

Data and Backend



Huỳnh Tuấn Anh - ĐHNT

1

Nội dung

- Local key-value storage:
 - Shared preferences, localstorage, get_storage, flutter_secure_storage: Dữ liệu key-value đơn giản, nhỏ gọn
 - Hive: Key-value database
- Read and Write Files
- SQLite: sqflite
- JSON and Serialization
- Firebase
- MongoDB

Huỳnh Tuấn Anh - ĐHNT

2

Shared Preferences



Huỳnh Tuấn Anh - ĐHNT

3

Shared preferences plugin

- Plugin đa nền tảng hỗ trợ lưu trữ dữ liệu đơn giản: UserDefaults trên iOS và macOS, SharedPreferences trên Android.
- Dữ liệu được ghi vào đĩa không đồng bộ: Không đảm bảo dữ liệu sẽ được ghi sau khi thao tác ghi được trả về, vì vậy nên cẩn thận khi sử dụng plugin này để ghi các dữ liệu quan trọng của ứng dụng.

- Sử dụng:

- Khai báo dependencies pubspec.yaml
- import package:

```
dependencies:
  shared_preferences:
```

```
import 'package:shared_preferences/shared_preferences.dart';
```

Huỳnh Tuấn Anh - ĐHNT

4

Ví dụ: Counter App: class _MyHomePageState

```
@override
void initState() {
  _getSavedCounter();
}
```

```
void _getSavedCounter () async{
  SharedPreferences sharedPreferences = await SharedPreferences.getInstance();
  _counter = (sharedPreferences.getInt('counter') ?? 0);
  setState(() {
  });
}
```

```
void _incrementCounter() async {
  setState(() {
    _counter++;
  });
  SharedPreferences sharedPreferences = await SharedPreferences.getInstance();
  await sharedPreferences.setInt('counter', _counter);
}
```

Huỳnh Tuấn Anh - ĐHNT

5

Methods

- **Static:**
 - `getInstance()` → `Future<SharedPreferences>`
- `clear()` → `Future<bool>`: Xóa User Preferences và trả về giá trị true khi hoàn tất.
- `containsKey(String key)` → `bool`
- `get(String key)` → `dynamic`
- `getBool(String key)` → `bool`
- `getDouble(String key)` → `double`
- `getInt(String key)` → `int`
- `getKeys()` → `Set<String>`: Trả về tất cả các key được lưu trữ

Huỳnh Tuấn Anh - ĐHNT

6

Methods

- `getString(String key) → String`
- `getStringList(String key) → List<String>`
- `reload()` → `Future<void>`: Nạp giá trị sau cùng từ nền tảng lưu trữ
- `remove(String key) → Future<bool>`
- `setBool(String key, bool value) → Future<bool>`
- `setDouble(String key, double value) → Future<bool>`
- `setInt(String key, int value) → Future<bool>`
- `setString(String key, String value) → Future<bool>`
- `setStringList(String key, List<String> value) → Future<bool>`

Huỳnh Tuấn Anh - ĐHNT

7

Read and Write Files



Huỳnh Tuấn Anh - ĐHNT

8

Recipe

1. Xác định đường dẫn cục bộ.
2. Tạo một tham chiếu đến vị trí tệp.
3. Ghi dữ liệu vào tệp.
4. Đọc dữ liệu từ tệp.

Xác định đường dẫn cục bộ

- Sử dụng `path_provider` (pub.dev): Hỗ trợ quyền truy cập vào hai vị trí hệ thống tệp:

- *Temporary directory*: Thư mục tạm thời (cache) mà hệ thống có thể xóa bất kỳ lúc nào.

```
Future<String> get _tempDirPath async{
  final temp = await getTemporaryDirectory();
  return temp.path;
}
```

- *Documents directory*: Một thư mục dành riêng cho app mà chỉ nó mới có thể truy cập. Hệ thống chỉ có thể xóa thư mục này nếu app bị xóa.

```
Future<String> get _docDirPath async{
  final directory = await getApplicationDocumentsDirectory();
  return directory.path;
}
```

Tạo một tham chiếu đến vị trí tệp

- Trả về một đối tượng File
- Sử dụng lớp File trong thư viện dart:io.

```
Future<File> getLocalFile(String fileName) async{
  String docPath = await _docDirPath;
  return File('$docPath/$fileName');
}
```

Ghi dữ liệu vào file

- Tạo tham chiếu đến đối tượng file
- Ghi dữ liệu vào file, sử dụng các phương thức:
 - `Future<File> writeAsString(String contents, {FileMode mode: FileMode.write, Encoding encoding: utf8, bool flush: false});`
 - FileMode: write, append
 - `void writeAsStringSync(String contents, {FileMode mode: FileMode.write, Encoding encoding: utf8, bool flush: false});`
 - file tự động đóng khi ghi xong dữ liệu

```
Future<File> writeCounter(int counter) async{
  final File file = await getLocalFile('counter.txt');
  // Write the file.
  return file.writeAsString('$counter');
}
```

Đọc dữ liệu từ file

■ Sử dụng các phương thức của lớp File:

- `Future<String> readAsString ({Encoding encoding: utf8})`: Đọc toàn bộ nội dung của file
- `Future<List<String>> readAsLines ({Encoding encoding: utf8})`: Đọc toàn bộ nội dung file dưới dạng các dòng văn bản.
- `String readAsStringSync({Encoding encoding: utf8})`: Đọc toàn bộ nội dung của file một cách đồng bộ.
- `List<String> readAsLinesSync({Encoding encoding: utf8})`: Đọc toàn bộ nội dung file dưới dạng các dòng văn bản một cách đồng bộ.

Đọc dữ liệu từ file

■ Ví dụ:

```
Future<int> readCounter(String fileName) async {
  try {
    final file = await getLocalFile(fileName);
    String counter = await file.readAsString();
    return int.parse(counter);
  } catch (e) {
    return 0;
  }
}
```

Hiển thị Image từ local File

- Sử dụng widget Image

- Image.file(File)

- Chọn file từ bộ nhớ local: Sử dụng thư viện image_picker (pub.dev)

```
var image = await imagePicker.pickImage(source: ImageSource.gallery); //camera: sd camera
if(image == null)
  return null;
else
  setState(() {
    _image = File(image.path);
  });

...
Container(
  child: Image.file(_image),
)
```

JSON and Serialization



Json Data Types

- In JSON, values must be one of the following data types:
 - a string
 - a number
 - an object (JSON object)
 - an array
 - a boolean
 - *null*
- JSON values **cannot** be one of the following data types:
 - a function
 - a date
 - *undefined*

Serializing JSON manually using dart:convert

- Sử dụng gói thư viện dart:convert:
 - `import 'dart:convert'`
 - Phương thức: `Map<String , dynamic> jsonDecode(String str)`: Chuyển một chuỗi thành một đối tượng Map (gồm các cặp Key-Value).
 - Phương thức `jsonEncode()`: Nhận một đối tượng và chuyển đối tượng đó thành chuỗi Json

```
void main() {
  var jsonString = '{
    "name": "Tuan",
    "email": "tuan@gmail.com"
  }';
  Map<String , dynamic> user = jsonDecode(jsonString);
  print("Chao ${user['name']}");
}
```

Serializing JSON inside model classes

- Cài đặt class có các phương thức sau:

- Named constructor dùng để khởi tạo một thể hiện của lớp từ cấu trúc Map. Ví dụ: `User.fromJson()`
- Phương thức `toJson()`: chuyển một thể hiện của lớp thành một Map.

```
class User {
    final String name;
    final String email;

    User(this.name, this.email);

    User.fromJson(Map<String, dynamic> json)
        : name = json['name'],
          email = json['email'];

    Map<String, dynamic> toJson() =>
    {
        'name': name,
        'email': email,
    };
}
```

Serializing JSON inside model classes

```
void main() {
    var jsonString = '{
        "name": "Tuan",
        "email": "tuan@gmail.com"
    }';
    Map<String, dynamic> userMap = jsonDecode(jsonString);
    User user = User.fromJson(userMap);
    print("Chao ${user.name}");
}
```

- Để chuyển một User thành một chuỗi Json, không cần phải gọi phương thức `toJson()`, phương thức `jsonEncode()` đã làm công việc này:
 - `String json = jsonEncode(user);`

JSON Array to List Object

- B1: Serializing JSON inside model classes
 - Ví dụ: Định nghĩa lớp Photo với 2 phương thức:
 - Constructor `Photo.fromJson(Map<String, dynamic> json)`
 - Phương thức `toJson()` trả về một `Map<String, dynamic>`
- B2: Sử dụng phương thức `json.decode` để trả về một List các object
 - Ví dụ: `jsonString` là một chuỗi biểu diễn một mảng Array các photo

```
List<Photo> photos;
photos = (json.decode(jsonString) as List).map((item) =>
    Photo.fromJson(item)).toList();
```


Ví dụ:

- Tại địa chỉ: <https://jsonplaceholder.typicode.com/photos> chứa một Json Array

```
[
  {
    "albumId": 1,
    "id": 1,
    "title": "accusamus beatae ad facilis cum similique qui sunt",
    "url": "https://via.placeholder.com/600/92c952",
    "thumbnailUrl": "https://via.placeholder.com/150/92c952"
  },
  {
    "albumId": 1,
    "id": 2,
    "title": "reprehenderit est deserunt velit ipsam",
    "url": "https://via.placeholder.com/600/771796",
    "thumbnailUrl": "https://via.placeholder.com/150/771796"
  },
]
```

Ví dụ:

- Đọc chuỗi Json Array này, phân tích và hiển thị trên màn hình của ứng dụng:
 - Các thư viện sử dụng:
 - import 'package:http/http.dart' as http;




Huỳnh Tuấn Anh - ĐHNT
23

class Photo

```
class Photo{
  final int albumId;
  final int id;
  final String title;
  final String url;
  final String thumbnailUrl;

  Photo({this.albumId, this.id, this.title, this.url, this.thumbnailUrl});

  factory Photo.fromJson(Map<String, dynamic> json){
    return Photo(
      albumId : json['albumId'] as int,
      id : json['id'] as int,
      title : json['title'] as String,
      url : json['url'] as String,
      thumbnailUrl : json['thumbnailUrl'] as String);
  }
}
```



Phương thức cần thiết để Decode một chuỗi Json thành một đối tượng Photo

Huỳnh Tuấn Anh - ĐHNT
24

Chuyển Json Array thành List<Photo>

```
Future<List<Photo>> fetchPhotos() async{
  final response = await http.get('https://jsonplaceholder.typicode.com/photos');
  if(response.statusCode==200)
  {
    List<Photo> photos;
    var list = json.decode(response.body) as List;
    photos = list.map((item) => Photo.fromJson(item)).toList();
    return photos;
  }
  else{
    print("Không tải được Album");
    throw Exception("Khong tai duoc Album");
  }
}
```

Chuyển chuỗi Json
Array thành danh
sách các đối tượng
Photo

PhotoPage: StatefulWidget

```
class _PhotosPageState extends State<PhotosPage> {
  Future<List<Photo>> photos;
  @override
  void initState() {
    super.initState();
    photos = fetchPhotos();
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(...), // AppBar
      body: FutureBuilder<List<Photo>>(...), // FutureBuilder
    ); // Scaffold
  }
}
```

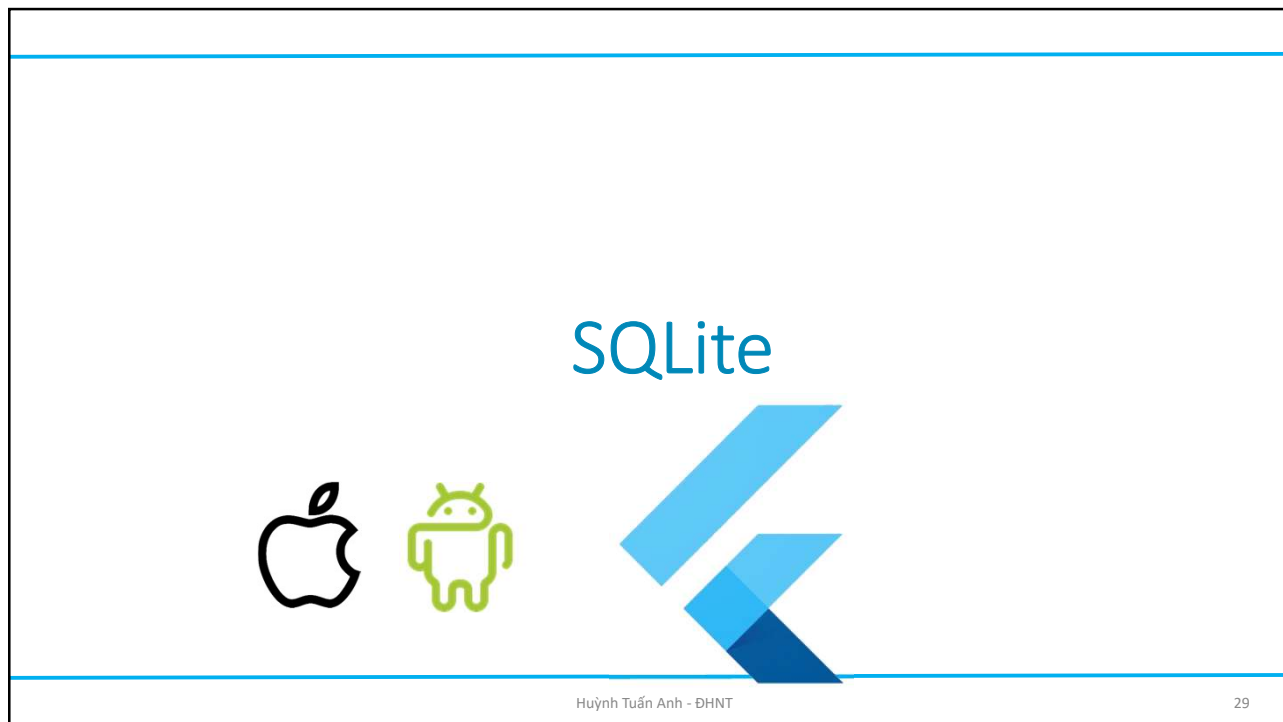
FutureBuilder: Widget làm việc với dữ liệu không đồng bộ


- future: Đối tượng dữ liệu không đồng bộ (Async)
- builder: Phương thức trả về một Widget hiển thị dữ liệu không đồng bộ

```
body: FutureBuilder<List<Photo>>(  
  future: photos,  
  builder: (context, snapshot) {  
    if(snapshot.hasError) {  
      print("Lỗi xảy ra");  
      return Text("Lỗi xảy ra");  
    }  
    return snapshot.hasData  
      ? Photolist(photos: snapshot.data,)  
      : Center(child: CircularProgressIndicator(),);  
  },  
)
```

PhotoList

```
class PhotoList extends StatelessWidget {  
  List<Photo> photos;  
  PhotoList({Key key, this.photos}):super(key:key);  
  @override  
  Widget build(BuildContext context) {  
    return GridView.extent(  
      maxCrossAxisExtent: 200, padding: EdgeInsets.all(5),  
      mainAxisSpacing: 5, crossAxisSpacing: 5,  
      children: List.generate(photos.length, (index) => Container(  
        decoration: BoxDecoration(  
          border: Border.all(color: Colors.blue),  
        ),  
        child: Image.network(photos[index].thumbnailUrl),  
      )),  
    );  
  }  
}
```



 SQLite

- Plugin hỗ trợ SQLite trong flutter
 - `sqlite`: <https://pub.dev/packages/sqlite>
 - Cài đặt:

```
dependencies:  
  ...  
  sqlite: ^1.3.0
```
 - Import: `import 'package:sqlite/sqlite.dart';`
- SV tìm hiểu thêm về `sqlite` online

Huỳnh Tuấn Anh - ĐHNT 30

SQLite database

- Cơ sở dữ liệu quan hệ rút gọn thường dùng trong các hệ thống mobile
- Cơ sở dữ liệu SQLite trong mobile là một tệp trong hệ thống tệp được xác định bằng một đường dẫn.
 - Phương thức: `getDatabasesPath()` trả về đường dẫn thư mục cơ sở dữ liệu mặc định trên Android và thư mục tài liệu trên iOS
- Các bước làm việc với SQLite database
 - Thiết kế mô hình dữ liệu
 - Mở cơ sở dữ liệu
 - Thực hiện câu truy vấn tạo bảng
 - Sử dụng cơ sở dữ liệu (thực hiện truy vấn CRUD)

Huỳnh Tuấn Anh - ĐHNT

31

Open Database

```
// Get a Location using getDatabasesPath
var databasesPath = await getDatabasesPath();
String path = databasesPath + '/demo.db';
// open the database, Nếu database chưa tồn tại onCreate sẽ được gọi
Database database = await openDatabase(path, version: 1,
  onCreate: (Database db, int version) async {
    // When creating the db, create the table
    await db.execute(
      'CREATE TABLE Test (id INTEGER PRIMARY KEY, name TEXT, value INTEGER, num REAL)');
    db.execute(
      'CREATE TABLE Users (id INTEGER PRIMARY KEY, name TEXT, phone TEXT, email TEXT)');
  });

void closeDatabase() async{
  await database!.close();
}
void deleteDB(){
  deleteDatabase(_path!);
}
```

Huỳnh Tuấn Anh - ĐHNT

32

Chèn dữ liệu

```
await database.transaction((txn) async {
  int id1 = await txn.rawInsert(
    'INSERT INTO Test(name, value, num) VALUES("some name", 1234, 456.789)');
  print('inserted1: $id1');
  int id2 = await txn.rawInsert(
    'INSERT INTO Test(name, value, num) VALUES(?, ?, ?)',
    ['another name', 12345678, 3.1416]);
  print('inserted2: $id2');
});
```

Phương thức rawInsert trả về id của bản ghi cuối cùng được chèn vào bảng

Ví dụ: INSERT

```
Future<int> insert(User user) async {
  int id = await database.transaction(
    (txn) async {
      int id = await txn.rawInsert(
        'INSERT INTO $tableName(name, phone, email) VALUES( ?, ?, ?)',
        [user.name, user.phone, user.email],
      );
      return id;
    }
  );
  return id;
}
```

```
Future<int> insert(User user) async {
  int id = await database.rawInsert(
    'INSERT INTO $tableName(name, phone, email) VALUES( ?, ?, ?)',
    [user.name, user.phone, user.email],
  );
  return id;
}
);
return id;
}
```

Update

```
Future<int> update(User newUser, int id) async {
  int count = await database!.transaction((txn) async {
    int count = await txn.rawUpdate(
      'UPDATE $tableName SET name = ?, phone = ?, email = ? WHERE id = ?',
      [newUser.name, newUser.phone, newUser.email, id]
    );
    return count;
  });
  return count;
}
```

```
Future<int> update(User newUser, int id) async {
  int count = await database!.rawUpdate(
    'UPDATE $tableName SET name = ?, phone = ?, email = ? WHERE id = ?',
    [newUser.name, newUser.phone, newUser.email, id]
  );
  return count;
}
```

Huỳnh Tuấn Anh - ĐHNT

35

Query

```
List<Map> list = await database.rawQuery('SELECT * FROM Test');
```

Kết quả:

```
[
  {'name': 'updated name', 'id': 1, 'value': 9876, 'num': 456.789},
  {'name': 'another name', 'id': 2, 'value': 12345678, 'num': 3.1416}
];
```

Trả về kết quả là danh sách User:

```
Future<List<User>> getUsers() async{
  List<Map> list = await database.rawQuery("SELECT * FROM Users");
  return list.map((userJson) => User.fromJson(userJson)).toList();
}
```

Huỳnh Tuấn Anh - ĐHNT

36

Firebase



Huỳnh Tuấn Anh - ĐHNT

37



Nội dung

- Firebase Authentication
- Firebase Firestore Database
- Firebase Storage
- Firebase Cloud Messaging

Huỳnh Tuấn Anh - ĐHNT

38

Firebase

- Firebase là nền tảng phát triển ứng dụng Backend-as-a-Service (BaaS) cung cấp các dịch vụ backend được lưu trữ trên máy chủ như cơ sở dữ liệu thời gian thực, lưu trữ đám mây, xác thực, báo cáo sự cố, máy học, cấu hình từ xa và lưu trữ cho các tệp tĩnh của ứng dụng mà không cần phải duy trì máy chủ riêng cho ứng dụng.
- Thư viện sử dụng: `firebase_core`:
 - https://pub.dev/packages/firebase_core
 - Là một Flutter plugin để sử dụng Firebase Core API, cho phép kết nối với nhiều ứng dụng Firebase.
 - FlutterFire: Các plugin do Google phát triển để làm việc với Firebase





Làm việc với Firebase

- Tạo dự án trên Firebase console.
- Tạo dự án Flutter trong Android Studio.
- Kết nối dự án Flutter với dự án trên Firebase (sinh viên tự tìm hiểu)
 - Android: <https://firebase.google.com/docs/flutter/setup?platform=android>
 - iOS: <https://firebase.google.com/docs/flutter/setup?platform=ios>
- Cloud Firestore: Cơ sở dữ liệu NoSQL trên Firebase
- `cloud_firestore`: API làm việc với csdl Cloud Firestore do Google cung cấp (pub.dev)
- `firebase_auth`: API dùng để xác thực người dùng
- `firebase_storage`: API dùng để làm việc với dịch vụ lưu trữ trên firebase

Firebase Authentication

- Khái niệm xác thực người dùng
- Một số phương pháp xác thực do Firebase cung cấp:
 - Google
 - Email/Password
 - Phone
 - Anonymous

Huỳnh Tuấn Anh - ĐHNT 41

Provider	Status	
 Email/Password	Enabled	
 Phone	Enabled	
 Google	Enabled	
 Play Games	Disabled	
 Game Center	Disabled	
 Facebook	Disabled	
 Twitter	Disabled	
 GitHub	Disabled	
 Yahoo	Disabled	
 Microsoft	Disabled	
 Apple	Disabled	
 Anonymous	Disabled	


42

Firebase Authentication

- Authentication: Xác minh và cho phép người dùng đăng nhập vào ứng dụng
- Cung cấp các dịch vụ hỗ trợ (backend service), các SDK để sử dụng, và các thư viện giao diện người dùng được tạo sẵn để xác thực người dùng với ứng dụng
- Các phương thức xác thực:
 - Mật khẩu
 - Số điện thoại
 - Các nhà cung cấp định danh liên hợp như: Google, Facebook, Twitter...
- Có thể đăng nhập người dùng vào ứng dụng Firebase bằng cách sử dụng FirebaseUI để nhúng giải pháp xác thực hoàn chỉnh hoặc sử dụng các Firebase Authentication SDK để tích hợp thủ công một hay nhiều phương thức xác thực vào ứng dụng.

firebase_auth plugin


- flutter plugin cung cấp các Firebase Authentication API
- Cho phép các ứng dụng iOS, Android thực hiện xác thực bằng mật khẩu, số điện thoại và các nhà cung cấp định danh như: Google, Facebook, Twitter
- Để sử dụng xác thực firebase cần khai báo hai thư viện: `firebase_core`, `firebase_auth`
- Một số thư viện hỗ trợ:
 - `sms_autofill`: Tự động lấy số điện thoại thiết bị, tự động lấy/điền code SMS từ firebase dùng để xác thực.
 - `google_sign_in`: Thư viện kết hợp với `firebase_auth` dùng để đăng nhập theo phương pháp Google
 - `sign_button`: Thư viện dùng để thiết kế các nút bấm đăng nhập
- Cần phải khởi tạo `FlutterFire` trước khi sử dụng:
 - `Firestore.initializeApp(options: DefaultFirebaseOptions.currentPlatform)`
- Tạo một thể hiện của `FirebaseAuth`: `FirebaseAuth auth = FirebaseAuth.instance;`



firebase_ui_auth

- Plugin tích hợp với UI cho phép thực hiện các phương pháp xác thực với firebase một cách dễ dàng
- Link hướng dẫn:
 - <https://firebase.flutter.dev/docs/ui/auth/>
 - Từ khóa: FlutterFire UI for Auth
- Địa phương hóa giao diện của firebase_ui_auth
 - [Localization | FlutterFire](#)
 - <https://firebase.flutter.dev/docs/ui/auth/localization/>

Huỳnh Tuấn Anh - ĐHNT 45



Đăng nhập bằng Google

- Nguyên tắc
 - Tạo một đối tượng AuthCredential: credential
 - Đăng nhập bằng FirebaseAuth với đối tượng credential:


```
FirebaseAuth.instance.signInWithCredential(credential);
```

Huỳnh Tuấn Anh - ĐHNT 46

Đăng nhập bằng Google

```
Future<UserCredential> _signInWithGoogle() async {
  // Trigger the authentication flow
  final GoogleSignInAccount googleUser = await GoogleSignIn().signIn();
  // Obtain the auth details from the request
  final GoogleSignInAuthentication googleAuth = await googleUser.authentication;
  // Create a new credential
  final credential = GoogleAuthProvider.credential(
    accessToken: googleAuth.accessToken,
    idToken: googleAuth.idToken,
  );
  // Once signed in, return the UserCredential
  return await FirebaseAuth.instance.signInWithCredential(credential);
}
```

Đăng nhập thành công: đối tượng *UserCredential* khác null sẽ được trả về

Đăng nhập với Email/Password

- Cần phải đăng ký Email/Password với Firebase Authentication
 - Email: Email đăng ký với Google Firebase Authentication
 - Password: Password dùng để đăng nhập ứng dụng, không phải password dùng để đăng nhập Email
- Đăng ký Email/Password

Registration

```

showLoaderDialog(context, "Đang thực hiện");
try {
    await FirebaseAuth.instance.createUserWithEmailAndPassword(
        email: "abc_test@gmail.com",
        password: "12345678",
    );
    Navigator.pop(context); // tắt loader Dialog
    setState(() {
        registerMessage = "Đăng ký thành công";
    });
} on FirebaseAuthException catch(e){
    Navigator.pop(context); // tắt loader Dialog
    if(e.code=='weak-password')
        setState(() {
            registerMessage = "Sử dụng password mạnh hơn";
        });
    else
        if(e.code=='email-already-in-use')
            setState(() {
                registerMessage = "Email đã tồn tại";
            });
}
}

```

Huỳnh Tuấn Anh - ĐHNT 49

Loader Dialog

```

showLoaderDialog(BuildContext context, String state){
    AlertDialog alert=AlertDialog(
        content: new Row(
            children: [
                CircularProgressIndicator(),
                Container(margin: EdgeInsets.only(left: 7),child:Text(state)),
            ],),
    );
    showDialog(barrierDismissible: false,
        context:context,
        builder:(BuildContext context){
            return alert;
        },
    );
}

```

Huỳnh Tuấn Anh - ĐHNT 50

Login bằng Email/Password

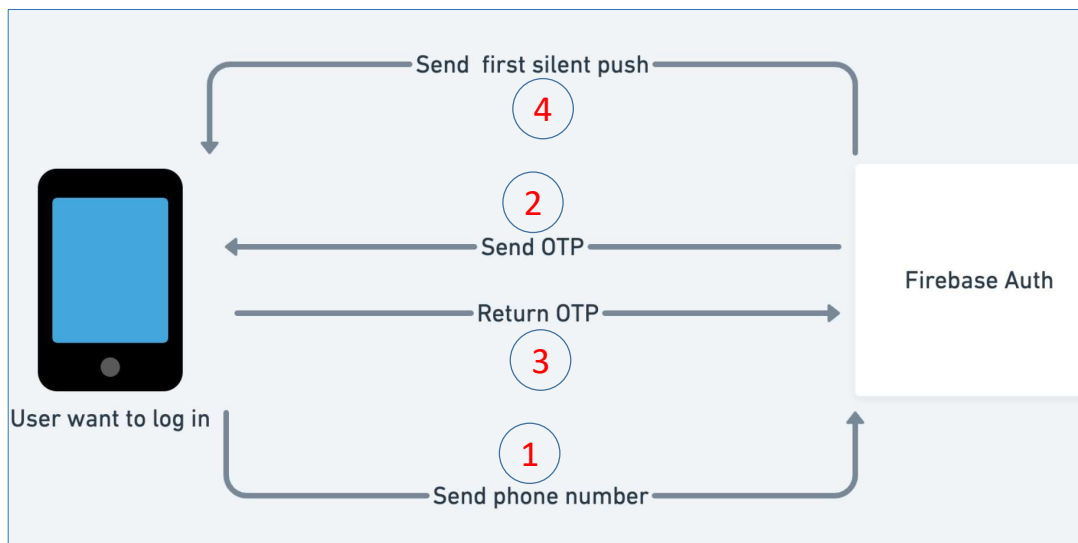
```
try {
  await FirebaseAuth.instance.signInWithEmailAndPassword(
    email: "abc_test@gmail.com", password: "12345678");
  // Chuyển đến trang home của ứng dụng sau khi đăng nhập thành công
  Navigator.of(context).pushAndRemoveUntil(MaterialPageRoute(
    builder: (context) => HomePage()), (Route<dynamic> route) => false);

} on FirebaseAuthException catch(e){
  if(e.code == 'user-not-found')
    setState(() {
      _loginMessage = "Email chưa được đăng ký";
    });
  else if (e.code == 'wrong-password')
    setState(() {
      _loginMessage = "Không đúng password đã đăng ký";
    });
}
```

Huỳnh Tuấn Anh - ĐHNT

51

Xác thực số điện thoại



Huỳnh Tuấn Anh - ĐHNT

52

Thiết lập

- Bật phương thức xác thực bằng số điện thoại trên Firebase Console.
- Android: Điền mã SHA-1 vào dự án liên kết với Firebase trên Firebase Console
- iOS: Trong XCode, Bật thông báo đẩy cho dự án của bạn và đảm bảo khóa xác thực APN của bạn được định cấu hình với Firebase Cloud Messaging (FCM)
- Web: Đảm bảo rằng bạn đã thêm application domain của mình trên Firebase, trong **OAuth redirect domains**..
- Chú ý: Đăng nhập bằng số điện thoại chỉ khả dụng trên thiết bị thực và trên web. Để kiểm tra quy trình xác thực của bạn trên trình giả lập thiết bị, hãy đăng ký một số điện thoại test với smsCode với Firebase để kiểm tra.



Trong mục Authentication bật phương thức xác thực Phone và tiến hành một số bước cấu hình bổ sung cho dự án

 Phone

 Enable



Phone Authentication requires additional configuration steps. Follow the steps for your platform.

[Apple](#) 

[Android](#) 

[Web](#) 

Sử dụng sms_autofill để tự động lấy số điện thoại thiết bị


▪ Khai báo: `final SmsAutoFill _autoFill = SmsAutoFill();`

▪ Đọc số điện thoại của thiết bị:

```
// Khai báo controller cho một TextField để hiển thị/nhập số điện thoại
TextEditingController txtPhone = TextEditingController();
.....

// Hiển thị UI để chọn số điện thoại trên thiết bị
txtPhone.text = await _autoFill.hint;
```

Continue with

 (555) 521-5554

NONE OF THE ABOVE

55

```
await FirebaseAuth.instance.verifyPhoneNumber(
  phoneNumber: '+44 7123 123 456',
  1 verificationCompleted: (PhoneAuthCredential credential) {},
  2 verificationFailed: (FirebaseAuthException e) {},
  3 codeSent: (String verificationId, int? resendToken) {},
  4 codeAutoRetrievalTimeout: (String verificationId) {},
);
```

1. Callback được gọi tự động để xử lý SMS code trên các thiết bị Android
2. Callback được gọi tự động khi xác thực bị lỗi
3. Callback được gọi khi mã SMS gửi đến thiết bị có cài số điện thoại cần xác thực, được sử dụng để nhắc người dùng nhập mã xác thực
4. Callback được gọi khi hết thời gian xác thực

verificationCompleted

- Trình xử lý này sẽ chỉ được gọi trên các thiết bị Android hỗ trợ giải mã SMS tự động.
- Khi mã SMS được gửi tới thiết bị, Android sẽ tự động xác minh mã SMS mà không yêu cầu người dùng nhập mã theo cách thủ công.

```

FirebaseAuth auth = FirebaseAuth.instance;

await auth.verifyPhoneNumber(
  phoneNumber: '+44 7123 123 456',
  verificationCompleted: (PhoneAuthCredential credential) async {
    // ANDROID ONLY!

    // Sign the user in (or link) with the auto-generated credential
    await auth.signInWithCredential(credential);
  },
);

```

Huỳnh Tuấn Anh - ĐHNT

57

verificationFailed

- Nếu Firebase trả về lỗi, chẳng hạn như số điện thoại không chính xác hoặc nếu vượt quá quota SMS cho dự án, một FirebaseAuthException sẽ được gửi đến trình xử lý này.

```

FirebaseAuth auth = FirebaseAuth.instance;

await auth.verifyPhoneNumber(
  phoneNumber: '+44 7123 123 456',
  verificationFailed: (FirebaseAuthException e) {
    if (e.code == 'invalid-phone-number') {
      print('The provided phone number is not valid.');
```

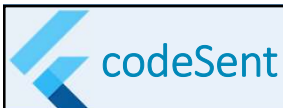
```

    // Handle other errors
  },
);

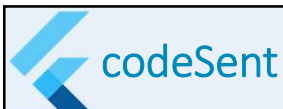
```

Huỳnh Tuấn Anh - ĐHNT

58



- Khi Firebase gửi mã SMS đến thiết bị, trình xử lý này được kích hoạt bằng một `verifyId` và `resendToken` (`resendToken` chỉ được hỗ trợ trên thiết bị Android, thiết bị iOS sẽ luôn trả về giá trị null).
- Sau khi được kích hoạt, đây sẽ là thời điểm tốt để cập nhật giao diện người dùng ứng dụng của bạn để nhắc người dùng nhập mã SMS mà họ mong đợi. Khi mã SMS đã được nhập, bạn có thể kết hợp ID xác minh với mã SMS để tạo `PhoneAuthCredential` dùng để đăng nhập



```

FirebaseAuth auth = FirebaseAuth.instance;
await auth.verifyPhoneNumber(
  phoneNumber: '+44 7123 123 456',
  codeSent: (String verificationId, int? resendToken) async {
    // Update the UI - wait for the user to enter the SMS code
    String smsCode = 'xxx';
    // Create a PhoneAuthCredential with the code
    PhoneAuthCredential credential = PhoneAuthProvider.credential(verificationId:
      verificationId, smsCode: smsCode);
    // Sign the user in (or link) with the credential
    await auth.signInWithCredential(credential);
  },

```

codeAutoRetrievalTimeout

- Trên các thiết bị Android hỗ trợ giải mã SMS tự động, trình xử lý này sẽ được gọi nếu thiết bị không tự động giải quyết một tin nhắn SMS trong một khung thời gian nhất định. Khi khung thời gian đã trôi qua, thiết bị sẽ không cố gắng giải quyết bất kỳ tin nhắn đến nào nữa.
- Theo mặc định, thiết bị đợi trong 30 giây và có thể tùy chỉnh bằng tham số timeout

```

FirebaseAuth auth = FirebaseAuth.getInstance();

await auth.verifyPhoneNumber(
  phoneNumber: '+44 7123 123 456',
  timeout: const Duration(seconds: 60),
  codeAutoRetrievalTimeout: (String verificationId) {
    // Auto-resolution timed out...
  },
);

```

Huỳnh Tuấn Anh - ĐHNT

61

Xác thực khi sử dụng máy ảo hay số test (số không có thật)

- Trong trường hợp sử dụng máy ảo hoặc số điện thoại test, sẽ không có tin nhắn xác thực đến thiết bị do đó quá trình xác thực sẽ không tự động hoàn tất (do callback `verificationCompleted` không được gọi)
- Mỗi số điện thoại thật trong thời gian ngắn chỉ được test với một số lần giới hạn (khoảng 5 lần) quá số lần này số điện thoại đó bị tính là spam và bị block một thời gian, khoảng vài giờ, sau đó số này mới được sử dụng lại. Nên sử dụng số điện thoại đăng ký test với firebase để test kỹ ứng dụng trước khi test trên số thật

Phone numbers for testing (optional) ⓘ

Phone number

+1 650-555-1234

Verification code

Add

+1 650-555-1234

123456

Khai báo số điện thoại để test, mã số tin nhắn xác thực, số điện thoại này có thể không có thật nhưng phải đúng format của một số và không được sử dụng một số điện thoại có thật, tốt nhất là theo format gợi ý của firebase

Huỳnh Tuấn Anh - ĐHNT

62

Xác thực khi sử dụng máy ảo hay sdt test

1. Sử dụng phương thức `FirebaseAuth.instance.verifyPhoneNumber(...)`
2. Trong callback `codeSent` lấy mã xác minh (`verificationId`)

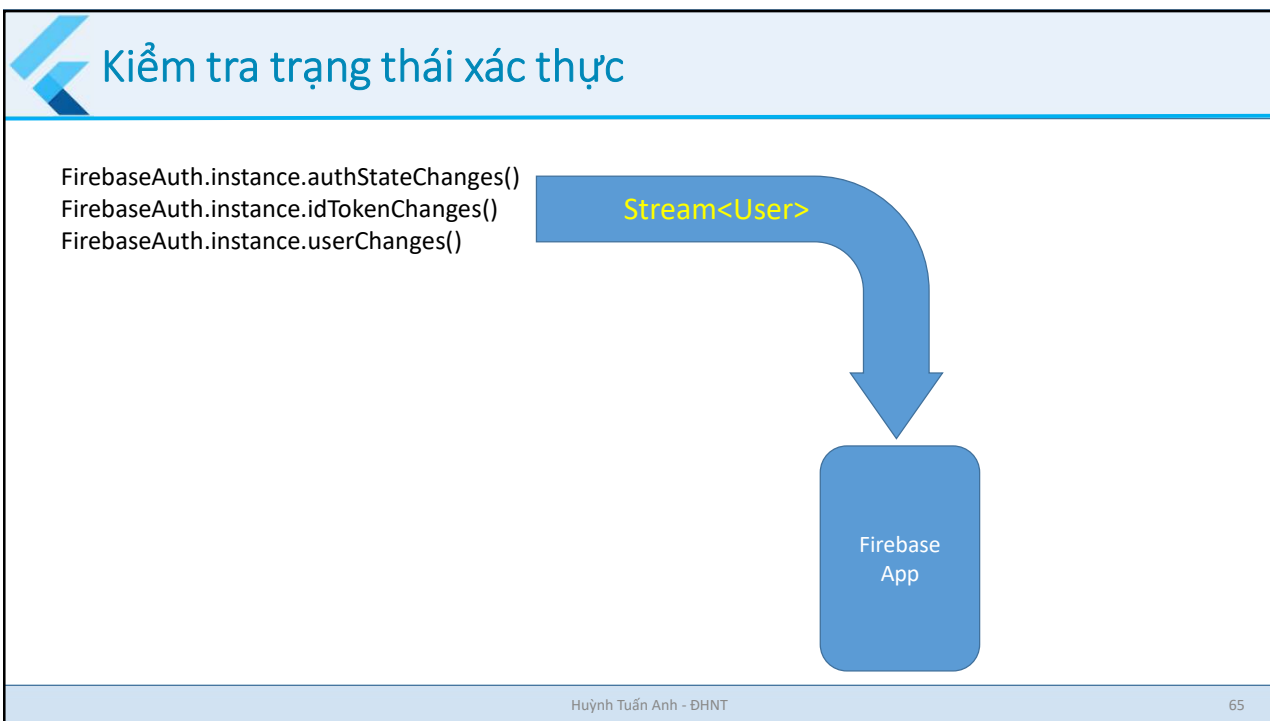
```
codeSent: (verificationId, forceResendingToken){
  verifyPhoneId = verificationId;
},
```

3. Sử dụng mã xác minh này và `smsCode` đã đăng ký với firebase để tạo đối tượng ủy nhiệm đăng nhập

```
PhoneAuthCredential credential = PhoneAuthProvider.credential(verificationId: verificationId, smsCode: smsCode);
await FirebaseAuth.instance.signInWithCredential(credential);
```

Trạng thái xác thực (Authentication state)

- Firebase Auth cung cấp nhiều phương pháp và tiện ích cho phép bạn tích hợp xác thực an toàn vào ứng dụng Flutter mới hoặc hiện có của mình. Trong nhiều trường hợp, bạn sẽ cần biết về trạng thái xác thực của người dùng, chẳng hạn như họ đã đăng nhập hay đã đăng xuất.
- Firebase Auth cho phép bạn đăng ký theo thời gian thực tới trạng thái này thông qua một Stream.
 - Sau khi được gọi, luồng cung cấp một sự kiện ngay lập tức về trạng thái xác thực hiện tại của người dùng và sau đó cung cấp các sự kiện tiếp theo bất cứ khi nào trạng thái xác thực thay đổi.



Các phương pháp kiểm tra trạng thái xác thực người dùng

Phương pháp kiểm tra trạng thái xác thực

FirestoreAuth.instance.	Sự kiện để xảy ra sự kiểm tra xác thực	
authStateChanges()		
idTokenChanges()	<ul style="list-style-type: none"> - Ngay sau khi người nghe đã được đăng ký. - Khi người dùng đã đăng nhập. - Khi người dùng hiện tại đã đăng xuất. 	<ul style="list-style-type: none"> - Sự kiện xác thực được phát ra khi có sự thay đổi token của user hiện tại - An ID token is force refreshed by calling: <code>FirebaseAuth.instance.currentUser.getIdTokenResult(true)</code>.
instance.userChanges()		<ul style="list-style-type: none"> - Sự kiện xác thực được phát ra khi có sự thay đổi token của user hiện tại

Huỳnh Tuấn Anh - ĐHNT 66

Kiểm tra trạng thái người dùng có đăng nhập hay không

- Sử dụng biến toàn cục *Login*. Thiết lập giá trị ban đầu login = false.
- Đăng nhập thành công: *Login* = true
- Đăng xuất khỏi ứng dụng *Login* = false
- Kiểm tra trong một trang của ứng dụng, có thể đặt trong initState như sau:

```
@override
void initState() {
  // TODO: implement initState
  FirebaseAuth.instance.userChanges().listen((User user) {
    if(user!=null)
      login = true;
    else
      login = false; // có thể đặt trong setState?
  });
  super.initState();
}
```

Sử dụng biến *Login* này trong các sự kiện xử lý của trang ứng dụng

Duy trì trạng thái xác thực

- Đảm bảo rằng trạng thái xác thực của người dùng của bạn được duy trì khi khởi động lại ứng dụng hoặc tải lại trang.
- Trên các nền tảng gốc như Android và iOS, hành vi này không thể cấu hình và trạng thái xác thực của người dùng sẽ tồn tại trên thiết bị giữa các lần khởi động lại ứng dụng.
 - Xóa dữ liệu đã lưu trong bộ nhớ cache của ứng dụng thông qua cài đặt thiết bị --> Xóa mọi trạng thái hiện có đang được lưu trữ.
- Trên nền tảng web, trạng thái xác thực của người dùng được lưu trữ trong bộ nhớ cục bộ. Có thể thay đổi hành vi mặc định này để chỉ duy trì trạng thái xác thực cho phiên hiện tại hoặc hoàn toàn không duy trì. Để định cấu hình các cài đặt này, hãy gọi phương thức setPersistence ().

Duy trì trạng thái xác thực

- Trên nền tảng web, trạng thái xác thực của người dùng được lưu trữ trong bộ nhớ cục bộ. Nếu cần, có thể thay đổi hành vi mặc định này để chỉ duy trì trạng thái xác thực cho phiên hiện tại hoặc hoàn toàn không duy trì bằng cách sử dụng phương thức `setPersistence()`.
 - Chú ý trên các nền tảng gốc (native) một ngoại lệ, `UnimplementedError`, sẽ được ném ra
- Ví dụ: Để tắt việc duy trì trạng thái xác thực


```
await FirebaseAuth.instance.setPersistence(Persistence.NONE);
```

Cloud Firestore - thư viện cloud_store

- https://pub.dev/documentation/cloud_firestore/latest/cloud_firestore/cloud_firestore-library.html
- <https://firebase.flutter.dev/docs/firestore/usage>
- CollectionReference
 - Một đối tượng `CollectionReference` có thể được sử dụng để thêm các documents, lấy các `DocumentReference` và truy vấn các documents
- DocumentReference
 - `DocumentReference` tham chiếu đến một document location trong cơ sở dữ liệu `FirebaseFirestore` và có thể được sử dụng để write, read hoặc listen tại location đó.
 - Tài liệu tại vị trí được tham chiếu có thể tồn tại hoặc không. Một `DocumentReference` cũng có thể được sử dụng để tạo một `CollectionReference` cho một bộ sưu tập con.

Thư viện cloud_store

▪ QuerySnapshot:

- Chứa kết quả câu truy vấn. Nó có thể chứa không hay nhiều đối tượng DocumentSnapshot.

▪ DocumentSnapshot

- Một DocumentSnapshot chứa dữ liệu đọc từ một document trong CSDL FirebaseFirestore.
- Dữ liệu có thể được trích xuất từ DocumentSnapshot bằng phương thức `data()`, trả về một Map<String, dynamic>

▪ QueryDocumentSnapshot: Lớp mở rộng từ DocumentSnapshot

- Một QueryDocumentSnapshot chứa dữ liệu được đọc từ tài liệu trong cơ sở dữ liệu FirebaseFirestore của bạn như một phần của truy vấn.

Thư viện cloud_store

- FieldValue: Giá trị Sentinel (lính canh) có thể được sử dụng khi ghi các document field với phương thức `set ()` hoặc `update ()`.

▪ Các *static* method:

- `delete()` → FieldValue
- `increment(num value)` → FieldValue
- `serverTimestamp()` → FieldValue
- `arrayRemove(List elements)` → FieldValue
- `arrayUnion(List elements)` → FieldValue

Sử dụng Cloud Firestore

- import:
 - `import 'package:cloud_firestore/cloud_firestore.dart';`
- Khởi tạo FlutterFire trước khi sử dụng:
 - `await Firebase.initializeApp();`
- Tạo một Firestore instance:
 - `FirebaseFirestore firestore = FirebaseFirestore.instance;`
 - Mặc định getter này sẽ trả về một default Firebase App (khi cài đặt FlutterFire trên hệ thống của bạn). Nếu muốn sử dụng một Firestore với một Firebase App thứ hai, sử dụng phương thức `instanceFor` :
 - `FirebaseApp secondaryApp = Firebase.app('SecondaryApp');`
 - `FirebaseFirestore firestore = FirebaseFirestore.instanceFor(app: secondaryApp);`

Huỳnh Tuấn Anh - ĐHNT

73

Sử dụng Cloud Firestore: Collections & Documents

- Để làm việc với một Collection (gần giống với bảng trong CSDL quan hệ), sử dụng một đối tượng [CollectionReference](#):
 - `CollectionReference users = FirebaseFirestore.instance.collection('users');`
- Thêm một Document vào một Collection: Sử dụng phương thức `add` của đối tượng `CollectionReference`

```
Future<void> addUser() {
  return users.add({
    'full_name': "Minh Thanh",
    'company': "ABC bakery",
    'age': 40
  }).then((value) => print("User Added"))
    .catchError((error) => print("Failed to add user: $error"));
}
```

Huỳnh Tuấn Anh - ĐHNT

74

Writting Data

- Thêm một Document mới vào một Collection: Sử dụng phương thức `add` với tham số là một `Map<String, dynamic>` trên đối tượng `CollectionReference`
 - Phương thức `add` trả về một đối tượng `Future<DocumentReference>`
 - Document được thêm vào có ID được Firestore sinh ra một cách tự động

```
Future<DocumentReference> addDocument() async{
  CollectionReference users = FirebaseFirestore.instance.
    collection('users');

  var doc= await users.add({
    'full_name': 'Minh Thanh',
    'company': 'ABC Bakery',
    'age': 40
  });
  return doc;
}
```

Writting Data

- Thêm một Document có Id do người dùng chỉ định: Sử dụng phương thức `set` của đối tượng `DocumentReference` với tham số là một `Map<String, dynamic>`
 - Nếu Document có Id đã tồn tại, document này sẽ bị ghi đè

```
Future<void> setDocument(String id) async{
  CollectionReference users = FirebaseFirestore.instance.
    collection('users');
  var doc= await users.doc(id).set({
    'full_name': 'Minh Thanh',
    'company': 'ABC Bakery',
    'age': 40
  });
  return doc;
}
```

Writing Data

- Update Document: Sử dụng phương thức `update` của đối tượng `DocumentReference` với đối số là một `Map<string, dynamic>`

```
Future<void> updateDocument(String id) async{
  var asset_image = await rootBundle.load('assets/images/sample.jpg');
  var avatar = asset_image.buffer.asUint8List();
  CollectionReference users = FirebaseFirestore.instance.
    collection('users');
  var doc= await users.doc(id).update({
    'phone': "12345678",
    'info.address.location': GeoPoint(53.483959, -2.244644),
    'info.avatar': Blob(avatar)
  });
}
```

Field values

- string: Up to 1,048,487 bytes (1 MiB - 89 bytes). Only the first 1,500 bytes of the UTF-8 representation are considered by queries.
- number: integer (64-bit signed), float (64-bit double precision)
- Boolean: true, false
- map: {key:value...}. Được sắp xếp theo key. VD: {a: "foo", b: "bar", c: "qux"}.
- array: mảng không chứa mảng khác. VD: [1, 2, 3, 1]
- null
- geopoint: By latitude, then longitude

Field values

- reference: By path elements (collection, document ID, collection, document ID...)
 - VD: projects/[PROJECT_ID]/databases/[DATABASE_ID]/documents/[DOCUMENT_PATH].
- Byte: Up to 1,048,487 bytes (1 MiB - 89 bytes). Only the first 1,500 bytes are considered by queries.
- Date and time: When stored in Cloud Firestore, precise only to microseconds; any additional precision is rounded down.

Delete document

1. Lấy đối tượng DocumentReference, docRef, của Document cần xóa
2. Gọi phương thức delete trên đối tượng docRef

```
Future<bool> deleteUser(String id) async{
    CollectionReference users = FirebaseFirestore.instance.
        collection('users');

    try {
        await users.doc(id).delete();
        return true;
    } catch(e) {
        return false;
    }
}
```


Delete field

▪ Gọi phương thức delete trên lớp FieldValue

```
Future<bool> deleteField() async{
  CollectionReference users = FirebaseFirestore.instance.
    collection('users');

  try{
    await users.doc('Id123').update({
      'phone':FieldValue.delete(),
    });
    return true;
  }catch(e){
    return false;
  }
}
```

Transactions

- Transaction là một cách để đảm bảo rằng hoạt động ghi chỉ xảy ra bằng cách sử dụng dữ liệu mới nhất có sẵn trên máy chủ. Các giao dịch không bao giờ áp dụng ghi một phần và hoạt động ghi thực hiện khi giao dịch kết thúc thành công.
- Khi sử dụng các transaction, hãy lưu ý rằng:
 - Thao tác đọc phải đến trước khi thao tác ghi
 - Giao dịch sẽ không thành công khi client offline, họ không thể sử dụng dữ liệu trong bộ nhớ cache.
- Bạn không nên trực tiếp sửa đổi trạng thái ứng dụng bên trong giao dịch, vì trình xử lý có thể thực hiện nhiều lần. Thay vào đó, bạn nên trả về một giá trị ở cuối trình xử lý, cập nhật trạng thái ứng dụng khi giao dịch đã hoàn tất.

Transaction

```
void transaction(){
    // Tạo một tham chiếu đến document mà transaction sẽ sử dụng.
    DocumentReference documentReference = FirebaseFirestore.instance
        .collection('users').doc('id123');
    FirebaseFirestore.instance.runTransaction((transaction) async{
        DocumentSnapshot snapshot = await transaction.get(documentReference);
        if (!snapshot.exists) {
            throw Exception("User does not exist!");
        }
        int newFollowerCount = snapshot.data()['followers'] + 1;
        transaction.update(documentReference, {'followers': newFollowerCount});
        // Return giá trị khi giao dịch hoàn tất
        return newFollowerCount;
    }).then((value) => print("Follower count updated to $value"))
        .catchError((error) => print("Failed to update user followers: $error"));
}
```

Huỳnh Tuấn Anh - ĐHNT

83

Batch write

- Firestore cho phép bạn thực hiện nhiều hoạt động ghi dưới dạng một lô (batch) duy nhất có thể chứa các kết hợp bất kỳ của các hoạt động: *set*, *update*, *delete*.
- Thực hiện:
 - Tạo một thể hiện của WriteBatch thông qua phương thức batch()
 - Thực hiện các hoạt động trên batch.
 - Gọi commit() trên batch khi các hoạt động trên batch sẵn sàng. Phương thức commit đảm bảo các hoạt động ghi trong batch được được xem là một đơn vị nguyên tử duy nhất. Ngoài ra commit còn ngăn không cho bất kỳ các hoạt động nào ở tương lai được thêm vào.

Huỳnh Tuấn Anh - ĐHNT

84

Batch Write

```
Future<void> batchDelete() async{
    CollectionReference users = FirebaseFirestore.
        instance.collection('users');
    WriteBatch batch = FirebaseFirestore.instance.batch();
    QuerySnapshot querySnapshot = await users.get();
    querySnapshot.docs.forEach((doc) {
        batch.delete(doc.reference);
    });
    return batch.commit();
}
```

Huỳnh Tuấn Anh - ĐHNT

85

Truy vấn dữ liệu: Document & Query Snapshots

- Khi thực hiện một truy vấn, Firestore trả về một QuerySnapshot hoặc một DocumentSnapshot.
- **QuerySnapshot**: QuerySnapshot được trả về từ truy vấn một Collection và cho phép bạn kiểm tra Collection, chẳng hạn như có bao nhiêu Document tồn tại bên trong nó, cấp quyền truy cập vào các Document trong Document, xem bất kỳ thay đổi nào kể từ query cuối cùng và nhiều hơn nữa.
- **DocumentSnapshot**: được trả về từ một query hoặc bằng cách truy cập trực tiếp vào Document. Ngay cả khi không có Document nào tồn tại trong cơ sở dữ liệu, một snapshot sẽ luôn được trả về.

Huỳnh Tuấn Anh - ĐHNT

86

QuerySnapshot

- Truy vấn Collection user và trả về đối tượng Future<QuerySnapshot>, duyệt qua các Document trong thuộc tính *docs* của đối tượng này

```
void querySnapshot() async{
  var snapshot = await FirebaseFirestore.
    instance.collection('users').get();
  snapshot.docs.forEach((doc)=>
    print(doc['first_name']));
}
```

Querying

- Filtering: Để lọc ra các Documents trong một Collection thỏa mãn một điều kiện nào đó, ta sử dụng phương thức where:

```
void filtering() async{
  var snapshot = await FirebaseFirestore.instance.
    collection('users').where('age', isGreaterThan: 20).
    orderBy('age').limit(5).get();
  snapshot.docs.forEach((doc)=>
    print(doc['last_name']));
}
```

Ví dụ: Truy vấn và hiển thị một document

```
Widget build(BuildContext context) {
  CollectionReference users = FirebaseFirestore.instance.collection('users');
  return FutureBuilder<DocumentSnapshot>(
    future: users.doc('doc123').get(),
    builder: (context, snapshot){
      if (snapshot.hasError) {
        return Text("Something went wrong");
      }
      if (snapshot.connectionState == ConnectionState.done) {
        Map<String, dynamic> data = snapshot.data.data();
        return Text("Full Name: ${data['full_name']} ${data['last_name']}");
      }
      return Text("loading");
    },
  );
}
```

Huỳnh Tuấn Anh - ĐHNT

89

DocumentSnapshot

- Truy vấn một Document trong Collection users và trả về đối tượng Future<DocumentSnapshot>

```
void documentSnapshot(String docId) async{
  var snapshot = await FirebaseFirestore.instance
    .collection('users').doc(docId).get();
  if(snapshot.exists) {
    Map<String, dynamic> doc = snapshot.data();
    print(doc['last_name']);
  }
}
```

- Phương thức `get(dynamic field)` có thể trả về các field lồng bên trong DocumentSnapshot:

```
dynamic last_name = snapshot.get('last_name');
```

Huỳnh Tuấn Anh - ĐHNT

90

Collections & Documents: Read data

- One-time Read: Để đọc một collection hay một document một lần: Gọi phương thức `Query.get` hay `DocumentReference.get`
 - Sử dụng `FutureBuilder Widget` để hỗ trợ quản lý trạng thái của request trên UI.
- Realtime changes: Sử dụng `Stream` để kết nối giữa client và `Firestore`
 - `FlutterFire` cung cấp sự hỗ trợ xử lý các thay đổi trong thời gian thực đối với Collections và Documents. Một sự kiện mới được cung cấp theo yêu cầu ban đầu và mọi thay đổi tiếp theo đối với Collections /Documents mỗi khi xảy ra sự thay đổi (thêm, xóa, sửa) sẽ được tự động cập nhật trên các widget trong `StreamBuilder` (sử dụng `Stream` này).
 - Sử dụng phương thức `snapshots()` để lấy data `Stream`

```
Stream<QuerySnapshot> collectionStream = FirebaseFirestore.instance.  
    collection('users').snapshots();  
Stream<DocumentSnapshot> documentStream = FirebaseFirestore.instance.  
    collection('users').doc('ABC123').snapshots();
```

Huỳnh Tuấn Anh - ĐHNT

91

Stream<QuerySnapshot>

- Sử dụng `Stream<QuerySnapshot>` kết hợp với `StreamBuilder Widget` để bind kết quả một truy vấn với các Widget được xây dựng trong `StreamBuilder`.
 - Khi dữ liệu liên quan đến `QuerySnapshot` thay đổi, các Widget sẽ tự động được rebuild để hiển thị dữ liệu mới.
- Ví dụ:
 - `FirebaseFirestore.instance.collection("users").snapshots()`: Trả về một `Stream<QuerySnapshot>`, mỗi `QuerySnapshot` chứa toàn bộ các Document của Collection `users`.
 - `FirebaseFirestore.instance.collection("users").where("age",isGreaterThan: 30).snapshots() --> Stream<QuerySnapshot>;`

Huỳnh Tuấn Anh - ĐHNT

92

Stream<DocumentSnapshot>

- Tương tự như Stream<QuerySnapshot> nhưng chỉ làm việc trên mỗi Document.
- Kết hợp với StreamBuilder để bind một Document với các widget nhằm tự động rebuild để hiển thị dữ liệu khi dữ liệu thay đổi.
- Ví dụ:
 - `Firestore.instance.collection("journals").doc("abc").snapshots(); --> Stream<DocumentSnapshot>`

Ví dụ

```
Widget build(BuildContext context) {
  CollectionReference users = Firestore.instance.collection('users');
  return StreamBuilder<QuerySnapshot>(
    stream: users.snapshots(),
    builder: (context, snapshot) {
      if (snapshot.hasError) {return Text('Something went wrong');}
      if (snapshot.connectionState == ConnectionState.waiting){
        return Text("Loading");
      }
      return new ListView(
        children: snapshot.data.docs.map((DocumentSnapshot document){
          return new ListTile(
            title: new Text(document.data() ['full_name']),
            subtitle: new Text(document.data() ['company']),
          );
        }).toList(),
      );
    },
  );
}
```

listening metadata change

- Mặc định, listeners không cập nhật nếu có sự thay đổi chỉ ảnh hưởng đến siêu dữ liệu. Nếu bạn muốn nhận các sự kiện khi siêu dữ liệu của document hay query thay đổi, bạn có thể sử dụng thuộc tính `includeMetadataChanges` cho phương thức `snapshots`:

```
FirestoreFirestore.instance
  .collection('users')
  .snapshots(includeMetadataChanges: true);
```

Access Data Offline

- Firestore cung cấp khả năng ngoại tuyến ra bên ngoài. Khi đọc và ghi dữ liệu, Firestore sử dụng *cơ sở dữ liệu cục bộ tự động đồng bộ hóa với máy chủ*. Chức năng Cloud Firestore vẫn tiếp tục khi người dùng ngoại tuyến và tự động xử lý việc di chuyển dữ liệu (migration) khi họ lấy lại kết nối.
- Chức năng này được bật theo mặc định, tuy nhiên nó có thể bị tắt nếu cần. Các cài đặt phải được đặt trước khi thực hiện bất kỳ tương tác nào với Firestore:

```
//Web
await FirestoreFirestore.instance.enablePersistence();
// All other platforms.
FirestoreFirestore.instance.settings =
  Settings(persistenceEnabled: false);
```


Access Data Offline

- Nếu bạn muốn xóa mọi dữ liệu duy trì (persisted data), bạn có thể gọi phương thức `clearPersistence()`.

```
await FirebaseFirestore.instance.clearPersistence();
```

- Gọi các phương thức để cập nhật setting hoặc xóa persistence phải được thực hiện trước khi sử dụng Firestore. Nếu được gọi sau đó, chúng sẽ có hiệu lực đối với yêu cầu tiếp theo của Firestore (ví dụ: khởi động lại ứng dụng).

Configure Cache Size

- Khi tính năng persistence được bật, Firestore lưu trữ mọi tài liệu để truy cập ngoại tuyến. Sau khi vượt quá kích thước bộ nhớ cache, Firestore sẽ cố gắng xóa dữ liệu cũ hơn, không sử dụng. Bạn có thể định cấu hình các kích thước bộ nhớ cache khác nhau hoặc vô hiệu hóa quá trình xóa:

```
FirebaseFirestore.instance.settings =  
    Settings(cacheSizeBytes: Settings.CACHE_SIZE_UNLIMITED);
```

- Disable and Enable Network Access

```
await FirebaseFirestore.instance.disableNetwork();
```

```
await FirebaseFirestore.instance.enableNetwork();
```

Tham khảo

- Flutter – Firebase:
 - <https://flutter.dev/docs/development/data-and-backend/firebase>

Huỳnh Tuấn Anh - ĐHNT 99

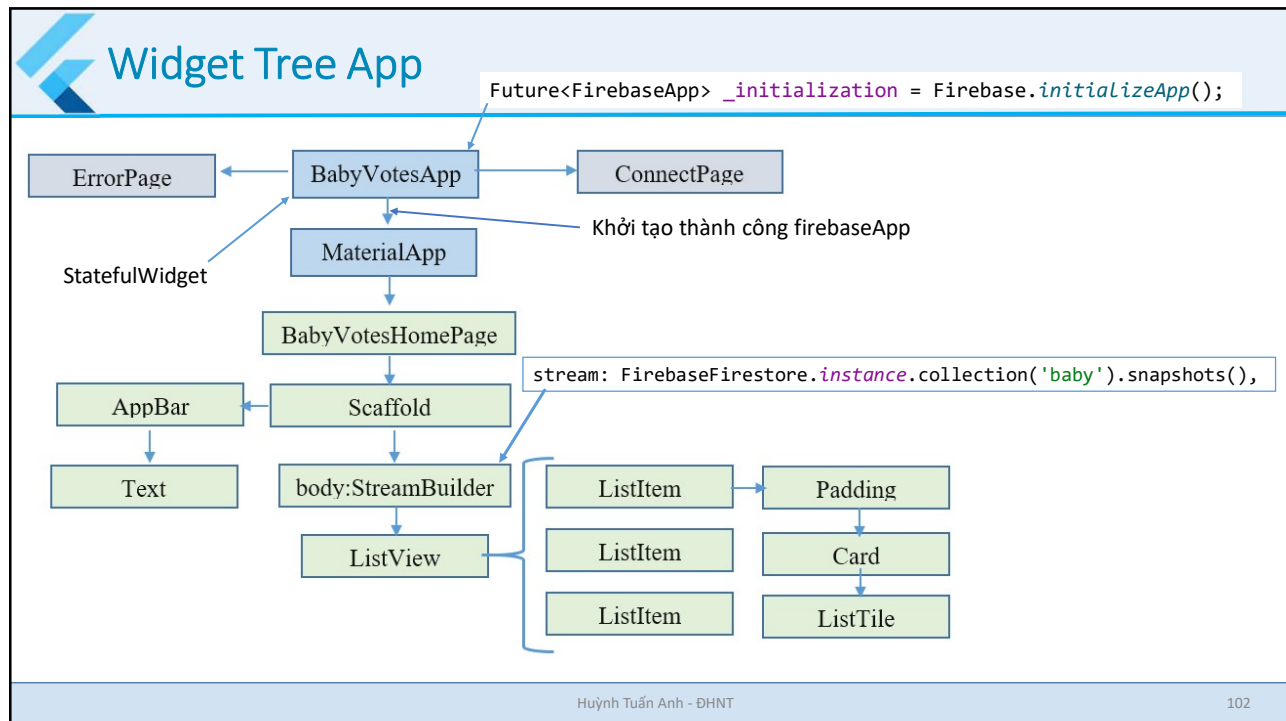
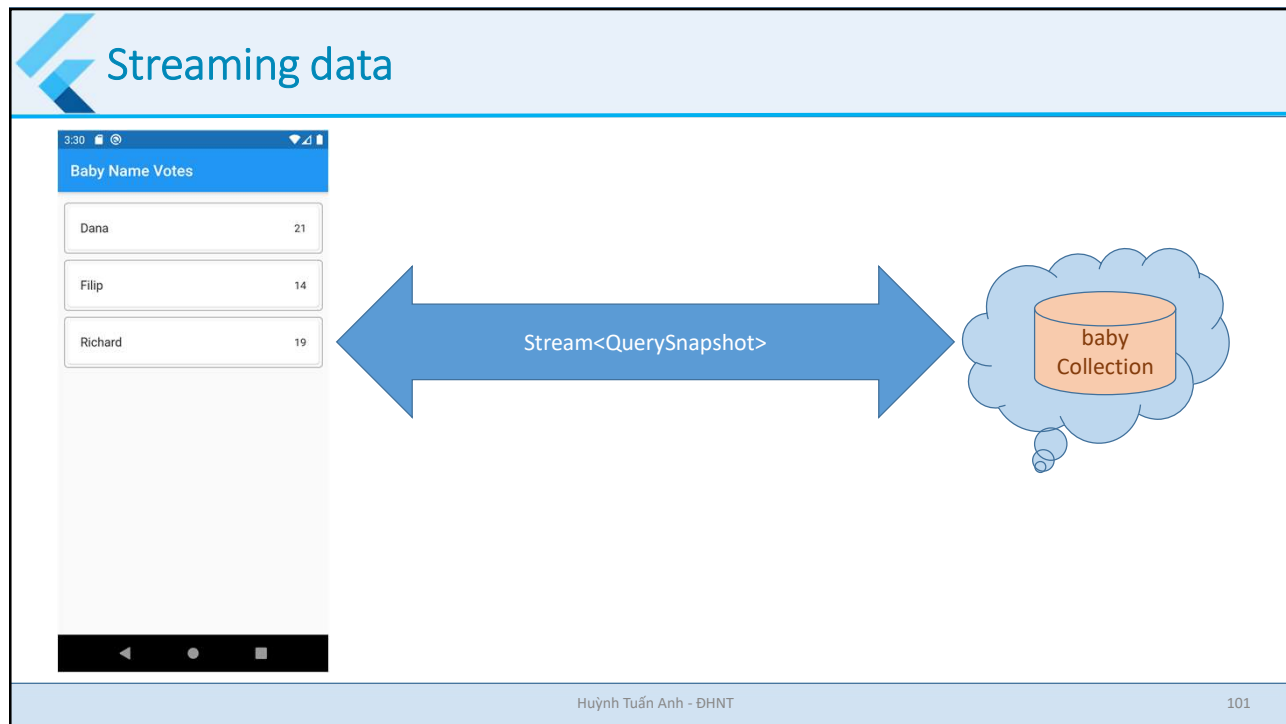
Ví dụ

- Thiết kế ứng dụng Baby Name Votes
 - Dữ liệu được lưu trữ trên Firebase Firestore
 - Khi người sử dụng click vào tên trong danh sách số lượt vote cho tên đó sẽ tự động tăng lên một và dữ liệu sẽ lưu vào Firestore



The screenshot shows a mobile application interface with a blue header titled "Baby Name Votes". Below the header, there is a list of three items, each consisting of a name in a white box and a vote count in a blue box. The items are: Dana with 21 votes, Filip with 14 votes, and Richard with 19 votes. The app is running on a device with a status bar at the top showing 3:30 and various icons, and an Android navigation bar at the bottom.

Huỳnh Tuấn Anh - ĐHNT 100



Record class

```
class Record{
  final String name;
  final int votes;
  final DocumentReference reference;
  Record.fromMap(Map<String, dynamic> map, {this.reference})
    : assert(map['name']!=null),
      assert(map['votes']!=null),
      name=map['name'],
      votes=map['votes'];

  Record.fromSnapshot(DocumentSnapshot snapshot)
    : this.fromMap(snapshot.data(), reference:snapshot.reference);

  @override
  String toString() => "Record<$name:$votes>";
}
```

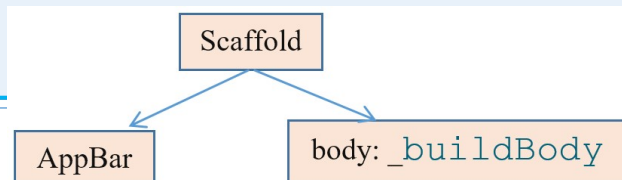
Tham chiếu đến một Document trong Collection trên CSDL Firestore

Huỳnh Tuấn Anh - ĐHNT

103

BabyVotesHomePage

- Là một Stateful widget



```
Widget _buildBody(BuildContext context){
  //Firebase.initializeApp();
  return StreamBuilder<QuerySnapshot>(
    stream: FirebaseFirestore.instance.collection('baby').snapshots(),
    builder: (context, snapshot) {
      if(snapshot.hasError)
        return _buildError(context);
      else
        if(snapshot.hasData)
          return _buildList(context, snapshot.data.docs);
        return Center(child: CircularProgressIndicator());
    },
  );
}
```

Stream<QuerySnapshot>

List<QueryDocumentSnapshot>

Huỳnh Tuấn Anh - ĐHNT

104

```
Widget _buildList(BuildContext context, List<DocumentSnapshot> docs) {
  return ListView(
    padding: EdgeInsets.only(top: 10),
    children: List.generate(docs.length,
      (index) => _buildListItem(context, docs[index])),
  );
}
```

↓
DocumentSnapshot

ListItem

```
_buildListItem(BuildContext context, DocumentSnapshot data) {
  final Record record = Record.fromSnapshot(data);
  return Padding(
    key: ValueKey(record.name),
    padding: const EdgeInsets.symmetric(horizontal: 8.0, vertical: 4.0),
    child: Container(
      decoration: BoxDecoration(
        border: Border.all(color: Colors.grey),
        borderRadius: BorderRadius.circular(5.0),
      ),
      child: Card(
        child: ListTile(
          title: Text(record.name),
          trailing: Text(record.votes.toString()),
          onTap: () => record.reference.update({'votes': FieldValue.increment(1)}),
        ),
      ),
    ),
  );
}
```

Richard

19

Firebase.initializeApp()

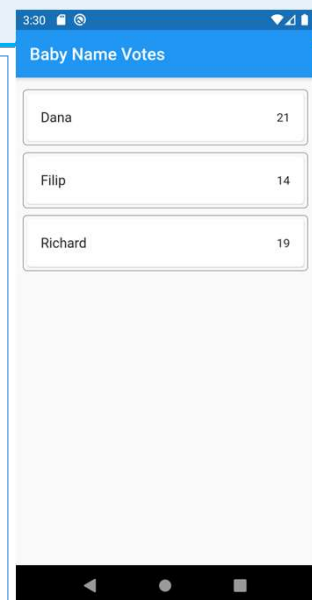
```
class BabyVotesPage extends StatelessWidget {
  final Future<FirebaseApp> _initialization = Firebase.initializeApp();
  @override
  Widget build(BuildContext context) {
    return FutureBuilder(
      future: _initialization,
      builder:(context, snapshot){
        if(snapshot.connectionState==ConnectionState.done)
          return BabyVotesHomePage();
        else
          if(snapshot.hasError)
            return ErrorPage('Lỗi kết nối!');
            return ConnectPage("Đang kết nối");
      },);
  }
}
```

Huỳnh Tuấn Anh - ĐHNT

107

Bài tập

- Thiết kế ứng dụng Baby Name Votes sử dụng Firebase và Provider

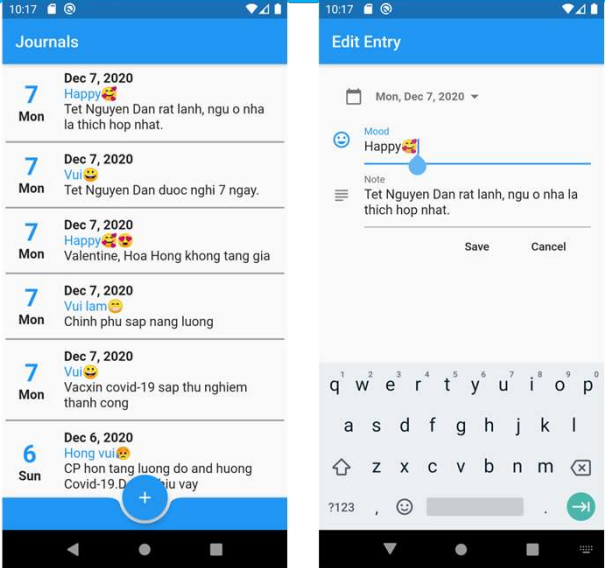


Huỳnh Tuấn Anh - ĐHNT

108

Bài tập

- Yêu cầu: Thiết kế ứng dụng với version 3:
 - Dữ liệu được lưu trữ trên firebase firestore.
 - Sử dụng Provider.

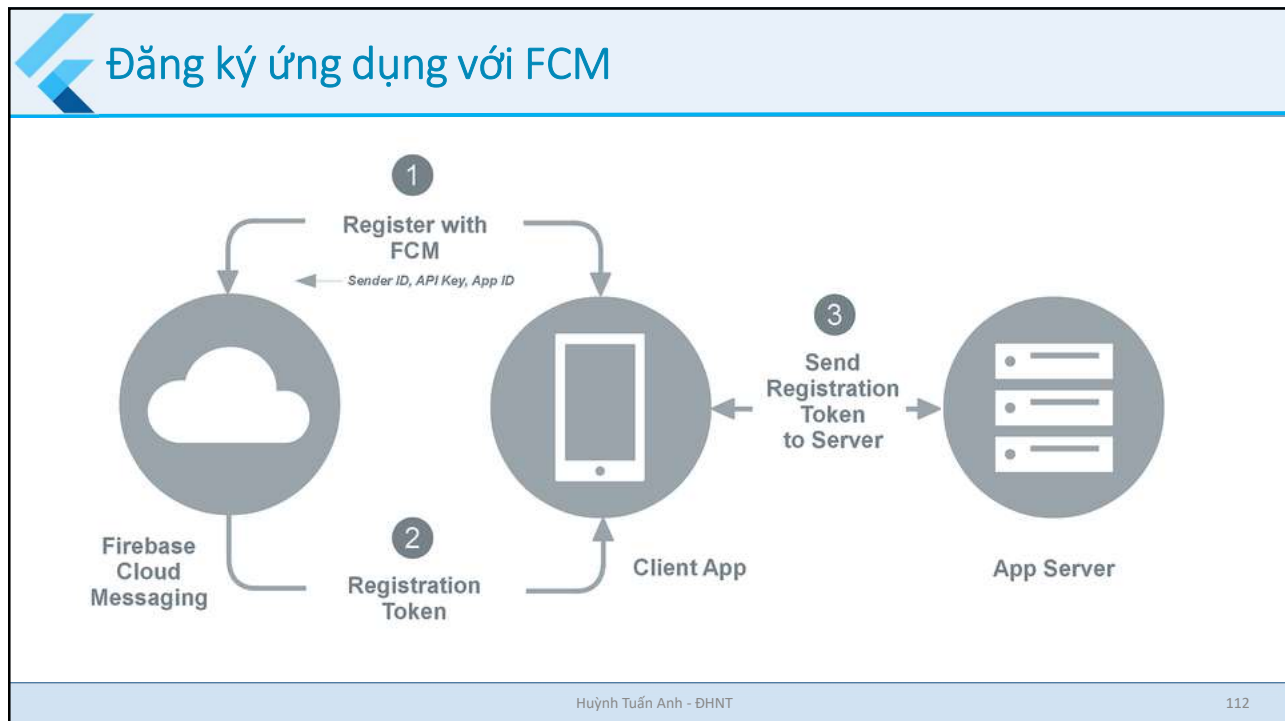
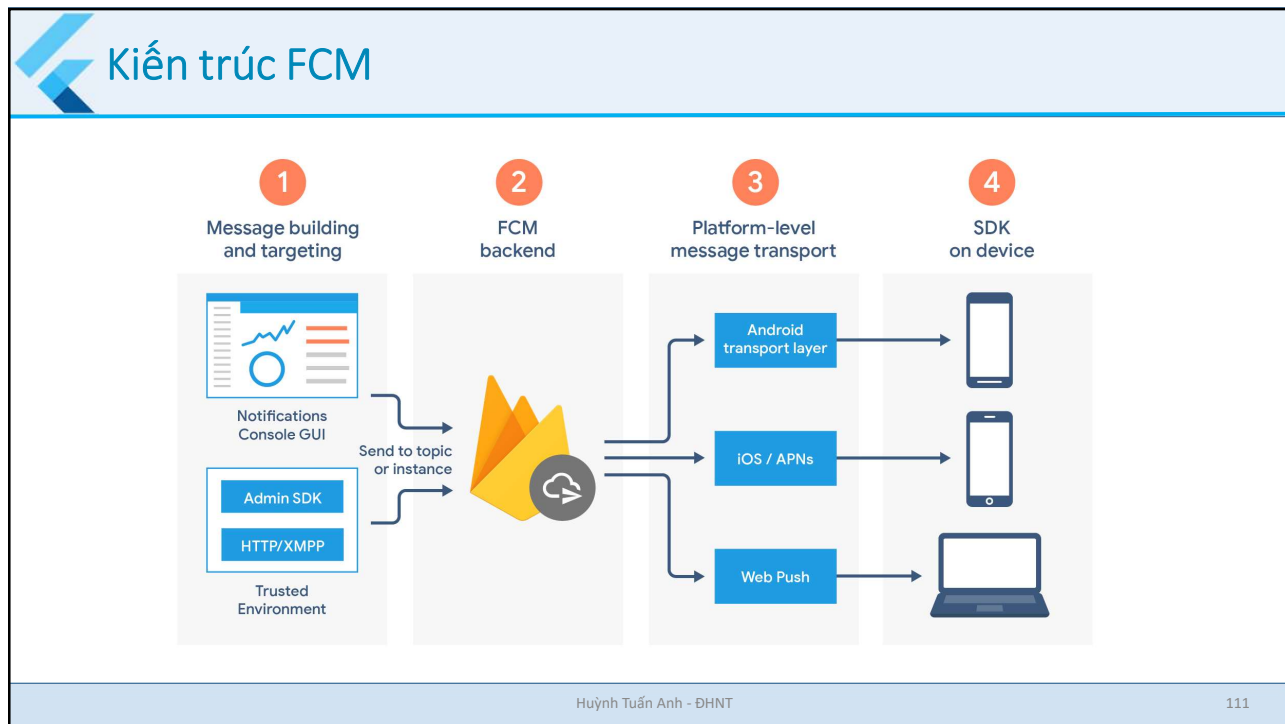


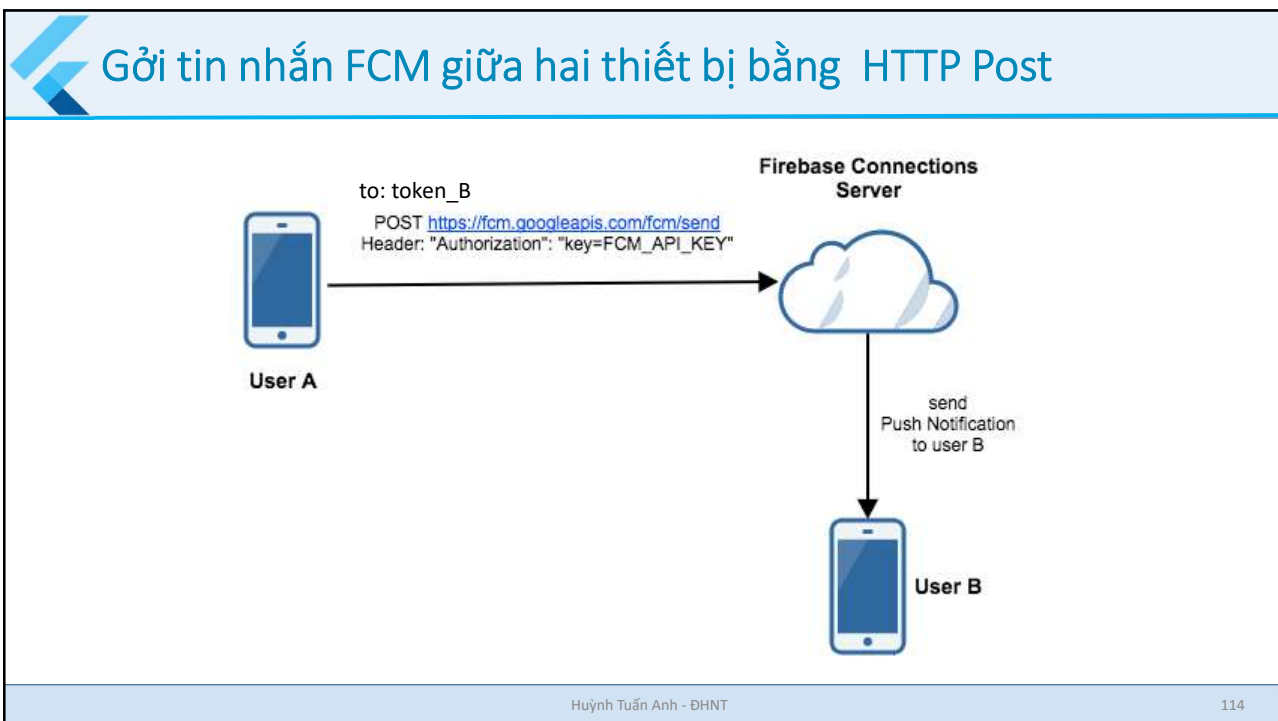
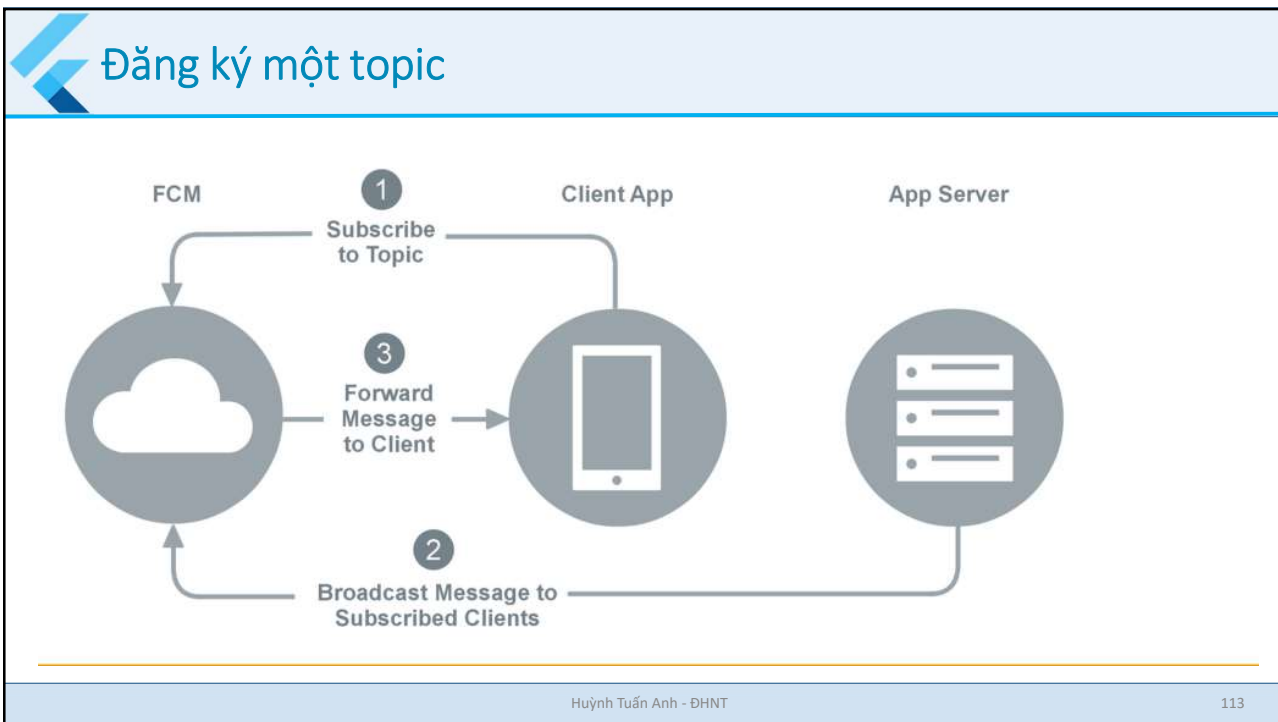
Huỳnh Tuấn Anh - ĐHNT 109

Firestore Cloud Messaging (FCM)

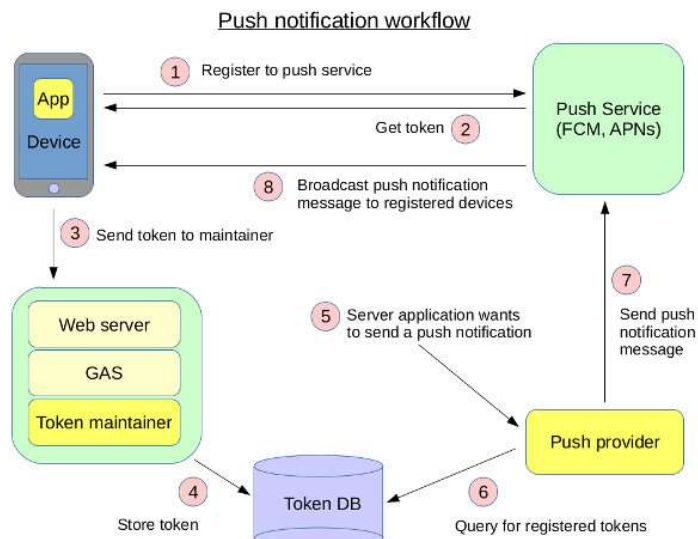
- FCM là một giải pháp nhắn tin đa nền tảng cho phép các ứng dụng gửi tin nhắn miễn phí một cách đáng tin cậy.
- Sử dụng FCM, bạn có thể thông báo cho ứng dụng khách rằng có email mới hoặc dữ liệu khác có sẵn để đồng bộ hóa. Bạn có thể gửi tin nhắn thông báo để tăng mức độ tương tác lại và giữ chân người dùng. Đối với các trường hợp sử dụng như nhắn tin trò chuyện, tin nhắn có thể truyền dung lượng lên đến 4 KB cho một ứng dụng khách.
- Các thư viện hỗ trợ:
 - firebase_messaging: Thư viện hỗ trợ việc gửi thông báo thông qua FCM
 - flutter_local_notifications: Thư viện dùng để hiển thị các thông báo

Huỳnh Tuấn Anh - ĐHNT 110





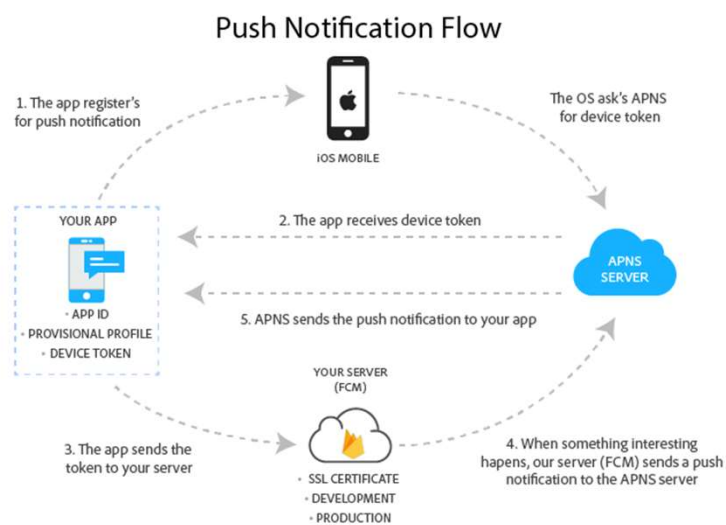
Push Notification Flow



Huỳnh Tuấn Anh - ĐHNT

115

Push Notification Flow in iOS



Huỳnh Tuấn Anh - ĐHNT

116

Cloud Messaging

- import thư viện:
 - `import 'package:firebase_messaging/firebase_messaging.dart';`
- Khởi tạo FlutterFire trước khi sử dụng: `await Firebase.initializeApp();`
- Khởi tạo đối tượng FirebaseMessaging
 - `FirebaseMessaging messaging = FirebaseMessaging.instance;`

Receiving messages

- Cloud Messaging package kết nối các ứng dụng với dịch vụ Nhắn tin qua đám mây của Firebase (FCM). Các trường hợp phổ biến sử dụng thông báo là:
 - Hiển thị thông báo
 - Đồng bộ hóa dữ liệu tin nhắn một cách "âm thầm" trên thiết bị
 - Cập nhật lại giao diện người dùng
- Ba trạng thái người dùng ứng với các cách xử lý khác nhau của tin nhắn
 - Foreground: Ứng dụng đang chạy, đang xem và đang sử dụng
 - Background: Ứng dụng đang chạy tuy nhiên nó không hiện hoạt (người dùng nhấn Home)
 - Terminate: Thiết bị bị khóa hay ứng dụng không chạy
- Để nhận được tin nhắn, ứng dụng phải được mở ít nhất một lần (từ khi cài đặt, khi buộc dừng)

Requesting permission (Apple & Web)

- Android App: Không bắt buộc phải yêu cầu cấp quyền
- On iOS, macOS & web: Yêu cầu phải cấp quyền trước khi nhận tin nhắn trên thiết bị:

```
NotificationSettings settings = await FirebaseMessaging.  
instance.requestPermission(  
  alert: true,  
  announcement: false,  
  badge: true,  
  carPlay: false,  
  criticalAlert: false,  
  provisional: false,  
  sound: true,  
);
```

NotificationSettings

- Thuộc tính `settings.authorizationStatus` có thể trả về một giá trị được sử dụng để xác định quyết định tổng thể của người dùng:
 - `authorized`: Người dùng đã cấp quyền.
 - `denied`: Người dùng từ chối cấp quyền
 - `notDetermined`: Người dùng vẫn chưa chọn cấp quyền hay không.
 - `provisional`: Người dùng đã cấp quyền tạm thời
- Trên Android `authorizationStatus` luôn có giá trị là `authorized`

Các dạng message

- Notification only message: payload chứa thuộc tính "notification", thuộc tính này sẽ được sử dụng để hiển thị thông báo hiển thị cho người dùng.
- Data only message: Còn được gọi là "thông báo im lặng", payload này chứa các cặp key / value tùy chỉnh trong thuộc tính "data" có thể được sử dụng theo các cách phù hợp. Những tin nhắn này được cho là có "mức độ ưu tiên thấp"
- Notification & Data messages: Payloads chứa cả hai thuộc tính "notification" và "data".

Các callback để xử lý các dạng message

	Foreground	Background	Terminated
Notification	onMessage	onBackgroundMessage	onBackgroundMessage
Data	onMessage	onBackgroundMessage (<i>see below</i>)	onBackgroundMessage (<i>see below</i>)
Notification & Data	onMessage	onBackgroundMessage	onBackgroundMessage



Các callback để xử lý các dạng message

- Data message only được các thiết bị coi là mức độ ưu tiên thấp khi ứng dụng của bạn ở chế độ nền hoặc kết thúc và sẽ bị bỏ qua. Tuy nhiên, bạn có thể tăng mức độ ưu tiên một cách rõ ràng bằng cách gửi các thuộc tính bổ sung trên tải trọng FCM:
 - Trên Android, đặt trường "priority" thành "high"
 - Trên Apple (iOS & macOS), đặt trường "content-available" thành "true"



Tin nhắn thông báo

```
String notificationMessage(){
    var message = {
        "to": "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
        'priority': 'high',
        "notification": {
            "title": "Portugal vs. Denmark",
            "body": "great match!"
        },
    },
};
return jsonEncode(message);
}
```



Tin nhắn dữ liệu

```
String dataMessage(){
    var message = {
        "to": "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
        'priority': 'high',
        "data": {
            "Nick" : "Mario",
            "body" : "great match!",
            "Room" : "PortugalVSDenmark"
        }
    };
    return jsonEncode(message);
}
```



Tin nhắn thông báo và dữ liệu

```
String notificationDataMessage(){
    var message = {
        "to": "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
        'priority': 'high',
        "notification": {
            "title": "Portugal vs. Denmark",
            "body": "great match!"
        },
        "data": {
            "Nick" : "Mario",
            "body" : "great match!",
            "Room" : "PortugalVSDenmark"
        }
    };
    return jsonEncode(message);
}
```

Tin nhắn gửi tới một Topic

```
String notificationDataMessageTopic(String topic){
    var message = {
        'to': '/topics/$topic',
        'priority': 'high',
        "notification":{
            "title":"Portugal vs. Denmark",
            "body":"great match!"
        },
        "data":{
            "Nick" : "Mario",
            "body" : "great match!",
            "Room" : "PortugalVSDenmark"
        }
    };
    return jsonEncode(message);
}
```

Đăng ký/hủy đăng ký nhận tin nhắn từ một topic

- Những thiết bị đăng ký với topic sẽ nhận tin nhắn gửi trong topic đó
- Đăng ký topic:

```
await FirebaseMessaging.instance.subscribeToTopic(topic);
```

- Hủy đăng ký topic:

```
await FirebaseMessaging.instance.unsubscribeFromTopic(topic);
```

- topic: chuỗi, tên của topic

Gửi tin nhắn bằng HTTP Post

```
try {
    await http.post(
        Uri.parse('https://fcm.googleapis.com/fcm/send'),
        headers: <String, String>{
            'Content-Type': 'application/json; charset=UTF-8',
            "Authorization" : authorization_key,
        },
        body: notificationDataMessageTopic("test_fcm"),
    );
    print('FCM request for device sent!');
} catch (e) {
    print(e);
}
```

Cloud Messaging Server key
"key=AAAAKQ09..."

Authorization_key: Token của server key được lấy tại thẻ Cloud Messaging trong Project setting trên Firebase Console

Nhận tin nhắn khi app ở foreground

```
RemoteMessage _remoteMessage;
@override
void initState() {
    // TODO: implement initState
    super.initState();
    FirebaseMessaging.onMessage.listen((newMessage) {
        setState(() {
            _remoteMessage = newMessage;
        });
    });
}
```

```
String title = _remoteMessage.notification.title; // Tiêu đề thông báo
String body = _remoteMessage.notification.body; // Nội dung thông báo
Map<String, dynamic> data = _remoteMessage.data; // Data của tin nhắn
```

Nhận tin nhắn khi ứng dụng ở background

1. Định nghĩa một callback ở mức toàn cục để xử lý thông báo FCM ở khi app ở background

```
Future<void> _firebaseMessagingBackgroundHandler(RemoteMessage message) async {
  // If you're going to use other Firebase services in the background, such as Firestore,
  // make sure you call `initializeApp` before using other Firebase services.
  await Firebase.initializeApp();
  print('Handling a background message ${message.messageId}');
}
```

2. Khai báo một channel cho các cập nhật thông báo

```
AndroidNotificationChannel channel;
```

3. Khai báo FlutterLocalNotificationsPlugin cho các thông báo local trong ứng dụng, không bắt buộc với các thông báo FCM

```
FlutterLocalNotificationsPlugin flutterLocalNotificationsPlugin;
```

Huỳnh Tuấn Anh - ĐHNT

131

Nhận tin nhắn khi ứng dụng ở background

4. Trong main, khai báo callback onBackgroundMessage để hiển thị các thông báo FCM gửi đến khi app ở background

```
await Firebase.initializeApp();
FirebaseMessaging.onBackgroundMessage(_firebaseMessagingBackgroundHandler);
```

5. Trong main, khai báo kênh thông báo (đối với android)

```
FirebaseMessaging.onBackgroundMessage(_firebaseMessagingBackgroundHandler);
if (!kIsWeb) {
  channel = const AndroidNotificationChannel(
    'high_importance_channel', // id
    'High Importance Notifications', // title
    'This channel is used for important notifications.', // description
    importance: Importance.max,
  );
}
```

Huỳnh Tuấn Anh - ĐHNT

132

Nhận tin nhắn khi ứng dụng ở background

5. Trong main, tạo một kênh thông báo (đối với android), kênh này sẽ được sử dụng trong tập AndroidManifest.xml để ghi đè kênh FCM mặc định để bật thông báo cập nhật

```
flutterLocalNotificationsPlugin = FlutterLocalNotificationsPlugin();
await flutterLocalNotificationsPlugin
    .resolvePlatformSpecificImplementation<
        AndroidFlutterLocalNotificationsPlugin>()?.createNotificationChannel(channel);
```

6. Cập nhật các tùy chọn hiển thị thông báo foreground của iOS để cho phép cập nhật thông báo.

```
await FirebaseMessaging.instance.
    setForegroundNotificationPresentationOptions(
        alert: true,
        badge: true,
        sound: true,
    );
```

Hiển thị thông báo trên thanh trạng thái

Sử dụng plugin flutter_local_notification để hiển thị thông báo trên thanh trạng thái

```
FirebaseMessaging.onMessage.listen((RemoteMessage message) {
  RemoteNotification notification = message.notification;
  AndroidNotification android = message.notification?.android;
  if (notification != null && android != null && !kIsWeb) {
    flutterLocalNotificationsPlugin.show(
      notification.hashCode,
      notification.title,
      notification.body,
      notificationDetail
    );
  }
  print("plugin: Notification show");
});
```

```
notificationDetail= NotificationDetails(
  android: AndroidNotificationDetails(
    channel.id,
    channel.name,
    channel.description,
    // TODO add a proper drawable resource to android, for now using
    // one that already exists in example app.
    icon: 'launch_background',
    importance: Importance.max,
    priority: Priority.max,
    showWhen: false,
  ),
);
```

Xử lý khi nhấn vào thông báo trên thanh trạng thái

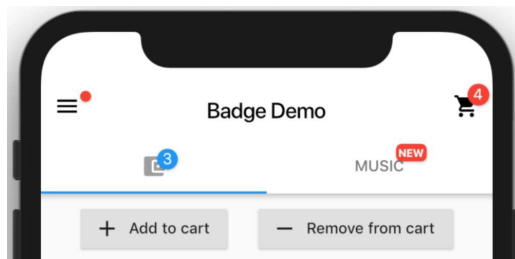
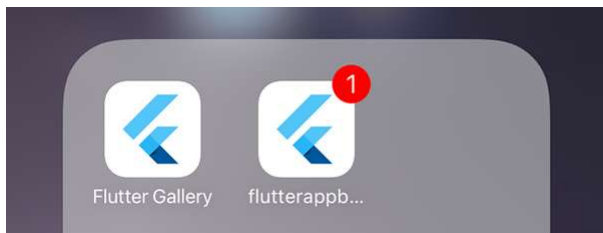
```
FirebaseMessaging.onMessageOpenedApp.listen((RemoteMessage message) {
  print('A new onMessageOpenedApp event was published!');
  ///Mã lệnh xử lý ở đây
  ///thường là mở một page để hiển thị chi tiết hơn
  ///ví dụ
  Navigator.push(context, MaterialPageRoute(
    builder: (context) =>
      MessageView(message: message, openedApplication: true,)));
});
```

Chú ý:

Callback `onMessageOpenApp` chỉ được gọi khi người dùng nhấn vào thông báo do FCM gửi đến thanh trạng thái. Nếu thông báo được hiển thị bởi `flutter_local_notification` thì callback này sẽ không được gọi

Cách tiếp cận khác

- Một số thiết bị được các nhà cung cấp đã chỉnh sửa lại bản Android, do đó flutter_local_notification hoạt động không ổn định (vd Huawei, Xiaomi...)
- Sử dụng các plugin badges, flutter_app_badger để thay thế

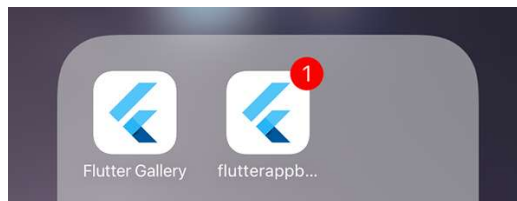


Huỳnh Tuấn Anh - ĐHNT

137

Xử lý tin nhắn nhận được

- Tin nhắn nhận được background: Nhận và hiển thị ở thanh trạng thái của ứng dụng
- Tin nhắn nhận được ở foreground: *Lưu tin nhắn*, cập nhật số lượng tin nhắn trên các trên các: badges, flutter_app_badger ...
 - Cập nhật số lượng tin nhắn trên icon ứng dụng ở màn hình thiết bị: `FlutterAppBadger.updateBadgeCount(count);`
 - Xóa số lượng tin nhắn ở màn hình thiết bị: `FlutterAppBadger.removeBadge();`

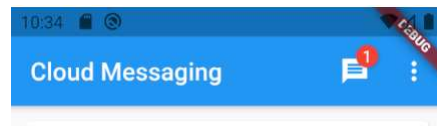


Huỳnh Tuấn Anh - ĐHNT

138

Nhận tin nhắn ở Foreground

```
Badge(
  showBadge: count>0,
  badgeContent: Text("$count", style: TextStyle(color: Colors.white)),
  position: BadgePosition.topEnd(top: 3, end: 5),
  child: IconButton(
    icon: Icon(Icons.message_sharp),
    onPressed: () {
      if(count>0) {...}
    },
  ), // IconButton
), // Badge
```



Huỳnh Tuấn Anh - ĐHNT

139

Nhận tin nhắn ở Foreground

```
FirebaseMessaging.onMessage.listen((RemoteMessage message) {
  SystemSound.play(SystemSoundType.click);
  /// Xử lý message gửi đến,
  /// thường là lưu vào danh sách
  receivedMessage = message;
  setState((){
    count++;
    _updateCountMessage(count);
  });
});
```

```
void _updateCountMessage(int num) async{
  SharedPreferences sharedPreferences =
    await SharedPreferences.getInstance();
  await sharedPreferences.setInt("count", num);
}
```

Huỳnh Tuấn Anh - ĐHNT

140

Một số chú ý

- Nếu sử dụng các plugin badges, flutter_app_badger, ta không cần phải sử dụng plugin flutter_local_notifications
 - Một số đoạn mã lệnh liên quan đến flutter_local_notifications trong hàm main nên loại bỏ (mã lệnh tạo channel)
- Để xem chi tiết tin nhắn, người sử dụng có thể nhấn vào các child (thường là một IconButton) của các badge để chuyển hướng tới trang/tab hiển thị chi tiết tin nhắn. Sau khi chuyển hướng, nội dung của "badge" sẽ reset = 0

```
if(count>0) {
  //Mã lệnh chuyển hướng sang trang hiển thị các tin nhắn ở đây
  _updateCountMessage(0); // Cập nhật số lượng tin nhắn chưa xem
  setState(() {
    count =0; // Cập nhật nội dung của badge
  });
}
```

Huỳnh Tuấn Anh - ĐHNT

141

Firebase Storage

- Dịch vụ của firebase cho phép người dùng lưu trữ một cách nhanh chóng và dễ dàng các nội dung phục vụ cho người dùng như: ảnh, video
- Cần phải thiết lập các rule trên firebase console. Ví dụ:

```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

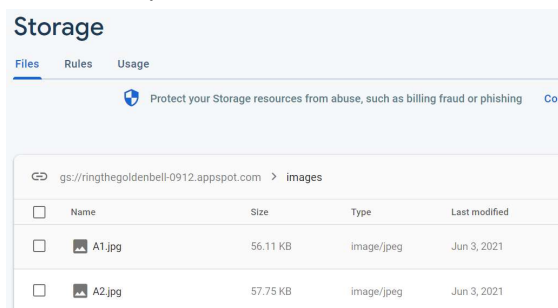
Huỳnh Tuấn Anh - ĐHNT

142

Firebase Storage

■ Các thư viện hỗ trợ

- `firebase_storage`: Thư viện hỗ trợ để sử dụng dịch vụ cloud storage của firebase
- `firebase_ui_storage`: Giao diện tích hợp các pp xác thực
- `image_picker`: Thư viện hỗ trợ chọn ảnh trên thiết bị
- `file_picker`: Thư viện hỗ trợ chọn tập tin trên thiết bị



Huỳnh Tuấn Anh - ĐHNT

143

Lưu trữ trên FirebaseStorage

■ Bước 1: Tạo một thể hiện FirebaseStorage

- `FirebaseStorage _storage = FirebaseStorage.instance;`

■ Bước 2: Tạo tham chiếu đến vị trí file sẽ lưu trong Firebase storage

- `Reference reference = _storage.ref().child("images").child("anh.jpg");`

■ Bước 3: Upload file lên trên Firebase Storage:

- `UploadTask uploadTask = await _uploadTask(reference);`

■ Bước 4: Lấy đường dẫn lưu trữ file, lưu đường dẫn này để truy cập file sau này

- `String url = await reference.getDownloadURL();`
- Lưu trữ đường dẫn này, có thể lưu trong CSDL cloud firestore để truy cập file sau này

Huỳnh Tuấn Anh - ĐHNT

144

Upload File


Trước khi Upload file lên firebase cần phải gọi đoạn mã sau để lấy đường dẫn tệp tin trên máy

```
_pickedFile = await picker.getImage(source: ImageSource.gallery);
```


```
FirebaseStorage _storage = FirebaseStorage.instance;
// Tạo tham chiếu đến file sẽ lưu trong firebase
Reference reference =
    _storage.ref().child("images").child("/${q.table}${q.id}.jpg");
UploadTask uploadTask = await _uploadTask(reference);
uploadTask.whenComplete(() async{
    String url = await reference.getDownloadURL();
    ... await Lưu url...
}).onError((error, stackTrace) async{
    _errorHandle("Lỗi xảy ra");
});
```

Upload File

```
Future<UploadTask> _uploadTask(Reference reference) async {
    final metadata = SettableMetadata(
        contentType: 'image/jpeg',
        customMetadata: {'picked-file-path': _pickedFile.path});
    UploadTask uploadTask;
    if (kIsWeb)
        uploadTask = reference.putData(await _pickedFile.readAsBytes(), metadata);
    else
        uploadTask = reference.putFile(File(_pickedFile.path), metadata);
    return Future.value(uploadTask);
}
```


 Bài tập

Huỳnh Tuấn Anh - ĐHNT147

 MongoDB

- MongoDB Cơ sở dữ liệu NoSQL mã nguồn mở:
- Các khái niệm
 - Database
 - Collection
 - Document
 - Field
 - _id
- MongoDB Realm: Cơ sở dữ liệu đám mây được MongoDB thiết kế cho các ứng dụng Mobile dựa trên dữ liệu


Huỳnh Tuấn Anh - ĐHNT148



MongoDB

- Node.js® là một trình chạy JavaScript (Javascript Runtime) được xây dựng trên công cụ JavaScript V8 của Chrome.
 - Môi trường thực thi JavaScript đa nền tảng, mã nguồn mở, là một công cụ được sử dụng phổ biến cho nhiều dự án
 - Nodejs không hỗ trợ đa luồng, nó là một máy chủ đơn luồng.
- MongoDB Cơ sở dữ liệu NoSQL mã nguồn mở
- Các khái niệm
 - Database
 - Collection
 - Document
 - Field
 - _id

Huỳnh Tuấn Anh - ĐHNT 149



Huỳnh Tuấn Anh - ĐHNT 150

Sử dụng cơ sở dữ liệu MongoDB trong ứng dụng mobile

- Truy cập MongoDB database trực tiếp thông qua thư viện mongo_dart
 - Thư viện không chính thống, chậm, còn nhiều lỗi
- Truy cập MongoDB database thông qua một server, ví dụ như Node.js
 - Viết RESTful API phía server: Các API thực hiện chức năng CRUD
 - Phía Client (Ứng dụng mobile) sử dụng plugin http/dio để gọi các RESTful API