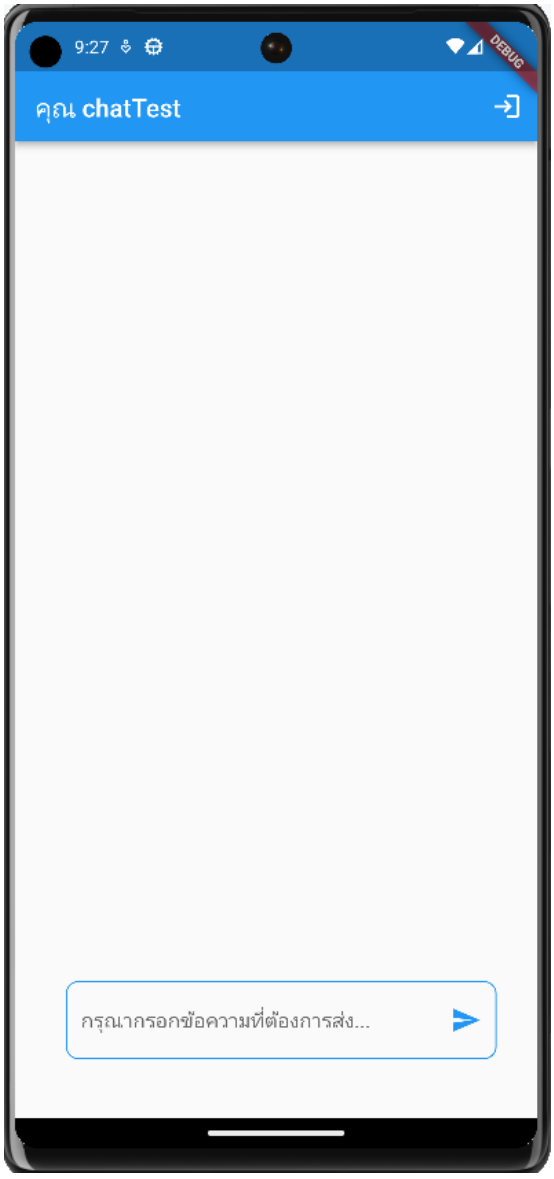
ภาพประกอบ 8.84 ภาพแสดงตัวอย่าง code ในส่วนของการกรอก และส่งข้อความ

```dart
void onSentMessage() async {
  if (msg.text.isNotEmpty) {
    Map<String, dynamic> message = {
      'sendby': nameUser,
      'message': msg.text,
      'time': FieldValue.serverTimestamp(),
      // 'uidRest': queueModel.uidRest,
    };

    await _firestore.collection('chatTable').add(message);
    msg.clear();
  } else {
    print("Enter Some Text");
  }
}
```

ภาพประกอบ 8.85 ภาพแสดงตัวอย่าง code void function onSentMessage()

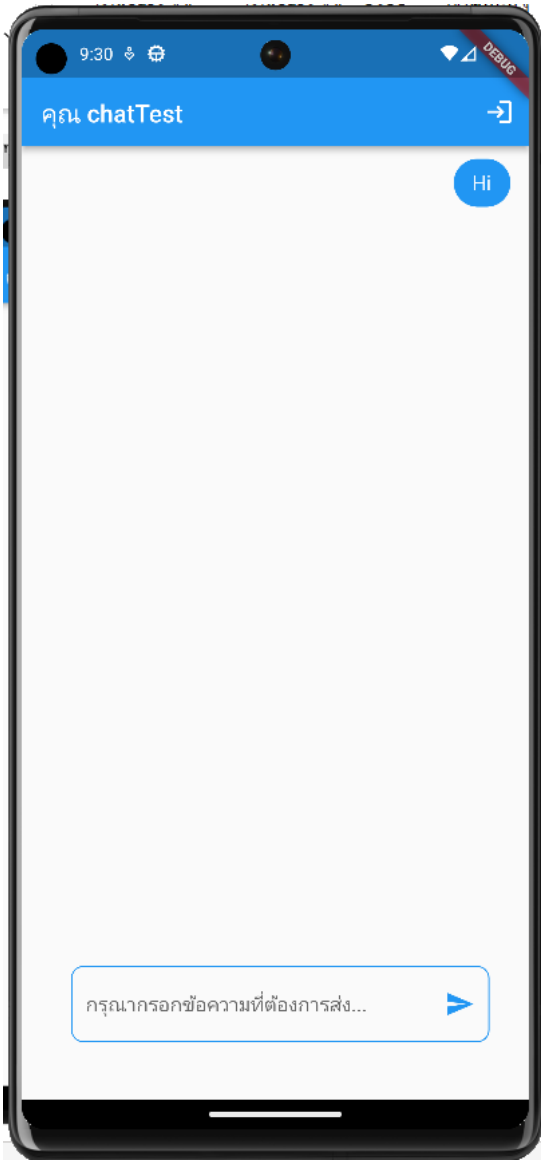
จากภาพประกอบ 8.85 เป็น void function ที่เขียนขึ้นมาเพื่อใช้รับค่าจาก TextField เมื่อมีการกดส่งข้อความ และบันทึกไปยัง CloudFirestore database

8. สั่ง run โปรเจค ทำการล็อกอินเพื่อไปยังหน้า ChatPage ดังภาพประกอบ 8.86



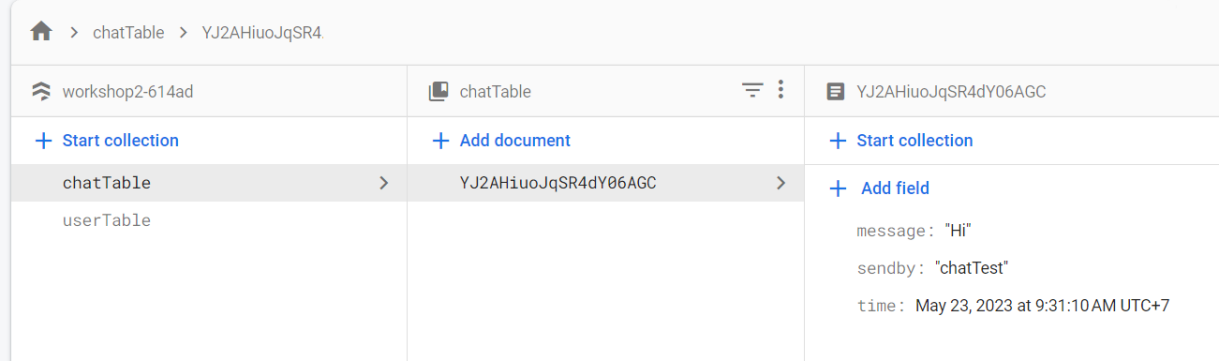
ภาพประกอบ 8.86 ภาพแสดงหน้า ChatPage

9.ทดสอบพิมพ์ข้อความแล้วทำการส่งข้อความจะได้ ดังภาพประกอบ 8.87



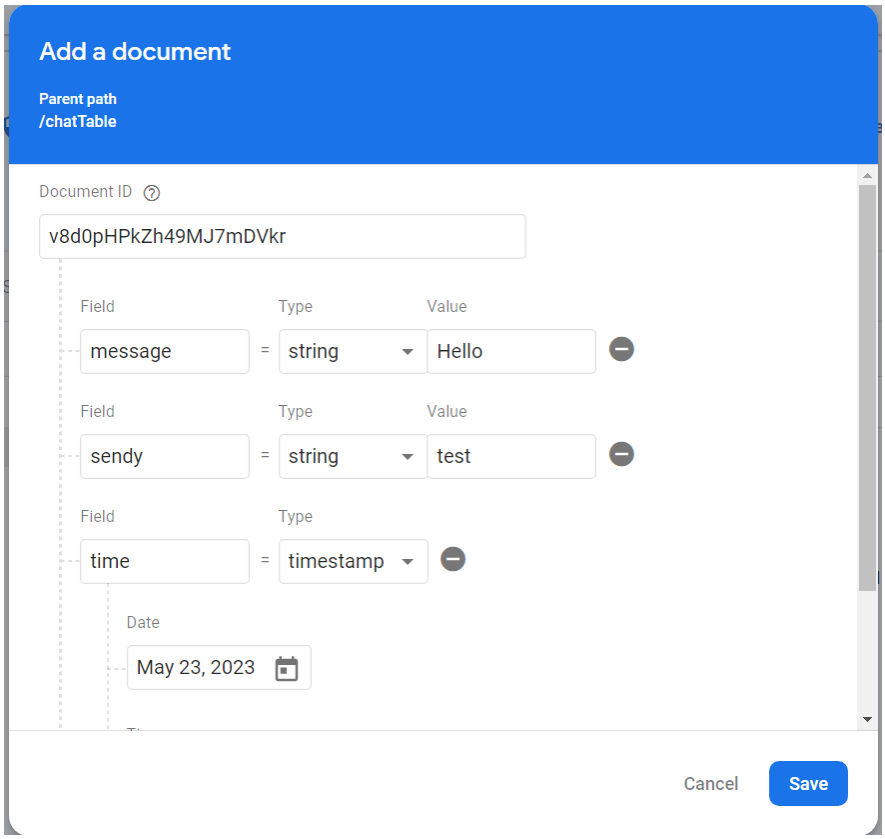
ภาพประกอบ 8.87 ภาพแสดงหน้า ChatPage ที่มีข้อความ

จากภาพประกอบ 8.87 ที่เราได้ส่งข้อความและแสดงผล เมื่อเราไปที่ CloudFirestore Database จะพบข้อมูลที่เราได้ทำการส่งโดยจะมี ข้อความที่ส่ง เวลาที่ส่ง ชื่อผู้ส่ง ดังภาพประกอบ 8.88



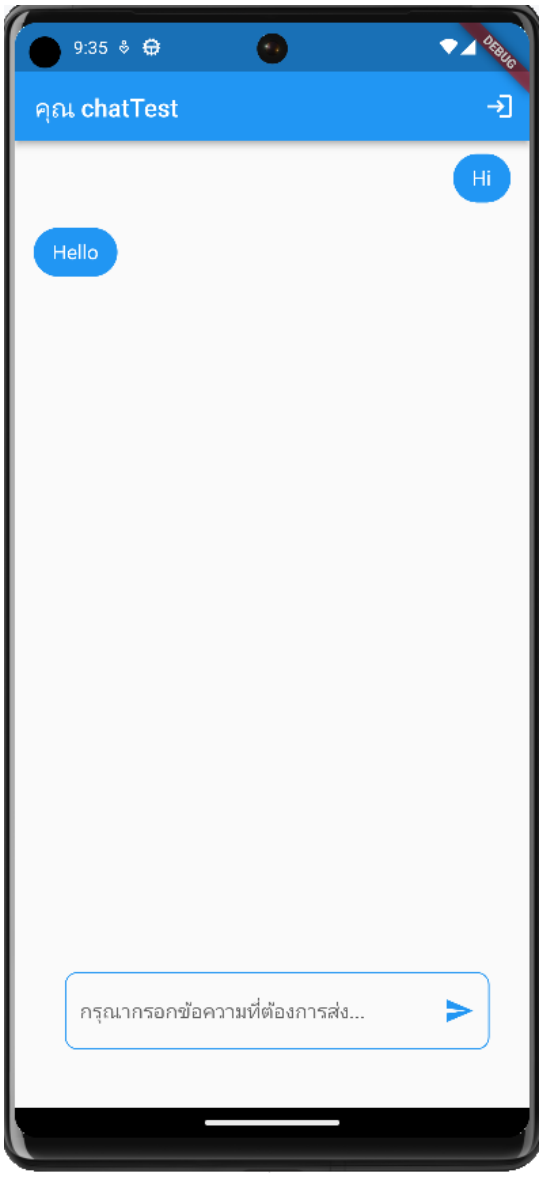
ภาพประกอบ 8.88 ภาพแสดงฐานข้อมูล CloudFirestore Database collection chatTable

10. เพิ่ม document ข้างใน collection chatTable ดังภาพประกอบ 8.89



ภาพประกอบ 8.89 ภาพแสดงการเพิ่ม document ข้างใน collection chatTable

จากภาพประกอบ 8.89 เมื่อเราเพิ่มข้อความ ชื่อผู้ส่ง วันเวลา ให้ทำการกดบันทึกและไปดูที่หน้าแอปพลิเคชันของเรา ดังภาพประกอบ 8.90



ภาพประกอบ 8.90 ภาพแสดงหน้าแอปพลิเคชันหลังเพิ่ม document ข้างใน collection chatTable

จากภาพประกอบ 8.90 เมื่อเราลองเพิ่ม document ข้างใน collection chatTable ประกอบด้วยข้อความ วันเวลา ชื่อผู้ส่ง หากต้องการทดสอบพิมพ์ข้อความโต้ตอบให้ทำการสร้างบัญชีผู้ใช้เพิ่มขึ้นมา