Ciclo de vida de componentes

Android (Parte II)

{desafío} latam_

Utilizar elementos del ciclo de vida para la implementación de un aplicativo Android que resuelve un problema.

- Unidad 1: Kotlin para el desarrollo de aplicaciones.
- Unidad 2: Ciclo de vida de componentes.
- Unidad 3: Arquitectura en Android.
- Unidad 4: Programación asíncrona en Android.

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

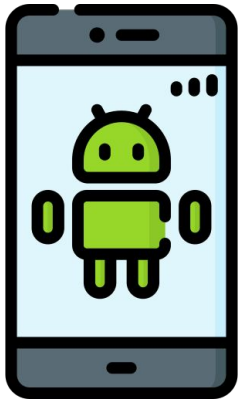
- *Reconocer los aspectos fundamentales del ciclo de vida de Activities y Fragments en Android para el desarrollo de aplicaciones*

{desafío}
latam_

Responde en el chat: ¿Qué es el ciclo de vida de Activities?

/* Añadir Fragments dinámicamente */

Ejemplo añadir Fragments dinámicamente



En la sesión anterior vimos cómo se podían manejar fragments con FragmentFactory. Ahora, en tanto, veremos que los fragments pueden ser agregados de forma más dinámica. Para esto debes tener claro algunos conceptos:

- Un Fragment es una clase que extiende de `Fragment()`
- Un Fragment requiere de, al menos, una Activity para existir

FragmentManager: es la clase responsable de realizar acciones en los fragments, como por ejemplo, agregarlos, eliminarlos o reemplazarlos, y agregarlos al backstack.

/* Pasarle parámetros al fragment */

Pasarle parámetros al fragment

Primero debemos darnos cuenta que un Fragment, en una app, no es más que una clase la cual extiende de `Fragment()`, así que si queremos pasar parámetros a esa clase, basta con especificarlos en su constructor. Por ejemplo:

```
class ConversationFragment(val conversationId: Int) : Fragment() {
...
}
```

Sin embargo, si recordamos lo que se mencionó en **FragmentFactory**, cada vez que queramos hacer uso de ese fragment, debemos pasar todos los parámetros necesarios.

Pasarle parámetros al fragment

En la actualidad existe más de una alternativa al momento de pasar parámetros a un fragment, pero primero debemos saber de donde provienen estos datos, específicamente, necesitamos saber si vienen desde un activity u otro fragment.

- Activity → Fragment
- Fragment → Fragment

Activity → Fragment

- **Companion Object**: podemos pasar un parámetro haciendo uso del companion object, algo que vimos en una clase anterior.

- **ViewModel compartido**: este caso se trata de compartir cambios entre vistas con objetos tipo LiveData o StateFlow a través del ViewModel

Fragment → Fragment

- **ViewModel compartido**: Como se menciona en el caso de activity a fragment, si dos fragments comparten el mismo ViewModel, entonces es posible que los cambios ocurridos en el `Fragment A` se vean reflejados en el `Fragment B`

- **Fragment Result API**: En algunos casos, es posible que desee pasar un valor único entre dos fragmentos o entre un fragmento y su actividad de host. A partir de la versión 1.3.0 de Fragment, cada FragmentManager implementa `FragmentResultOwner`. Esto significa que un FragmentManager puede actuar como un almacén central de resultados de fragmentos.

Este cambio permite que los componentes se comuniquen entre sí al establecer resultados de fragmentos y escuchar esos resultados sin requerir que estos componentes tengan referencias directas entre sí.

ViewModel

ViewModel es un componente fundamental en el desarrollo de aplicaciones Android que ayuda a gestionar y retener datos relacionados con la interfaz de usuario (UI) de una manera más eficiente y persistente.

A diferencia de las actividades y fragmentos, que pueden recrearse durante cambios de configuración (como girar la pantalla), un ViewModel sobrevive a estos cambios.

Puedes visitar el siguiente enlace para obtener más información acerca de ViewModel.

¿Cómo utilizar Fragment Result API?

Para pasar datos a un `FragmentA` desde un `FragmentB`, en `FragmentB` podemos hacer:

```
setFragmentResult("requestKey", bundleOf(
                "bundleKey1" to 1001,
                "bundleKey2" to "esto es un string",
                "bundleKey3" to true,
            )
        )
```

{desafío} latam_

¿Cómo utilizar Fragment Result API?

Luego en `FragmentA` “escuchamos” ese cambio:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setFragmentResultListener("requestKey") { requestKey, bundle ->
        val result1 = bundle.getInt("bundleKey1")
        val result2 = bundle.getString("bundleKey2")
        val result3 = bundle.getBoolean("bundleKey3")
    }
}
```

/* Utilizando el ciclo de vida a nuestro favor */

Utilizando el ciclo de vida a nuestro favor

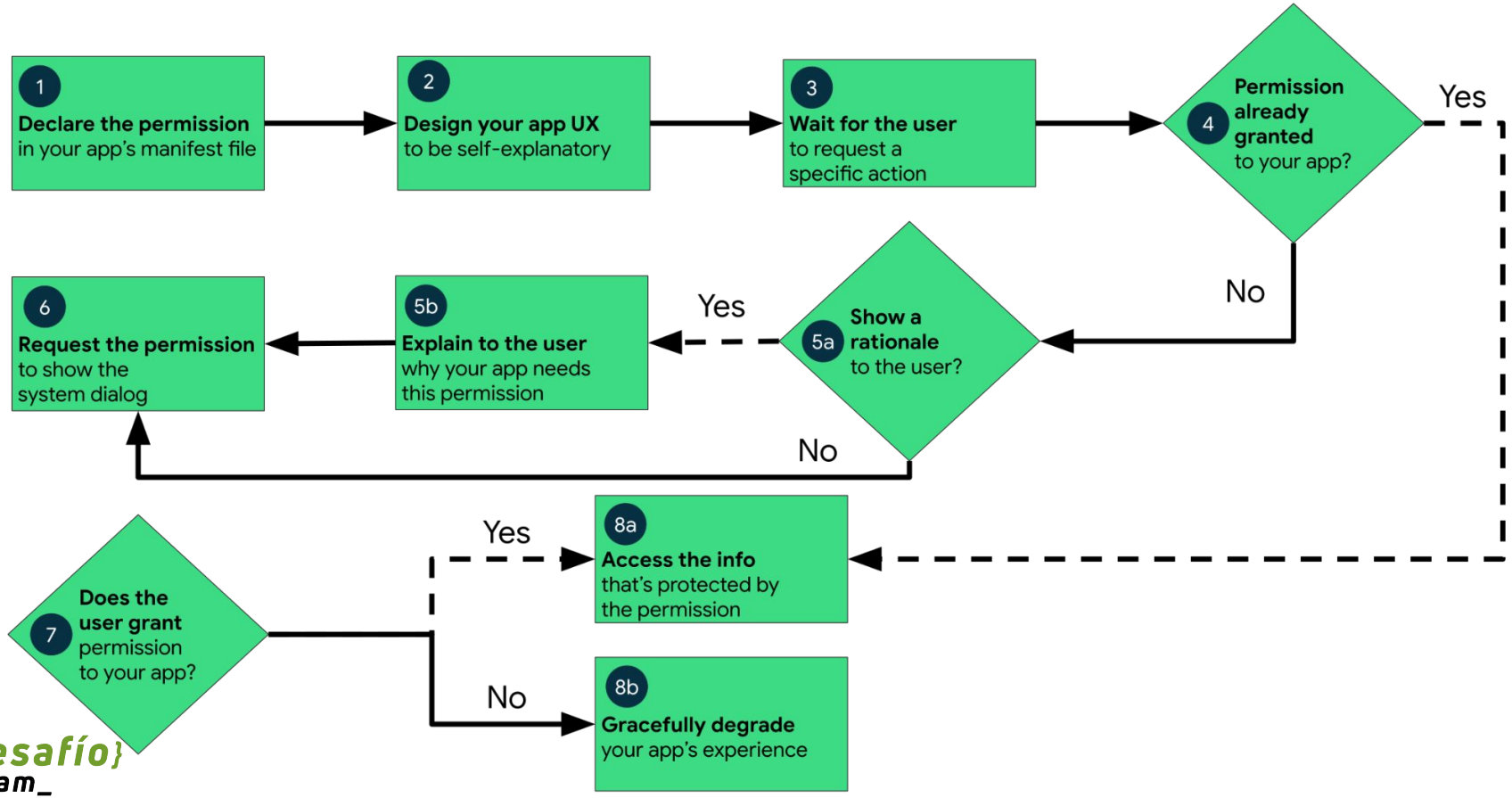
Es necesario entender correctamente cada uno de los estados del ciclo de vida de la app y así evitar errores que nos lleven a:

- ANR (Application Not Responding)
- Crashes
- Pérdida de información de los usuarios
- Largos procesos en foreground

Lo que se traduce finalmente como una mala experiencia de usuario

/* Solicitud de permisos sensibles */

Solicitud de permisos sensibles



{desafío} latam_

Solicitud de permisos sensibles

Para la solicitud de permisos se deben seguir los siguientes pasos

1. Declarar los permisos necesarios en el Manifest.
2. Diseñar experiencia de usuario que sea autoexplicativa.
3. No solicitar permisos durante la ejecución si estos no son requeridos para el correcto funcionamiento.
4. Validar si los permisos ya han sido otorgados antes de preguntar.
5. Ser claros al explicar por qué la app requiere dichos permisos.
6. En caso de que el usuario no otorgue permiso, respetar la decisión del usuario.

¡Revisemos este desarrollo de manera más detallada!

Solicitud de permisos sensibles

El código necesario para la solicitud de permisos ha sido optimizado en los últimos años.

```
val requestPermissionLauncher =
    registerForActivityResult(RequestPermission()
    ) { isGranted: Boolean ->
        if (isGranted) {
                // Ejecutar la función que requiere el permiso
        } else {
// Explicar al usuario por qué la app requiere permiso
        }
    }
```

Solicitud de permisos sensibles

Antes de preguntar por algún permiso, es bueno validar si ya fue concedido anteriormente:

```
when {
    ContextCompat.checkSelfPermission(
            CONTEXT,
            Manifest.permission.REQUESTED_PERMISSION
            ) == PackageManager.PERMISSION_GRANTED -> {
        // You can use the API that requires the permission.
    }
    shouldShowRequestPermissionRationale(...) -> {
        showInContextUI(...)
    }
    else -> {

requestPermissionLauncher.launch(Manifest.permission.REQUESTED_PERMISSION)
    }
}
```

¡Recomendaciones que no debes olvidar!

- No pidas permisos que NO necesites.
- Respeta la decisión del usuario.

Próxima sesión...

- *Se revisará una **guía de ejercicios** donde se aprenderá a utilizar elementos del ciclo de vida en Activities y Fragments para dar solución a un problema planteado*

{desafío}
latam_

Academia de
talentos digitales