Giới thiệu Android

Android là gì?

- **Android** là một hệ điều hành mã nguồn mở do **Google** phát triển, dùng chủ yếu cho **điện thoại thông minh** và **máy tính bảng**.
- Đây là hệ điều hành di động phổ biến nhất thế giới, chiếm hơn 70% thị phần toàn cầu.
- Android dựa trên **nhân Linux**, cho phép các nhà phát triển xây dựng ứng dụng bằng ngôn ngữ lập trình như **Java** hoặc **Kotlin**.

Điểm nổi bật của Android

- **Mã nguồn mở**: bất kỳ ai cũng có thể sử dụng và tùy biến.
- **Kho ứng dụng khổng lồ**: Google Play Store với hàng triệu ứng dụng.
- **Đa dạng thiết bị**: có mặt trên nhiều dòng smartphone, từ cao cấp đến phổ thông.
- **Hệ sinh thái mạnh**: hỗ trợ đồng bộ với các dịch vụ Google (Gmail, Maps, Drive...).

Android app là gì?

- Một **Android app** là ứng dụng được thiết kế để chạy trên thiết bị Android.
- Người dùng có thể cài đặt ứng dụng từ **Google Play Store** hoặc bằng file cài đặt **APK**.
- Ứng dụng Android thường bao gồm:
    - **Giao diện người dùng (UI)**: các màn hình, nút bấm, văn bản, hình ảnh.
    - **Logic xử lý (Logic)**: cách ứng dụng phản hồi hành động của người dùng.
    - **Dữ liệu (Data)**: thông tin được lưu trữ cục bộ (trong điện thoại) hoặc trực tuyến (qua API, server).

Android Studio là gì?

Giới thiệu

- **Android Studio** là môi trường phát triển tích hợp (IDE – Integrated Development Environment) chính thức do **Google** phát hành để lập trình ứng dụng Android.
- Đây là công cụ mạnh mẽ, được xây dựng dựa trên IntelliJ IDEA, hỗ trợ lập trình bằng **Kotlin**, **Java** và **C++**.

Chức năng chính

- **Trình soạn thảo code thông minh**: gợi ý cú pháp, tự động hoàn thành code.
- **Thiết kế giao diện trực quan**: kéo thả (drag & drop) hoặc viết bằng XML/Compose.
- **Trình giả lập Android (Android Emulator)**: chạy thử ứng dụng mà không cần thiết bị thật.
- **Trình quản lý Gradle**: tự động build, quản lý thư viện và cấu hình project.
- **Debug & Profiler**: theo dõi lỗi, hiệu năng CPU, bộ nhớ, mạng khi chạy ứng dụng.
- **Tích hợp Git/GitHub**: hỗ trợ quản lý mã nguồn và làm việc nhóm.

Tại sao chọn Android Studio?

- Công cụ **chính thức từ Google**, luôn được cập nhật.
- Tích hợp đầy đủ công cụ cần thiết để phát triển một ứng dụng Android hoàn chỉnh.
- Dễ cài đặt, có sẵn plugin và hỗ trợ nhiều hệ điều hành (Windows, macOS, Linux).

Gradle là gì?

Giới thiệu

- **Gradle** là công cụ **tự động build (xây dựng dự án)** được Android Studio sử dụng mặc định.
- Nói một cách dễ hiểu, Gradle giống như **"người thợ" đứng sau hậu trường**, đảm nhiệm việc:
    - Biên dịch code Kotlin/Java thành file chạy trên Android.
    - Gắn kết các thư viện bên ngoài (dependencies) vào project.
    - Tối ưu hóa mã, nén hình ảnh, thu nhỏ dung lượng khi phát hành.
    - Tạo ra file cài đặt **APK** hoặc **AAB**.

Vì sao cần Gradle?

- **Tiết kiệm công sức**: Thay vì lập trình viên phải làm thủ công từng bước (biên dịch, copy file, thêm thư viện), Gradle tự động làm tất cả.
- **Quản lý thư viện dễ dàng**: Chỉ cần thêm 1 dòng trong file `build.gradle`, Gradle sẽ tải và tích hợp thư viện.
- **Hỗ trợ nhiều môi trường**: Có thể cấu hình cho bản **debug** (dùng khi phát triển) và bản **release** (dùng để phát hành) khác nhau.

Các file Gradle quan trọng trong Android

- **settings.gradle**: Xác định các module có trong project.
- **build.gradle (project-level)**: Khai báo chung cho toàn bộ project (phiên bản Gradle, plugin...).
- **build.gradle (app-level)**: Nơi cấu hình chính của ứng dụng, ví dụ:
    - `applicationId` (tên gói ứng dụng).
    - `minSdk` và `targetSdk`.
    - `dependencies` (thư viện bên ngoài như Retrofit, Room, Compose...).

Ví dụ dễ hiểu

Trong `build.gradle (app)` có đoạn:

```
dependencies {
    implementation "androidx.core:core-ktx:1.9.0"
    implementation "androidx.compose.ui:ui:1.3.0"
}
```

Nghĩa là: "Hãy cài thêm 2 thư viện hỗ trợ Kotlin và Compose UI cho ứng dụng này."
Gradle sẽ tự động tải về và tích hợp khi build.

JDK và SDK là gì?

1) JDK (Java Development Kit)

- **JDK** là bộ công cụ dành cho lập trình viên Java.
- Nó giống như một **"hộp đồ nghề"** để bạn có thể viết, biên dịch và chạy chương trình Java.
- Thành phần chính trong JDK:
  - **Java Compiler (javac)**: biên dịch code `.java` thành bytecode `.class`.
  - **Java Runtime Environment (JRE)**: môi trường để chạy chương trình Java.
  - **Các thư viện Java chuẩn**: cung cấp sẵn hàng nghìn hàm, lớp tiện ích.
- Với Android: Dù ứng dụng viết bằng **Kotlin**, nhưng Kotlin chạy trên **JVM (Java Virtual Machine)**, nên Android Studio vẫn cần **JDK** để biên dịch.

Hiểu đơn giản: **JDK = bộ công cụ giúp Android Studio dịch và chạy code Kotlin/Java.**

2) SDK (Software Development Kit)

- **SDK** là **bộ công cụ phát triển phần mềm** dành riêng cho một nền tảng.
- Với Android, ta có **Android SDK**, chứa mọi thứ cần để lập trình ứng dụng Android.
- Thành phần chính trong Android SDK:
  - **API của Android**: các thư viện cho phép truy cập vào tính năng của điện thoại (camera, GPS, Bluetooth…).
  - **Build Tools**: công cụ biên dịch và build ứng dụng.
  - **Emulator**: giả lập thiết bị Android để test ứng dụng.
  - **Platform Tools (adb, fastboot)**: kết nối và điều khiển thiết bị Android thật.

Hiểu đơn giản: **SDK = bộ công cụ giúp Android Studio giao tiếp với hệ điều hành Android và thiết bị.**

3) Sự khác nhau

- **JDK**: tập trung vào **ngôn ngữ Java/Kotlin** (biên dịch và chạy code).
- **SDK**: tập trung vào **nền tảng Android** (cung cấp API, công cụ để tạo ứng dụng chạy trên Android).

Ví dụ dễ hiểu:

- Bạn muốn xây một ngôi nhà.
- **JDK** = bộ dụng cụ cơ bản (búa, cưa, thước).
- **SDK** = vật liệu & bản thiết kế đặc thù để xây nhà kiểu Android (gạch, xi măng, cửa sổ theo chuẩn Android).

👉 Cả hai kết hợp lại → bạn có thể **xây dựng và chạy ứng dụng Android**.

Một sản phẩm Android app chuẩn sẽ như thế nào?

1) Giao diện người dùng (UI) thân thiện

- Thiết kế theo **Material Design** của Google.
- Hỗ trợ **Dark Mode** và **đa kích thước màn hình**.
- Nút bấm, chữ, màu sắc rõ ràng, dễ nhìn.
- Điều hướng (Navigation) trực quan, không rối mắt.

2) Kiến trúc rõ ràng, dễ bảo trì

- Sử dụng mô hình **MVVM (Model – View – ViewModel)** hoặc **Clean Architecture**.
- Chia code thành các phần riêng:
  - **UI (View)**: hiển thị dữ liệu.
  - **ViewModel**: quản lý trạng thái, xử lý logic.
  - **Repository/Model**: làm việc với API hoặc cơ sở dữ liệu.
- Giúp dự án dễ mở rộng và tránh lỗi khi phát triển lâu dài.

3) Hiệu năng mượt mà

- Ứng dụng phải **khởi động nhanh**.
- Chuyển màn hình trơn tru, không giật lag.
- Tối ưu bộ nhớ và pin, tránh chạy tác vụ nặng trên UI thread.

4) Bảo mật dữ liệu

- Không lưu **mật khẩu hoặc API key** trực tiếp trong code.
- Dữ liệu nhạy cảm được mã hóa.
- Kết nối mạng sử dụng **HTTPS**.
- Tuân thủ chính sách bảo mật của Google Play.

5) Khả năng hoạt động offline

- Dùng **Room database** để lưu dữ liệu cục bộ.
- Cho phép người dùng xem lại dữ liệu khi không có Internet.
- Đồng bộ tự động khi có mạng trở lại.

6) Quản lý phiên bản & phát hành

- Có **version code** và **version name** rõ ràng (ví dụ: 1.0.0, 1.0.1...).
- Build **debug** cho lập trình viên, build **release** cho phát hành.
- Tối ưu mã bằng **ProGuard/R8** khi release.

7) Kiểm thử và chất lượng

- Có test cơ bản: **Unit test** (logic nhỏ), **UI test** (giao diện).

- Ứng dụng chạy ổn định trên nhiều thiết bị và phiên bản Android khác nhau.
- Tránh crash bất ngờ bằng cách xử lý lỗi đầy đủ.

8) Trải nghiệm người dùng (UX)

- Hướng dẫn người mới (onboarding).
- Có icon, màu sắc, tên app chuyên nghiệp.
- Thông báo (notification) đúng lúc, không gây phiền.
- Đáp ứng đúng nhu cầu chính của người dùng, không rườm rà.

👉 Tóm lại, một **Android app chuẩn** phải:

- **Đẹp** (UI/UX thân thiện).
- **Chắc chắn** (kiến trúc, bảo mật).
- **Mượt** (hiệu năng tốt).
- **Tiện dụng** (hỗ trợ offline, đồng bộ online).
- **Đáng tin cậy** (ít lỗi, kiểm thử kỹ, phát hành theo chuẩn Google Play).

Cơ sở dữ liệu và Room trong Android

1) Cơ sở dữ liệu (Database) là gì?

- **Cơ sở dữ liệu (CSDL)** là nơi dùng để **lưu trữ và quản lý dữ liệu** một cách có tổ chức.
- Bạn có thể hình dung CSDL giống như một **cuốn sổ ghi chép điện tử**:
  - Dữ liệu được ghi theo dạng **bảng (table)** gồm hàng và cột.
  - Ví dụ: Bảng *SinhVien* có các cột: `id`, `ten`, `lop`, `diem`.
- Lợi ích của cơ sở dữ liệu:
  - Lưu trữ được **nhiều dữ liệu lớn** mà không bị mất.
  - Có thể **tìm kiếm, thêm, sửa, xóa** nhanh chóng.
  - Đảm bảo dữ liệu có **cấu trúc rõ ràng** và tránh trùng lặp.

So sánh: Có CSDL và không có CSDL

- **Không có CSDL**: tưởng tượng bạn ghi tên và điểm của hàng trăm sinh viên ra giấy rời. Khi muốn tìm 1 sinh viên, bạn phải lật từng tờ giấy → rất chậm, dễ mất giấy, dễ nhầm lẫn.
- **Có CSDL**: giống như bạn có một **bảng Excel điện tử**. Chỉ cần gõ tên, dữ liệu hiện ngay; có thể sắp xếp theo điểm, lọc theo lớp; dữ liệu lưu bền vững và có thể sao lưu.

👉 Như vậy, CSDL giúp dữ liệu **có tổ chức, dễ tìm, dễ quản lý và an toàn hơn rất nhiều**.

👉 Với ứng dụng Android: cơ sở dữ liệu thường được dùng để **lưu thông tin người dùng, lịch sử sử dụng, dữ liệu offline** để xem khi không có mạng.

Tại sao mobile cần có cơ sở dữ liệu?

- Ứng dụng di động không phải lúc nào cũng có Internet, nhưng vẫn cần hiển thị thông tin → CSDL cho phép **lưu trữ cục bộ** để dùng offline.

- Dữ liệu người dùng (tài khoản, cài đặt, lịch sử thao tác) cần được **lưu bền vững** để khi mở lại app không bị mất.
- Một số ứng dụng có lượng dữ liệu lớn (danh bạ, tin nhắn, sản phẩm thương mại điện tử) → không thể chỉ lưu tạm trên bộ nhớ RAM, mà cần CSDL để quản lý.
- CSDL giúp **đồng bộ**: dữ liệu lưu cục bộ trên máy, sau đó khi có Internet thì cập nhật với server.

Ví dụ:

- App ghi chú: lưu ghi chú trong CSDL để xem lại ngay cả khi offline.
- App mua sắm: danh sách sản phẩm được tải về và lưu trong CSDL, giúp tìm kiếm nhanh mà không cần gọi API liên tục.
- App học tập: lưu bài tập đã tải về, để học sinh làm ngay cả khi không có mạng.

2) Room trong Android là gì?

- **Room** là một thư viện do Google cung cấp, giúp lập trình viên Android làm việc với **SQLite Database** một cách dễ dàng hơn.
- Nếu **SQLite** giống như viết SQL thủ công (khó nhớ cú pháp, dễ lỗi), thì **Room** giống như một lớp học có **trợ giảng**:
    - Tự động chuyển đổi dữ liệu thành đối tượng Kotlin.
    - Tự động kiểm tra câu lệnh SQL có đúng không.
    - Giúp code ngắn gọn, dễ đọc, dễ bảo trì.

2.1 Các thành phần chính của Room

1. **Entity**: đại diện cho một bảng trong cơ sở dữ liệu.

```
@Entity(tableName = "sinhvien")
data class SinhVien(
    @PrimaryKey val id: Int,
    val ten: String,
    val lop: String,
    val diem: Float
)
```

→ Lớp `SinhVien` chính là bảng `sinhvien` với 4 cột.

9. **DAO (Data Access Object)**: nơi định nghĩa các hàm để truy vấn (insert, update, delete, query).

```
@Dao
interface SinhVienDao {
    @Insert
    suspend fun themSV(sinhVien: SinhVien)

    @Query("SELECT * FROM sinhvien")
    suspend fun layTatCa(): List<SinhVien>
}
```

→ Giúp bạn thao tác dữ liệu mà **không cần viết câu SQL dài dòng**.

18. **Database**: lớp kết nối đến CSDL và gom tất cả các DAO.

```
@Database(entities = [SinhVien::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun sinhVienDao(): SinhVienDao
}
```

2.2 Lợi ích của Room

- **Dễ sử dụng**: Code Kotlin thuần, không cần viết SQL phức tạp.
- **An toàn**: Room kiểm tra câu lệnh SQL ngay khi biên dịch → tránh lỗi khi chạy.
- **Hỗ trợ LiveData/Flow**: Khi dữ liệu thay đổi, UI tự động cập nhật.
- **Tích hợp tốt với MVVM**: DAO gọi từ Repository, Repository cung cấp cho ViewModel.

3) Tóm lại

- **Cơ sở dữ liệu** = nơi lưu dữ liệu có tổ chức (giống như một cuốn sổ điện tử).
- **Room** = thư viện giúp làm việc với cơ sở dữ liệu SQLite dễ dàng, ngắn gọn và an toàn hơn.

👉 Nhờ Room, ứng dụng Android có thể **lưu dữ liệu cục bộ** (offline), đảm bảo trải nghiệm mượt mà ngay cả khi không có Internet.

Jetpack Compose trong Android – Giải thích dễ hiểu

Jetpack Compose là gì?

- Jetpack Compose là **cách mới để thiết kế giao diện ứng dụng Android** do Google phát triển.
- Thay vì phải viết giao diện bằng nhiều file XML phức tạp, lập trình viên giờ chỉ cần dùng **ngôn ngữ Kotlin** để mô tả giao diện.
- Có thể hình dung Compose giống như **xếp lego**: mỗi mảnh lego là một thành phần giao diện (nút bấm, chữ, hình ảnh…). Ghép các mảnh lại với nhau sẽ tạo thành màn hình hoàn chỉnh.

Tại sao cần Jetpack Compose?

1. **Dễ hiểu, dễ viết hơn**: Thay vì viết hàng trăm dòng XML và Java, giờ chỉ cần vài dòng Kotlin.
2. **Nhanh hơn**: Khi dữ liệu thay đổi, giao diện sẽ tự động cập nhật, không cần viết thêm nhiều đoạn code điều khiển.
3. **Hiện đại**: Compose được thiết kế theo phong cách mới, hỗ trợ sẵn **Material Design 3** (chuẩn thiết kế của Google).
4. **Tiết kiệm thời gian**: Có thể xem trước giao diện ngay trong Android Studio, không cần cài app lên điện thoại.

Ví dụ minh họa dễ hiểu

- Nếu muốn hiện chữ “Xin chào”, trước đây phải viết XML và code để kết nối.
- Với Compose chỉ cần:

```
@Composable
fun LoiChao() {
    Text("Xin chào")
}
```

👉 Một hàm nhỏ đã đủ để hiển thị chữ trên màn hình.

Vai trò của Jetpack Compose

- **Giúp lập trình viên tập trung vào ý tưởng**, không mất nhiều công sức với kỹ thuật rườm rà.
- **Ứng dụng đẹp và hiện đại hơn**, vì Compose gắn liền với chuẩn thiết kế mới.
- **Phù hợp với người mới học**: dễ tiếp cận hơn so với cách làm truyền thống.

👉 Nói ngắn gọn: **Jetpack Compose là cách làm giao diện Android đơn giản, nhanh chóng và hiện đại**, giúp việc phát triển ứng dụng trở nên dễ dàng hơn rất nhiều.

MVVM trong Android – Giải thích dễ hiểu

MVVM là gì?

- **MVVM** là viết tắt của **Model – View – ViewModel**, một cách tổ chức code khi làm ứng dụng.
- Bạn có thể hình dung MVVM giống như **chia việc trong một nhóm** để làm việc hiệu quả hơn:
    - **Model**: quản lý dữ liệu (giống như "người thủ kho" giữ thông tin).
    - **View**: giao diện người dùng (giống như "người bán hàng" trực tiếp nói chuyện với khách).
    - **ViewModel**: trung gian giữa hai bên (giống như "quản lý cửa hàng" – nhận yêu cầu từ khách, đi lấy dữ liệu từ kho, rồi đưa lại cho người bán hàng).

👉 Nhờ vậy, mỗi người (mỗi lớp) có nhiệm vụ rõ ràng, không ai làm việc chồng chéo.

Nếu không dùng MVVM (cách truyền thống)

- Toàn bộ code (giao diện + dữ liệu + xử lý) để chung một chỗ.
- Giống như một người **vừa làm thủ kho, vừa làm quản lý, vừa bán hàng**.
- Hậu quả:
    - Code rối, khó đọc, khó sửa.
    - Khi có lỗi → khó tìm nguyên nhân.
    - Muốn thêm tính năng mới → dễ gây hỏng chỗ khác.

Khi có MVVM

- Công việc được **chia nhỏ**:
    - **View** chỉ hiển thị giao diện và gửi sự kiện (bấm nút, nhập dữ liệu).
    - **ViewModel** nhận sự kiện, xử lý logic, yêu cầu dữ liệu từ Model.
    - **Model** cung cấp dữ liệu (API, cơ sở dữ liệu cục bộ).
- Giống như một cửa hàng chuyên nghiệp:
    - Người bán hàng (View) chỉ tập trung phục vụ khách.
    - Quản lý (ViewModel) điều phối công việc.
    - Thủ kho (Model) cung cấp đúng dữ liệu khi cần.

👉 Kết quả:

- Code gọn gàng, dễ hiểu, dễ sửa.
- Có thể thay đổi giao diện mà không ảnh hưởng dữ liệu.
- Dễ dàng phát triển thêm tính năng mới.

Tóm lại

- **Không MVVM** = mọi việc dồn hết cho một người, dễ rối, khó kiểm soát.
- **Có MVVM** = chia việc rõ ràng, ai lo phần việc nấy → ứng dụng rõ ràng, dễ phát triển, ít lỗi.

👉 Đây là lý do MVVM trở thành kiến trúc phổ biến nhất khi làm ứng dụng Android hiện nay.