Contents

Lesson 1. Build your First App

1. Core of file

- Kotlin files: chứa logic cốt lõi của ứng dụng.
- Res folder: chứa các tài nguyên tĩnh như bố cục, hình ảnh, chuỗi …
- Manifests: Bao gồm các thông tin cần thiết của ứng dụng mà hệ điều hành cần biết để chạy ứng dụng.
- Gradle scripts: xây dựng và chạy ứng dụng.

2. Activity và Layout

- Activity: Là lớp core android chịu trách nhiệm về draw giao diện và nhận các sự kiện đầu vào.
- Layout: xác định app sẽ trông như thế nào bằng cách xác định các Text, Image, Button (nằm trong file XML) xuất hiện trên màn hình.
- Activity và Layout được kết nối với nhau bởi quá trình gọi là Layout Inflation, trong đó các dạng View ở bố cục được inflate thành các đối tượng View Kotlin trong bộ nhớ. Các View được tổ chức thành cây phân cấp View và để tìm các View cụ thể trong file XML, ta dùng phương thức findViewById.

3. Gradle scripts

- Mô tả những thiết bị mà ứng dụng dùng để chạy.
- Biên dịch mã và tài nguyên thành mã thực thi.
- Khai báo và quản lý mã và các thư viện phụ thuộc.
- Đăng ký ứng dụng với Google Play để người dùng có thể tải xuống và sử dụng.
- Khi run app, một loạt các lệnh Gradle biên dịch mã nguồn, các dependence và bất kỳ tài nguyên nào được liên kết trong gói ứng dụng Android hoặc APK.
- APK: là định dạng các tệp được sử dụng để phân phối và cài đắt các ứng dụng của Android.

- Build.gradle (project): gồm repositories và dependencies xác định kho lưu trữ có sẵn cho toàn bộ dự án. Repositories là các server từ xa, nơi bất kì mã bên ngoài nào sẽ được tải xuống từ đó. Dependences là các mã bên ngoài chẳng hạn như thư viện, mà dự án phụ thuộc vào.
- Build.gradle (app): chịu trách nhiệm xác định cấu hình module của ứng dụng này, nó bào gồm những plug-in để biết cách xây dựng các dự án Android và Kotlin, những phiên bản các hệ điều hành mà ứng dụng hỗ trợ, các dependencies mà gradle app sẽ tải xuống.

4. Android Compatibility

- Là một thành phần phụ thuộc và cũng là một phần của Androidx.

Lesson 2. Layouts

1. View Groups & View Hierarchy

- Tất cả các view đều có thuộc tính height, width và nền, chúng đều có thể tương tác. Có các loại View cơ bản như TextView, Button, ImageView, CheckBox, Menu, Slider, ColourPicker …
- Đơn vị thể hiện vị trí và kích thước là dp. Dp là một đơn vị trừu tượng dựa trên mật độ điểm ảnh vật lý của màn hình. VD: 1dp là 1pixel trên màn hình 160inch và là 3pixel trên màn hình 480inch.
- Các chế độ xem tạo nên một layout được tổ chức thành một View Hierarchy. Views mà nhiệm vụ chính là giữ các Views khác được gọi là ViewGroups
- LinearLayout là một ViewGroup mà có thể sắp xếp các Views theo chiều ngang hoặc chiều dọc, phù hợp với các layout có nội dung đơn giản.
- ConstrainLayout phân cấp chế độ xem phẳng và được tối ưu hóa để bố trí các chế độ xem của nó, bố cục ràng buộc của nó. ConstrainLayout sắp xếp một lượng nhỏ View hoặc ViewGroups trong một layout phức tạp rất tốt.

2. Layout Editor

- Project View, Android: Hiển thị phân cấp dự án theo cách sắp xếp trực quan hơn cho các dự án Android so với các phân cấp tệp thô.
- Design and Text tabs: Cho phép chuyển đổi giữa biểu diễn trực quan và layout.
- Palette: Liệt kê các view phổ biến cho các thành phần UI và layouts mà bạn có thể thêm vào thiết kế bố cục.
- Component Tree: Hiển thị layout hierarchy dưới dạng cây các view.
- Design Editor: Hiển thị bản xem trước màn hình.
- Atrributes pane: Hiển thị các thuộc tính của các chế độ xem được chọn.

3. Adding and Styling a TextView

- TextView hiển thị để format và style text.
- Styles là một tài nguyên, được định nghĩa trong styles.xml, có thể định nghĩa màu sắc, font, kích tước text …
- Styles làm các thành phần layut trông chắc chắn hơn.
- Dùng styles làm code dễ đọc hơn.

4. EditText

- Extend từ TextView.
- Hệ thống Android tự động xác nhận tính hợp lệ của đầu vào dựa trên trường input.
- Lựa chọn inputType hẹp nhất giúp người dùng tránh được các lỗi.

5. Button

- Các buttons là mở rộng của TextView.
- Người dùng mong đợi rằng các buttons có thể click được và khi click vào thì có các sự kiện được xảy ra.

6. Data Binding

- Cho phép kết nối các bố cục với 1 Activity hoặc Fragment tại thời điểm biên dịch thay vì phải duyệt qua hệ thống phân cấp chế độ xem để tìm trong runtime như findViewById. Điều này sẽ tiết kiệm thời gian hơn để tăng tốc ứng dụng người dùng. Trình biên dịch sẽ tạo ra một class được gọi là lớp ràng buộc khi Activity được tạo, sau đó ta có thể truy cập các view thông qua class này mà không cần thêm bất kì nỗ lực nào.

7. Constrain Layout

- Là sự kết nối hoặc căn giữa các thành phần UI với các thành phần UI khác hoặc với layout parent hoặc với các hướng dẫn vô hình.
- Lợi ích
    - Có thể dễ dàng responsive với các màn hình.
    - Thường phẳng hơn view hierachy.
    - Tối ưu layout với các view.
    - Free-form -đặt các view ở mọi nơi và editor giúp thêm các ràng buộc.
    - ConstrainLayout hiệu quả hơn với nhỏ hơn 15 View.
- Cách thức hoạt động của ràng buộc
    - Nếu không chỉ định ràng buộc, nó luôn xuất hiện ở vị trí (0, 0) của màn hình.
    - Ở định vị tuyệt đối, chỉ định tọa độ của một view trong hệ tọa độ của view parent.
    - Ở định vị tương đối, chỉ định tọa độ của một view trong mối quan hệ với các view khác, bao gồm cả chế độ chính.
- Ratio

    - Chế độ xem cần có một tỷ lệ khung hình nhất định để chứa hình ảnh, bất kể hướng màn hình hoặc kích thước hiển thị là bao nhiêu.
    - Tạo một layout với các hình vuông đáp ứng với các hình vuông khác.
- Chaining: Liên các các view như một hàng ngang hoặc hàng dọc, sau đó cư sử với nhau như một group.
    - Spread chain là mặc định và các phần tử spread là như nhau.
    - Spread inside chain sử dụng tất cả các không gian có sẵn với phần đầu và phần đuôi được dán vào cha mẹ.
    - Weighted chain sử dụng tất cả các không gian và thay đổi kích thước các phần tử để lấp đầy nó, dựa trên các giá trị được đặt trong trọng số ngang của ràng buộc layout, hoặc các thuộc tính trọng lượng dọc ràng buộc layout.
    - Packed chain sử dụng không gian tối thiểu.
    - Packed chain with bias có thể thêm bias để di chuyển nhóm các phần tử trên trục.

Lesson 3: App Navigation

1. Navigation pattern

- Action bar: Xuất hiện ở trên cùng màn hình ứng dụng, chứa các tính năng điều hướng cũng như nhãn hiệu ứng dụng, overflow menu và Drawer.
- Up button: Xuất hiện ở thanh Action bar và đưa ta trở lại màn hình trước đó mà người dùng đã điều hướng đến trong ứng dụng.
- Overflow menu: danh sách thả xuống các thư mục trong Action bar có thể chứa các điểm điều hướng đến.
- Navigation drawer: Một menu cùng với header có thể trượt ra từ cạnh của ứng dụng.
- Navigtion graph: là một đồ thị chứa tất cả các destinations, các màn hình có thể được điều hướng từ một hoạt động duy nhất đều được chứa trong này

2. Fragment

- Hỗ trợ năng động và linh hoạt hơn các UI được thiết kế trên các màn hình lớn như máy tính bảng.
- onCreateView(): trong Fragment ta không cần phải inflate layout trong OnCreate() giống như trong Activity, thay vào đó là trong method này.
- Context: Sử dụng thuộc tính này bên trong Fragment đẻ truy cập các tài nguyên như chuỗi và hình ảnh.
- The Activity Layout: Các UI Fragment chứa một bố cục và chiếm một vị trí bên trong.
- Fragment có thể cung cấp các UI và phải đi kèm với Activity như khung chứa, Fragment hoạt động như View trong Activity.
- Có thể điều hướng các Activity khác nhau hoặc giữa các phân đoạn khác nhau trong cùng một Acticity.

- Back stack: được sắp xếp dựa trên thứ tự mà Activity hoặc Fragment được mở, khi điều hướng, các Activity được đưa vào Back stack và khi điều hướng ngược lại, các Activity sẽ lần lượt bị pop ra khỏi Back stack và đi đến Activity cần điều hướng.

3. Navigation component

- Nguyên tắc 1: Các ứng dụng có điểm bắt đầu cố định, đó là màn hình mà người dùng nhìn thấy. Điểm đến này cũng là điểm cuối của màn hình mà người dùng nhìn thấy khi họ nhấn nút quay lại.
- Nguyên tắc thứ 2: Hoạt động dựa trên nguyên tắc Back stack.
- Nguyên tắc thứ 3; Up button và Back button hoạt động như nhau khi điều hướng ứng dụng trở lại. Trong khi Back button điều hướng ứng dụng vào các ứng dụng khác và thường là trình khởi chạy thì Up button điều hướng ứng dụng về màn hình đầu tiên và sẽ không hiển thị Up button tại thời điểm đó.

4. Thao tác Back Stack

- PopTo Not-Inclusive: Pop mọi thứ từ Back stack cho tới khi tìm thấy tham chiếu Frament Transaction.
- PopTo Inclusive: Pop mọi thứ từ Back stack bao gồm cả tham chiếu Frament Transaction.

5. Các thuộc tính của Menu

- ID: dùng bởi navigation để xác định nơi cần điều hướng.
- Title: Chuỗi được dùng để hiển thị trên menu.
- setHasOptionsMenu: Thông báo với Android rằng có một menu.
- onCreateOptionsMenu: Xác định nơi inflate menu.
- onOptionsItemSelected: Được gọi khi menu item đượ chọn.

6. Safe Arguments

- Các Fragment chứa các argument dưới dạng một gói Android (Android bundle), đó là một cấu trúc dữ liệu mà ta sử dụng một khóa duy nhất chẳng hạn như một chuỗi để tìm nạp một giá trị được liên kết.
- Safe Arguments giúp tránh được lỗi null và sai khác kiểu đối số vì điều hướng tạo ra hành động và lớp đối số từ biểu đồ điều hướng.

7. Intent

- Intent là một đối tượng mang thông điệp, cho phép request một hành động từ một vài component trong ứng dụng.
- Explicit Intent: được sử dụng để khởi chạy một activity sử dụng tên lớp của Activity đó và thường chỉ được dùng để khởi chạy các hoạt động khác trong cùng một ứng dụng.
- Implicit Intent: Loại Intents này chỉ ra hành động cần được thực hiện (action) và dữ liệu cho hành động đó (data). Khi sử dụng implicit intent, hệ thống Android sẽ tìm kiếm tất cả thành phần thích hợp để start bằng cách cách so sánh nội dung của Intent đc gửi với các Intent filter đc khai báo trong ứng dụng khác. Nếu intent đc gửi đó khớp với intent filter trong một component hoặc một ứng dụng nào đó, thì ngay lập tức hệ thống sẽ khởi động thành phần đó và cung cấp cho nó intent ban đầu đc gửi.

Nếu nhiều intent filter tương thích thì hệ thống sẽ hiển thị hộp thoại để người dùng có thể chọn ứng dụng nào sẽ sử dụng.

Lesson 4: Activity & Fragment LifeCycle

1. States

- Running: Khi Activity được kích hoạt, và được hệ thống để vào BackStack, nó sẽ bước vào trạng thái active. Với trạng thái active, người dùng hoàn toàn có thể nhìn thấy và tương tác với Activity của ứng dụng.
- Pause: Trạng thái này khá đặc biệt. Trạng thái tạm dừng, trạng thái này xảy ra khi mà Activity đang chạy, người dùng vẫn nhìn thấy, nhưng Activity khi này lại bị che một phần bởi một thành phần nào đó. Chẳng hạn như khi bị một dialog đè lên. Cái sự che Activity này không phải hoàn toàn. Chính vì vậy mà Activity đó tuy được người dùng nhìn thấy nhưng không tương tác được.
- Stop: Trạng thái này khá giống với trạng thái tạm dừng trên kia. Nhưng khi này Activity bị che khuất hoàn toàn bởi một thành phần giao diện nào đó, hoặc bởi một ứng dụng khác. Và tất nhiên lúc này người dùng không thể nhìn thấy Activity của bạn được nữa.Hành động mà khi người dùng nhấn nút Home ở System Bar để đưa ứng dụng của bạn về background, cũng khiến Activity đang hiển thị trong ứng dụng rơi vào trạng thái dừng này.
- Dead: Nếu Activity được lấy ra khỏi BackStack, chúng sẽ bị hủy và rơi vào trạng thái này. Trường hợp này xảy ra khi user nhấn nút Back ở System Bar để thoát một Activity. Hoặc lời gọi hàm finish() từ một Activity để "kill chính nó". Cũng có khi ứng dụng ở trạng thái background quá lâu, hệ thống có thể sẽ thu hồi tài nguyên bằng cách dừng hẳn các Activity trong ứng dụng, làm cho tất cả các Activity đều vào trạng thái này.Khi vào trạng thái dead, Activity sẽ kết thúc vòng đời của nó.

2. Callbacks

- onCreate(): Hàm này được gọi khá sớm, ngay khi activity được kích hoạt và thầm chí người còn chưa thấy gì cả thì callback này đã đc gọi rồi. Ngoài ra thì bạn nên biết là callback này chỉ được gọi một lần duy nhất khi Activity được khởi tạo. Nó có thể được gọi lại nếu hệ thống xóa Activity này đi để lấy lại tài nguyên của hệ thống, nhưng rất hiếm khi xảy ra. Và nó còn có thể được gọi lại nếu bạn xoay màn hình (ngang/dọc). Nó đước dùng để load giao diện cho Activity ở giai đoạn này load database do chỉ được gọi một lần duy nhất trong vòng đời của Activity.
- onStart(): Sau khi gọi đến onCreate(), hệ thống sẽ gọi đến onStart(). Hoặc hệ thống cũng sẽ gọi lại onStart() sau khi gọi onRestart() nếu trước đó nó bị che khuất bởi Activity nào khác (một màn hình khác hoặc một ứng dụng khác) che hoàn toàn và rơi vào onStop(). Khi hệ thống gọi đến callback này thì Activity được nhìn thấy bởi người dùng và nhưng chưa tương tác được. Bởi đặc tính này mà onStart() ít được dùng đến.

- onResume(): Khi hệ thống gọi đến callback thì người dùng đã nhìn thấy và đã tương tác được với giao diện. onResume() được gọi khi Activity được khởi tạo rồi và bước qua onStart() trên kia. Hoặc khi Activity bị một giao diện nào khác che đi một phần (hoặc toàn phần), rồi sau đó quay lại Activity hiện tại. Callback này được gọi rất nhiều lần trong một vòng đời của nó. Chính đặc điểm này của onResume() mà có thể tận dụng để quay lại tác vụ mà người dùng đang bị dang dở khi onPause() (được nói đến dưới đây) được gọi.
- onPause(): Được gọi khi một thành phần nào đó che Activity hiện tại mà người dùng vẫn nhìn thấy Activity đó (nhìn thấy chứ không tương tác được. Bạn có thể tưởng tượng rằng onPause() cũng sẽ được gọi khá nhiều lần trong một vòng đời Activity. Theo như Google thì onPause() được gọi đến khá nhanh, nếu muốn lưu trữ dữ liệu thì nên lưu những gì nhanh gọn.
- onStop(): onStop() được gọi khi Activity không còn được nhìn thấy nữa, có thể một màn hình nào khác che lên hoàn toàn, có thể một ứng dụng nào đó vào foreground, hoặc người dùng nhấn nút Home để về màn hình chính. Có thể tận dụng onStop() để lưu trữ dữ liệu ứng dụng. Hoặc để giải phóng các tài nguyên đang dùng. Ngưng các API còn đang gọi dang dở.
- onDestroy(): callback này để giải phóng các tài nguyên hệ thống mà ở onStop() bạn chưa gọi đến.

3. Lifecycle Observation

- Dùng để quan sát LifeCycleOwner, như là một Activity hay Fragment.
- Observer Pattern: một design pattern liên quân đến đối tượng Subject, theo dõi các đối tượng được gọi là Observer. Những Observer quan sát các đối tượng. Khi trạng thái của Subject thay đổi, nó thông báo cho tất cả các Observer nó.

4. Process Shutdown

- Hệ điều hành điều chỉnh các ứng dụng chạy trong nền để các ứng dụng trước có thể chạy mà không gặp sự cố. Điều này bao gồm việc giới hạn số lượng ứng dụng xử lý ở chế độ nền có thể làm, và trong một số trường hợp cụ thể, thậm chí tắt toàn bộ quy trình ứng dụng, bao gồm mọi hoạt dộng liên kết với ứng dụng.

5. onSaveInstanceBundle

- Là một callback, nơi mà có thể lưu trữ dữ liệu khi hệ điều hành Android phá hủy ứng dụng.

6. Configuration Change

- Khi rotation phone, các callback được xảy ra theo thứ tự sau: onStop, onDestroy, onCreate, onStart, onResume.

Lesson 5: App Architecture (UI layer)

1. Architecture

- Bản thiết kế các lớp ứng dụng và mối quan hệ giữa chúng để code được tổ chức tốt hơn.
- UI Controller: được dùng để mô tả Activity hoặc Fragment, nó chịu trách nhiệm cho các nhiệm vụ liên quan đến UI như là hiển thị các view, bắt input người dùng.
- ViewModel: làm các tác vụ decision-making, mục đích là giữ data để hiển thị trên Fragment hoặc Activity liên kết với chúng. Bên cạnh đó, chúng cũng thực hiện các tính toán đơn giản và chuyển đổi data để có thể hiển thị trên UI Controller. ViewModel giữ các LiveData.

2. ViewModel

- Một abstract class giữ dữ liệu UI của ứng dụng ngay cả khi xảy ra configuration changes.
- Lưu trữ được data lớn hơn nhiều so với saveInstanceBundle.
- ViewModel giữ data cho UI, và không bao giờ tham chiếu đến Fragment hay Activity.

3. The benefits of a good architecture

- Organized.
- Dễ debug.
- Ít xảy ra các vấn đề liên quan đến LifeCycle.
- Tạo thành các modul

4. Livedata

- LiveData là một lớp lưu trữ dữ liệu có thể quan sát được (Subject), nó bao quanh data.
- LiveData nhận biết vòng đời, có nghĩa là nó tôn trọng vòng đời của các thành phần ứng dụng khác, chẳng hạn như Activity, Fragment hoặc Service.
- Các Observer như Activity, Fragment quan sát sự thay đổi của nó và tự động cập nhật qua các method.
- Nếu Fragment hay Activity ở trong background, chúng sẽ không nhận được thông báo nhưng khi chúng quay trở lại màn hình, nó sẽ gọi các observer code cùng với giá trị data gần nhất.

Lesson 6: App Architecture (Persistence)

1. SQL databases

- Cơ sở dữ liệu SQL lưu trữ dữ liệu trong các bảng hàng và cột
    - Giao điểm của hàng và cột được gọi là trường.

- Các trường chứa dữ liệu, tham chiếu đến các trường khác hoặc tham chiếu đến các bảng khác.
- Mỗi hàng chứa một thực thể được xác định bằng một ID duy nhất, thường được sử dụng làm khóa chính của nó.
- Mỗi cột được xác định bằng một tên duy nhất cho mỗi bảng.

2. SQLite

- SQLite triển khai một công cụ cơ sở dữ liệu SQL
  - Độc lập (không yêu cầu các thành phần khác).
  - Không có máy chủ (không yêu cầu phần phụ trợ máy chủ).
  - Không cấu hình (không cần phải định cấu hình cho ứng dụng của bạn).
  - Độc lập (không yêu cầu các thành phần khác) Không có máy chủ (không yêu cầu phần phụ trợ máy chủ) Không cấu hình (không cần phải định cấu hình cho ứng dụng của bạn).

3. Data Access Object (DAO)

- Xác định một interface tùy chỉnh để truy cập cơ sở dữ liệu.
- Cung cấp các tiện ích như *@Insert, @Delete, @Update, @Query*, vì vậy không cần viết câu lệnh truy vấn.
- Khi đã xác định các DAO, Room tạo ra ánh xạ giữa các truy vấn và các hàm.

4. Multithreading and Coroutines

- Multithreading
  - Nhiều luồng thực thi trong một tiến trình.
  - Trình lập lịch tính toán đến những thứ như ưu tiên và đảm bảo tất cả các luồng chạy và kết thúc.
  - Luồng chính chạy ở foreground và các luồng phụ chạy ở background. Luồng chính xử lý tất cả các cập nhật đến UI, nó cũng là luồng gọi các xử lý click, các UI khác và lifecycle callbacks.
- Coroutines
  - Coroutines tránh luồng chính bị delay bởi các tác vụ dài từ background như load database.
  - Có khả năng báo lỗi hiệu quả với Exception.
  - Thực hiện chính xác cùng một việc với callback.
  - Asynchronous: Coroutines chạy độc lập với các bước thục thi của chương trình.
  - Non-blocking: hệ thống không block luồng chính.
  - Job: là bất kể thứ gì có thể bị hủy bỏ.

- Dipatcher: gửi các coroutines để chạy trên các luồng khác nhau.
- Scope: kết hợp thông tin, bao gồm một job và dipatcher để định nghĩa với context nơi coroutines chạy.