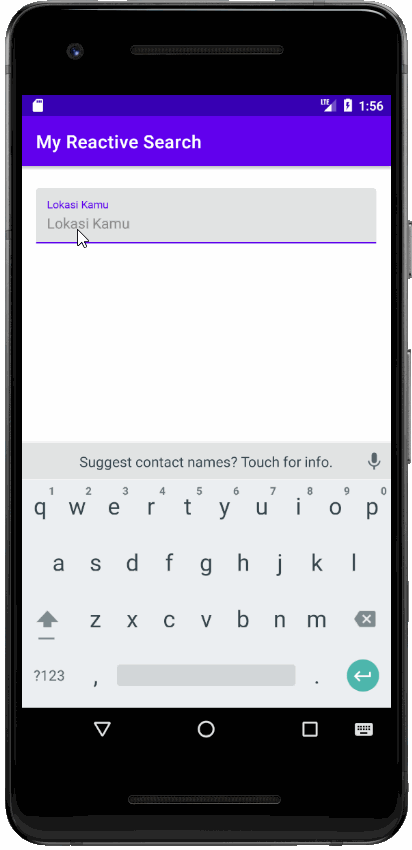
**Coroutine Flow**

**Tujuan**

Pada codelab kali ini Anda akan belajar membuat AutoCompleteTextView yang menerapkan reactive programming. Target Anda adalah menghasilkan output seperti di bawah ini:

[](https://www.dicoding.com/academies/165/tutorials/10349?from=10344)

**Logika Dasar**

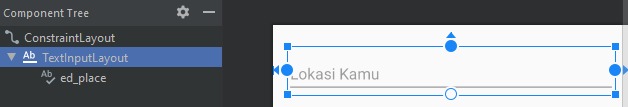
Menjalankan aplikasi → Menulis beberapa huruf untuk mencari tempat → Melakukan pencarian menggunakan Mapbox API → Menampilkan hasil di dalam AutoCompleteTextView → Pilih lokasi kamu.

**Codelab Reactive Search**

Pada codelab latihan kali ini Anda akan menggunakan Flow untuk membuat reactive search.

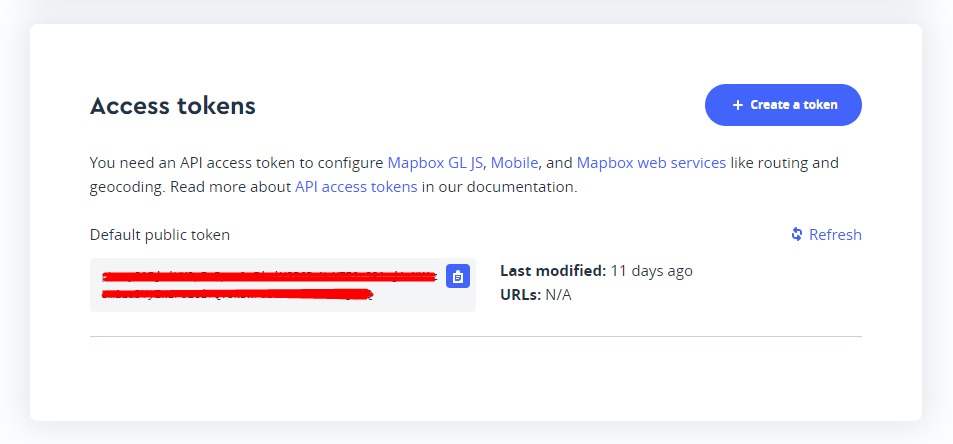
1. Buat Project baru di Android Studio dengan kriteria sebagai berikut:

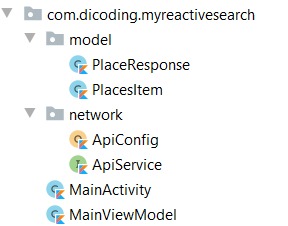
|  |  |
| --- | --- |
| Nama Project | **MyReactiveSearch** |
| Target & Minimum Target SDK | **Phone and Tablet, Api level 21** |
| Tipe Activity | **Empty Activity** |
| Activity Name | **MainActivity** |
| Language | **Kotlin** |

1. Pertama kita akan menambahkan library-library yang dibutuhkan terlebih dahulu di **build.gradle(Module: app)** seperti berikut:
   1. dependencies {
   2. ...
   4. **implementation "com.squareup.retrofit2:retrofit:2.9.0"**
   5. **implementation "com.squareup.retrofit2:converter-gson:2.9.0"**
   6. **implementation "com.squareup.okhttp3:logging-interceptor:4.9.0"**
   7. **implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.9"**
   8. **implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.9"**
   9. **implementation "androidx.activity:activity-ktx:1.1.0" //by viewModels()**
   10. **implementation "androidx.lifecycle:lifecycle-runtime-ktx:2.2.0" //lifecycleScope**
   11. **implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.2.0" //asLiveData**
   12. }
2. Selanjutnya kita mulai dengan membuat desain aplikasinya. Buka activity\_main.xml dan buat desain seperti berikut:  
   [](https://www.dicoding.com/academies/165/tutorials/10349?from=10344)
   1. <?xml version="1.0" encoding="utf-8"?>
   2. <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
   3. xmlns:app="http://schemas.android.com/apk/res-auto"
   4. xmlns:tools="http://schemas.android.com/tools"
   5. android:layout\_width="match\_parent"
   6. android:layout\_height="match\_parent"
   7. **android:layout\_marginTop="8dp"**
   8. **android:padding="8dp"**
   9. tools:context=".MainActivity">
   11. **<com.google.android.material.textfield.TextInputLayout**
   12. **android:layout\_width="match\_parent"**
   13. **android:layout\_height="wrap\_content"**
   14. **android:layout\_margin="8dp"**
   15. **android:hint="@string/your\_place"**
   16. **app:layout\_constraintEnd\_toEndOf="parent"**
   17. **app:layout\_constraintStart\_toStartOf="parent"**
   18. **app:layout\_constraintTop\_toTopOf="parent">**
   19. **<AutoCompleteTextView**
   20. **android:id="@+id/ed\_place"**
   21. **style="@style/Widget.MaterialComponents.AutoCompleteTextView.FilledBox"**
   22. **android:layout\_width="match\_parent"**
   23. **android:layout\_height="wrap\_content"**
   24. **android:hint="@string/your\_place" />**
   25. **</com.google.android.material.textfield.TextInputLayout>**
   27. </androidx.constraintlayout.widget.ConstraintLayout>

Jangan lupa untuk menambahkan *resource*teks-nya juga di **strings.xml**seperti berikut:

* 1. <resources>
  2. <string name="app\_name">My Reactive Search</string>
  3. **<string name="your\_place">Lokasi Kamu</string>**
  4. </resources>

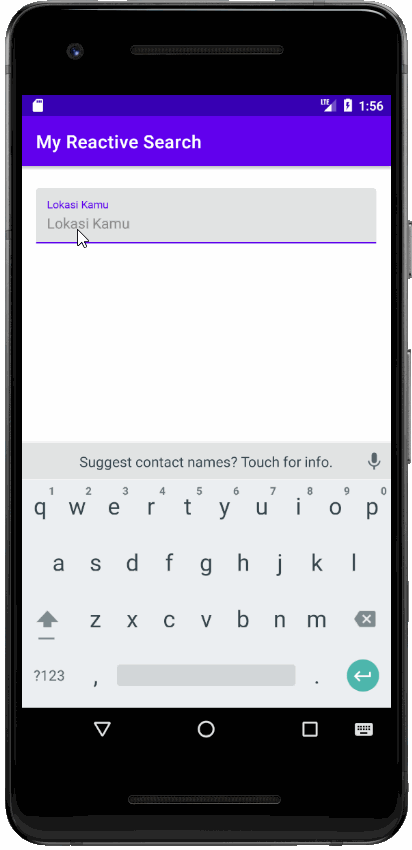
1. Sebelum melangkah lebih lanjut, di awal telah dijelaskan bahwa Anda akan membuat sebuah form lokasi dengan fitur AutoComplete. Untuk mendapatkan data suatu tempat, Anda akan menggunakan bantuan **Mapbox API**. Untuk itu silakan mendaftar terlebih dahulu pada halaman berikut: <https://account.mapbox.com/auth/signup/> untuk mendapatkan *access token*.
2. Setelah usai, lanjutkan dengan [Sign in](https://account.mapbox.com/auth/signin/) dan Anda akan mendapatkan *access token* pada halaman awal seperti ini:  
   [](https://www.dicoding.com/academies/165/tutorials/10349?from=10344)
3. Kemudian cobalah untuk melakukan suatu pencarian dengan menggunakan format berikut:  
   [**https://api.mapbox.com/geocoding/v5/mapbox.places/{KATA\_KUNCI\_TEMPAT}.json?access\_token={ACCESS\_TOKEN\_ANDA}&autocomplete=true**](https://api.mapbox.com/geocoding/v5/mapbox.places/icoding.json?access_token=pk.eyJ1IjoiYXJpZmFpemluIiwiYSI6ImNrYTI2c3R1cjAwNXAzbm1zaDYyZW1ra2cifQ.okSWF0zf58rWkhoVuYjShQ&autocomplete=true)  
     
   **Catatan:**  
   Anda dapat melihat documentation Search Service pada Mapbox pada halaman berikut : <https://docs.mapbox.com/api/search/#geocoding>Berikut ini adalah contoh hasil JSON setelah memanggil endpoint tersebut:  
   [](https://www.dicoding.com/academies/165/tutorials/10349?from=10344)  
   **Tips:**  
   Anda bisa menggunakan extension [JsonFormatter](https://chrome.google.com/webstore/detail/json-formatter/bcjindcccaagfpapjjmafapmmgkkhgoa?hl=id) supaya tampilan JSON menjadi lebih rapi.
4. Langkah selanjutnya kita buat model berdasarkan hasil JSON dari API tadi. Buat package baru dengan cara **klik kanan pada nama package → new → package → beri nama model**. Selanjutnya **buat dua kelas** model di dalamnya seperti berikut:
   * **PlacesItem**
     1. data class PlacesItem(
     2. @field:SerializedName("place\_name")
     3. val placeName: String
     4. )
   * **PlaceResponse**
     1. data class PlaceResponse(
     2. @field:SerializedName("features")
     3. val features: List<PlacesItem>
     4. )
5. Lanjut dengan membuat konfigurasi Retrofit. **Buat package baru** dengan nama **network**dan **buat dua kelas** berikut:
   * **ApiService**
     1. interface ApiService {
     2. @GET("mapbox.places/{query}.json")
     3. suspend fun getCountry(
     4. @Path("query") query: String,
     5. @Query("access\_token") accessToken: String,
     6. @Query("autocomplete") autoComplete: Boolean = true
     7. ): PlaceResponse
     8. }
   * **ApiConfig**
     1. object ApiConfig {
     2. private fun provideOkHttpClient(): OkHttpClient {
     3. return OkHttpClient.Builder()
     4. .addInterceptor(HttpLoggingInterceptor().setLevel(HttpLoggingInterceptor.Level.BODY))
     5. .connectTimeout(120, TimeUnit.SECONDS)
     6. .readTimeout(120, TimeUnit.SECONDS)
     7. .build()
     8. }
     10. fun provideApiService(): ApiService {
     11. val retrofit = Retrofit.Builder()
     12. .baseUrl("https://api.mapbox.com/geocoding/v5/")
     13. .addConverterFactory(GsonConverterFactory.create())
     14. .client(provideOkHttpClient())
     15. .build()
     16. return retrofit.create(ApiService::class.java)
     17. }
     18. }
6. Setelah Retrofit sudah siap, saatnya membuat ViewModel untuk memanggil retrofit. **Buat kelas baru** dengan nama **MainViewModel**dan tuliskan kode berikut:
   1. @FlowPreview
   2. @ExperimentalCoroutinesApi
   3. class MainViewModel : ViewModel() {
   5. private val accessToken = "ACCESS\_TOKEN\_ANDA"
   6. val queryChannel = BroadcastChannel<String>(Channel.CONFLATED)
   8. val searchResult = queryChannel.asFlow()
   9. .debounce(300)
   10. .distinctUntilChanged()
   11. .filter {
   12. it.trim().isNotEmpty()
   13. }
   14. .mapLatest {
   15. ApiConfig.provideApiService().getCountry(it, accessToken).features
   16. }
   17. .asLiveData()
   18. }

Jangan lupa untuk mengubah value pada variabel accessToken dengan token yang Anda dapatkan dari Mapbox.  
  
Sehingga struktur package sekarang menjadi seperti ini:  
[](https://www.dicoding.com/academies/165/tutorials/10349?from=10344)

1. Terakhir, masuk ke **MainActivity**dan inisialisasi viewmodel terlebih dahulu:
   1. class MainActivity : AppCompatActivity() {
   3. **val viewModel : MainViewModel by viewModels()**
   5. override fun onCreate(savedInstanceState: Bundle?) {
   6. super.onCreate(savedInstanceState)
   7. setContentView(R.layout.activity\_main)
   8. }
   9. }

Selanjutnya kirimkan query ketika form diisi dengan kode berikut:

* 1. class MainActivity : AppCompatActivity() {
  3. val viewModel : MainViewModel by viewModels()
  5. override fun onCreate(savedInstanceState: Bundle?) {
  6. super.onCreate(savedInstanceState)
  7. setContentView(R.layout.activity\_main)
  9. **val edPlace = findViewById<AutoCompleteTextView>(R.id.ed\_place)**
  11. **edPlace.addTextChangedListener(object : TextWatcher {**
  12. **override fun afterTextChanged(s: Editable?) {**
  13. **}**
  14. **override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {**
  15. **}**
  16. **override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {**
  17. **lifecycleScope.launch {**
  18. **viewModel.queryChannel.send(s.toString())**
  19. **}**
  20. **}**
  21. **})**
  22. }
  23. }

1. Terakhir, ambil hasil pencarian dari model dan tampilkan di AutoCompleteTextView seperti ini:
   1. class MainActivity : AppCompatActivity() {
   3. ...
   5. override fun onCreate(savedInstanceState: Bundle?) {
   7. ...
   9. **viewModel.searchResult.observe(this, Observer { placesItem ->**
   10. **val placesName = arrayListOf<String?>()**
   11. **placesItem.map {**
   12. **placesName.add(it.placeName)**
   13. **}**
   14. **val adapter = ArrayAdapter(this@MainActivity, android.R.layout.select\_dialog\_item, placesName)**
   15. **adapter.notifyDataSetChanged()**
   16. **edPlace.setAdapter(adapter)**
   17. **})**
   18. }
   19. }
2. Sehingga hasil akhir kode pada **MainActivity**akan terlihat seperti ini:
   1. class MainActivity : AppCompatActivity() {
   3. val viewModel : MainViewModel by viewModels()
   5. override fun onCreate(savedInstanceState: Bundle?) {
   6. super.onCreate(savedInstanceState)
   7. setContentView(R.layout.activity\_main)
   9. edPlace.addTextChangedListener(object : TextWatcher {
   10. override fun afterTextChanged(s: Editable?) {
   11. }
   13. override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {
   14. }
   16. override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
   17. lifecycleScope.launch {
   18. viewModel.queryChannel.send(s.toString())
   19. }
   20. }
   21. })
   23. viewModel.searchResult.observe(this, Observer { placesItem ->
   24. val placesName = arrayListOf<String?>()
   25. placesItem.map {
   26. placesName.add(it.placeName)
   27. }
   28. val adapter = ArrayAdapter(this@MainActivity, android.R.layout.select\_dialog\_item, placesName)
   29. adapter.notifyDataSetChanged()
   30. edPlace.setAdapter(adapter)
   31. })
   32. }
   33. }
3. Karena membutuhkan akses internet, jangan lupa menambahkan permission pada **AndroidManifest**seperti berikut:
   1. <?xml version="1.0" encoding="utf-8"?>
   2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
   3. package="com.dicoding.myreactivesearch">
   5. **<uses-permission android:name="android.permission.INTERNET"/>**
   7. <application
   8. ...
   9. </application>
   11. </manifest>
4. Hore! Sekarang Anda sudah dapat menjalankan projek ini dan melihat hasilnya:  
   [](https://www.dicoding.com/academies/165/tutorials/10349?from=10344)Anda sudah dapat menerapkan flow untuk melakukan pencarian API secara dinamis. Mantap!

**Bedah Kode**

**Konfigurasi Retrofit**

Secara keseluruhan, konfigurasi Retrofit hampir sama dengan modul yang sudah kita pelajari sebelumnya. Apa bedanya? Sekarang ApiService memiliki Query dan Path. Apa itu?

1. @GET("mapbox.places/{query}.json")
2. suspend fun getCountry(
3. @Path("query") query: String,
4. @Query("access\_token") accessToken: String,
5. @Query("autocomplete") autoComplete: Boolean = true
6. ): PlaceResponse

Lihatlah kembali endpoint API yang Anda gunakan :

[https://api.mapbox.com/geocoding/v5/mapbox.places/{KATA\_KUNCI\_TEMPAT}.json?access\_token={ACCESS\_TOKEN\_ANDA}&autocomplete=true](https://api.mapbox.com/geocoding/v5/mapbox.places/icoding.json?access_token=pk.eyJ1IjoiYXJpZmFpemluIiwiYSI6ImNrYTI2c3R1cjAwNXAzbm1zaDYyZW1ra2cifQ.okSWF0zf58rWkhoVuYjShQ&autocomplete=true)  
  
Kita bagi Endpoint tersebut menjadi 4 bagian:

* **Base URL** : **https://api.mapbox.com/geocoding/v5/** yaitu URL dasar yang tidak berubah-ubah, kita mendefinisikannya di dalam kelas **ApiConfig**.
* **Endpoint URL** :**mapbox.places/{KATA\_KUNCI\_TEMPAT}.json**, bagian yang di dalam tanda kurung kurawal merupakan Path karena kita perlu mengubah isinya setiap kali melakukan pencarian. Di dalam Retrofit untuk mengisinya menggunakan @Path.
* **Query 1** : **access\_token={ACCESS\_TOKEN\_ANDA}**, untuk awal query selalu menggunakan tanda tanya**(?)**. Di dalam Retrofit, untuk mengisinya menggunakan @Query.
* **Query 2** : **autocomplete=true**, untuk memisahkan antar query selalu menggunakan tanda dan**(&)**. Di dalam Retrofit caranya sama dengan query 1. Hanya saja karena kita selalu menginginkan autoComplete bernilai True maka kita bisa memberikan *default value* seperti cara di atas.

Apa beda **AutoComplete**yang bernilai True dan False?

* Jika memilih **False**maka kita hanya mencari tempat yang awalnya sesuai dengan yang kita ketik saja. Misal “coding” maka yang muncul hanya “Codingle”, CodingSans” dan “Codington”.
* Jika memilih **True**maka kita juga mencari tempat yang mengandung kata-kata yang kita ketik. Misal “coding, maka yang muncul bisa jadi “Dicoding”, “YukCoding,  “Coding Road”.

Anda bisa melihat documentation untuk lebih jelasnya : <https://docs.mapbox.com/api/search/#geocoding>

**BroadcastChannel dan Flow pada ViewModel**

Lihatlah kode berikut pada **MainViewModel**:

1. val queryChannel = BroadcastChannel<String>(Channel.CONFLATED)

BroadcastChannel digunakan untuk membuat channel yang bertujuan untuk berkomunikasi antar coroutine, seperti mengirim dan menerima value. Berbeda dengan Flow yang merupakan cold stream, Channel merupakan contoh dari hot stream. Sehingga Anda dapat mengirimkan data dari luar channel tersebut.  
  
Selanjutnya perhatikan tipe yang digunakan di sini yaitu CONFLATED, artinya hanya nilai terakhir saja yang disimpan, sedangkan untuk nilai yang sebelumnya dihiraukan. Secara garis besar, ia mirip seperti LiveData, hanya saja dibutuhkan Coroutine untuk membacanya

**Catatan :**  
BroadcastChannel merupakan fitur yang masih dalam perkembangan dan masih bisa berubah-ubah sebelum *stable release*. Karena itulah kita berikan *flag*seperti ini:

1. @FlowPreview
2. @ExperimentalCoroutinesApi

Kemudian untuk mengirimkan broadcast dari **MainActivity**, cukup menggunakan fungsi send() seperti berikut:

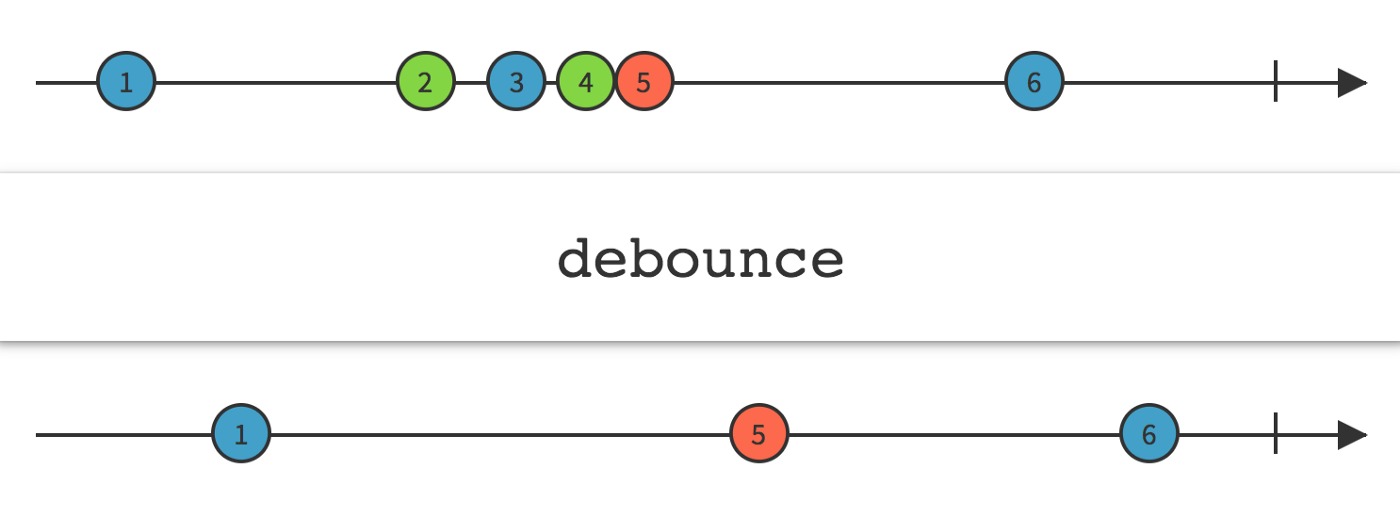
1. lifecycleScope.launch {
2. viewModel.queryChannel.**send**(s.toString())
3. }

Untuk melakukan fungsi ini Anda harus melakukannya di dalam Coroutine. Karena itulah kita menggunakan lifecycleScope yang didapat dari library [lifecycle-runtime-ktx](https://androidx.tech/artifacts/lifecycle/lifecycle-runtime-ktx/). lifecycleScope digunakan untuk membuat scope coroutine yang *aware*terhadap lifecycle. Sehingga ketika lifecycle dihapus -seperti dalam keadaan onPause/onStop- maka Coroutine juga akan dibersihkan. Alhasil, aplikasi terhindar dari *memory leak*.

Selanjutnya lihatlah kode berikut pada **MainViewModel**:

1. val searchResult = queryChannel.asFlow()
2. .debounce(300)
3. .distinctUntilChanged()
4. .filter {
5. it.trim().isNotEmpty()
6. }
7. .mapLatest {
8. ApiConfig.provideApiService().getCountry(it, accessToken).features
9. }
10. .asLiveData()

Berikut ini adalah fungsi dari masing-masing operator:

* asFlow : Membuat kita bisa mengubah channel menjadi Flow.
* debounce : Memastikan agar eksekusi selanjutnya berjalan jika ada jeda 300 millisecond. Hal ini akan sangat bermanfaat ketika melakukan pencarian. Dengan menggunakan debounce, Anda mencegah aplikasi untuk melakukan request API di setiap kali mengetikkan huruf, namun menunggu kata tersebut diketik terlebih dahulu. Hal ini tentunya akan menghemat *bandwith*dan memberikan *user experience* yang menyenangkan bagi pengguna.  
  [](https://www.dicoding.com/academies/165/tutorials/10349?from=10344)
* distincUntilChanged : Berfungsi kala request sebelumnya dengan sekarang, sama. Jadi, tidak perlu melakukan request lagi. Misalnya pengguna menulis “dico”, kemudian menghapus satu huruf menjadi “dic”, kemudian menulis lagi menjadi “dico”. Karena setelah 300 millisecond nilainya sama, maka aplikasi tidak akan melakukan request lagi.
* filter : melakukan filter jika yang diketik hanya spasi atau masih kosong, sehingga bisa mencegah melakukan request yang tidak berguna.
* mapLatest : Berfungsi kala  melakukan transformasi flow menjadi data yang baru dengan hanya menghasilkan nilai terakhir, sehingga ketika ada nilai baru maka data yang lama akan digantikan.
* asLiveData : Mengubah Flow menjadi LiveData, merupakan extension yang di dapat dari library [lifecycle-livedata-ktx](https://androidx.tech/artifacts/lifecycle/lifecycle-livedata-ktx/).

Selamat Anda sudah belajar membuat reactive search dengan menerapkan Reactive Programming. Masih banyak hal lain yang bisa kita eksplorasi dengan menggunakan Coroutine Flow. Kita akan melanjutkan perjalanan kita bersama Flow pada modul berikutnya. Semangat!

Untuk mendalami lebih lanjut Anda dapat melihat tautan berikut:

* [Coroutine MapLatest](https://github.com/square/retrofit/tree/master/retrofit-adapters/rxjava2)
* [Deepdive into Reactive Spring with Coroutines and Kotlin Flow by Sébastien Deleuze](https://www.youtube.com/watch?v=BoidEr_ZCGc)
* [Lessons learnt using Coroutines Flow in the Android Dev Summit 2019 app](https://medium.com/androiddevelopers/lessons-learnt-using-coroutines-flow-4a6b285c0d06)

Source code dapat Anda unduh di:

[Source Code MyReactiveSearch](https://github.com/dicodingacademy/a165-android-expert-labs/tree/master/MyReactiveSearch)