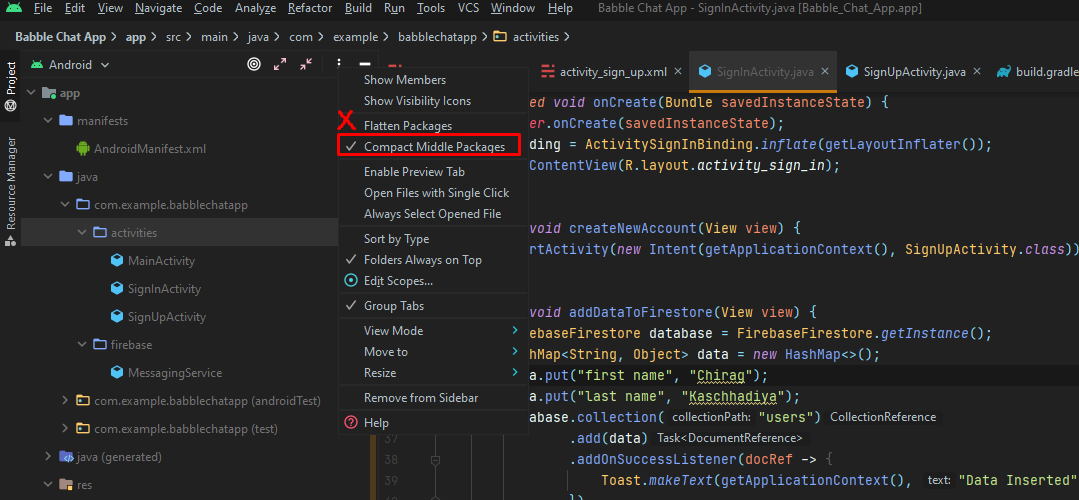
1. Tạo các giao diện cơ bản cho app, tạo các resources cần thiết trong xml
2. Setup FireBase service vào app
3. Implement Cloud Messaging vào app
4. Implement Cloud Store (lưu trữ dạng NoSQL) vào app
5. Implement các function vào button + text
6. Setup MultiDex (tạo nhiều DEX files)
7. Functions
8. Thử nghiệm tính năng (Test)

1.

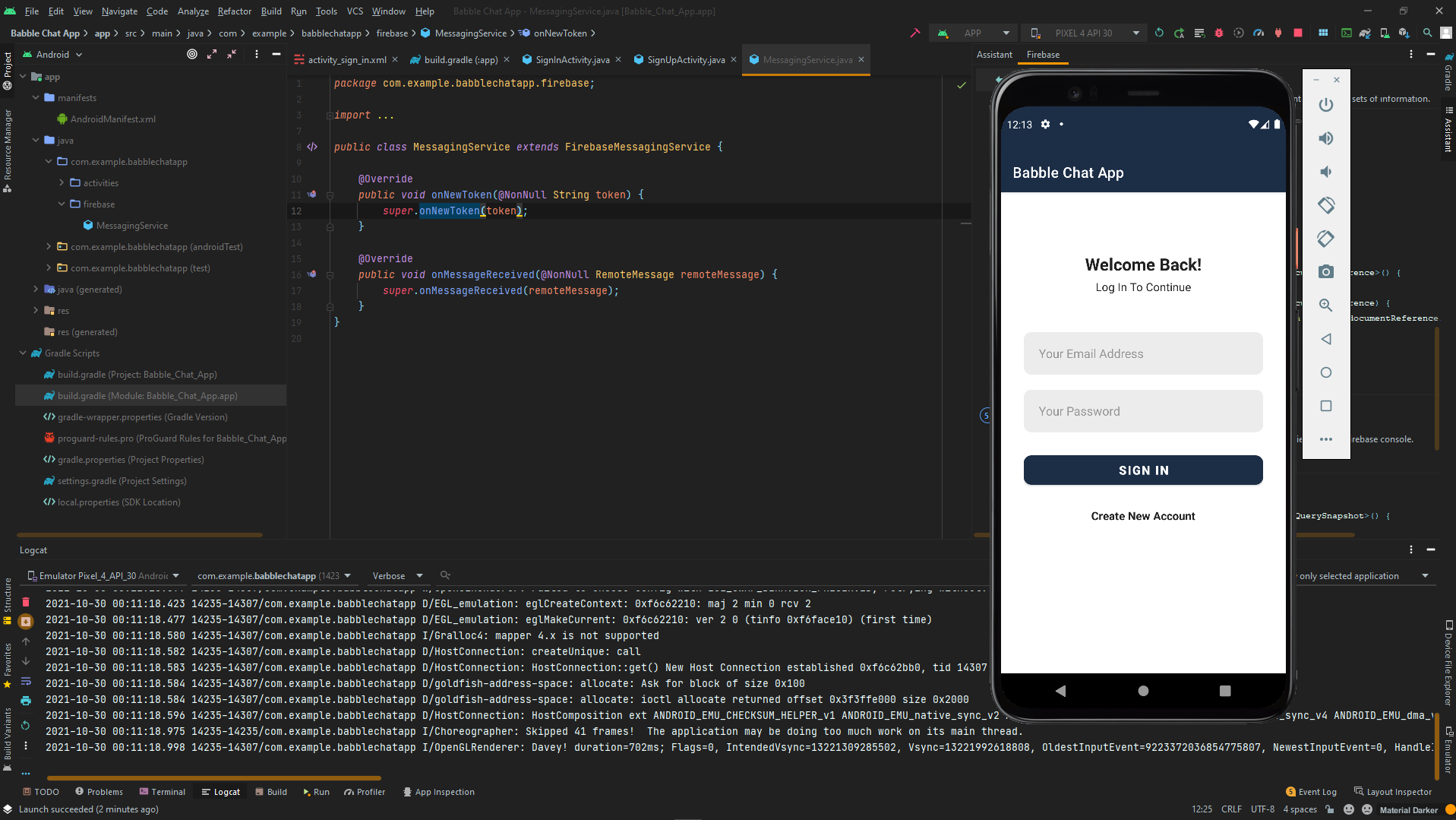


*Note: Có 1 trick em thấy khá là hay là nếu chúng ta bỏ tick Flatten Package thì nó lại trở về dạng Compact Middle Package (kiểu dạng như Nested Folder Tree) ạ, cái này khá tiện khi manage nhiều package lồng nhau ạ. Nhiều lúc em restart Android Studio nó cứ tự động Flatten Package khá là khó chịu nên em tìm hiểu thì ra trick này :’D*

2. Setup FireBase service vào app

Các bước setup Firebase khá là đơn giản + đã có doc hướng dẫn trên mạng nên em sẽ chỉ tóm tắt ạ:

* Chúng ta Login vào FireBase Console ([console.firebase.google.com](https://console.firebase.google.com/))
* Tạo tài khoản (nếu chưa có)
* Create New Project -> điền đầy đủ thông tin về app của chúng ta để link project với FireBase *(****chú ý****: có thể enable hoặc disable GG Analytics tùy theo nhu cầu, em disable ạ)*
* Nếu create thành công thì trong Android Studio > Tools giờ sẽ có option FireBase. Click vào đó để mở Assistant lên
* Chọn Cloud Messaging -> sau đó làm theo hướng dẫn được chỉ định ở trong doc *(sơ lược là add thêm id của FireBase vào phần Plugins trong AndroidManifest.xml* ***và*** *add thêm implementation của các dịch vụ tương ứng vào Build Gradle với Gradle Settings ạ)*

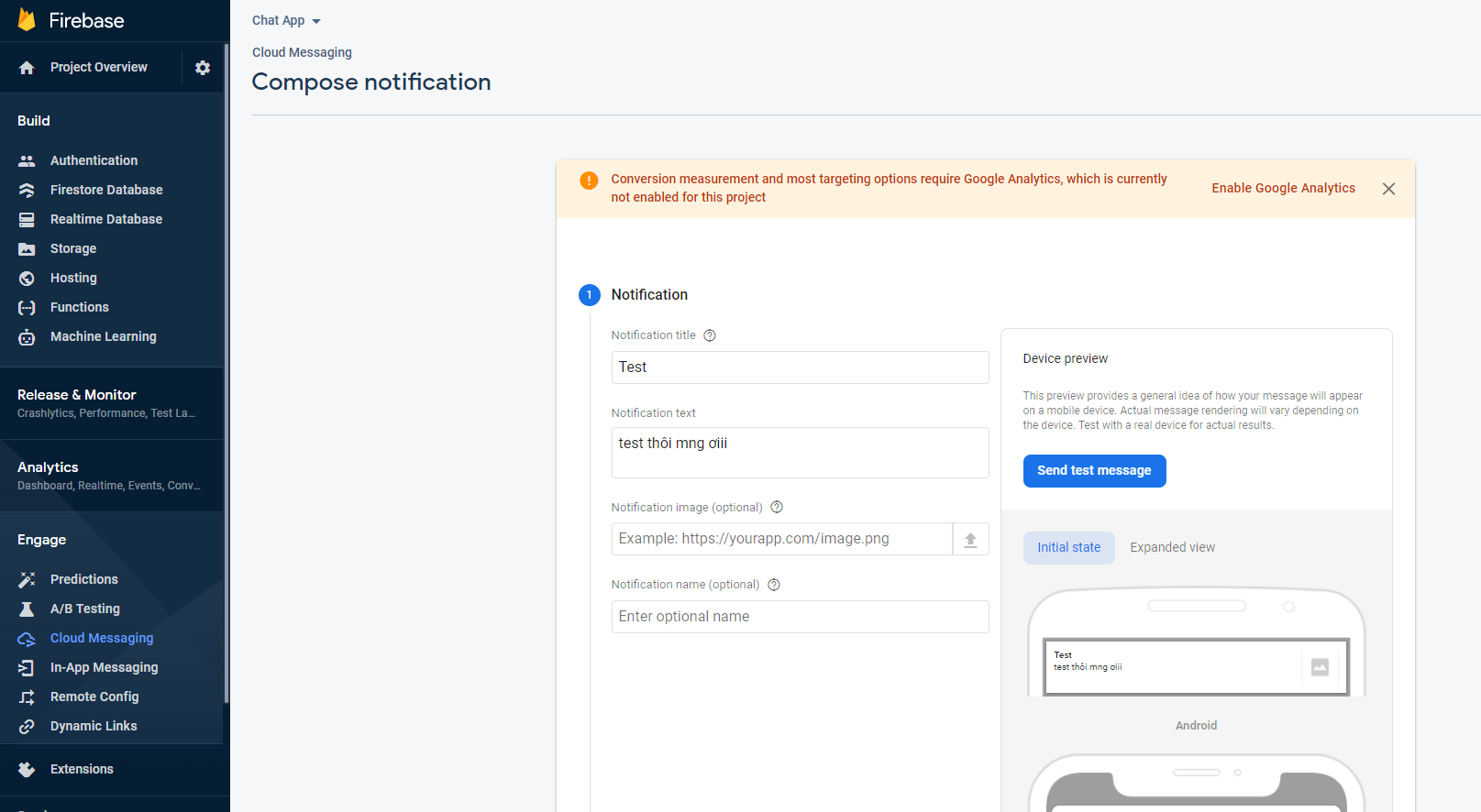


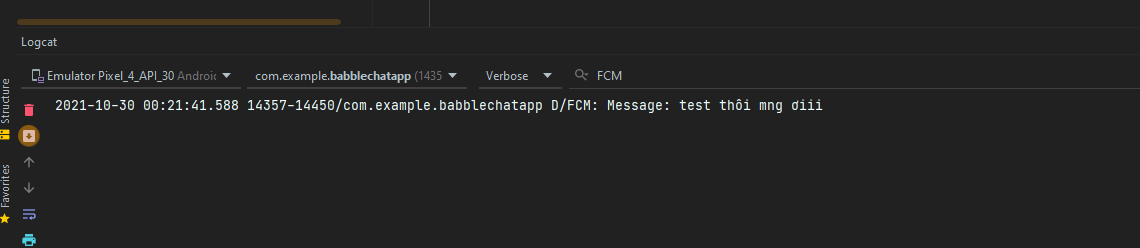
4. Implement FCM (FireBase Cloud Messaging)

Tại đây em implement Firebase Cloud Messaging (FCM) theo như hướng dẫn của Firebase Assistant như sau:

* Tạo child class cho FirebaseMessagingService
* Implement function để tạo 1 Token mỗi khi App chạy
* Implement hàm nhận message.

Cuối cùng, em có chạy thử FCM để xem setup đã thành công chưa bằng các sử dụng: *Test New Notification tại Firebase console (giao diện nó như Dashboard ạ)* **và** sau đó *Log.d trong app* để log ra ngoài màn hình kết quả đã chạy được:



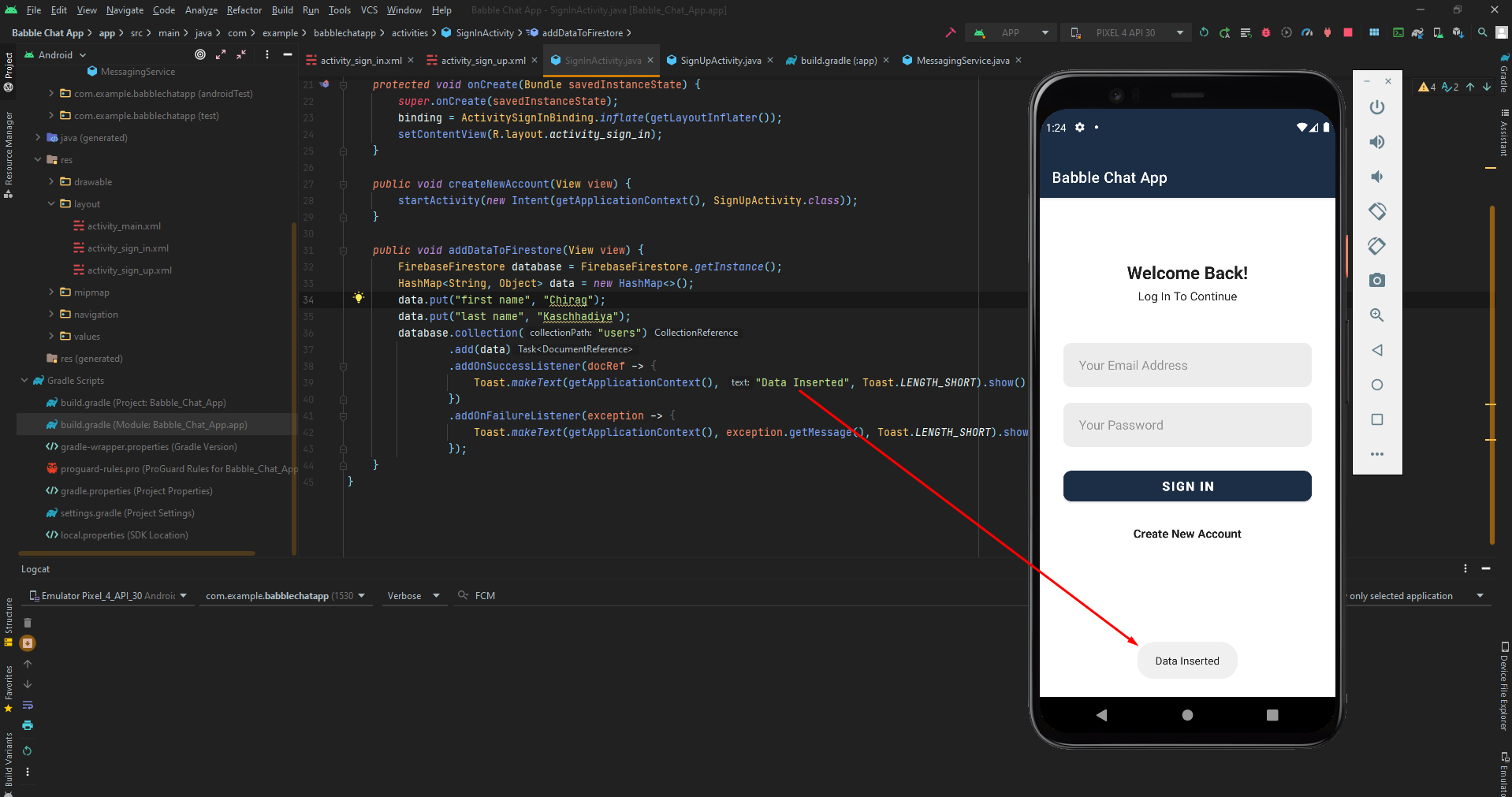


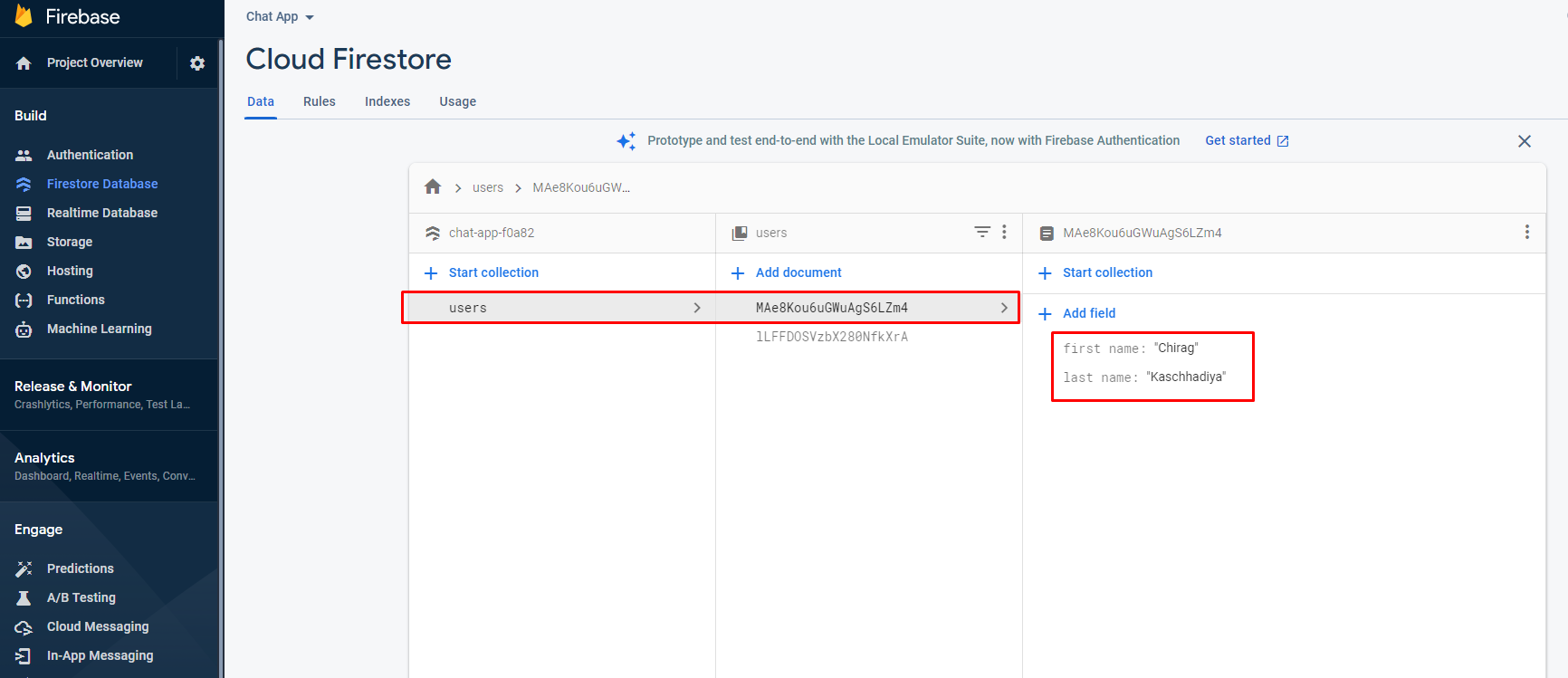
Và nhận được noti em tự gửi (***test thôi mng ơiii***), tức là đã setup Cloud Messaging thành công ạ.

4. Implement FireStore

Cấu trúc dữ liệu em dùng để lưu trữ thông tin người dùng là HashMap (1 dạng Symbol table) avg time complexity của CTDL này là O(1) (Constant time) rất tiện cho việc tra cứu (trên thực tế thì time cost là O(m), với m là số bước để tìm ra địa chỉ chính xác nếu có collision xảy ra (trùng hashCode), tùy trường hợp mà O(m) có thể rất gần với hằng số hoặc xấp xỉ <N/số mã băm được sử dụng>)

Để test FireStore, em test luôn giá trị trong code và xem thông tin này có được add vào DataBase không.. Kết quả test thu về ám chỉ đã thành công ạ:





5. Implement các function vào button và text

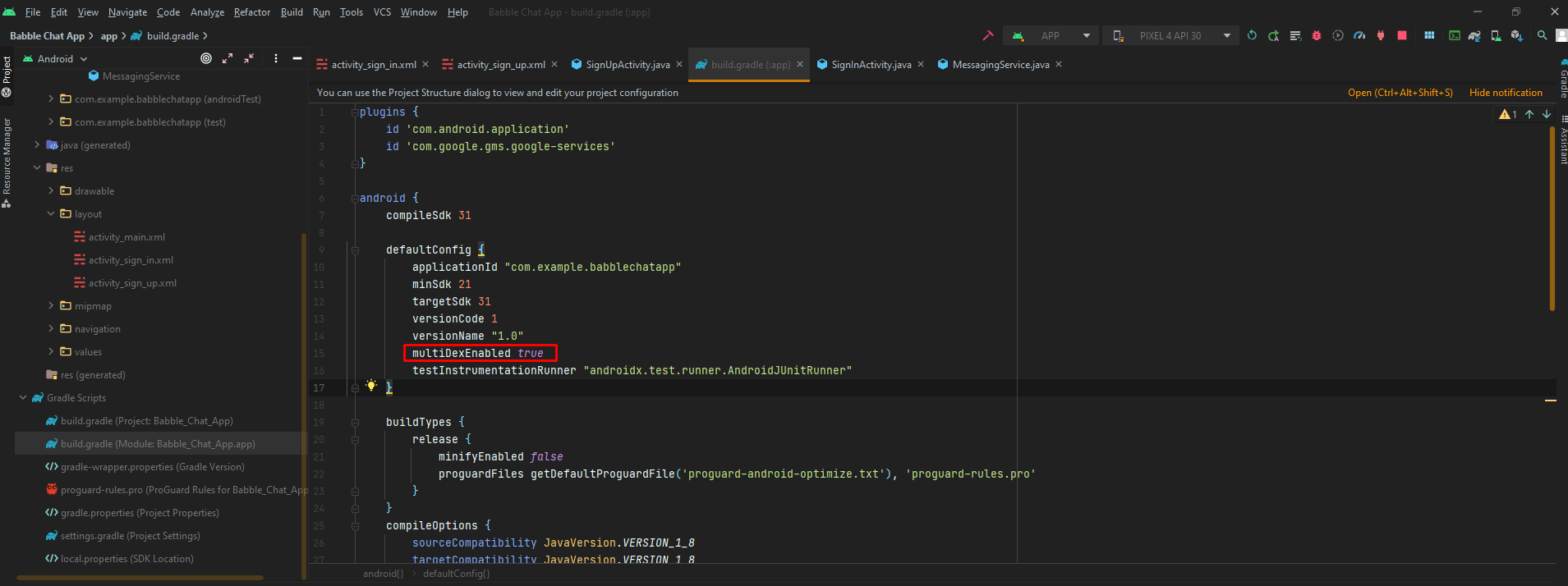
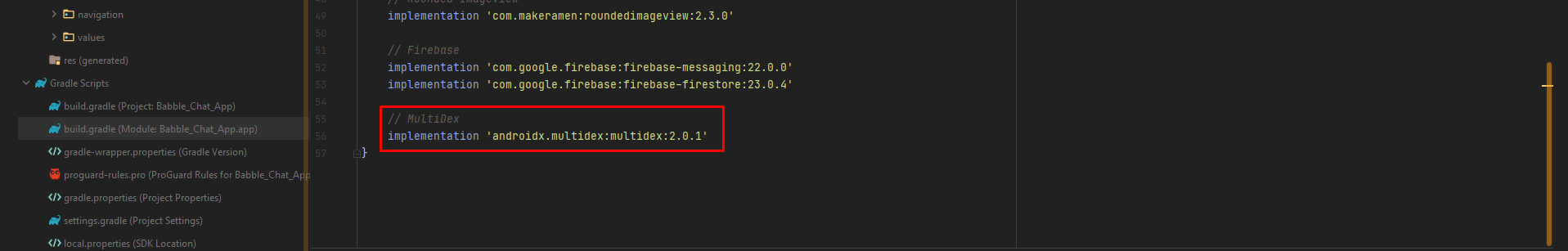
Hiện đại em thấy nhiều coder implement luôn click event vào .xml luôn, tuy nhiên trong bài này em dùng cách cổ điển là implement 1 Listener trong file .java để nghe event và react mỗi khi có event xảy ra tại đó ạ, lí do là bởi:

* **Bad practice khi phải public tất cả các hàm cần dùng:** Để implement click event vào .xml, coder bắt buộc phải để “public” cho các hàm được dùng, trong khi dùng listener thì có thể để nguyên “private” ạ, cái này có thể không ảnh hưởng nhiều tới kết quả chạy nhưng mà lại là good practice khi lập trình Java ạ (để scope nhỏ nhất có thể), tránh cho việc bị sơ ý modify bởi các class khác cùng package
* **Không thể bind nhiều hàm lên cùng 1 component nếu dùng onClick tại .xml:** Sử dụng Listener có thể bind nhiều hàm khác nhau lên cùng 1 nút trong khi android:onClick lại không thể bind nhiều hàm ạ. Bởi em đã lường trước sẽ bind nhiều hàm (thực hiện nhiều các chức năng khác nhau) lên 1 tương tác của người dùng nên em xài luôn Listener từ đầu ạ.
* **IntelliSense báo lỗi đỏ gây nhiễu mắt:** IntelliSense trong Android Studio không nhận biết được hàm mà không nằm trong MAIN LAUNCHER của app ạ -> nếu bind function backToLogin() vào button của activity\_sign\_up.xml và chỉ define hàm này trong SignUpActivity (ko phải MAIN LAUNCHER), nó sẽ hiện đỏ báo lỗi rằng không có function nào như vậy trong MAIN LAUNCHER (SignInActivity). Cái này không ảnh hưởng đến app sẽ hơi khó nhìn khi coder phải duyệt lại file ạ
* **Khó kiểm soát và tra cứu:** mỗi 1 component trong .xml còn thêm rất nhiều thông tin về thiết kế (margin, size, color, etc..) và đặt đúng thứ tự hiển thị, nên các hàm có liên quan tới nhau sẽ không thể ở gần nhau nếu chúng khác component, trong khi đó .java lại có thể xếp các hàm tương quan sao cho hợp bố cục, logic code thông thường

Về phần code e đã comment đầy đủ và em nghĩ thầy có thể thấy rõ hơn trong mã nguồn ạ. Về chủ ý cơ bản, em chỉ bind listener vào các View/Button tương ứng:

* buttonSignIn: bind Store data
* Create New Account: bind vào function để chuyển sang activity Sign Up thông qua intent
* Back To Login: bind vào function để chuyển sang activity Sign In thông qua intent

6. Setup MultiDex

APK mặc định của Android chỉ tạo 1 DEX file và từ đó chỉ lưu trữ được tối đa 2 ^ 16 byte (giới hạn của 1 DEX file), việc này làm giới hạn lượng lưu trữ thông tin của app (bất tiện cho các app nhắn tin bởi chúng ta muốn lưu nhiều tin nhắn nhất có thể (có thể gồm ảnh, text, sticker etc). Để đè (overwrite) giới hạn mặc định này, chúng ta sẽ phải implement và bật chế độ nhiều DEX file lên.

7.Functions

Về mã nguồn cụ thể và chức năng chính em đã chú thích kĩ trong file code ạ. Có gì thầy có thể dùng communication diagram hoặc sequence diagram của em để tra cứu ạ.

Ở mục này em chủ yếu sẽ giải thích tại sao một số các functions có nhiều cách khác làm nhưng em lại chọn cách cụ thể như trong file code ạ:

* Image Encode: encode image thành base-64 string, cái này rất tốt cho các ảnh nhỏ mà nhiều ạ, bởi sau khi encode thành xâu kí tự mình có thể embed luôn thẳng vào page mà không cần phải mở connection và tạo thêm request để fetch ảnh về ạ (tiết kiệm được request và load về, càng nhiều ảnh càng tiết kiệm đc nhiều request). Tuy nhiên thì bởi base-64 có dung lượng lớn hơn ảnh gốc nên nếu “chi phí tải dung lượng ảnh” > “chi phí phải đợi request và load từ server” thì sử dụng mới có lãi ạ.
* Signup(): thực hiện signup sau khi gọi tới hàm check các input (ở đây e có gặp 1 chút issue khi qua 1 ngày khởi động lại máy ko kết nối được tới firestore DB để lưu trữ thông tin ng dùng sign up ạ -> em fix bằng thêm cái này vào build.gradle dependencies ạ: implementation "io.grpc:grpc-okhttp:1.32.2")

8. Thử nghiệm tính năng (test)

Trong phần thử nghiệm tính năng, bởi em là lập trình viên luôn nên vô hình chung em test White Box ạ :D em sử dụng kĩ thuật test phân vùng (valid và invalid), bởi thời gian có hạn nên em chưa test được thêm theo mô hình test giá trị biên ạ.

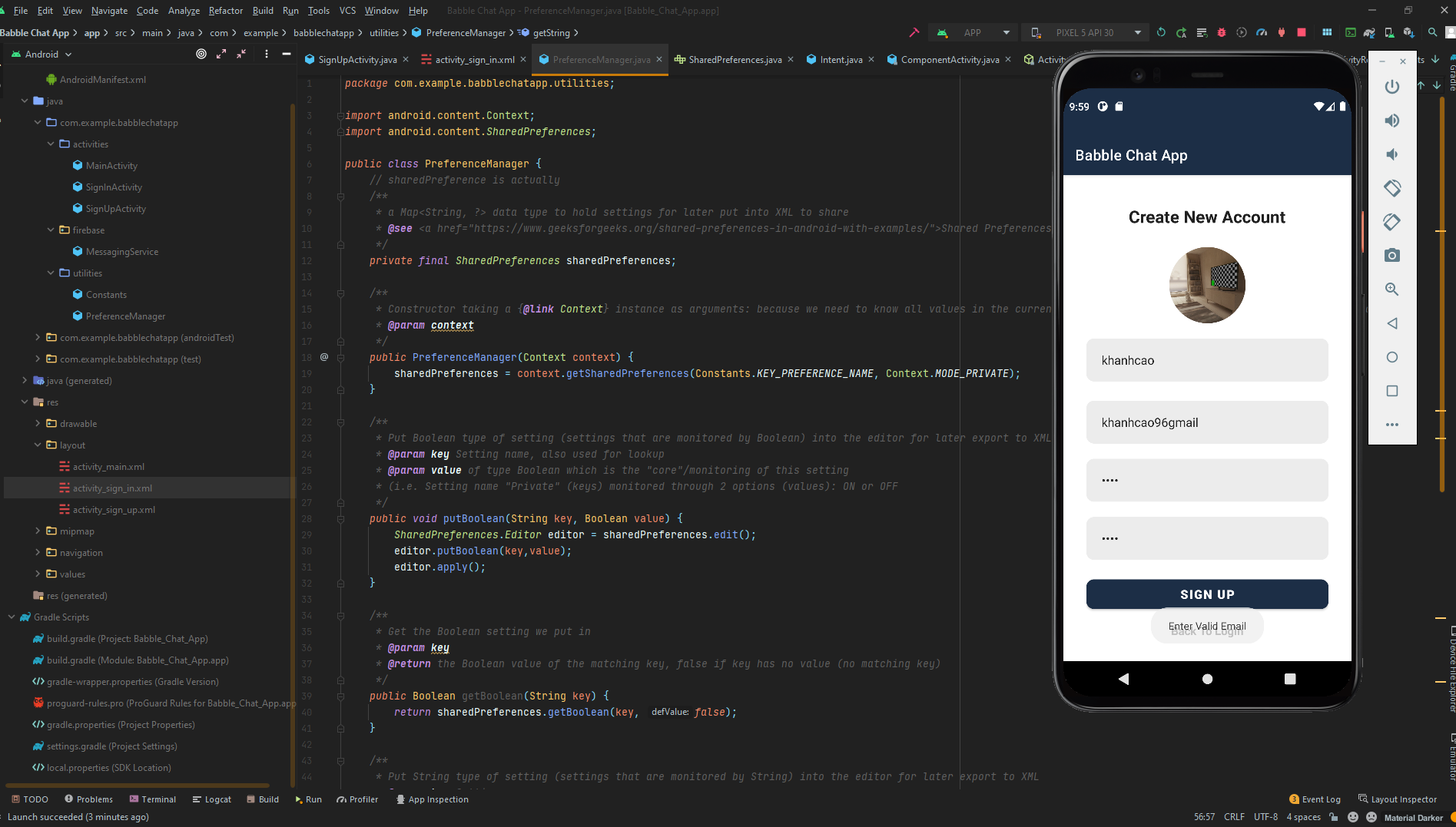
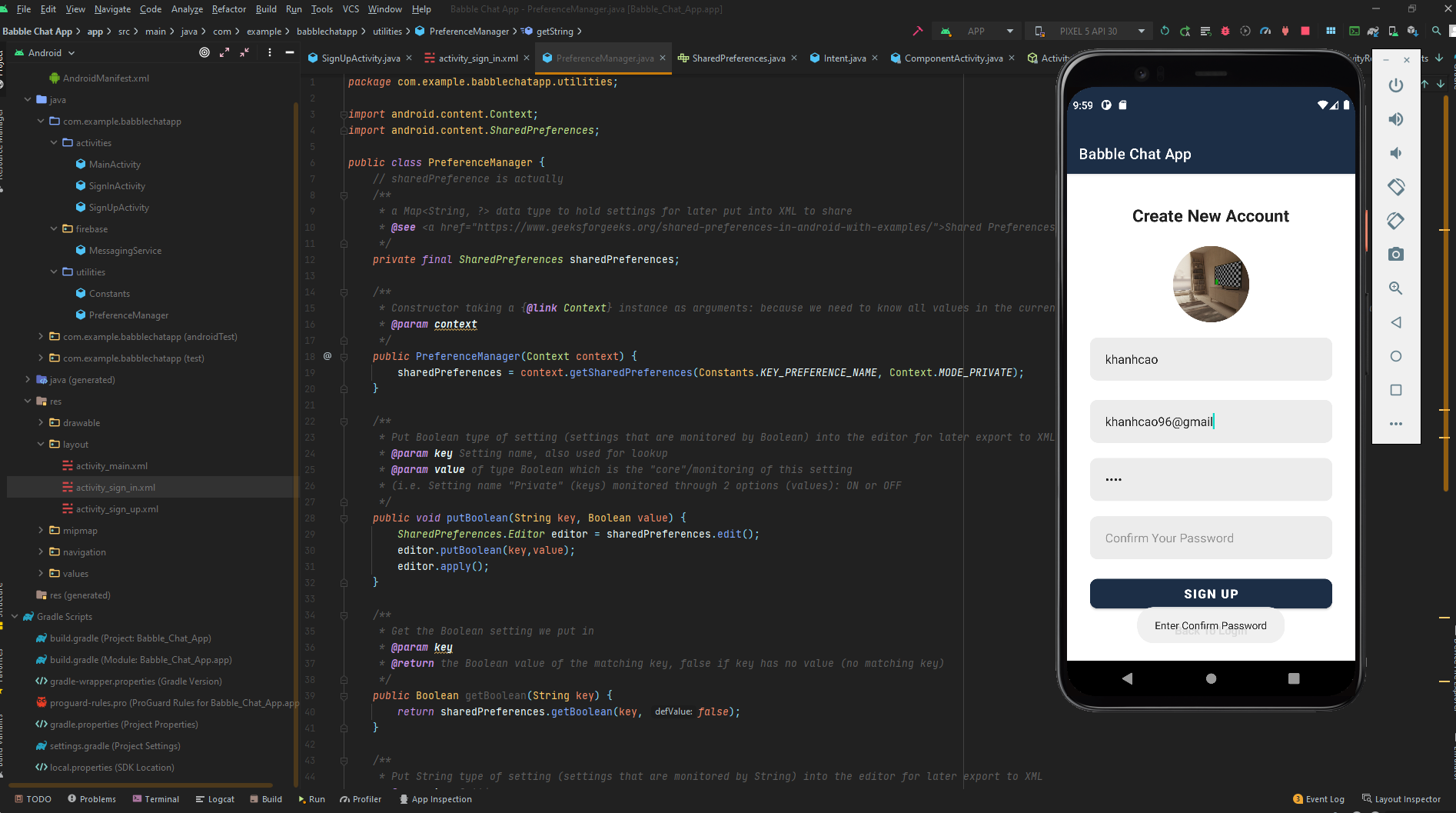
Usecase 1. Tạo tài khoản:

Input: Ảnh, tên, email, mật khẩu (2 lần)

Output:

* Luồng chính (dữ liệu hợp lệ): user được đưa lại về màn hình login, dữ liệu input thành dữ liệu đầu vào database của FireBase chứa thông tin và mã băm tra cứu người dùng đó
* Luồng mở rộng (dữ liệu không hợp lệ): Thông báo ra ngoài màn hình (Toast) yêu cầu người dùng nhập lại

Test vùng invalid (nhận biết được và yêu cầu người dung nhập lại)



Test vùng valid (thành công)

