



**BINAR**  
ACADEMY

# Navigation Component

**Gold** - Chapter 3 - Topic 5

---

**Selamat datang di Chapter 3 Topic 5 online course  
Android dari Binar Academy!**



## Hii Teman Teman 🙌

Pada topik sebelumnya, kita sudah belajar konsep perpindahan antar screen dengan dengan **Intent**.

Nah, di **Topik 5** ini kita mendalami metode lain untuk perpindahan antar screen, yaitu **Navigation Component**.

Yuk, lanjut!



**Detailnya, kita bakal bahas hal-hal berikut ini:**

- Pengenalan Android Jetpack
- Pengenalan Navigation Component
- Implementasi Navigation Component





**Ngomongin pindah memindah, untuk membuat sistem bernavigasi antar screen, ada cara lain loh selain pake Intent.**

**Apakah metode navigasi itu?**

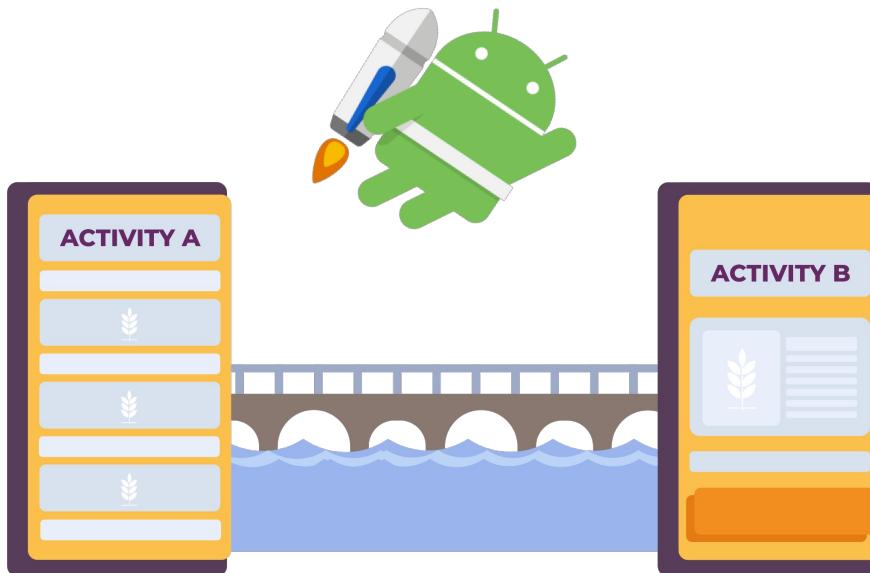


## Masih inget kan sama konsep Perpindahan Antar Screen?

Inget gambar disamping?

Untuk menjalankan Activity dalam aplikasi Android, sistem perlu menjalankan services untuk membawa data ke activity yang berbeda.

Nah, di Topik sebelumnya, kita udah memperdalam tentang metode Intent. Tapi, ternyata masih ada lagi loh fitur yang bisa kita pake, salah satunya adalah Navigation Component~



Navigation Component ini adalah salah satu fitur baru yang ada dalam paket (package) servis baru Google, yaitu Android Jetpack.

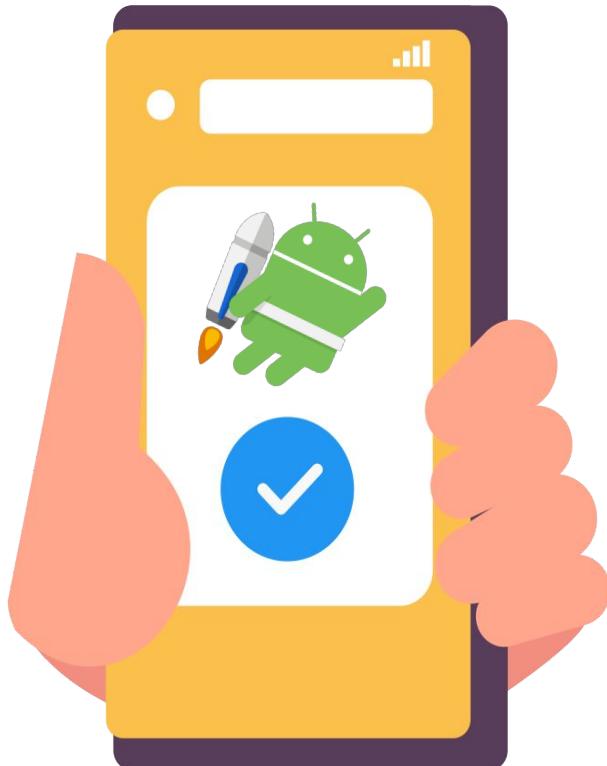
Fitur dari Android Jetpack ini akan kita bahas beberapa.

Namun, untuk memantapkan skill kita di bidang navigasi activity ini, kita bakal perdalam khusus tentang fitur Navigation component dalam Android Jetpack~



Ternyata banyak ya jalan menuju Roma~

Sebelum kita masuk ke pembahasan Navigation Component, kita akan bahas sekilas tentang **Android Jetpack!**



Android Jetpack adalah sebuah inovasi dari Google.

Membuat para developer Android terbantu dan cepat, dalam mengembangkan sebuah aplikasi yang berkualitas dan tangguh dengan sekumpulan library, tools dan panduan di dalamnya.

Berdasarkan hal itu, Google juga memperkenalkan 'Architecture Components' yang sebelumnya pernah kita bahas.

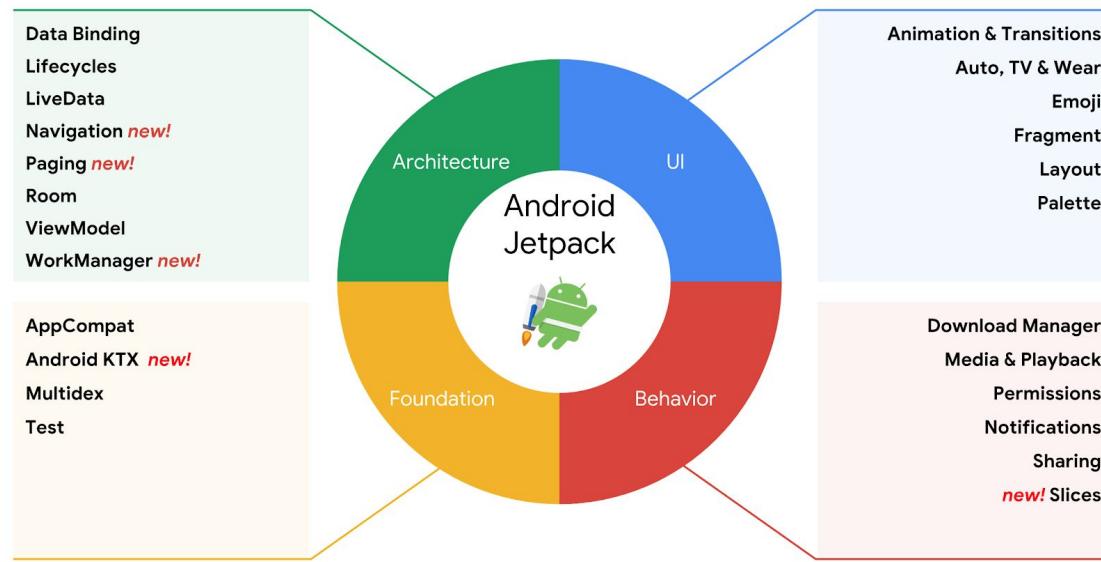


Tentunya package ini punya beberapa kelebihan dibandingkan dengan metode lain, yaitu :

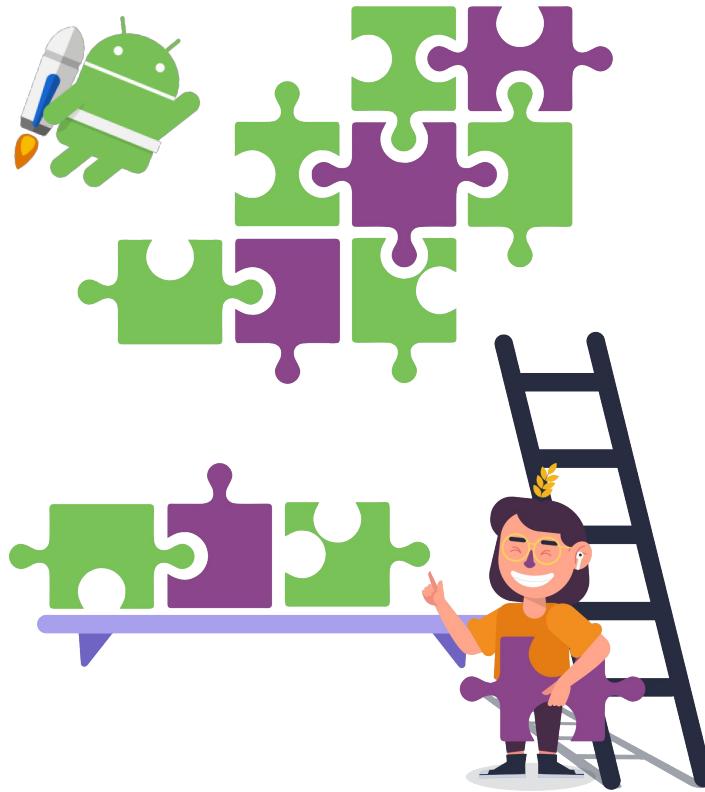
- Sangat mempercepat development aplikasi Android.
- Menghapus code boilerplate.
- Membuat aplikasi berkualitas tinggi dan kuat karena memungkinkan lebih sedikit error dan mengurangi *memory leak*.
- Bisa melihat alur navigasi dengan lebih jelas dan mudah karena adanya visualisasi dari Android Studio.



Komponen Android Jetpack menyatukan **Support Library** dan **Architecture Components** yang ada, lalu menyusunnya menjadi empat kategori seperti pada Gambar.



Untuk melihat lebih lengkapnya, ada di <https://developer.android.com/jetpack/>



Komponen Android Jetpack disediakan sebagai **library "unbundled"** yang bukan merupakan bagian dari platform Android dasar.

Ini berarti kita dapat memakai setiap komponen sesuai dengan kebutuhan project kita.

Library Android Jetpack unbundled telah dipindahkan ke library androidx.\*



Data Binding

Lifecycles

LiveData

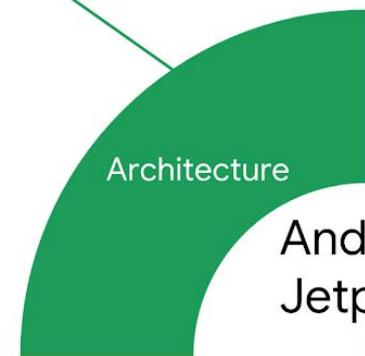
Navigation *new!*

Paging *new!*

Room

ViewModel

WorkManager *new!*



Setiap kategori, punya buanyak banget komponen. Supaya makin jadi master Android, komponen-komponen itu bakal kita bahas di chapter lainnya.

Nah di topik kali ini, kita coba fokus untuk mempelajari tentang **Navigation Component**.

Yap, kalo kamu liat digambar samping ada tulisan *New*. Artinya fitur ini masih baru di update sama Google~



**Kembali ke laptop!**

Karena udah tercerahkan tentang Jetpack, kita mulai yuk bahas **Navigation Component!**



## Apa itu Navigation Components?

Definisinya berkaitan sama Activity.

Coba deh, baik Activity maupun Fragment, sesuai namanya, adalah kumpulan aktivitas yang terjadi di aplikasi kita untuk suatu siklus tertentu.

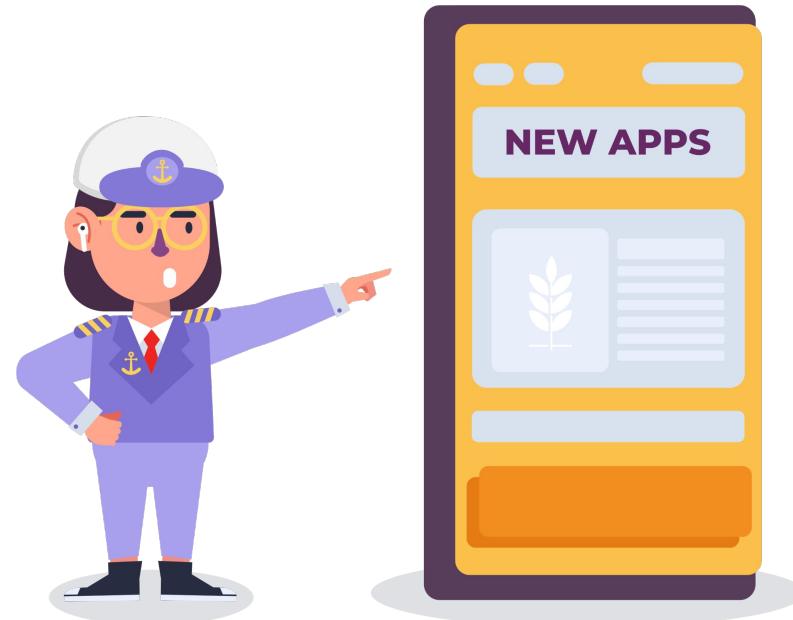
Activity jadi nggak fleksibel, dan kurang fleksibilitas ini membuat kita harus berbagi data antara screen untuk membuat satu fitur yang utuh..





Nggak cuma itu, transisi menjadikan **Activity sebagai arsitektur yang kurang ideal untuk membangun navigasi dalam aplikasi kita.**

Akhirnya, Google memperkenalkan komponen Navigation sebagai framework untuk menyusun UI dalam aplikasi kita, dengan fokus pada pembuatan aplikasi single activity sebagai arsitektur yang dipilih.

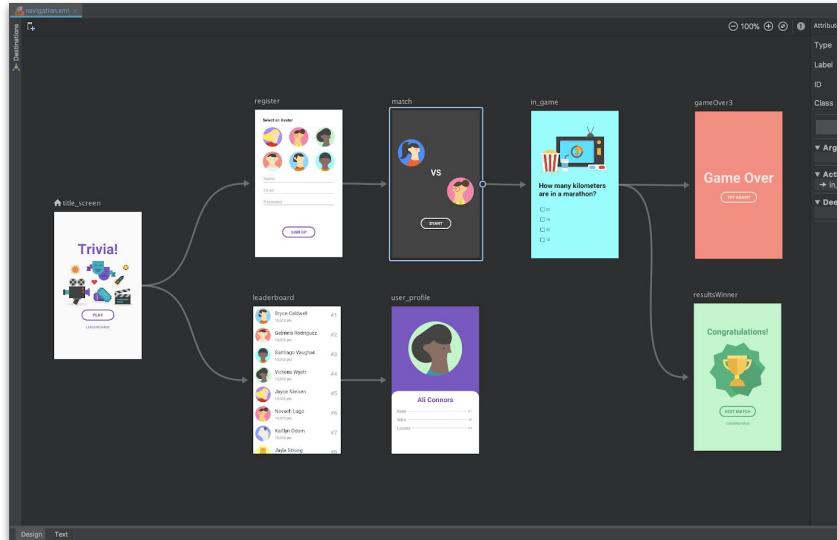




Navigation Component mendukung bawaan untuk fragment. Sehingga kita jadi bisa dapetin semua manfaat Architecture Components seperti **“Lifecycle”** dan **“ViewModel”**.

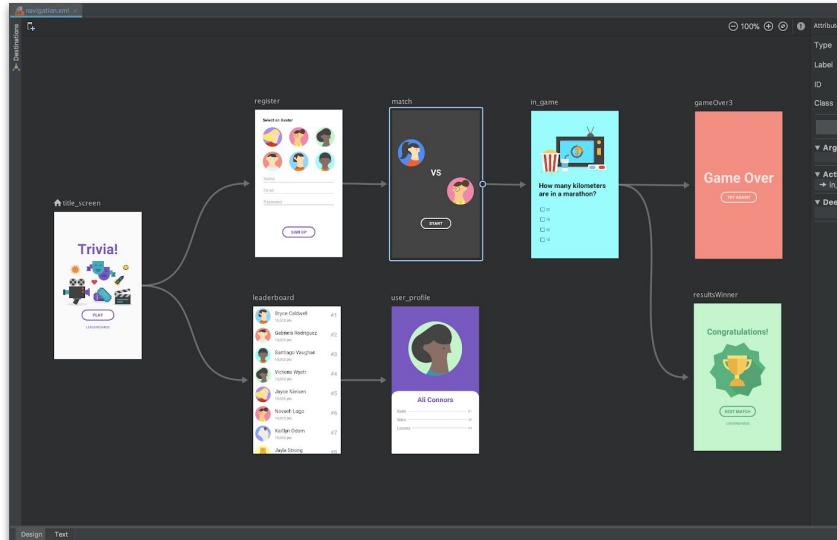
Sistem ini juga memungkinkan Navigation untuk menangani kompleksitas **“FragmentTransaction”** untuk kita.





Selanjutnya, komponen Navigation memungkinkan kita untuk menyatakan transisi/tujuan.

Secara otomatis membangun flow yang tepat, termasuk dukungan penuh untuk **deep link**, dan menyediakan bantuan untuk menghubungkan Navigation ke widget UI yang sesuai. Misalnya kayak navigation drawer dan bottom navigation.



Wuess ada lagi. Navigation Editor di Android Studio versi 3.2(Canary build) ke atas memungkinkan kita melihat dan mengelola properti **navigasi** kita **secara visual**.



Nggak cuma ngebuat navigasi jadi lebih simpel, Navigation Component juga memberikan keuntungan seperti :

- Menangani *fragment transaction*.
- Menangani aksi *Up* dan *Back* secara default.
- Menyediakan standar resources untuk animations dan transitions.
- Menerapkan dan menangani *deep linking*.





- Menerapkan Navigation UI patterns, seperti navigation drawer dan bottom navigation tanpa berpusing ria.
- Adanya fitur Safe Args sebuah plugin Gradle yang menyediakan keamanan saat bernavigasi dan mengirimkan data antar destinations. Bisa dibilang sebagai 'pengganti extra' pada intent.
- Support ViewModel  
Kita dapat mengatur ViewModel ke navigation graph untuk berbagi data terkait UI antara graph destinations.



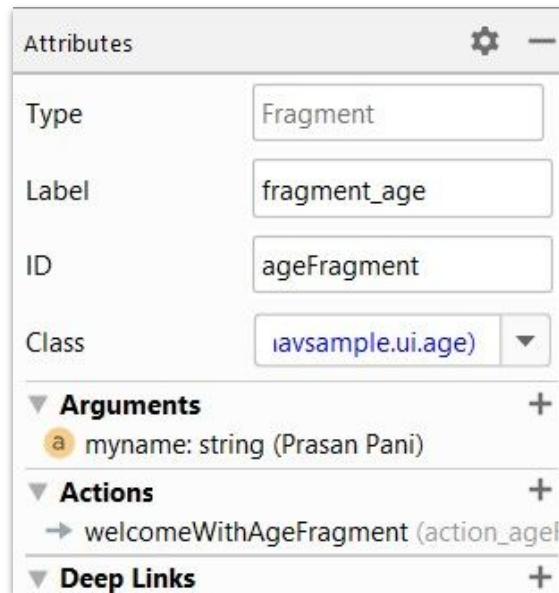


## Unsur dalam Navigation Component

Komponen navigation terdiri dari tiga bagian utama yang perlu kita ketahui:

1. **Navigation Graph**
2. **NavController**
3. **NavHost**

Yuk kita bahas satu persatu~



The screenshot shows the 'Attributes' panel from the Android Studio UI. It contains the following fields:

- Type: Fragment
- Label: fragment\_age
- ID: ageFragment
- Class: iavsample.ui.age)

Below these fields, there are three expandable sections:

- Arguments**: Contains an argument named "myname: string (Prasan Pani)".
- Actions**: Contains an action named "welcomeWithAgeFragment (action\_age)".
- Deep Links**: This section is currently collapsed.



### 1) Navigation Graph:

Sebuah file XML yang berisikan tentang semua informasi terkait relasi navigasi di satu lokasi yang terpusat. Ini mencakup :

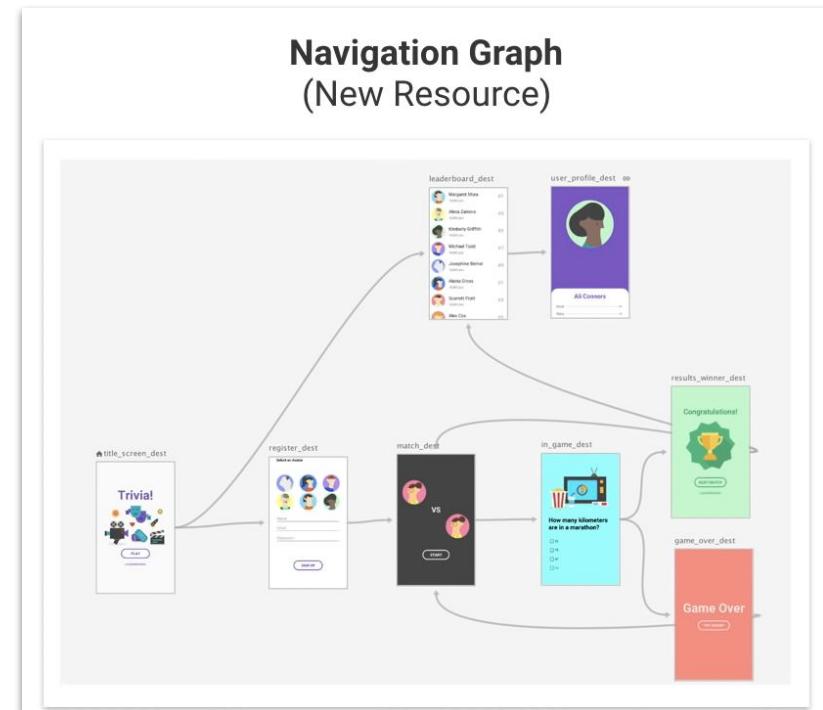
- **Destination**, merupakan Fragment atau Activity yang saling berkaitan atau berelasi.
- **Action**, merupakan navigasi/relasi dengan tujuan (destination) ke fragment/activity yang akan kita tuju.

Kita juga dapat mengatur animation (menggunakan animasi dari folder anim), launch option dan juga pop behaviour (action saat menekan button back).





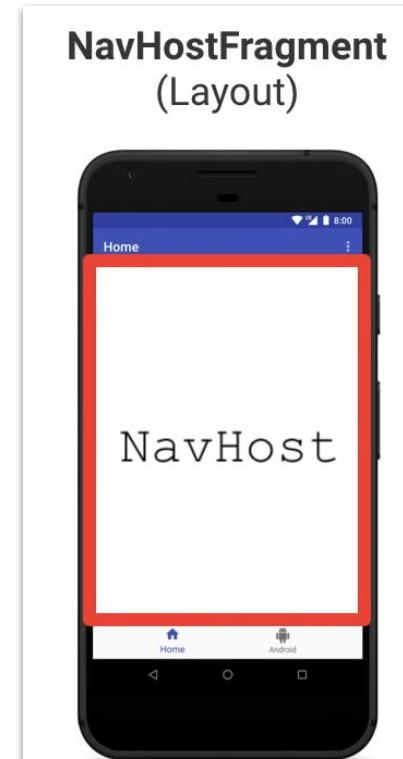
- **Argument**, digunakan untuk mendefinisikan data yang akan dikirim antar fragment/activity.
- **DeepLinks**, digunakan untuk bisa membuka suatu halaman menggunakan sebuah URL.





### 2. NavHost

Container kosong yang menampilkan tujuan dari *navigation graph* kita. Biasanya terletak di Activity dengan layout yang berisi fragment.





### 3. NavController

Object yang mengelola navigasi aplikasi dalam sebuah NavHost. NavController **mengatur pertukaran konten tujuan** di NavHost saat user menggunakan aplikasi kita.

**NavController**  
(Fragment)

```
findNavController().navigate(<Destination or Action id>)
```



Saat kita bernaavigasi dalam aplikasi, kita memberi tahu NavController bahwa kita ingin **menavigasi di sepanjang jalur tertentu dalam navigation graph kita** atau langsung ke tujuan tertentu.

NavController kemudian menunjukkan tujuan yang sesuai di NavHost.

## NavController (Fragment)

```
findNavController().navigate(<Destination or Action id>)
```

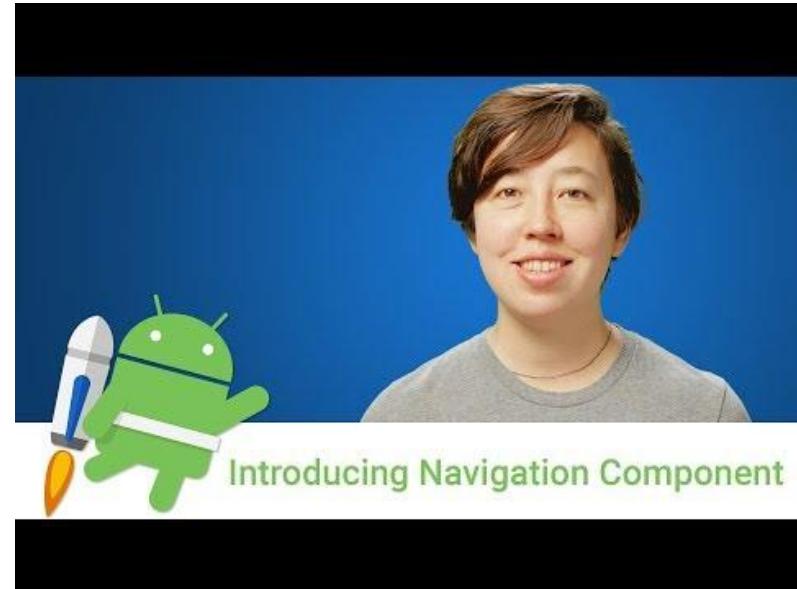


### Gimana? Sudah kebayang?

Fitur ini sebenarnya cukup baru di Android Studio.

Kalo kita paham fitur ini, kamu bakal lebih mudah melakukan deklarasi navigasi antar screen. Kenapa? karena UI-nya memudahkan kita melihat relasi antar screen dalam satu halaman kerja yang sama.

Kalau masih belum paham, bisa saksikan video disamping yah~



[Link Video Klik disini](#)



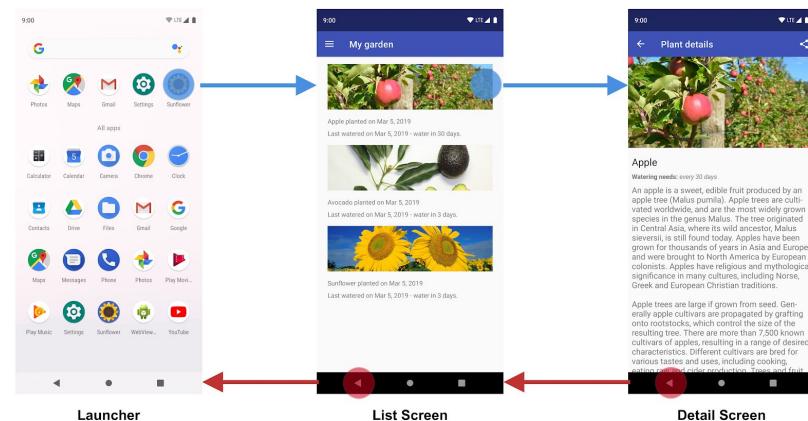
Definisi dasar dari Navigation Component udah, sekarang kita gali lebih dalam tentang **Konsep Navigation Component.**



## Prinsip Navigation

Untuk membuat Prinsip Navigasi terbaik, ada 5 Prinsip yang perlu kita ingat:

1. Start Destination yang Tetap
2. Navigation state dinyatakan sebagai destination stack
3. Up dan Back button identik
4. Button Up tidak pernah membawa kita keluar dari aplikasi
5. Deep linking menyimulasikan navigasi manual



Yuk kita bahas satu satu~

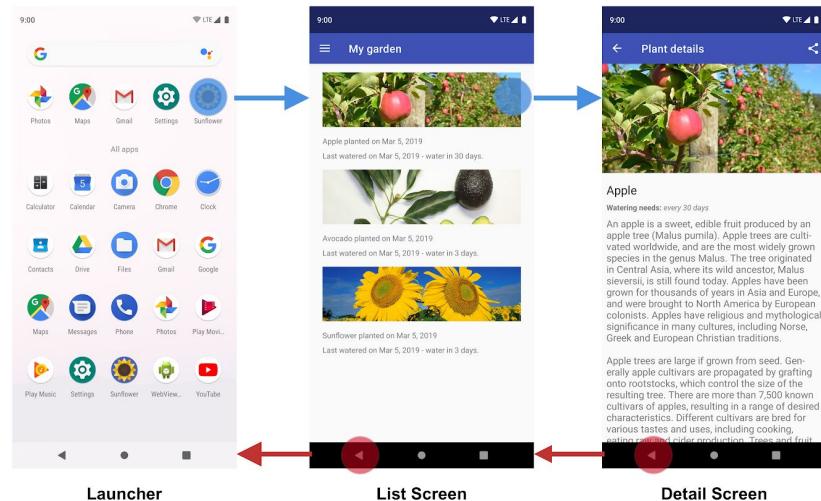


## 1. Start Destination yang Tetap

Setiap aplikasi yang kita buat harus memiliki **Start Destination yang tetap.**

Destination ini adalah layar pertama yang akan dilihat oleh user saat menjalankan aplikasi.

Destination juga merupakan layar terakhir yang dilihat oleh user saat kembali ke launcher setelah menekan tombol *Back*.



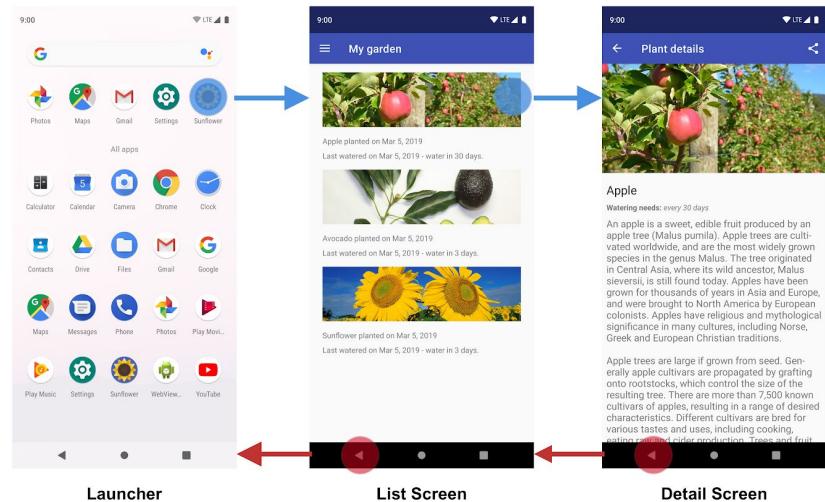


Coba deh kamu liat gambar disamping

Di gambar itu ada launcher. Layar pertama yang akan dilihat oleh user adalah *List Screen*, yang berisi list tanaman.

Layar ini juga **merupakan layar terakhir yang seharusnya dilihat user sebelum keluar dari aplikasi.**

Kalo kamu teken button Back dari list screen, maka mereka akan kembali ke launcher.

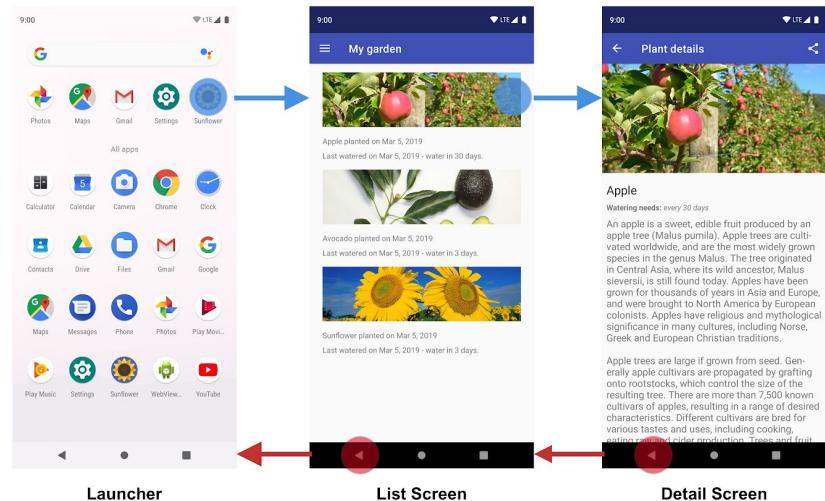




## 2. Navigation state dinyatakan sebagai destination stack

Saat aplikasi pertama kali diluncurkan, task baru akan dibuat untuk user dan aplikasi akan menampilkan tujuan awalnya.

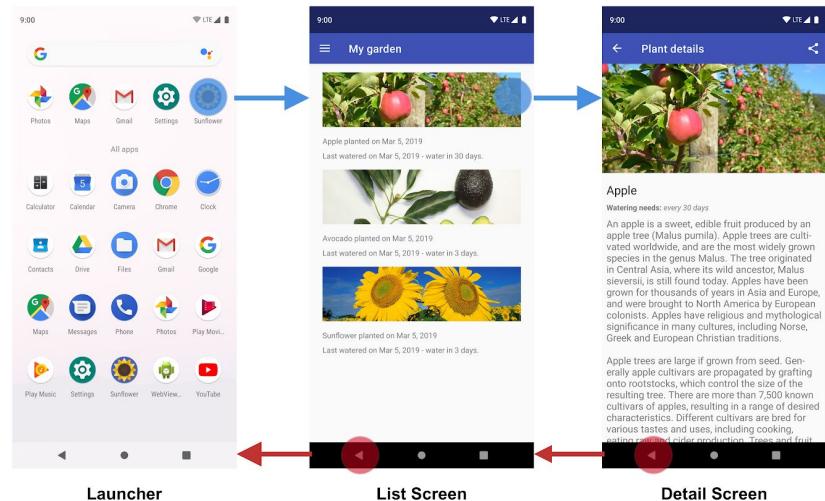
Hal ini menjadi **dasar dari “back stack”** dan merupakan **dasar untuk navigation state**.





Bagian atas stack adalah layar saat ini, dan tujuan sebelumnya dalam stack menunjukkan histori halaman yang pernah dibuka oleh kita.

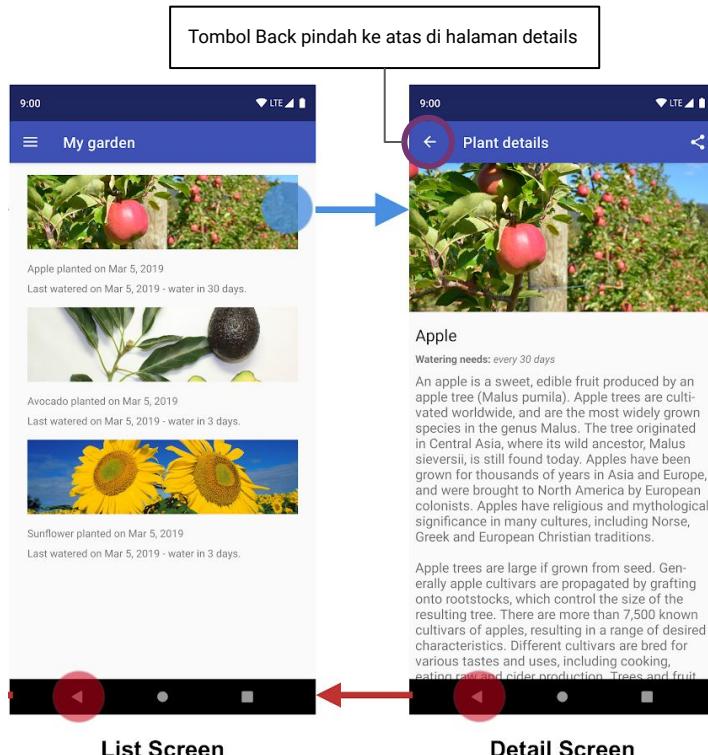
Tujuan awal aplikasi dalam back stack selalu berada di bagian bawah stack.





### 3. Up dan Back button identik

Tombol Back muncul di tombol navigasi sistem yang **biasanya ada di bagian bawah layar** dan digunakan untuk melakukan navigasi dalam urutan sesuai riwayat user.

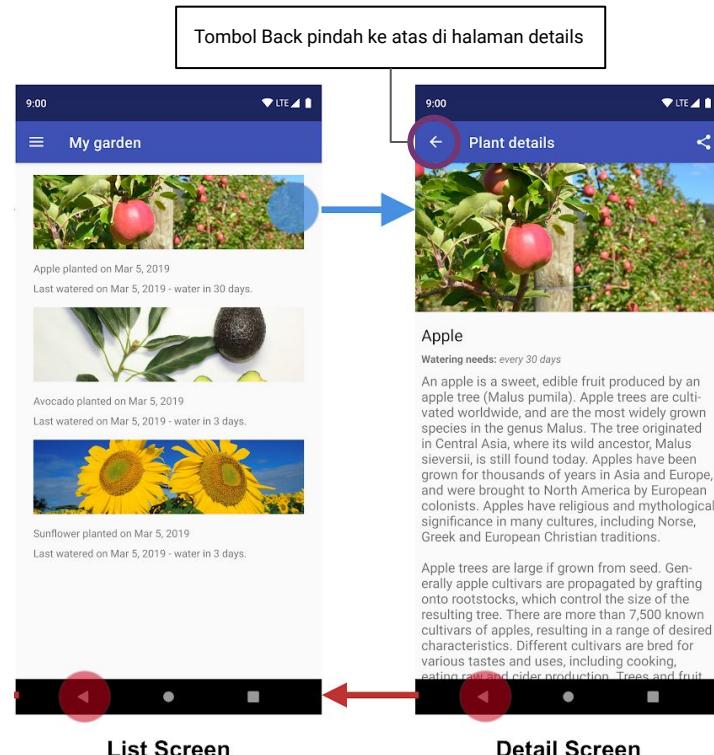




Saat kita menekan button *Back* pada tujuan saat ini (yang tampil dilayar) akan dimunculkan di bagian atas *back stack*.

Ketika kita berada di halaman detail. Tombol Back atas **akan muncul dalam panel aplikasi di bagian atas layar**. Pada saat kita menekan tombol Back atas, kita akan menavigasi ke tujuan sebelumnya.

Dalam aplikasi, button Up dan Back memiliki perilaku yang sama/identik.

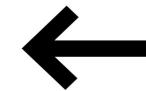




#### 4. Button Up tidak pernah membawa kita keluar dari aplikasi

Jika user berada di *Start Destination* aplikasi, **button Up tidak akan muncul**. Karena memang seharusnya *Up Button* tidak akan pernah membawa kita keluar dari aplikasi.

Namun button Back pada device akan ditampilkan dan tetap akan membawa kita keluar dari aplikasi.





Kayak gini nih.. saat aplikasi A diluncurkan menggunakan **deep link** pada fitur aplikasi lain, button Up akan melakukan transisi pada user untuk kembali ke aplikasi A dan bukan ke aplikasi yang memicu deep link tersebut.

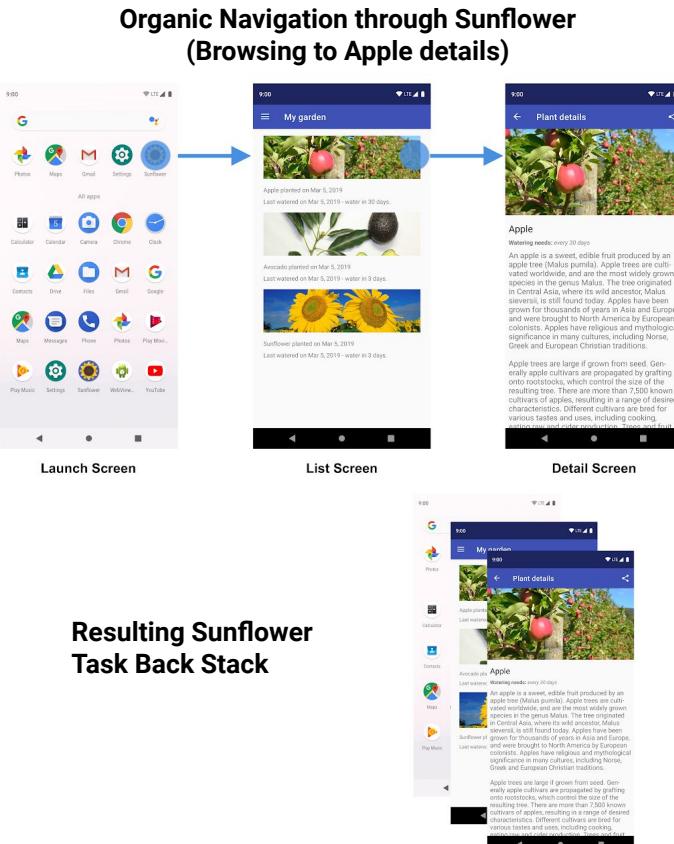
Tapi, kalo kamu teken tuh button Back, Ini akan membawa kita kembali ke aplikasi lain.





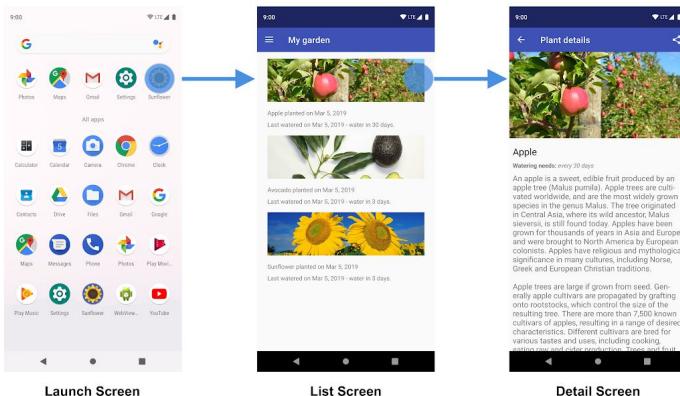
## 5. Deep linking menyimulasikan navigasi manual

Baik saat melakukan deep linking atau navigasi secara manual ke tujuan tertentu, kita bisa **menggunakan button Up untuk melakukan navigasi dalam tujuan untuk kembali ke tujuan awal.**





## Organic Navigation through Sunflower (Browsing to Apple details)



Saat melakukan deep linking ke tujuan dalam task aplikasi, Back stack yang ada untuk task aplikasi akan dihapus dan diganti dengan Back stack yang ditautkan.

## Resulting Sunflower Task Back Stack

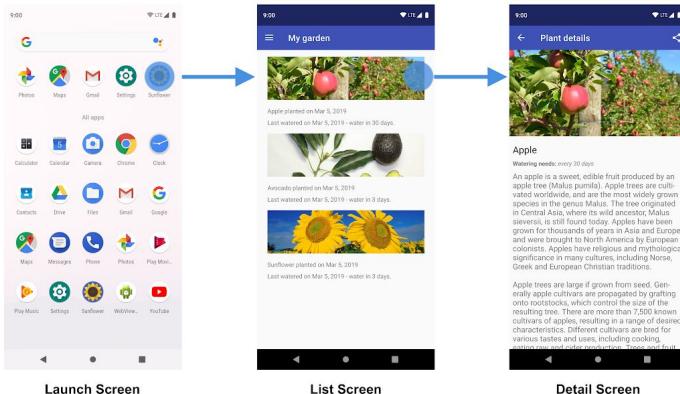




Sebagai contoh, anggaplah user sebelumnya telah meluncurkan aplikasi dari launch screen dan membuka layar detail pada apel.

Saat ini layar akan menunjukkan adanya task dengan layar paling atas yang merupakan layar detail yaitu apel.

## Organic Navigation through Sunflower (Browsing to Apple details)



## Resulting Sunflower Task Back Stack

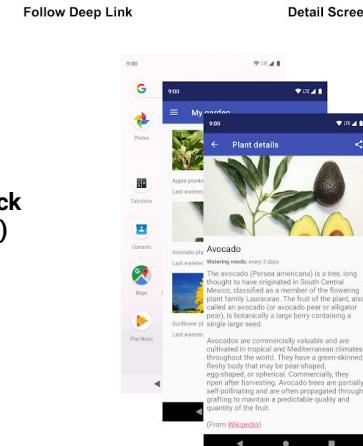
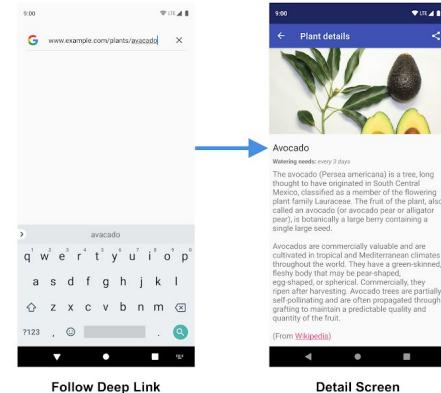




Pada tahap ini, user dapat mengetuk button Layar Utama untuk menempatkan aplikasi di background.

Selanjutnya, anggaplah aplikasi ini memiliki fitur deep link yang memungkinkan user langsung membuka layar detail tanaman tertentu berdasarkan namanya.

## Deep Link ke Hal. Plant Details (Linking to Avocado Details)



## Hasilnya, Sunflower Task Back Stack (Setelah User masuk ke Deep link)



## Prinsip Deep Linking

Deep linking adalah teknologi yang memungkinkan pengguna untuk membuka aplikasi melalui tautan yang dikirim melalui pesan instan, email, atau media sosial.

Dengan menggunakan deep linking, pengguna dapat langsung diarahkan ke halaman spesifik dalam aplikasi tanpa perlu melalui proses navigasi standar.

Hal ini memberikan pengalaman pengguna yang lebih intuitif dan efisien.

Untuk mencoba fitur deep linking pada aplikasi, silakan ikuti langkah-langkah berikut:

- Buka aplikasi yang telah dilengkapi dengan deep linking.

- Klik ikon browser atau tautan yang mengandung URL dengan format `http://www.example.com/plants/avacado`.

- Aplikasi akan otomatis membuka halaman "Plant details" yang menampilkan informasi tentang buah avokado.

- Pada halaman tersebut, klik tombol "Follow Deep Link" untuk melihat bagaimana tautan tersebut mempengaruhi stack navigasi.

- Setelah mengklik tombol tersebut, pengguna akan diarahkan ke halaman awal aplikasi.

- Stack navigasi akan menunjukkan bahwa pengguna baru saja masuk ke aplikasi melalui tautan yang dikirim melalui pesan instan.

- Hal ini menunjukkan bahwa deep linking berhasil mempertahankan state aplikasi dan memungkinkan pengguna untuk kembali ke halaman sebelumnya dengan mudah.

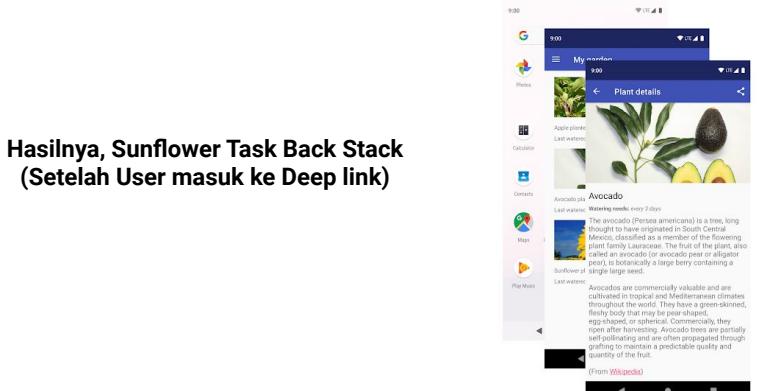
- Terakhir, pengguna dapat melanjutkan navigasi dalam aplikasi sesuai dengan tujuan mereka.

- Demikian penjelasan tentang prinsip deep linking pada aplikasi Android Jetpack. Semoga bermanfaat!

## Deep Link ke Hal. Plant Details (Linking to Avocado Details)



Follow Deep Link      Detail Screen



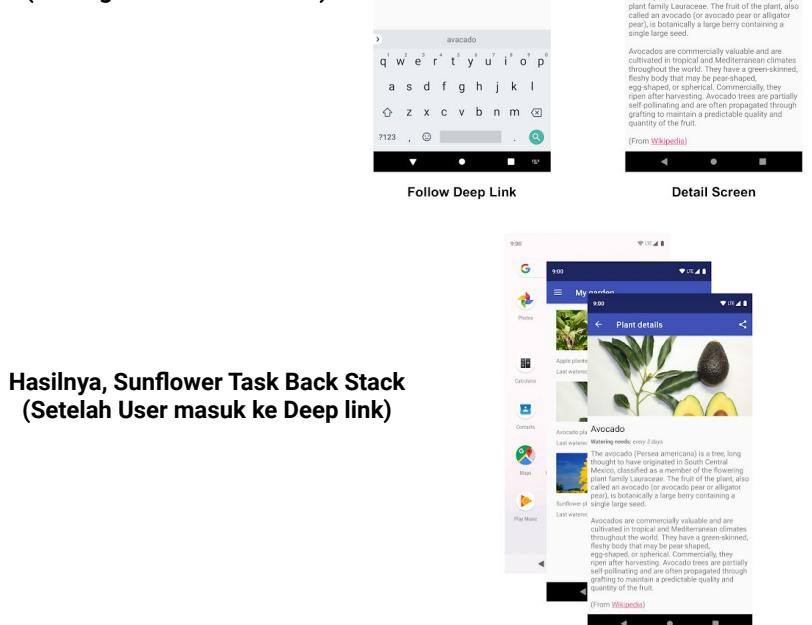
Detail Screen      Follow Deep Link

## Hasilnya, Sunflower Task Back Stack (Setelah User masuk ke Deep link)



**Deep Link ke Hal. Plant Details  
(Linking to Avocado Details)**

Perhatiin juga nih kalo Back stack aplikasi digantikan oleh “Back stack sintesis” dengan layar detail alpukat di bagian atas.



**Hasilnya, Sunflower Task Back Stack  
(Setelah User masuk ke Deep link)**

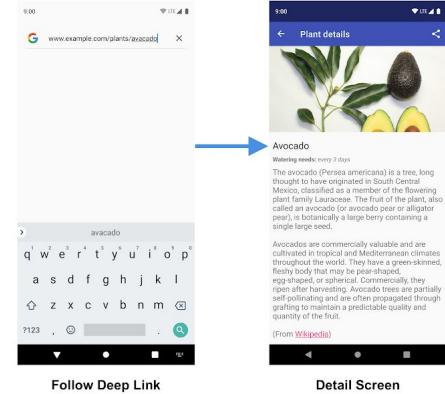


Layar utama aplikasi, yang merupakan tujuan awal juga ditambahkan ke Back stack.

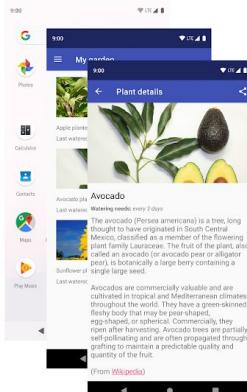
Hal ini penting karena Back stack sintetis haruslah realistik.

“Back stack sintesis” harus cocok dengan Back stack yang dapat diperoleh dengan melakukan navigasi dalam aplikasi secara benar. Back stack aplikasi yang asli akan hilang, termasuk data aplikasi yang menyatakan bahwa user berada di layar detail apel sebelumnya.

## Deep Link ke Hal. Plant Details (Linking to Avocado Details)



## Hasilnya, Sunflower Task Back Stack (Setelah User masuk ke Deep link)





**Wahh.. karena konsep ini baru banget kita dapetin, temen-temen boleh ambil nafas dulu untuk meresapinya.**

**Tarik nafas.. hembuskan..**

**Kalo udah siap, lanjut yuk ke bagian praktek. Langsung ya kita menuju ke langkah-langkah implementasi~**

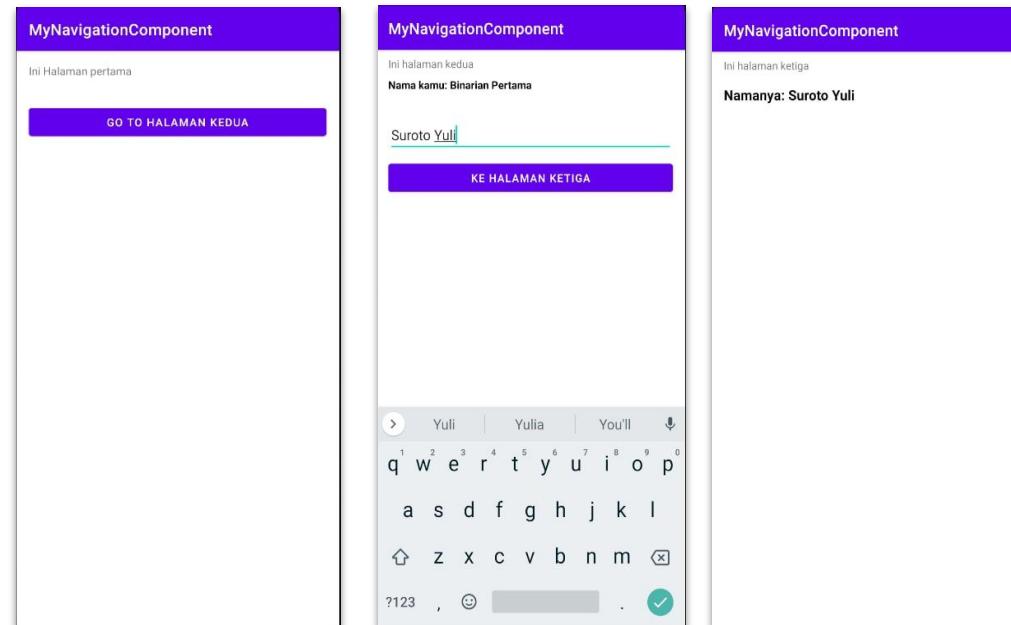


## Jadi, kita mau buat apa kah?

Untuk Praktek kali ini, Kita akan Buat Navigasi Pendaftaran Nama User nih.

Kurang lebih, kenampakannya akan seperti contoh disamping.

Yuk, cekidot!





## 1. Buat project + Setup Dependencies

Pertama-tama kita buat terlebih dahulu sebuah project dengan nama "MyNavigationComponent".

Kalo udah selesai dibuat, masukin dependency Navigation ke dalam "build.gradle(Module:app)" kita dan jangan lupa menambahkan plugin seperti pada code di samping.

### Di build.gradle(Module) :

```
plugins {
    ...
    id "androidx.navigation.safeargs"
}

android {
    ...
    buildFeatures {
        viewBinding true
    }
}

dependencies {
    ...
    // Navigation Component
    implementation 'androidx.navigation:navigation-fragment-ktx:2.4.1'
    implementation 'androidx.navigation:navigation-ui-ktx:2.4.1'
}
```

### Di build.gradle(Project) :

```
buildscript {
    dependencies {
        ...
        classpath "androidx.navigation:navigation-safe-args-gradle-plugin:2.4.1"
    }
}
```



Selain itu, untuk mendukung passing value antara tampilan yang terlibat dalam Navigation, kita perlu menambahkan plugin typesafe gradle.

Tambahkan navigation-safe-args-gradle-plugin di tag dependensi "build.gradle(Project...)" seperti pada Gambar di samping.

## Di build.gradle(Module) :

```
plugins {
    ...
    id "androidx.navigation.safeargs"
}

android {
    ...
    buildFeatures {
        viewBinding true
    }
}

dependencies {
    ...
    // Navigation Component
    implementation 'androidx.navigation:navigation-fragment-ktx:2.4.1'
    implementation 'androidx.navigation:navigation-ui-ktx:2.4.1'
}
```

## Di build.gradle(Project) :

```
buildscript {
    dependencies {
        ...
        classpath "androidx.navigation:navigation-safe-args-gradle-plugin:2.4.1"
    }
}
```



### 2. Buat 3 buah fragment

Lanjuut... Buatlah 3 buah blank fragment yang nantinya bakal digunakan untuk menavigasi antar halaman.

Buatnya pake nama ini ya :

- fragmentPertama,
- fragmentKedua, dan
- fragmentKetiga.

Kayak gambar di samping~

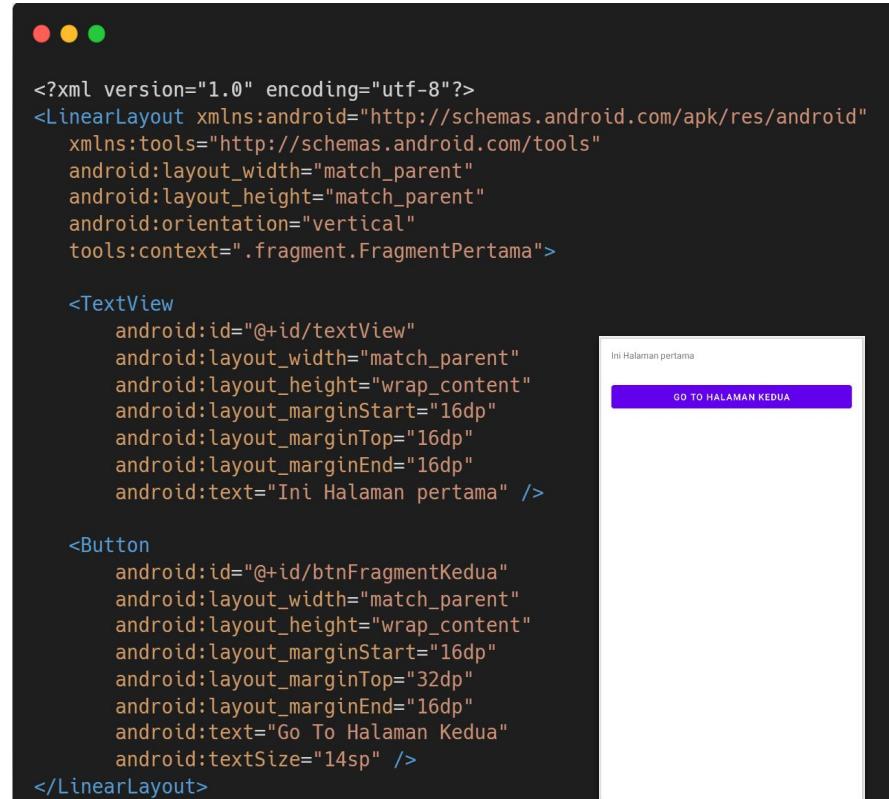




### 3. Setup Layout FragmentPertama

Pada fragment pertama, buatlah:

- Sebuah layout XML dengan satu textView yang menginformasikan bahwa ini adalah halaman pertama, dan
- Satu button yang akan digunakan untuk navigasi ke halaman kedua.



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".fragment.FragmentPertama">

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="16dp"
        android:text="Ini Halaman pertama" />

    <Button
        android:id="@+id/btnFragmentKedua"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="16dp"
        android:text="Go To Halaman Kedua"
        android:textSize="14sp" />
</LinearLayout>
```



## 4. Setup Layout FragmentKedua

Pada fragment kedua, buatlah:

- Sebuah layout dengan satu textView yang menginformasikan bahwa ini adalah halaman kedua.
- Dibawahnya buatlah text yang bertuliskan "Nama Kamu", dan text box input untuk user menuliskan nama.
- Di akhir, kembali buat satu button untuk navigasi ke halaman ketiga.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".fragment.FragmentKedua">

    <TextView
        android:id="@+id/textView3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="16dp"
        android:text="Ini halaman kedua" />

    <TextView
        android:id="@+id/tvNama"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="16dp"
        tools:text="Nama Kamu:" />

    <EditText
        android:id="@+id/etName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="16dp"
        android:ems="10"
        android:hint="Nama Kamu"
        android:inputType="text" />

    <Button
        android:id="@+id/btnToFragmentKetiga"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="16dp"
        android:text="Ke Halaman Ketiga" />
</LinearLayout>
```



## 5. Setup Layout Fragment Ketiga

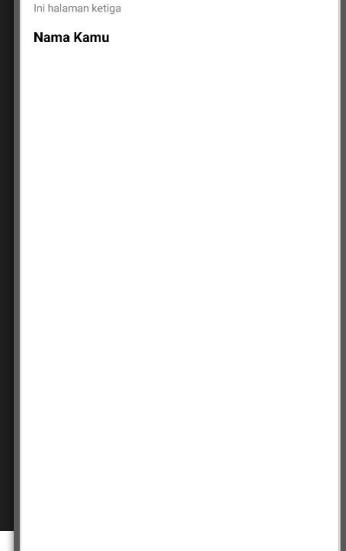
Pada fragment ketiga, buatlah:

- sebuah layout dengan satu textView yang menginformasikan bahwa ini adalah halaman ketiga.
- Dibawahnya, buat kembali satu textView yang bertuliskan “Nama Kamu”.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".fragment.FragmentKetiga">

    <TextView
        android:id="@+id/textView4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="16dp"
        android:text="Ini halaman ketiga" />

    <TextView
        android:id="@+id/tvName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="16dp"
        android:text="Nama Kamu"
        android:textSize="18sp" />
</LinearLayout>
```

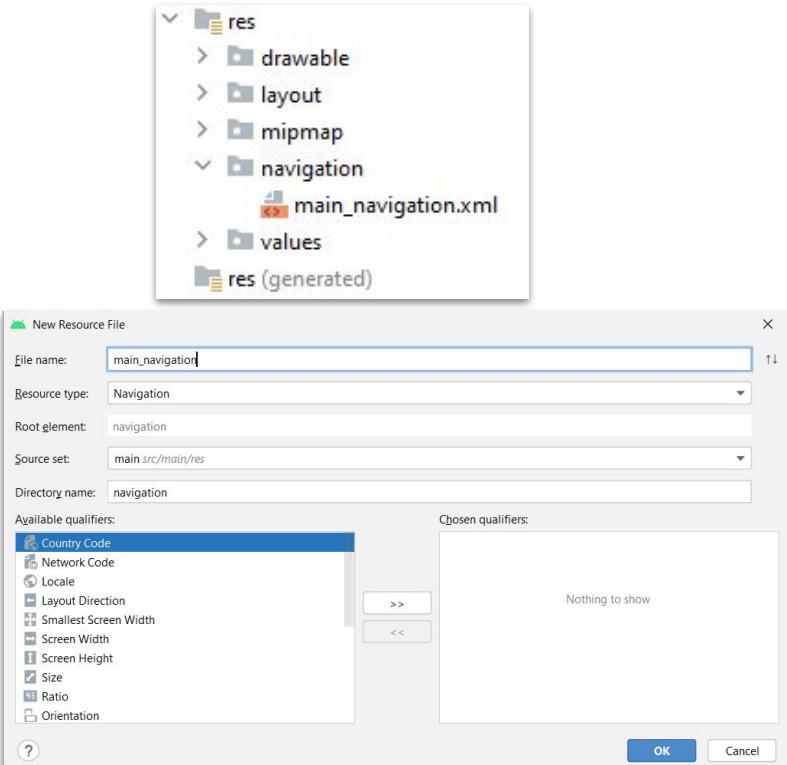




## 6. Setup NavigationGraph

Nah, langkah selanjutnya kalo semua layout udah dibuat saatnya kita setup navigation graph-nya.

Caranya dengan **klik kanan pada folder res > New > Android Resource File**.

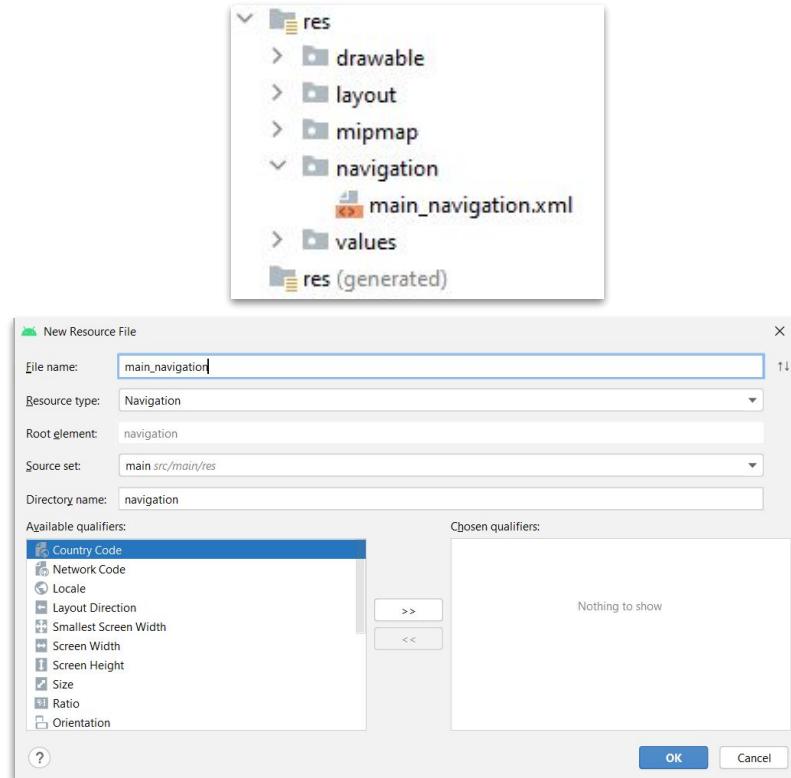




Lalu akan muncul sebuah jendela baru. Pada jendela tersebut kita dapat mengisinya dengan mengikuti seperti gambar yang ada di samping.

Setelahnya, kita klik button “OK”.

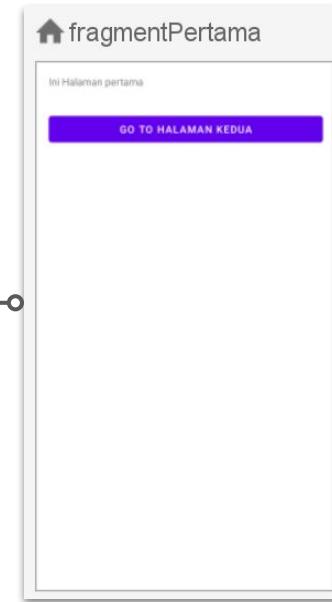
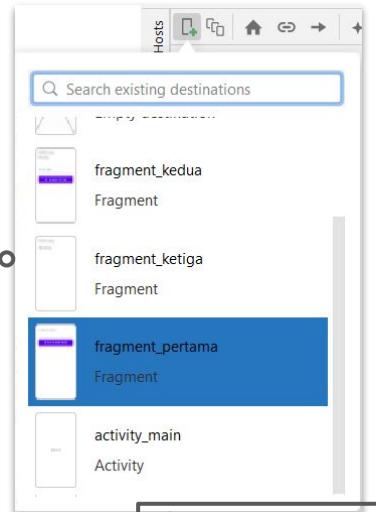
Jika sudah, maka pada folder res akan muncul sebuah folder baru dengan nama “**navigation**”, dan di dalamnya terdapat file yang sebelumnya sudah kita buat.





Langkah selanjutnya kita buka file yang sudah kita buat sebelumnya.

Lalu buatlah sebuah destination baru, dengan menekan icon add destination seperti pada gambar di samping dan pilih fragment pertama yang nantinya kita tetapkan sebagai **startDestination** atau tujuan awal pada saat kita membuka halaman navigation.



Nantinya pada halaman editor navigation graph, akan terdapat sebuah icon rumah yang menandakan bahwa fragment tersebut merupakan **startDestination**.



## 7. Setup NavHostFragment

Wah... tepuk tangan dulu nih kita udah selesai menambahkan **navGraph** baru dan menentukan sebuah **startDestination**.

Selanjutnya pada **mainActivity** kita harus membuat **navHostFragment**, supaya **navGraph** yang telah dibuat bisa terbaca pada aplikasi kita melalui **mainActivity**.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <fragment
        android:id="@+id/navContainer"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:navGraph="@navigation/main_navigation" />

</androidx.constraintlayout.widget.ConstraintLayout>
```



Tambahkan syntax di samping agar dapat menampilkan navGraph pada **mainActivity**.

Kalo udah, nantinya pada layout **mainActivity** akan berubah menjadi tampilan layout **startDestination** di mana pada project kali ini akan muncul layout **fragmentPertama**.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <fragment
        android:id="@+id/navContainer"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:navGraph="@navigation/main_navigation" />

</androidx.constraintlayout.widget.ConstraintLayout>
```



Akhirnya kita telah berhasil menyelesaikan setup navigation componentnya. Sekarang kita tambahkan beberapa action agar antar screen dapat berubah, juga dapat mengirimkan data.

Yuk, cuss~

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <fragment
        android:id="@+id/navContainer"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:navGraph="@navigation/main_navigation" />

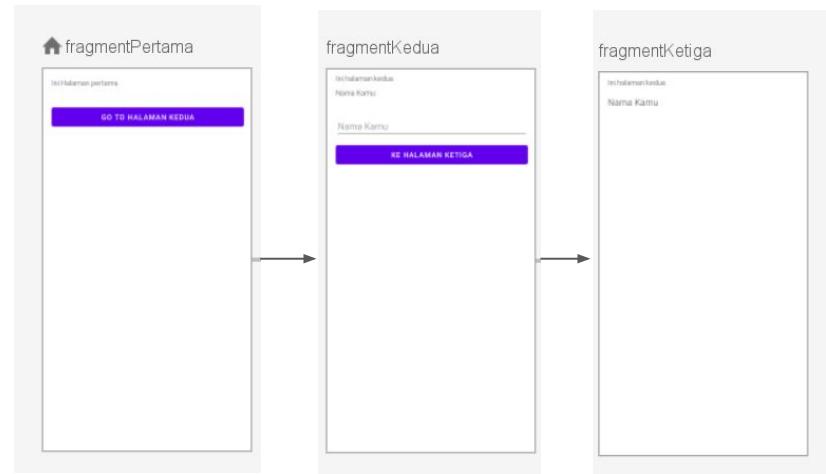
</androidx.constraintlayout.widget.ConstraintLayout>
```



## 8. Tambah Action Perpindahan Antar Screen

Kalo sebelumnya kita belajar untuk menambahkan destination pada navGraph. Lakukan hal tersebut lagi dan masukkan lah fragmentKedua dan juga fragmentKetiga.

Setelah itu kita tambahkan actionnya!



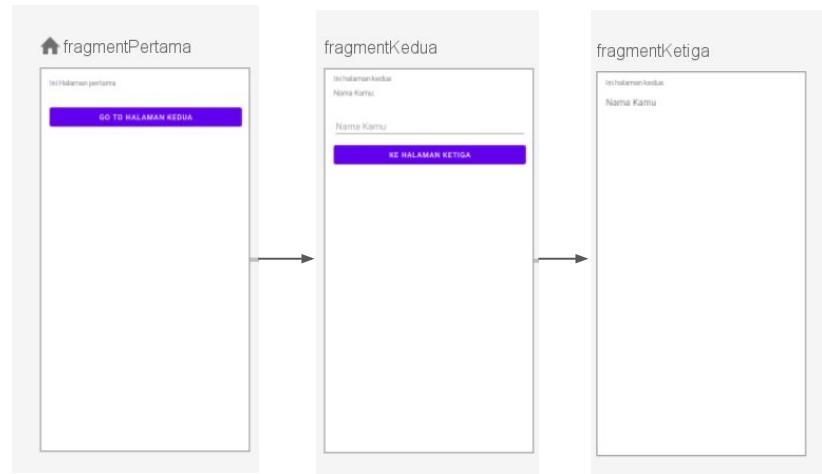


Caranya adalah mengarahkan kursor kita ke salah satu destination yang ada pada navGraph dan tekan dan tahan pada bagian lingkaran di tengah.

Setelah itu arahkan ke destination yang diinginkan. Buatlah action dengan mengurutkan:

**fragmentPertama > fragmentKedua > fragmentKetiga.**

Hasilnya bakal kayak gambar di samping





# Safe Args

## 9. Tambah SafeArgs pada fragment Ketiga

Nah inilah salah satu cara yang akan kita gunakan untuk mengirimkan data menggunakan navigation component.

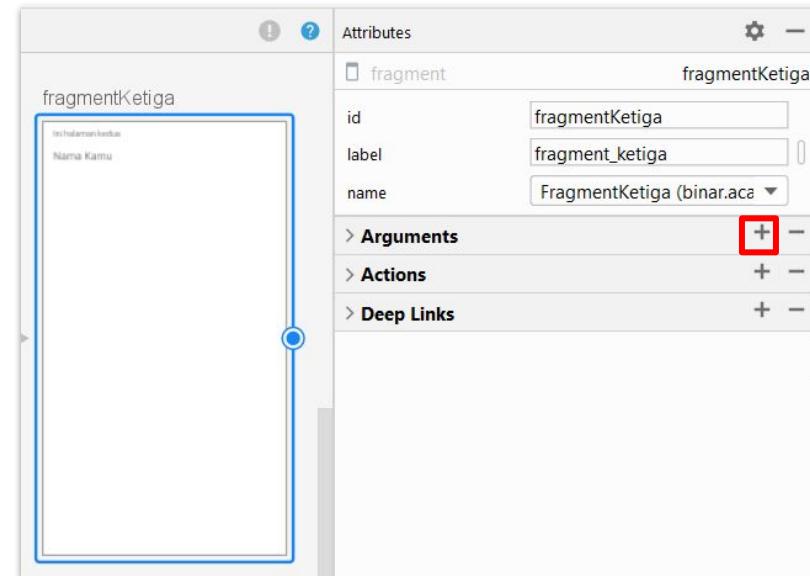
SafeArgs merupakan salah fitur yang terdapat pada navigation component yang **memungkinkan kita untuk mengirimkan data antar destinations secara aman.**





Untuk membuatnya, kita hanya perlu menekan destination fragmentKetiga.

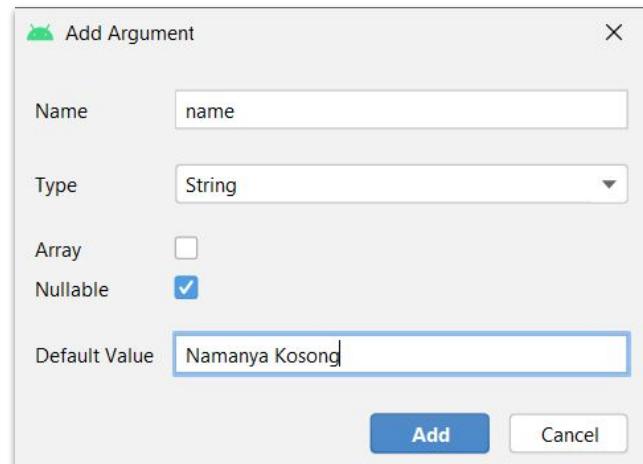
Lalu pada tab sebelah kanan akan muncul section Argument. Setelah itu klik icon + untuk menambahkan argument-Nya.





Kalo udah teken icon + sebelumnya, maka akan muncul dialog baru yang mana kita ditugaskan untuk mengisi beberapa field argument.

Isilah sesuai dengan gambar di samping. Kalo udah, kita klik button "Add".





## 10. Setup class fragmentPertama

Setelah semua setup kita selesaikan, mulailah kita untuk men-setup class fragment yang akan kita gunakan untuk navigasi antar fragment.

Kita mulai dari fragmentPertama.

```
class FragmentPertama : Fragment() {

    private var _binding: FragmentPertamaBinding? = null
    private val binding get() = _binding!!

    companion object {
        val EXTRA_NAME = "EXTRA_NAME"
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentPertamaBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.btnFragmentKedua.setOnClickListener {
            val mBundle = Bundle()
            mBundle.putString(EXTRA_NAME, "Binarian Pertama")
            it.findNavController().navigate(R.id.action_fragmentPertama_to_fragmentKedua, mBundle)
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }
}
```



Pada fragment ini, kita akan mencoba mengirimkan data antar screen menggunakan bundle.

Langkah pertama, kita buat terlebih dahulu variable dengan nama **EXTRA\_NAME**.

Setelah itu pada button fragment kedua, kita tambahkan action untuk mengirimkan data bundle ke screen lain.

```
class FragmentPertama : Fragment() {

    private var _binding: FragmentPertamaBinding? = null
    private val binding get() = _binding!!

    companion object {
        val EXTRA_NAME = "EXTRA_NAME"
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentPertamaBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.btnFragmentKedua.setOnClickListener {
            val mBundle = Bundle()
            mBundle.putString(EXTRA_NAME, "Binarian Pertama")
            it.findNavController().navigate(R.id.action_fragmentPertama_to_fragmentKedua, mBundle)
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }
}
```



Di sini kita mencoba menggunakan bundle untuk mengirimkan data antar screen.

Ini merupakan salah satu cara yang dapat digunakan ketika menggunakan navigationComponent.

```
class FragmentPertama : Fragment() {

    private var _binding: FragmentPertamaBinding? = null
    private val binding get() = _binding!!

    companion object {
        val EXTRA_NAME = "EXTRA_NAME"
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentPertamaBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.btnFragmentKedua.setOnClickListener {
            val mBundle = Bundle()
            mBundle.putString(EXTRA_NAME, "Binarian Pertama")
            it.findNavController().navigate(R.id.action_fragmentPertama_to_fragmentKedua, mBundle)
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }
}
```



Pada syntax :

```
navigate(R.id.action_fragmentPertama_to_fragmentKedua, mBundle)
```

di sini kita masukkan **actionId** yang ingin kita tuju. Karena kita ingin menavigasi ke halaman fragmentKedua, jadinya kita masukin deh actionId :

```
R.id.action_fragmentPertama_to_fragmentKedua.
```

```
class FragmentPertama : Fragment() {

    private var _binding: FragmentPertamaBinding? = null
    private val binding get() = _binding!!

    companion object {
        val EXTRA_NAME = "EXTRA_NAME"
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentPertamaBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.btnFragmentKedua.setOnClickListener {
            val mBundle = Bundle()
            mBundle.putString(EXTRA_NAME, "Binarian Pertama")
            it.findNavController().navigate(R.id.action_fragmentPertama_to_fragmentKedua, mBundle)
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }
}
```



## 11. Setup class fragmentKedua

Selanjutnya pada fragmentKedua buatlah sebuah variable dengan nama aName lalu kita isikan dengan syntax untuk mendapatkan data argument sesuai dengan key yang kita tentukan sendiri.

Lalu setelahnya kita coba masukkan ke dalam textView.

Kalo udah selesai semua, kita coba yuk main aplikasinya.

```
● ● ●

class FragmentKedua : Fragment() {

    private var _binding: FragmentKeduaBinding? = null
    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentKeduaBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // Mendapatkan data argument sesuai dengan key yang dimaksud
        val aName = arguments?.getString(FragmentPertama.EXTRA_NAME)

        binding.tvNama.text = "Nama kamu: $aName"
    }

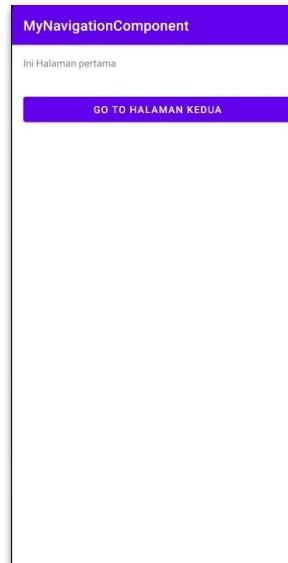
    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }
}
```



Setelah berhasil dijalankan, tampilan aplikasi akan langsung menampilkan halaman fragmentPertama.

Hal tersebut terjadi karena kita sudah menentukan kalau startDestination pada navGraph adalah fragmentPertama.

Jika kita menekan button Go To Halaman Kedua, maka kita akan pindah ke halaman fragmentKedua. **Di sini kita sudah membuat argument hardCode** yang memindahkan data bundle berupa **EXTRA\_NAME**.



Halaman 1



Halaman 2



Pada halaman fragmentKedua, kita akan melihat bahwa argument dari fragmentPertama sudah kita terima dan dapat kita tampilkan pada halaman fragmentKedua.

Ini artinya kita **sudah mengirimkan data antar screen** dengan salah satu cara yang disediakan oleh navigationComponent.



Halaman 1



Halaman 2



## 12. Lanjut setup class fragmentKedua

Pada kali ini, kita akan mencoba salah satu cari lagi di navigationComponent untuk membawa data ke halaman lain, yaitu dengan menggunakan **safeArgs**.

```
class FragmentKedua : Fragment() {  
    ....  
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
        super.onViewCreated(view, savedInstanceState)  
        ....  
        binding.btnToFragmentKetiga.setOnClickListener { view ->  
            if (binding.etName.text.isNullOrEmpty()) {  
                Toast.makeText(requireContext(), "Kolom Nama masih kosong",  
                    Toast.LENGTH_SHORT).show()  
            } else {  
                val actionToFragmentKetiga =  
                    FragmentKeduaDirections.actionFragmentKeduaToFragmentKetiga()  
                actionToFragmentKetiga.name = binding.etName.text.toString()  
                view.findNavController().navigate(actionToFragmentKetiga)  
            }  
        }  
        ....  
    }  
}
```



Tambahkan beberapa syntax di bagian onViewCreated untuk menambahkan action pada button supaya bisa berpindah ke halaman fragmentKetiga.

Sebelumnya, tambahin validasi pengecekan dulu supaya editText nggak kosong.

Kalo editText kosong nanti bisa muncul toast, kalo nggak kosong maka bakal berpindah ke halaman fragmentKetiga.

```
....  
class FragmentKedua : Fragment() {  
  
    ....  
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
        super.onViewCreated(view, savedInstanceState)  
        ....  
        binding.btnToFragmentKetiga.setOnClickListener { view ->  
            if (binding.etName.text.isNullOrEmpty()) {  
                Toast.makeText(requireContext(), "Kolom Nama masih kosong",  
                    Toast.LENGTH_SHORT).show()  
            } else {  
                val actionToFragmentKetiga =  
                    FragmentKeduaDirections.actionFragmentKeduaToFragmentKetiga()  
                actionToFragmentKetiga.name = binding.etName.text.toString()  
                view.findNavController().navigate(actionToFragmentKetiga)  
            }  
        }  
        ....  
    }  
}
```



## 12. Setup class fragmentKetiga

Terakhir yang perlu kita siapkan ada pada fragmentKetiga.

Pada fragment ini kita perlu memasukkan syntax yang digunakan untuk mendapatkan data argument dari fragmentKedua.

```
● ● ●

class FragmentKetiga : Fragment() {

    private var _binding: FragmentKetigaBinding? = null
    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentKetigaBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        val aName = FragmentKetigaArgs.fromBundle(arguments as Bundle).name

        binding.tvName.text = "Namanya: $aName"
    }

    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }
}
```



## Syntax

**FragmentKetigaArgs.fromBundle(arguments  
as Bundle).name**

bertujuan untuk mengambil data argument yang ada pada fragmentKetiga, dengan nama name. Setelahnya akan kita tampilkan pada textView.

Kalo udah siap semua, kita coba jalankan aplikasi kita.



```
class FragmentKetiga : Fragment() {

    private var _binding: FragmentKetigaBinding? = null
    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentKetigaBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        val aName = FragmentKetigaArgs.fromBundle(arguments as Bundle).name

        binding.tvName.text = "Namanya: $aName"
    }

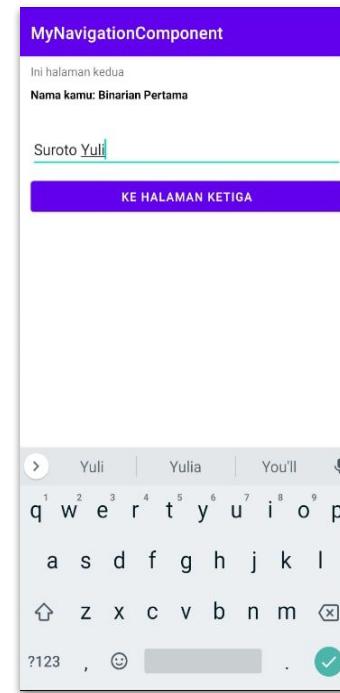
    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }
}
```



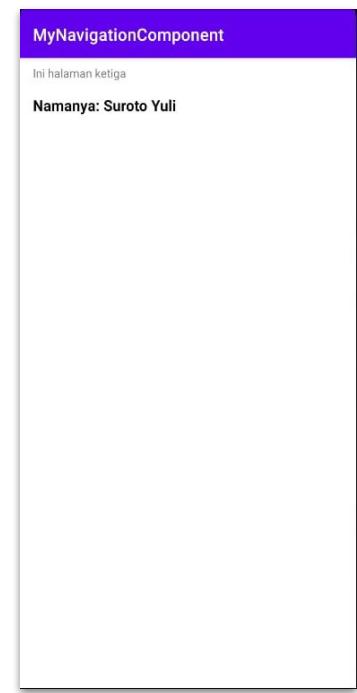
Pada editText di halaman fragmentKedua kita coba masukkan data sesuai yang kita inginkan.

Di sini kita masukkan value “Suroto Yuli”, setelahnya kita coba klik button “Ke Halaman Ketiga”.

Aplikasi akan menavigasi ke halaman fragmentKetiga dengan membawa argument name yang seharusnya berisi value “Suroto Yuli”.



Halaman 2



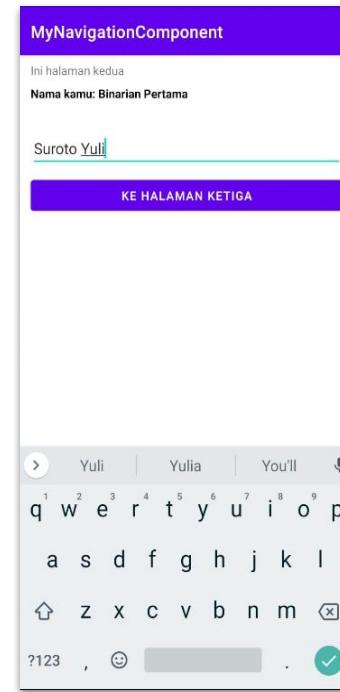
Halaman 3



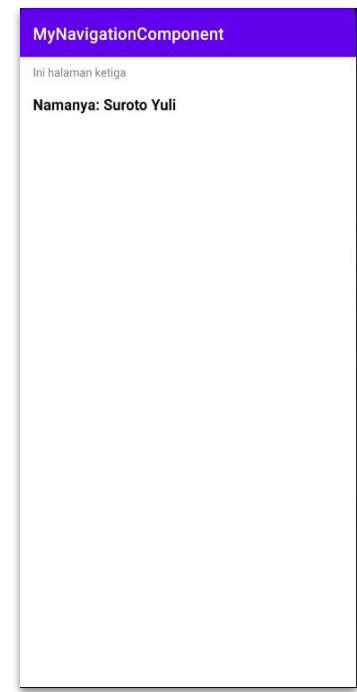
Pada halaman fragmentKetiga, di sini kita hanya menampilkan data argument yang telah dikirimkan dari fragmentKedua.

Di sini kita menampilkannya dengan menggunakan **safeArgs**.

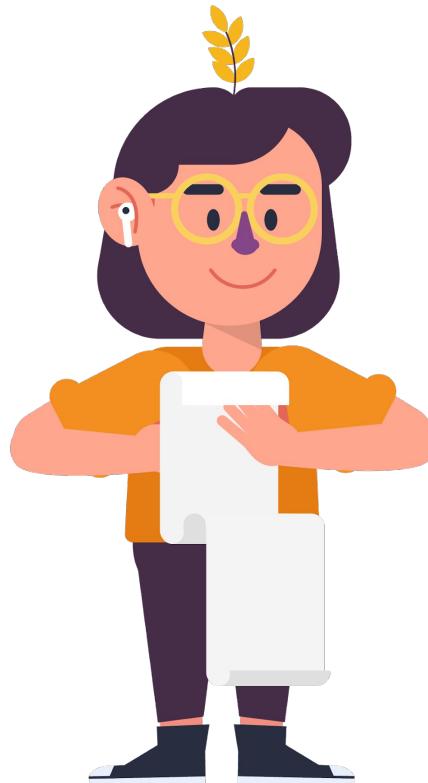
Cara ini merupakan cara lain yang dapat kita gunakan untuk mengirimkan data antar screen dengan navigationComponent.



Halaman 2



Halaman 3



## Biar Makin Mahir, Yuk, ikuti Challenge Berikut!

Buatlah Aplikasi sederhana yang mengimplementasikan Fragment menggunakan BottomNavigation dengan fitur Navigation Component !

***Pengumpulan tugas diserahkan kepada mentor***

Kita akan membahas bersama-sama di pertemuan berikutnya.

# Saatnya kita Quiz!





**1. Berikut adalah keuntungan dari menggunakan navigation component, kecuali...**

- A. Menggunakan safeArgs untuk transaksi antar fragment
- B. Menangani deep link dengan mudah
- C. Membantu meng-handle class



### 1. Berikut adalah keuntungan dari menggunakan navigation component, kecuali...

- A. Menggunakan safeArgs untuk transaksi antar fragment
- B. Menangani deep link dengan mudah
- C. Membantu meng-handle class

Yap! Navigation component digunakan untuk membantu kita untuk transaksi antar screen. Kita jadi terbantu dalam berbagai hal, diantaranya adalah memudahkan transaksi data antar fragment / screen, serta menangani deep link dengan mudah dalam sistem navigasi.



**2. Di bawah ini merupakan salah satu komponen yang benar dalam membuat navigasi, yaitu...**

- A. Intent
- B. Arguments
- C. Navigation

**2. Di bawah ini merupakan salah satu komponen yang benar dalam membuat navigasi, yaitu...**

- A. Intent
- B. Arguments
- C. Navigation

**Benar! Pastinya adalah Navigation!**

Google memperkenalkan komponen Navigation sebagai framework untuk menyusun UI dalam aplikasi kita, dengan fokus pada pembuatan aplikasi single activity sebagai arsitektur yang dipilih. Dengan dukungan bawaan untuk fragment, kita mendapatkan semua manfaat Architecture Components seperti Lifecycle dan ViewModel sembari memungkinkan Navigation untuk menangani kompleksitas FragmentTransaction untuk kita.



### 3. Di bawah ini, manakah cara yang benar untuk membuat navigation Graph

- A. Klik kanan pada folder Res > AndroidResourceFile > Pilih Navigation pada resource Type.
- B. Klik kanan pada folder Res > Pilih Navigation > Pilih blank Navigation
- C. Klik File pada toolbar > Pilih Navigation > Pilih blank Navigation



### 3. Di bawah ini, manakah cara yang benar untuk membuat navigation Graph ?

- A. Klik kanan pada folder Res > AndroidResourceFile > Pilih Navigation pada resource Type.
- B. Klik kanan pada folder Res > Pilih Navigation > Pilih blank Navigation
- C. Klik File pada toolbar > Pilih Navigation > Pilih blank Navigation

Yups, cara yang benar untuk membuat sebuah navigation Graph adalah klik kanan pada folder Res > Pilih AndroidResourceFile > Setelah itu akan muncul jendela baru, pada bagian resource type ubah data menjadi navigation.



#### 4. Di bawah ini manakah yang merupakan keuntungan dari penggunaan safeArgs

- A. Digunakan untuk mengamankan argument
- B. Digunakan untuk menghindari value argument null
- C. Digunakan untuk mendapatkan argument

#### 4. Di bawah ini manakah yang merupakan keuntungan dari penggunaan safeArgs

- A. Digunakan untuk mengamankan argument
- B. Digunakan untuk menghindari value argument null
- C. Digunakan untuk mendapatkan argument

Pada navigationComponent, safeArgs merupakan salah satu fitur yang disediakan untuk melakukan pengiriman data secara safe. Safe yang dimaksud adalah mengirimkan data argument antar screen dan juga digunakan untuk menghindari *null* yang terjadi ketika data tidak diisi.



5.



```
val toFragmentFour = harus_diisi_ap?  
view.findNavController().navigate(toFragmentFour)
```

**Perhatikan syntax di atas, syntax di atas digunakan untuk berpindah ke destination fragmentKeempat menggunakan safeArgs. Syntax apakah yang digunakan untuk melengkapinya?**

- A. `FragmentPertamaArgs.actionFragmentPertamaToFragmentKeempat()`
- B. `FragmentPertamaDirections.actionFragmentPertamaToFragmentKeempat()`
- C. `it.findNavController().navigate(R.id.action_fragmentPertama_to_fragmentKeempat, mBundle)`



5.



```
val toFragmentFour = harus_diisi_ap?  
view.findNavController().navigate(toFragmentFour)
```

Perhatikan syntax di atas, syntax di atas digunakan untuk berpindah ke destination fragmentKeempat menggunakan safeArgs. Syntax apakah yang digunakan untuk melengkapinya?

- A. FragmentPertamaArgs.actionFragmentPertamaToFragmentKeempat()
- B. **FragmentPertamaDirections.actionFragmentPertamaToFragmentKeempat()**
- C. `it.findNavController().navigate(R.id.action_fragmentPertama_to_fragmentKeempat, mBundle)`

Betul! Jawabannya adalah B. Untuk melakukan action navigasi dengan menggunakan safeArgs. Setelah menambahkan actionnya pada navigation Graph, kita perlu memanggil

**syntax** `NamaFragmetDirections.actionNamaFragmentAsalToNamaFragmentTujuan()`

## Referensi dan bacaan lebih lanjut~

1. [Pengembangan Aplikasi dengan Android Jetpack untuk Pemula – Coniolabs](#)
2. [Get started with the Navigation component | Android Developers](#)
3. [Navigation Component- The Complete Guide | by Muhamed Riyas M | Medium](#)





**Nah, selesai sudah pembahasan kita di Chapter 3 Topic 5 ini.**

**Daan, Topik ini juga mengakhiri bahasan Chapter 3 kita.**

**Selanjutnya, kita akan bahas Challenge yang akan kita kerjakan sebagai penutup Chapter ini~**



# Terima Kasih!



Chapter ✓

completed