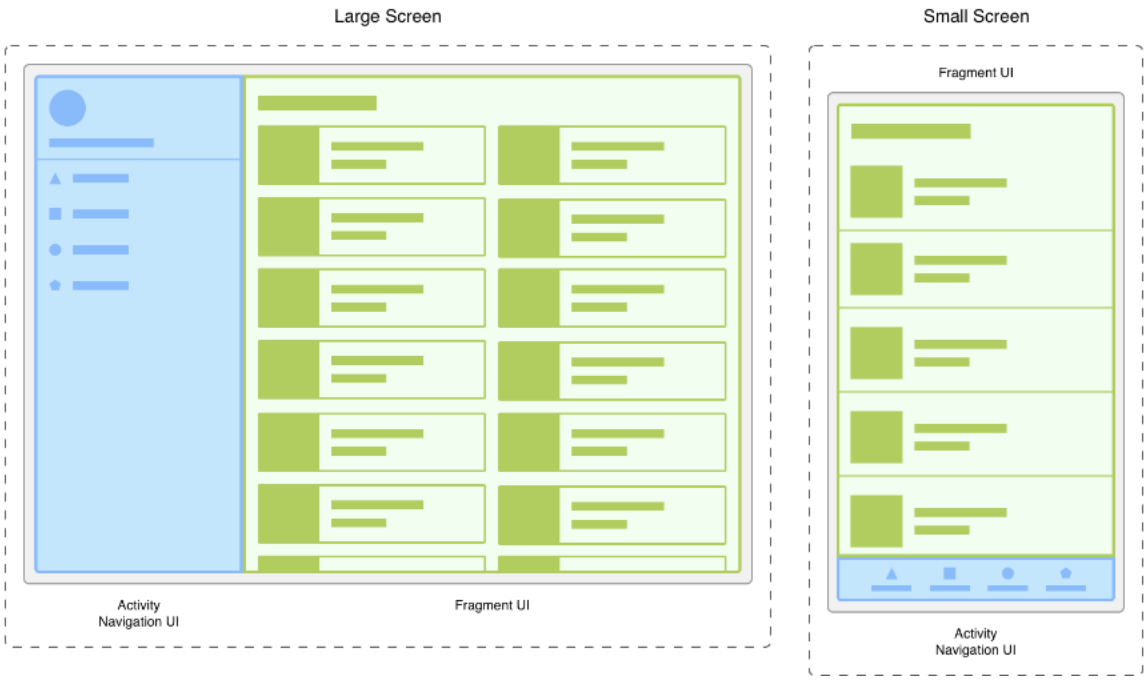
Informe de investigación

Fragments (Fragmentos):

Un fragmento es una porción reutilizable de la interfaz de usuario de una aplicación que se puede combinar con otros fragmentos y actividades para crear una UI multipanel.

Los fragmentos posees su propio ciclo de vida y eventos de modificaciones, esto les permite a los fragmentos el ser reutilizables y flexibles.

Los fragmentos son útiles en la construcción de UIs responsivas de modo que estos puedan crear una sola interfaz o dividirse en relación al tamaño de la pantalla:



En la figura 1 vemos una sola interfaz de navegación conformada por dos fragmentos adyacentes (Azul y verde), en la figura 2 (Small screen) se ven los

mismos dos fragmentos pero con una disposición distinta para acoplarse al tamaño de la pantalla sin reducir demasiado el tamaño de los componentes de cada fragmento.

Antes de definir un fragmento es necesario agregar las dependencias necesarias:

```
dependencies {
    val fragment_version = "1.8.1"

    implementation("androidx.fragment:fragment-ktx:$fragment_version")
}
```

Un fragmento en Kotlin se puede definir de la siguiente forma:

```
class ExampleFragment : Fragment(R.layout.fragment_example) {
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        // Inicializar componentes de la UI aquí
    }
}
```

Fragment manager

El Fragment manager es el responsable de gestionar los fragmentos que son utilizados en nuestra aplicación, es responsable de gestionar los fragmentos en una actividad, incluyendo la adición, eliminación y reemplazo de fragmentos.

El fragment manager puede ser invocado tanto en las actividades (padres) como en los fragmentos hijos.

En el padre:

```
val fragmentManager = supportFragmentManager
```

En el hijo:

```
val fragmentManager = parentFragmentManager
```

Fragment Transactions

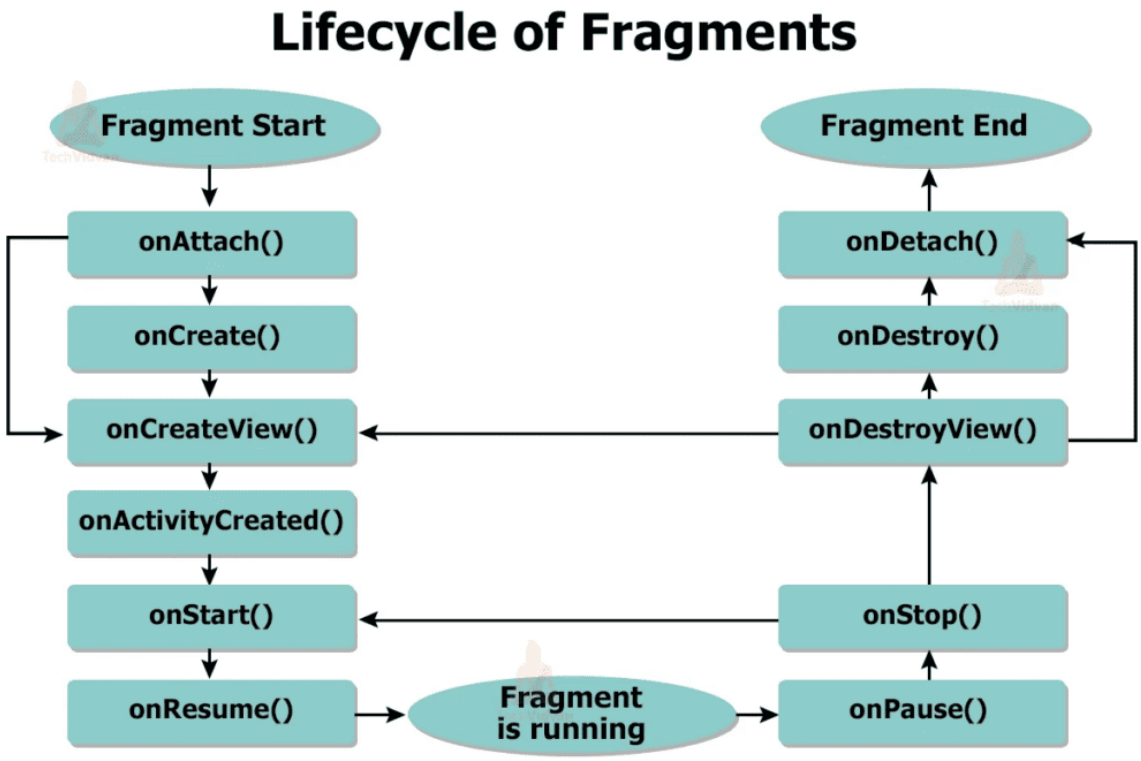
Una **FragmentTransaction** representa una operación atómica que se puede realizar en la colección de fragmentos. Las transacciones permiten añadir, quitar, reemplazar y realizar otras operaciones con los fragmentos de forma segura y controlada.

Las transacciones se realizan de la siguiente forma:

```
val fragmentManager = ...
val fragmentTransaction = fragmentManager.beginTransaction()
```

Ciclo de Vida de los Fragmentos

El ciclo de vida de un fragmento en Android es similar al de una actividad, pero con algunas diferencias clave. Los fragmentos tienen su propio conjunto de métodos de ciclo de vida que permiten gestionar su creación, estado y destrucción de manera más granular



Animar Transiciones entre Fragmentos

Las animaciones de transición entre fragmentos en Android mejoran la experiencia del usuario proporcionando una interfaz más fluida y atractiva. Las transiciones pueden aplicarse al añadir, reemplazar o eliminar fragmentos.

-Introducción a las Transiciones

> Las animaciones de transición entre fragmentos permiten visualizar los cambios en la interfaz de usuario de manera más agradable, mostrando cómo los fragmentos entran y salen de la pantalla.
>
> Mejoran la percepción de la navegación y el flujo de la aplicación, haciendo que las transiciones sean menos abruptas y más comprensibles para el usuario.

2. Tipos de Animaciones

- Predefinidas: Android proporciona animaciones predefinidas que se pueden utilizar fácilmente.

- Personalizadas: Los desarrolladores pueden crear animaciones personalizadas usando XML o programáticamente.

Las animaciones personalizadas pueden definirse en el directorio res/anim

```xml
<!-- res/anim/fade_out.xml -->
<?xml version="1.0" encoding="utf-8"?>
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="@android:integer/config_shortAnimTime"
    android:interpolator="@android:anim/decelerate_interpolator"
    android:fromAlpha="1"
    android:toAlpha="0" />
```

```xml
<!-- res/anim/slide_in.xml -->
<?xml version="1.0" encoding="utf-8"?>
<translate xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="@android:integer/config_shortAnimTime"
    android:interpolator="@android:anim/decelerate_interpolator"
    android:fromXDelta="100%"
    android:toXDelta="0%" />
```

Guardar Estado en Fragmentos

Guardar y restaurar el estado de los fragmentos en Android es esencial para asegurar que la interfaz de usuario y los datos del usuario se mantengan consistentes a través de cambios de configuración, como la rotación de la pantalla, o cuando el sistema decide destruir y recrear la actividad para liberar recursos.

Los datos de los fragmentos pueden perderse en cualquiera de estos casos:

Variables: cambios en las variables locales en el fragmento.

Estado de vista: cambio en cualquier dato que sea propiedad de una o más vistas en el fragmento.

SavedState: datos inherentes a esta instancia de fragmento que deben guardarse en onSaveInstanceState().

NonConfig: datos extraídos de una fuente externa, como un servidor o repositorio local, o datos creados por el usuario que se envían a un servidor una vez confirmados.

Para guardar y restaurar estados se usa una estructura de datos Clave:Valor a la cual se accede de la siguiente forma:

Para guardar estados:

```
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    outState.putBoolean(IS_EDITING_KEY, isEditing)
    outState.putString(RANDOM_GOOD_DEED_KEY, randomGoodDeed)
}
```

Para restaurar los estados guardados:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    isEditing = savedInstanceState?.getBoolean(IS_EDITING_KEY, false)
    randomGoodDeed = savedInstanceState?.getString(RANDOM_GOOD_DEED_KEY)
            ?: viewModel.generateRandomGoodDeed()
}
```

Comunicación entre Fragmentos

La comunicación entre fragmentos es una parte esencial del desarrollo de aplicaciones en Android, especialmente cuando se trabaja con interfaces de usuario complejas que requieren la interacción entre diferentes componentes de la interfaz.

1. Introducción a la Comunicación entre Fragmentos

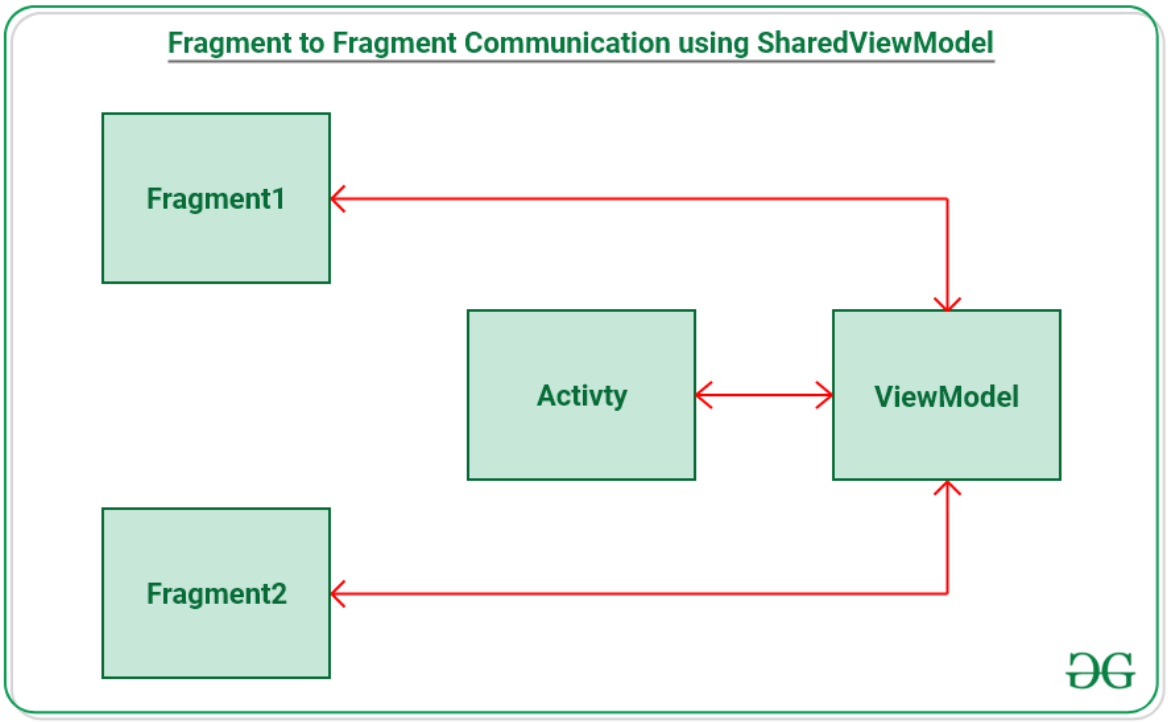
    Los fragmentos a menudo necesitan comunicarse entre sí para compartir datos y eventos. Dado que los fragmentos deben ser modulares y reutilizables, la comunicación debe ser manejada de manera estructurada para mantener la separación de responsabilidades.

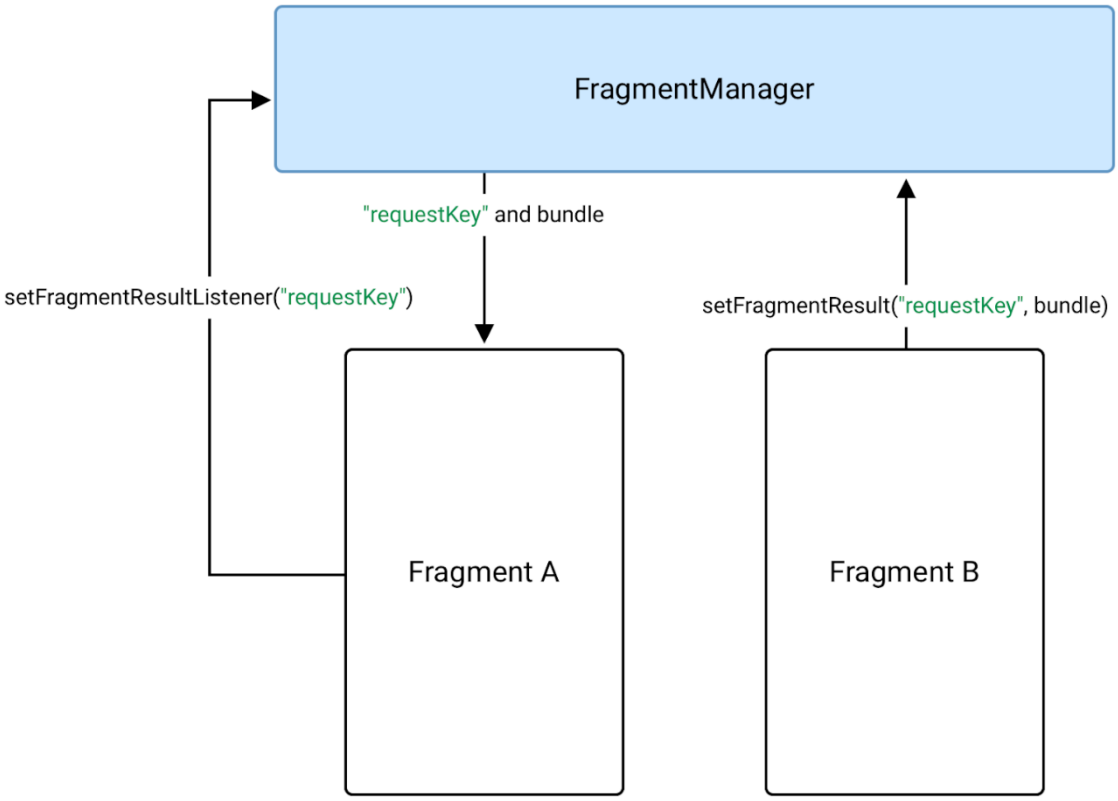
2. Métodos Comunes de Comunicación entre Fragmentos

    Utilizando la Actividad Anfitriona: La actividad actúa como un intermediario para pasar datos entre fragmentos.

    Interfaces de Callbacks: Definir interfaces en los fragmentos que la actividad anfitriona debe implementar.

    ViewModel Compartido: Utilizar ViewModel para compartir datos y eventos entre fragmentos.


Fragment to Fragment Communication using SharedViewModel
Fragment1
Activty
ViewModel
Fragment2


FragmentManager
"requestKey" and bundle
setFragmentResultListener("requestKey")
setFragmentResult("requestKey", bundle)
Fragment A
Fragment B