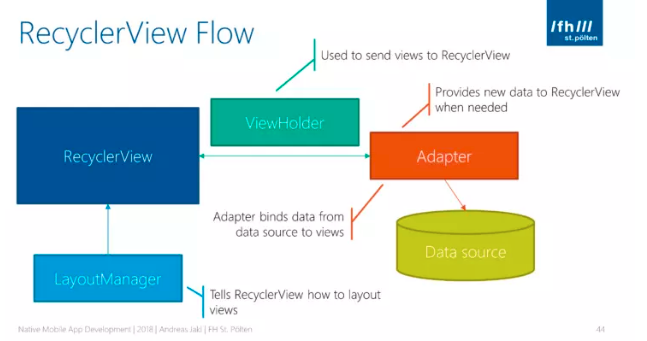
1. **RecyclerView**

**RecyclerView là một ViewGroup nó được dùng để chuẩn bị và hiện thị các View tương tự nhau.**

**Sử dụng RecyclerView** **khi** mà **data có các thành phần thay đổi trong quá trình chạy dựa trên hành động của người dùng hoặc các dữ kiện mạng.**

Một trong những lý do khiến **RecyclerView** được ưa chuộng là nó **là một framework có thể mở rộng, và** đặc biệt nó **cung cấp khả năng triển khai cả bố cục Hozizontal và Vertical.**

Nếu muốn sử dụng RecyclerView, bạn sẽ cần làm việc với các thành phần sau:



* RecyclerView.Adapter : đây là nơi xử lý dữ liệu và gán cho View.
* Layoutmanager: xác định ra vị trí của các item trong RecyclerView.
* ItemAnimator: Tạo hiệu ứng cho các hành động thêm , sửa, xóa các item điều này rất khó thực hiện với phiên bản tiền nhiệm (ListView)

Một khi RecyclerView được kết nối với Adapter , Adapter sẽ tạo ra đối tượng của các hàng (ViewHolder object) cho đến khi lấp đầy kích thước của RecyclerView và lưu trong HeapMemory . Sau đó sẽ không tạo thêm bất kỳ hàng nào để lưu trong bộ nhớ nữa Theo cách này nếu ngườ dùng trượt danh sách, các item đã trượt khỏi màn hình sẽ được giữ ở trong bộ nhớ để tái sử dụng lại sau và mỗi khi một hàng mới được chèn vào màn hình thì đối tượng ViewHolder được lưu trong bộ nhớ sẽ được mang ra để tái sử dụng và gán dữ liệu . Nếu không gán lại dữ liệu cho ViewHolder object thì sẽ hiện thị dữ liệu được gán trước đó. Theo cách này thì kể cả danh sách có 1000 item, thì chỉ có khoảng 7 đối tượng ViewHolder được tạo ra .

**LAYOUT**

Một RecyclerView cần có một layout manager và một adapter được khởi tạo.

Layout manager sẽ xác định các item bên trong RecyclerView được hiển thị như thế nào và khi nào phải tái sử dụng lại các item view ( những item đã bị trượt khỏi màn hình)

**LinearLayoutManager** : hiển thị các item trong danh sách có thể cuộn theo chiều dọc (horizontal) hoặc chiều ngang ( Vertical).

**GridLayoutManager** : hiển thị các item trong danh sách theo dạng lưới .

**StaggeredGridLayoutManager** : hiển thị các item trong danh sách theo dạng lưới so le nhau. Để tạo ra một custom layout manager, thì phải kế thừa [RecyclerView.LayoutManager](https://developer.android.com/reference/android/support/v7/widget/RecyclerView.LayoutManager) class.

Một chú ý quan trọng đối với Layout manager là nếu không khai báo layout manager thì RecyclerView sẽ không hiển thị và cửa số log cat sẽ thông báo:

**ADapter**

Adapter sẽ làm gần như tất cả các công việc cho RecyclerView nó kết nối Datasource vơi các View item.

1. Để vẽ được danh sách trên màn hình ,RecyclerView sẽ hỏi Adapter sẽ có tổng cộng bao nhiêu item. Và Adapter của chúng ta sẽ trả lời thông tin này ở trong hàm getItemCount().
2. Bất cứ khi nào RecyclerView quyết định nó cần tạo một ViewHolder và lưu trong bộ nhớ, nó sẽ gọi onCreateViewHolder (). Trong phương thức này, Adapter trả về bố cục xml .
3. Mỗi khi ViewHolder được tạo trước đó được sử dụng lại, RecyclerView sẽ bảo Adapter cập nhật dữ liệu của nó. Bạn tùy chỉnh quy trình này bằng cách ghi đè lên BindViewHolder().

**ItemAnimator**

RecyclerView.ItemAnimator sẽ tạo ra các hoạt ảnh cho ViewGroup khi các item thay đổi như thêm, sửa, xóa. DefaultItemAnimator được sử dụng cho các hoạt ảnh cơ bản và nó hoạt động khá ổn. Các bạn có thể xem thêm phần dưới này để có thêm thông tin.

### Sử dụng RecyclerView

Để sử dụng RecyclerView sẽ có 7 bước chính sau đây:

* Thêm RecyclerView support library vào gradle build file
* Định nghĩa ra model class để sử dụng data source
* Thêm RecyclerView vào trong activity mà bạn muốn hiển thị
* Tạo một tệp XML để xác định một item được biểu diễn như nào
* Tạo ra RecyclerView.Adapter và ViewHolder để gán dữ liệu cho các item
* Kết nối adapter với data source để đưa vào RecyclerView Mình sẽ trình bày chi tiết ở phần dưới ..

[**https://viblo.asia/p/su-dung-recyclerview-trong-android-RQqKLNe6l7z**](https://viblo.asia/p/su-dung-recyclerview-trong-android-RQqKLNe6l7z)

1. **Tab Layout, Viewpager**

**TabLayout provides a horizontal layout to display tabs.**

**View Page** user **for Slide between fragment susing ViewPager2 and Create swipe views with tabs using ViewPager2**

Benefits of ViewPager2

**Vertical orientation support**

**Right-to-left support**

**Modifiable fragment collections**

**DiffUtil**

<https://developer.android.com/guide/navigation/navigation-swipe-view>

<https://gist.github.com/mberberoglu/9524793>

<https://github.com/Pro-Grammerr/SlidingTabsUsingViewPager>

<https://developer.android.com/training/animation/vp2-migration>

**3. Permission**

Mục đích của permission là bảo vệ quyền riêng tư của người dùng Android. Các ứng dụng Android phải request các quyền truy cập nhảy cảm của người dùng (như danh bạ và sms), cũng như các tính năng nhất định của hệ thống (như camera và internet). Tuỳ thuộc vào tính năng, hệ thống có thể cung cấp permission tự động hoặc có thể nhắc người dùng phe duyệt yêu cầu.

**https://github.com/ParkSangGwon/TedPermission**

**4. Services and Notification**

Một Service là một thành phần (component) có thể thực hiện các hoạt động lâu dài trong background và nó không cung cấp một giao diện người dùng. Một thành phần khác của ứng dụng có thể start nó, và nó tiếp tục chạy trong background ngay cả khi người dùng chuyển sang ứng dụng khác.

***1 Foreground Service.***

Một Foreground Service thực hiện một số thao tác mà người dùng chú ý, có thể thấy rõ ràng. Ví dụ một ứng dụng nghe nhạc có thể chơi một bản nhạc và control nó bằng Foreground Service. Một điều bắt buộc là Foreground Service phải hiện thị một Notification. Foreground Service sẽ tiếp tục chạy ngay cả khi người dùng không tương tác với ứng dụng.

***2 Background Service.***

Một Background Service sẽ thực hiện các hoạt động mà không được người dùng chú ý trực tiếp. Ví dụ một ứng dụng sử dụng một service để thu gom bộ nhớ chẳng hạn thì service là một Background Service, hoạt động mà người dùng không cần thiết phải để ý.

***3 Bound Service.***

Một service được gọi là Bound khi một thành phần của ứng dụng ràng buộc với nó bởi lời gọi bindService(). Một Bound Service cung cấp một giao diện Client - Server cho phép các thành phần tương tác với nó: gửi yêu cầu, nhận kết quả và thậm chí là IPC. Một Bound Service chỉ chạy miễn là có một thành phần ràng buộc với nó. Có thể có nhiều thành phần ràng buộc với Bound Service cùng lúc, nhưng khi tất cả tháo bỏ ràng buộc (unbound) thì nó sẽ Destroy. Trước đây Service thường được chia là Started Service và Bound Service.

**5. Room**

Room là một thư viện bền bỉ của Android, nằm trong project Android Jetpack của Google. Theo tài liệu, Room cung cấp một lớp trừu tượng trên SQLite để cho phép truy cập cơ sở dữ liệu thông thạo trong khi khai thác toàn bộ sức mạnh của SQLite.

Các ứng dụng xử lý lượng dữ liệu có cấu trúc đáng kể có thể được hưởng lợi rất nhiều từ việc duy trì dữ liệu đó cục bộ. Trường hợp sử dụng phổ biến nhất là lưu vào bộ nhớ cache các phần dữ liệu có liên quan. Bằng cách đó, khi thiết bị không thể truy cập mạng, người dùng vẫn có thể duyệt nội dung đó khi họ đang offline. Mọi thay đổi nội dung do người dùng thực hiện sau đó sẽ được đồng bộ hóa với máy chủ sau khi thiết bị trực tuyến trở lại.

<https://viblo.asia/p/tim-hieu-ve-thu-vien-room-trong-android-07LKXLEEKV4>

Load ảnh

<https://viblo.asia/p/thu-vien-load-anh-trong-android-phan-2-su-khac-nhau-giua-glide-va-picasso-4P8566QG5Y3>

<https://viblo.asia/p/gioi-thieu-ve-thu-vien-load-anh-glide-va-so-sanh-voi-thu-vien-picasso-6BkGyKrER5aV>

**6. Fragment**

Each [Fragment](https://developer.android.com/reference/androidx/fragment/app/Fragment) instance has its own lifecycle. When a user navigates and interacts with your app, your fragments transition through various states in their lifecycle as they are added, removed, and enter or exit the screen.

**7. AppBarLayout**

If you want just to include the Toolbar without any scrolling effects you can use Toolbar. But if you want to make some scrolling effects like on the images you need to use AppBarLayout.

[**https://stackoverflow.com/questions/31328695/what-is-the-difference-between-appbarlayout-vs-toolbar**](https://stackoverflow.com/questions/31328695/what-is-the-difference-between-appbarlayout-vs-toolbar)

**8. Live Data**

LiveData is an observable data holder class. Unlike a regular observable, LiveData is lifecycle-aware, meaning it respects the lifecycle of other app components, such as activities, fragments, or services. This awareness ensures LiveData only updates app component observers that are in an active lifecycle state.

LiveData là một lớp nắm giữ dữ liệu và cho phép dữ liệu đó có thể quan sát được.

Không giống các kiểu dữ liệu có thể quan sát được khác, LiveData có thể nhận biết được vòng đời của các bên quan sát nó như activies, fragments, services để đảm bảo rằng LiveData chỉ gọi update các bên quan sát khi mà các bên đó còn hoạt động.

More Info: <https://viblo.asia/p/android-livedata-oOVlY1Yrl8W>

**9. Coroutines**

**Coroutines = Co + Routines** Ở đây, **Co** có nghĩa là hợp tác và **Routines** có nghĩa là các chức năng. Có nghĩa là khi các chức năng hợp tác với nhau, chúng ta gọi nó là Coroutines.

sử dụng **Kotlin Coroutines** trong android. Đây sẽ là 1 cách mới để quản lý background thread, code sẽ đơn giản hơn do giảm thiểu được số lượng callbacks.

Để xử lý task tốn nhiều thời gian mà không ảnh hưởng đến Main Thread thì đa số Dev đều dùng CallBack Pattern. Bằng cách này thì những task nào tốn nhiều thời gian sẽ được chạy dưới background thread. Khi task chạy xong thì sẽ được gọi lại và trả kết quả trong Main Thread.

Callbacks pattern là mẫu rất tốt. Tuy nhiên vẫn còn 1 vài hạn chế. Code sẽ rất nặng nếu sử dụng nhiều callback và sẽ trở nên khó đọc, khó debug.

Kotlin coroutines sẽ giúp bạn chuyển cấu trúc code CallBack thành code tuần tự (code nối tiếp nhau). Cấu trúc code tuần tự giúp dễ đọc hơn và với 1 vài ngôn ngữ lập trình cho phép sử dụng Exceptions.

Và tất nhiên, 2 cách code trên đều có chung 1 kết quả giống nhau.

Keyword suspend trong Kotlin để đánh dấu function hay function type chuyển thành Coroutines. Khi Coroutines được gọi (bằng suspend) thì sẽ thay vào đó là function sẽ được chờ cho đến khi có kết quả trả về như 1 function bình thường.

More info: [**https://viblo.asia/p/tim-hieu-kotlin-coroutines-trong-android-Do754pP35M6**](https://viblo.asia/p/tim-hieu-kotlin-coroutines-trong-android-Do754pP35M6)

[**https://viblo.asia/p/kotlin-coroutines-trong-android-ByEZkVy4KQ0**](https://viblo.asia/p/kotlin-coroutines-trong-android-ByEZkVy4KQ0)

**10. Retrofit, Okhttp, Gson**

Retrofit in Android – Always a topic that Android Developers always wonder about. So what is Retrofit? In short, Retrofit is a technology developed by Square, it is built on a lot of powerful technologies that allow to handle requests from client and server in the fastest and most effecient way. In short, Retrofit is a REST Client for Android and Java. Retrofit was developed to make the client-server connection process easy and fast. For Retrofit you can GET, POST, PUT, DELETE...

In simpler words, Retrofit is a type-safe HTTP Client for Java and Android. It makes it relatively easy to retrieve and upload JSON (or other structured data) through a REST-based web service. In Retrofit, you configure which converter is used to serialize the data. Normally, for JSON you use GSon, but you can add a custom converter to handle XML or other protocols. Retrofit uses the OkHttp library for HTTP requests.

Nói một cách dễ hiểu hơn, Retrofit chính là một type-safe HTTP Client cho Java và Android. Nó làm cho việc truy xuất và tải lên JSON (hoặc dữ liệu có cấu trúc khác) tương đối dễ dàng thông qua một dịch vụ web dựa trên REST. Trong Retrofit, bạn định cấu hình bộ chuyển đổi nào được sử dụng để tuần tự hóa dữ liệu. Thông thường, đối với JSON, bạn sử dụng GSon, nhưng bạn có thể thêm bộ chuyển đổi tùy chỉnh để xử lý XML hoặc các giao thức khác. Retrofit sử dụng thư viện OkHttp cho các yêu cầu HTTP.

More Info: <https://topdev.vn/blog/retrofit-trong-android/#what-is-retrofit>

**11. Room Database**

**12. Dependency injection**

Dependency Injection là một dạng design pattern được thiết kế với mục đích ngăn chặn sự phụ thuộc giữa các class, để khiến cho code dễ hiểu hơn, trực quan hơn, nhằm phục vụ cho mục đích bảo trì và nâng cấp code.

**Dependency injection (DI)** là một kỹ thuật lập trình giúp tách một class độc lập với các biến phụ thuộc. Với lập trình hướng đối tượng, chúng ta hầu như luôn phải làm việc với rất nhiều class trong một chương trình. Các class được liên kết với nhau theo một mối quan hệ nào đó. **Dependency** là một loại quan hệ giữa 2 class mà trong đó một class hoạt động độc lập và class còn lại phụ thuộc bởi class kia.

More Info: <https://codelearn.io/sharing/hieu-ro-ve-dependency-injection>

<https://viblo.asia/p/dependencies-injection-trong-android-phan-1-hieu-khai-niem-chinh-tu-ung-dung-cua-ban-07LKXokJ5V4>

\

**13. ViewBinding**

View Binding cung cấp cho bạn khả năng thay thế findViewById bằng các đối tượng ràng buộc được tạo ra để đơn giản hóa code, loại bỏ lỗi và tránh tất cả các boilerplate của findViewById.

Tại thời điểm này, có thể bạn đang tự hỏi rằng ViewBinding và DataBinding có giống nhau không? Cả hai đều generate class binding để ta có thể sử dụng tham chiếu tới view một cách trực tiếp, nhưng vẫn có một số điểm khác biệt.

* Để sử dụng DataBinding ta cần thêm tag layout vào file XML của layout.
* ViewBinding không support layout variable hay layout expression.

More Info:<https://viblo.asia/p/android-viewbinding-Ljy5Vq7zlra>

<https://viblo.asia/p/su-dung-view-binding-thay-the-cho-findviewbyid-trong-android-aWj53DwPK6m>

### **ViewBinding**

Only binding views to code.

### **DataBinding**

Binding data (from code) to views + ViewBinding (Binding views to code)

There are three important differences

1. With view binding, the layouts do not need a layout tag
2. You can't use viewbinding to bind layouts with data in xml (No binding expressions, no BindingAdapters nor two-way binding with viewbinding)
3. The main advantages of viewbinding are speed and efficiency. It has a shorter build time because it avoids the overhead and performance issues associated with databinding due to annotation processors affecting databinding's build time.

In short, there is nothing viewbinding can do that databinding cannot do (though at cost of longer build times) and there are a lot databinding can do that viewbinding can"t

**14. Data Binding**

DataBinding là một thư viện hỗ trợ để liên kết thành phần UI trong layout với nguồn dữ liệu sử dụng kiểu khai báo thay vì lập trình. Trước khi có thư viện này chúng ta gọi findViewById() để tìm kiếm TextView widget và ràng buộc tới Email như code ở dưới:

TextView txtView = findViewById(R.id.text\_demo);

txtView.setText(viewModel.getEmail());

Data binding là một trong những thành phần của Android Jetpack cho phép người lập trình liên kết dữ liệu tới UI.Nó xóa bỏ nhiều UI framework được gọi trong Activity của bạn,làm thêm dễ đọc và dễ bảo trì. Dưới đây là ví dụ 1 data binding đơn giản:

<TextView

android:text="@{viewmodel.userName}" />

Thật tuyệt vời chỉ với 1 dòng lệnh chúng ta có thể sử dụng username ở textview

More Info: <https://viblo.asia/p/ap-dung-databinding-trong-android-naQZRprm5vx>

**15. Architechture**

**MVC**

### **Model**

Model nghĩa là các dữ liệu cần thiết để hiển thị ở View. Model đại diện cho một tập hợp các lớp mô tả business logic (business model và data model). Nó cũng định nghĩa các business rules cho dữ liệu (nghĩa là cách mà dữ liệu thay đổi và được dùng)

### **View**

View đại diện cho các thành phần UI như XML, HTML. View sẽ hiển thị dữ liệu đã qua xử lý từ Controller. Model và View tương tác với nhau qua Observer pattern.

### **Controller**

Controller có trách nhiệm xử lý các yêu cầu (request) được gửi đến. Nó sẽ xử lý các dữ liệu của người dùng qua Model và trả về kết quả ở View

### **Đánh giá**

MVC rất tốt trong việc phân chia model và view. Chắc chắn sẽ dễ dàng test model vì nó không liên quan đến view và view không có gì nhiều để test (unit test). Tuy nhiên Controller vẫn còn nhiều hạn chế.

#### Mặt hạn chế của Controller

* Khả năng kiểm thử (test) - Controller bị ràng buộc với Android API nên sẽ khó để thực hiện unit test.
* Tính linh hoạt - Controller liên quan khá chặt chẽ với các view. Nếu chúng ta thay đổi view chúng ta sẽ phải thay đổi lại ở controller.
* Khả năng duy trì - Qua thời gian, controller sẽ ngày càng phình to ra do việc thêm code dẫn đến việc khó kiểm soát.

**MVP**

MVP là một User ***Interface Architectural Pattern*** (kiến trúc giao diện người dùng) được thiết kế để tạo điều kiện cho ***Automated Unit Testing*** và cải tiến ***Separation of Concerns*** trong việc trình bày logic (presentation logic).

**Mode**: là một interface xác định dữ liệu được hiển thị, hoặc dữ liệu này được thực hiện trong giao diện người dùng (UI)

**View**: là một interface thụ động dùng để hiện thị dữ liệu (là Model) và định hướng các lệnh người dùng (events) tới Presenter để Presenter hành động dựa trên các dữ liệu đó.

**Presenter**: hành động theo Model và View. Presenter lấy dữ liệu từ repositories (Model), sau đó định dạng dữ liệu và hiển thị lên View.

### **Đánh giá**

MVP sẽ "clear" hơn so với MVC. Chúng ta có thể dễ dàng viết unit test cho presenter vì nó không gắn với bất cứ view và API nào của Android và nó cũng cho phép chúng ta làm việc với các view khác miễn là view đó implement interface liên kết.

#### Mặt hạn chế của Presenter

* Khả năng duy trì - Cũng giống như Controller, Presenter dễ dàng bị thêm các business logic rải rác qua thời gian. Các developers sẽ rất khó để chia nhỏ presenter khi đã quá lớn.

**MVVM**

**Model**: Trong MVVM thì model sẽ thể hiện cho dữ liệu + trạng thái + các logic của đối tượng. **View** hoặc **Controller** vì vậy có thể được xử dụng lại dễ dàng

**View** : Liên kết các biến quan sát và hành động bởi **ViewModel**.Quan hệ giữa **View Model** và **View** là 1-n, nghĩa là nhiều **View** có thể liên kết với 1 **ViewModel**

**ViewModel**: **ViewModel là một abstraction của View. Nó sẽ lấy dữ liệu từ tầng Model, xửu lý UI logic sau đó hiển thị data có liên quan tới view. ViewModel sẽ không có bất kỳ behavior nào để tương tác với View. Như vậy để nhận biết khi nào cần hiển thị dư liệu, View sẽ đăng ký nhận notification từ ViewModel.**

### **Đánh giá**

Thực hiện Unit testing bây giờ sẽ rất dễ dàng, vì bạn thực sự không phụ thuộc vào view. Khi test bạn chỉ cần xác nhận rằng các biến observable được set thích hợp khi model thay đổi. Không cần phải tạo mockup cho view để test như ở mô hình MVP.

#### Mặt hạn chế của Presenter

* Khả năng duy trì - Khi view có thể gán cả biến và biểu thức, các logic không liên quan sẽ tăng dần theo thời gian, ảnh hưởng đến việc thêm code vào XML. Để phòng tránh điều nay, luôn luôn lấy giá trị trực tiếp từ ViewModel so với việc cố gắng tính toán và viết lại chúng khi gán biểu thức ở view.

**More Info:** <https://viblo.asia/p/mot-chut-ve-mvc-mvp-va-mvvm-3P0lPyMv5ox#_danh-gia-15>

<https://viblo.asia/p/android-architecture-tai-sao-chon-mvvm-hon-la-mvp-E375zXObZGW>