

# Android APP

- Thiết kế cấu trúc app Android

# Thiết kế cấu trúc app Android

## Cấu trúc ứng dụng điều khiển thiết bị thông minh

### 1. Tổng quan kiến trúc ứng dụng

Ứng dụng sẽ được thiết kế theo mô hình **client-server** với giao tiếp qua **REST API** hoặc **WebSocket** để đảm bảo tính **real-time** cho các thiết bị như smartlocker và thiết bị on/off. Dưới đây là các thành phần chính:

- **Frontend (Client):** Ứng dụng Android/iOS được phát triển bằng **Flutter** hoặc **React Native** để hỗ trợ cả hai nền tảng, cung cấp giao diện người dùng (UI) mượt mà và đồng nhất.
- **Backend (Server):** Server xử lý logic nghiệp vụ, quản lý thiết bị, tài khoản người dùng, và giao tiếp với thiết bị IoT qua giao thức **MQTT** hoặc **HTTP**.
- **Thiết bị IoT:** Smartlocker và các thiết bị on/off được kết nối qua Wi-Fi, sử dụng giao thức **MQTT** để gửi/nhận lệnh và trạng thái từ server.
- **Database:** Lưu trữ thông tin người dùng, thiết bị, quyền truy cập, và trạng thái thiết bị. Sử dụng **MySQL/PostgreSQL** hoặc **MongoDB** để đảm bảo hiệu suất và khả năng mở rộng.

### Kiến trúc tổng thể

```
graph LR; A["[Ứng dụng Android/iOS]"] <--> B["[REST API/WebSocket]"]; B <--> C["[Server]"]; C <--> D["[MQTT Broker]"]; D <--> E["[Thiết bị IoT]"]; C --- F["[Database]"];
```

### 2. Cấu trúc ứng dụng Android/iOS

## 2.1. Công nghệ phát triển

- **Framework:** Sử dụng **Flutter** (Dart) để phát triển ứng dụng đa nền tảng, giúp giảm thời gian phát triển và đảm bảo giao diện đồng nhất trên Android và iOS.
- **Thư viện hỗ trợ:**
  - **http** hoặc **dio** (Dart): Gửi yêu cầu REST API.
  - **mqtt\_client** (Dart): Kết nối với MQTT broker để giao tiếp real-time với thiết bị.
  - **qr\_code\_scanner**: Quét mã QR để thêm thiết bị.
  - **image\_picker**: Hiển thị hình ảnh từ camera của smartlocker.
  - **shared\_preferences**: Lưu trữ thông tin đăng nhập cục bộ.
  - **provider** hoặc **bloc**: Quản lý trạng thái (state management) của ứng dụng.

## 2.2. Cấu trúc thư mục (Flutter)

Cấu trúc thư mục được tổ chức theo mô hình **feature-based** để dễ dàng mở rộng và bảo trì:

```
lib/
├── features/
│   ├── authentication/
│   │   ├── screens/      # Màn hình đăng nhập, đăng ký
│   │   ├── models/      # Model cho user
│   │   ├── services/    # API calls cho authentication
│   │   └── widgets/     # Các widget tái sử dụng
│   ├── smart_locker/
│   │   ├── screens/    # Màn hình điều khiển smartlocker
│   │   ├── models/    # Model cho locker (trạng thái, hình ảnh...)
│   │   ├── services/   # Giao tiếp với server/MQTT
│   │   └── widgets/   # Widget hiển thị trạng thái, nút điều khiển
│   ├── device_control/
│   │   ├── screens/    # Màn hình điều khiển thiết bị on/off
│   │   ├── models/    # Model cho thiết bị on/off
│   │   ├── services/   # Giao tiếp với server/MQTT
│   │   └── widgets/   # Widget hiển thị trạng thái, nút on/off
│   └── device_management/
│       ├── screens/    # Màn hình thêm thiết bị, quét QR, config Wi-Fi
│       ├── models/    # Model cho thiết bị và quyền
│       ├── services/   # API calls cho quản lý thiết bị
│       └── widgets/   # Widget cho quét QR, chia sẻ thiết bị
└── core/
```

└─ config/	# Cấu hình ứng dụng (API URL, MQTT broker...)
└─ models/	# Các model chung (Device, User...)
└─ services/	# Dịch vụ chung (HTTP client, MQTT client...)
└─ utils/	# Các hàm tiện ích (logger, helper functions...)
└─ main.dart	# Điểm vào của ứng dụng

## 2.3. Giao diện người dùng (UI)

### Màn hình chính

- **Đăng nhập/Đăng ký:** Form nhập email/password, nút đăng ký, đăng nhập qua OAuth (nếu cần).
- **Danh sách thiết bị:** Hiển thị danh sách thiết bị đã thêm (smartlocker, thiết bị on/off...vv), trạng thái (online/offline, bật/tắt).

#### • Màn hình smartlocker

- **Hiển thị hình ảnh:** Hình ảnh từ camera của smartlocker được tải xuống từ server qua **WebSocket** hoặc **REST API** và lưu trực tiếp vào bộ nhớ cục bộ (local storage) của ứng dụng, xóa trên bộ nhớ tạm của sever.
- **Nút điều khiển:** Đóng/mở locker.
- **Trạng thái:** Hiển thị trạng thái (có hàng/không có hàng, bị mở trái phép).
- **Thông báo đẩy:** Sử dụng **Firebase Cloud Messaging (FCM)** để thông báo khi locker bị mở không phép.
- **Màn hình thiết bị on/off:**
  - Nút toggle để bật/tắt thiết bị.
  - Hiển thị trạng thái hiện tại (ON/OFF).
  - Mỗi thiết bị khi bấm vào có thể có nhiều nút tùy theo loại thiết bị
- **Quản lý thiết bị:**
  - Quét mã QR để thêm thiết bị (lấy device ID từ mã QR).
  - Config Wi-Fi cho thiết bị (sử dụng **SmartConfig** hoặc giao thức tương tự).
  - Chia sẻ thiết bị: Form nhập email người dùng để cấp quyền điều khiển.
  - Phân quyền: Chọn mức quyền (chỉ xem, điều khiển, quản lý).

### Công nghệ UI

- Sử dụng **Flutter Widgets** để xây dựng giao diện responsive.
- **Push Notification:** Tích hợp **Firebase Cloud Messaging (FCM)** để gửi thông báo khi locker bị mở trái phép hoặc trạng thái thiết bị thay đổi.

# 3. Phương thức kết nối và giao tiếp với server

## 3.1. Giao thức kết nối

- **REST API:** Sử dụng cho các tác vụ không yêu cầu real-time như đăng nhập, đăng ký, thêm thiết bị, chia sẻ quyền.
- **WebSocket:** Dùng để stream hình ảnh từ camera của smartlocker.
- **MQTT:** Giao tiếp real-time với thiết bị IoT (gửi lệnh đóng/mở locker, bật/tắt thiết bị, nhận trạng thái).

## REST API Endpoints

Dưới đây là các endpoint chính (giả định):

GET	/api/devices	# Lấy danh sách thiết bị của người dùng
POST	/api/devices/add	# Thêm thiết bị mới (gửi device ID từ QR)
POST	/api/devices/share	# Chia sẻ quyền điều khiển thiết bị
PUT	/api/devices/config-wifi	# Cấu hình Wi-Fi cho thiết bị
POST	/api/auth/register	# Đăng ký tài khoản
POST	/api/auth/login	# Đăng nhập
GET	/api/lockers/:id/status	# Lấy trạng thái smartlocker
POST	/api/lockers/:id/control	# Gửi lệnh đóng/mở locker
GET	/api/devices/:id/status	# Lấy trạng thái thiết bị on/off
POST	/api/devices/:id/control	# Gửi lệnh bật/tắt thiết bị

## WebSocket cho hình ảnh

- **Endpoint:** ws://server-url/lockers/{device\_id}/image-stream
- **Chức năng:** Server nhận hình ảnh từ thiết bị qua MQTT, trung chuyển qua WebSocket đến ứng dụng, không lưu trữ trên server.
- **Payload:** Dữ liệu hình ảnh dạng **base64** hoặc **binary stream**.

## MQTT Topics

- **Gửi lệnh đến thiết bị:**
  - devices/{device\_id}/control: Lệnh đóng/mở locker hoặc bật/tắt thiết bị.
  - Ví dụ payload: {"action": "open"} hoặc {"action": "on"}.
- **Nhận trạng thái:**

- `devices/{device_id}/status`: Trạng thái locker hoặc thiết bị on/off.
- `lockers/{device_id}/alert`: Cảnh báo khi locker bị mở trái phép.
- **Hình ảnh từ camera:**
  - `lockers/{device_id}/image`: Thiết bị gửi hình ảnh lên server qua topic này, server trung chuyển đến ứng dụng qua WebSocket.

## 3.2. Quy trình xử lý hình ảnh

1. **Thiết bị gửi hình ảnh:**
  - Smartlocker chụp ảnh từ camera và gửi lên server qua topic MQTT `lockers/{device_id}/image` (dữ liệu dạng **base64** hoặc **binary**).
2. **Server trung chuyển:**
  - Server nhận dữ liệu hình ảnh, không lưu trữ mà chuyển trực tiếp đến ứng dụng qua **WebSocket**.
  - Nếu ứng dụng yêu cầu hình ảnh theo nhu cầu (on-demand), server gửi yêu cầu qua MQTT đến thiết bị để chụp ảnh mới.
3. **Ứng dụng lưu trữ cục bộ:**
  - Ứng dụng nhận dữ liệu hình ảnh qua WebSocket, lưu vào bộ nhớ cục bộ (local storage) sử dụng **path\_provider** (thư mục tạm hoặc thư mục ứng dụng).
  - Hình ảnh được hiển thị trên UI và xóa khỏi local storage khi không cần thiết (ví dụ: sau khi người dùng đóng màn hình).

## Lưu ý tối ưu hóa

- **Nén hình ảnh:** Thiết bị nén hình ảnh (JPEG, chất lượng ~80%) trước khi gửi để giảm băng thông.
- **Tần suất gửi:** Hạn chế tần suất gửi hình ảnh (ví dụ: chỉ khi có yêu cầu hoặc khi trạng thái locker thay đổi).
- **Bộ nhớ cục bộ:** Ứng dụng giới hạn số lượng hình ảnh lưu trữ (ví dụ: chỉ lưu 5 hình ảnh gần nhất) và xóa tự động khi vượt giới hạn.

## 3.3. Quy trình giao tiếp

1. **Đăng nhập/Đăng ký:**
  - Gửi yêu cầu POST đến `/api/auth/login` hoặc `/api/auth/register`.
  - Nhận **JWT token** để xác thực các yêu cầu tiếp theo.
2. **Thêm thiết bị:**
  - Quét mã QR để lấy **device ID**.
  - Gửi POST `/api/devices/add` với device ID và thông tin Wi-Fi (SSID, password).
  - Thiết bị nhận cấu hình Wi-Fi qua **SmartConfig** và kết nối với MQTT broker.
3. **Điều khiển thiết bị:**
  - Gửi lệnh qua MQTT topic `devices/{device_id}/control`.
  - Nhận trạng thái qua topic `devices/{device_id}/status`.

#### 4. Stream hình ảnh:

- Kết nối WebSocket đến server để nhận stream từ camera của smartlocker.

#### 5. Thông báo:

- Server gửi thông báo qua FCM khi có sự kiện (locker bị mở trái phép, trạng thái thiết bị thay đổi).

## 4. Các biến số điều khiển

### 4.1. Biến trạng thái thiết bị

- **Smartlocker:**

- device\_id (String): Định danh thiết bị.
- status (Enum: locked, unlocked): Trạng thái đóng/mở.
- has\_item (Boolean): Tủ có hàng hay không.
- alert (Boolean): Cảnh báo khi bị mở trái phép.
- image\_path (String): Đường dẫn cục bộ đến hình ảnh được lưu trong ứng dụng.

- **Thiết bị on/off:**

- device\_id (String): Định danh thiết bị.
- status (Enum: on, off): Trạng thái bật/tắt.

- **Quyền truy cập:**

- user\_id (String): Định danh người dùng.
- device\_id (String): Định danh thiết bị.
- permission (Enum: view, control, manage): Mức quyền.

### 4.2. Model dữ liệu (Dart)

```
class SmartLocker {  
  final String deviceId;  
  final String status; // "locked" or "unlocked"  
  final bool hasItem;  
  final bool alert;  
  final String? imagePath; // Đường dẫn cục bộ đến hình ảnh  
  
  SmartLocker({  
    required this.deviceId,  
    required this.status,  
    required this.hasItem,  
    required this.alert,
```

```

        this.imagePath,
    });

factory SmartLocker.fromJson(Map<String, dynamic> json) {
    return SmartLocker(
        deviceId: json['device_id'],
        status: json['status'],
        hasItem: json['has_item'],
        alert: json['alert'],
        imagePath: json['image_path'],
    );
}

}

class OnOffDevice {
    final String deviceId;
    final String status; // "on" or "off"

    OnOffDevice({
        required this.deviceId,
        required this.status,
    });

    factory OnOffDevice.fromJson(Map<String, dynamic> json) {
        return OnOffDevice(
            deviceId: json['device_id'],
            status: json['status'],
        );
    }
}

```

## 5. Quy trình phát triển chi tiết

### 5.1. Thiết lập môi trường

- Cài đặt **Flutter SDK** và cấu hình Android/iOS.
- Tích hợp **Firebase Cloud Messaging (FCM)** cho thông báo đẩy.

- Thiết lập **MQTT broker** (Mosquitto hoặc AWS IoT).
- Thêm **path\_provider** để quản lý lưu trữ cục bộ.

## 5.2. Triển khai giao tiếp WebSocket cho hình ảnh

Dưới đây là ví dụ code kết nối WebSocket trong Flutter để nhận hình ảnh:

```
import 'package:web_socket_channel/io.dart';
import 'package:path_provider/path_provider.dart';
import 'dart:io';

class ImageService {
  IOWebSocketChannel? channel;

  void connect(String deviceId) {
    channel = IOWebSocketChannel.connect('ws://server-url/lockers/$deviceId/image-stream');
    channel!.stream.listen((data) async {
      // Giả định data là base64 string
      final bytes = base64Decode(data);
      final directory = await getTemporaryDirectory();
      final file = File('${directory.path}/locker_${deviceId}_${DateTime.now().millisecondsSinceEpoch}.jpg');
      await file.writeAsBytes(bytes);
      print('Image saved at: ${file.path}');
      // Cập nhật UI với đường dẫn file
    });
  }

  void disconnect() {
    channel?.sink.close();
  }
}
```

## 5.3. Lưu trữ và quản lý hình ảnh cục bộ

Ví dụ code lưu và xóa hình ảnh:

```

import 'package:path_provider/path_provider.dart';
import 'dart:io';

class LocalStorageService {
  // Lưu hình ảnh vào bộ nhớ tạm
  Future<String> saveImage(Uint8List imageBytes, String deviceId) async {
    final directory = await getTemporaryDirectory();
    final file = File('${directory.path}/locker_${deviceId}_${DateTime.now().millisecondsSinceEpoch}.jpg');
    await file.writeAsBytes(imageBytes);
    return file.path;
  }

  // Xóa hình ảnh cũ để tiết kiệm bộ nhớ
  Future<void> cleanOldImages(String deviceId) async {
    final directory = await getTemporaryDirectory();
    final files = directory.listSync().where((file) => file.path.contains('locker_${deviceId}')).toList();
    if (files.length > 5) { // Giới hạn 5 hình ảnh
      files.sort((a, b) => a.path.compareTo(b.path));
      for (var i = 0; i < files.length - 5; i++) {
        await File(files[i].path).delete();
      }
    }
  }
}

```

## 5.4. Tích hợp SmartConfig

Sử dụng **esp\_smartconfig** để cấu hình Wi-Fi cho thiết bị:

```

import 'package:esp_smartconfig/esp_smartconfig.dart';

Future<void> configureWifi(String ssid, String password) async {
  final provisioner = Provisioner.espTouch();
  await provisioner.start(ProvisioningRequest(
    ssid: ssid,
    password: password,
  ));
  provisioner.listen((response) {
    print('Device connected: ${response.ipAddress}');
  });
}

```

```
});  
}
```

## 5.5. Quét mã QR

Sử dụng **qr\_code\_scanner** để quét mã QR:

```
import 'package:qr_code_scanner/qr_code_scanner.dart';  
  
class QRScannerScreen extends StatefulWidget {  
  @override  
  _QRScannerScreenState createState() => _QRScannerScreenState();  
}  
  
class _QRScannerScreenState extends State<QRScannerScreen> {  
  final GlobalKey qrKey = GlobalKey(debugLabel: 'QR');  
  QRViewController? controller;  
  
  @override  
  Widget build(BuildContext context) {  
    return QRView(  
      key: qrKey,  
      onQRViewCreated: (QRViewController controller) {  
        this.controller = controller;  
        controller.scannedDataStream.listen((scanData) {  
          print('Scanned QR: ${scanData.code}');  
          // Gửi device ID tới server để thêm thiết bị  
        });  
      },  
    );  
  }  
}
```

## 6. Bảo mật

- **Xác thực:** Sử dụng **JWT** cho REST API.
- **Mã hóa:** HTTPS cho REST API, TLS cho MQTT và WebSocket.

- **Phân quyền:** Server kiểm tra quyền trước khi gửi hình ảnh hoặc lệnh điều khiển.
- **Bộ nhớ cục bộ:** Lưu trữ hình ảnh trong **temporary directory** để dễ dàng xóa, sử dụng **Flutter Secure Storage** cho thông tin nhạy cảm như JWT.

## 7. Các bước triển khai tiếp theo

1. Thiết kế **database schema** cho người dùng, thiết bị, và quyền truy cập.
2. Triển khai **backend server** sử dụng **Node.js**, **Python (FastAPI)**, hoặc **Spring Boot**.
3. Tích hợp **FCM** để gửi thông báo đẩy.
4. Kiểm thử giao tiếp MQTT và SmartConfig trên thiết bị thực tế.
5. Triển khai tính năng stream hình ảnh qua WebSocket.