

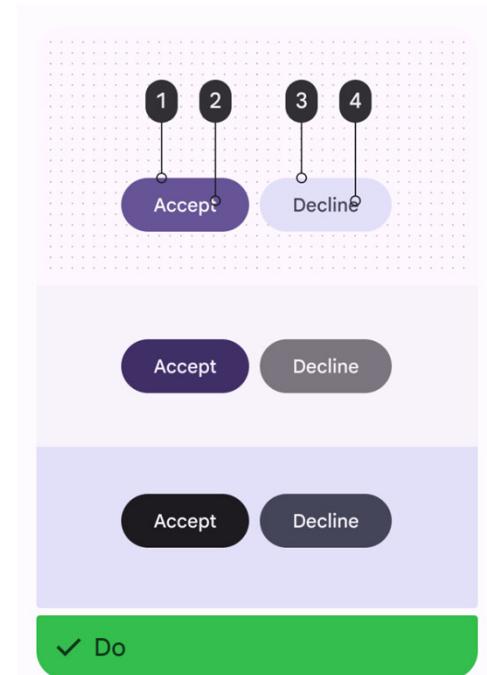
UE TAC

Avant de parler de navigation...

Petit aparté sur les couleurs...

Documentation officielle : <https://m3.material.io/styles/color/overview>

Primary		Secondary		Tertiary		Error
On Primary		On Secondary		On Tertiary		On Error
Primary Container		Secondary Container		Tertiary Container		Error Container
On Primary Container		On Secondary Container		On Tertiary Container		On Error Container
Primary Fixed	Primary Fixed Dim	Secondary Fixed	Secondary Fixed Dim	Tertiary Fixed	Tertiary Fixed Dim	
On Primary Fixed		On Secondary Fixed		On Tertiary Fixed		
On Primary Fixed Variant		On Secondary Fixed Variant		On Tertiary Fixed Variant		
Surface Dim		Surface		Surface Bright		Inverse Surface
Surface Container Lowest	Surface Container Low	Surface Container	Surface Container High	Surface Container Highest		Inverse On Surface
On Surface		On Surface Variant		Outline	Outline Variant	Inverse Primary
					Scrim	Shadow



Pair and layer color roles as intended to ensure expected visual results and accessibility. In this example, the two buttons mapped with (1) primary, (2) on primary, (3) secondary container, and (4) on secondary container stay legible as the contrast level changes.

...et les typographies

Ref officielle : <https://m3.material.io/styles/typography/overview>
Voir aussi <https://fonts.google.com/>

Vous pouvez utiliser vos polices

The screenshot shows the Google Fonts interface. On the left, there's a sidebar with navigation links for Fonts, Icons, Knowledge, and FAQ. The main area displays the 'Roboto' font family, which has 12 styles. It includes sections for readability tips, material design guidelines, and font optimization. A preview window on the left shows the text 'Type something' in the Roboto font at 40px. Below the preview, there are filters for Language, Writing system, and Technology, along with buttons for Variable and Color.

...et les typographies

Ref officielle : <https://m3.material.io/styles/typography/overview>

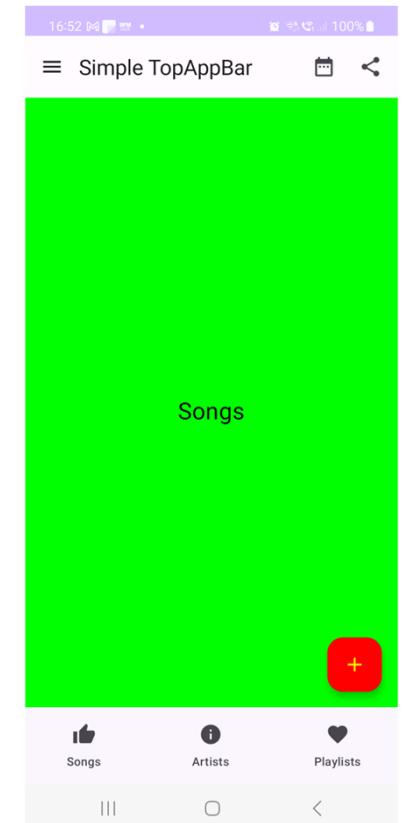
Voir aussi <https://fonts.google.com/>

The screenshot shows the Google Fonts interface. On the left, there's a sidebar with filters for 'Fonts', 'Noto', 'Icons', and 'Knowledge'. Below that is a 'Preview' section with a text input field labeled 'Type something' and a font size slider set to 40px. Underneath is a 'Filter' section with dropdowns for 'Language', 'Writing system', and 'Language'. Further down are sections for 'Technology' (with 'Variable' and 'Color' buttons), 'Everyone has the right to education', 'Edu Australia VIC WA NT Hand Guides', and 'Everyone has the right to education' again. On the right, the main content area shows the 'Google Fonts' logo, a search bar, and tabs for 'Specimen', 'Type tester', 'Glyphs', and 'About & license'. A large blue cloud-shaped callout bubble contains the text 'Vous pouvez récupérer la police'. Below it, the font 'Edu Australia VIC WA NT Hand Dots' is displayed, designed by Tina Anderson and Corey Anderson. A preview of the font is shown with the text 'Whereas disregard and contempt for human rights have resulted'. At the bottom, there are dropdowns for 'Select preview text', 'Writing system', and 'Language'.

Vous pouvez
récupérer la police

Ajouter des icônes à votre application

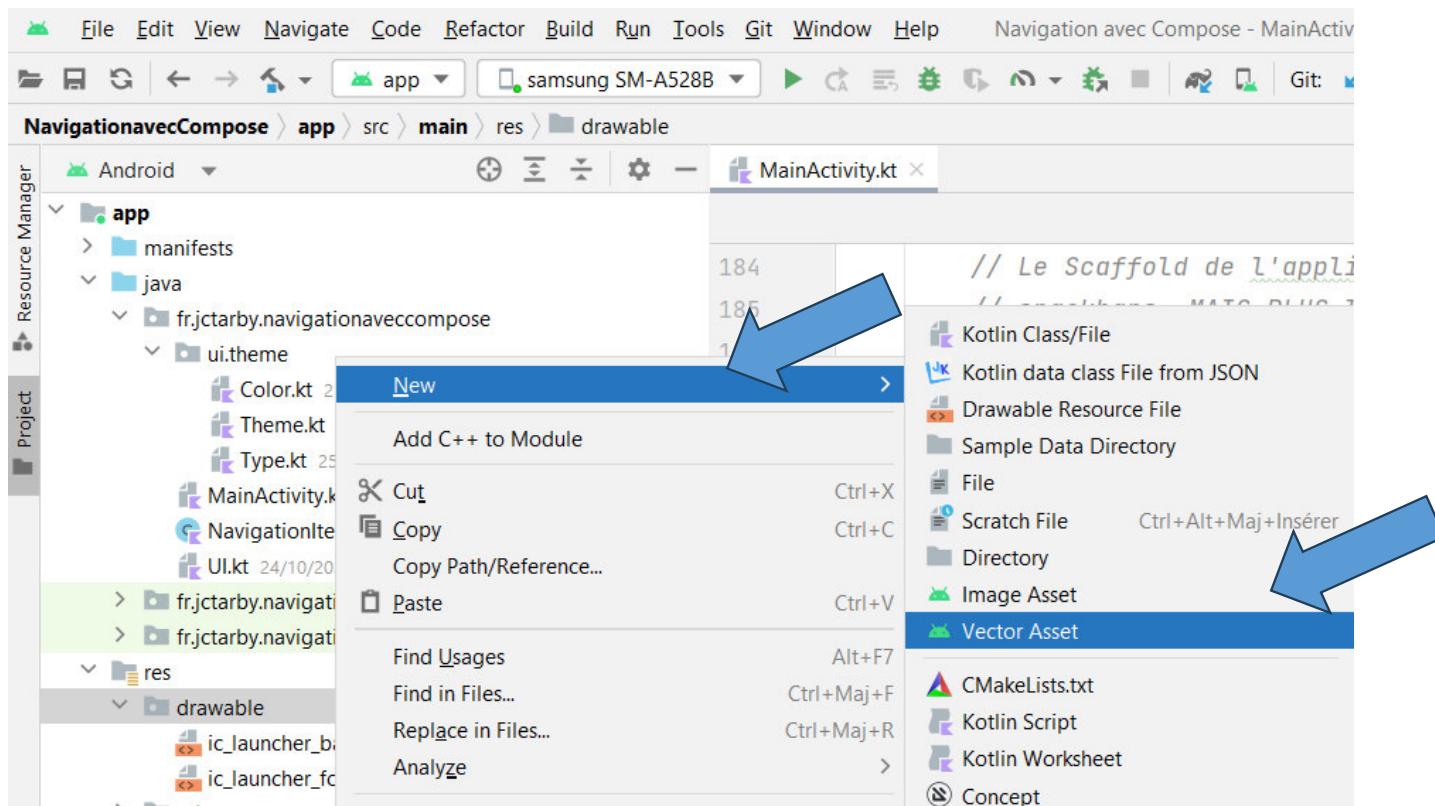
- Vous pouvez ajouter des icônes depuis Android Studio
 - Icônes fournies avec AS
 - Icônes fournies par Material3
 - Cf. <https://m3.material.io/styles/icons/overview> et <https://fonts.google.com/icons>
 - Vos propres fichiers



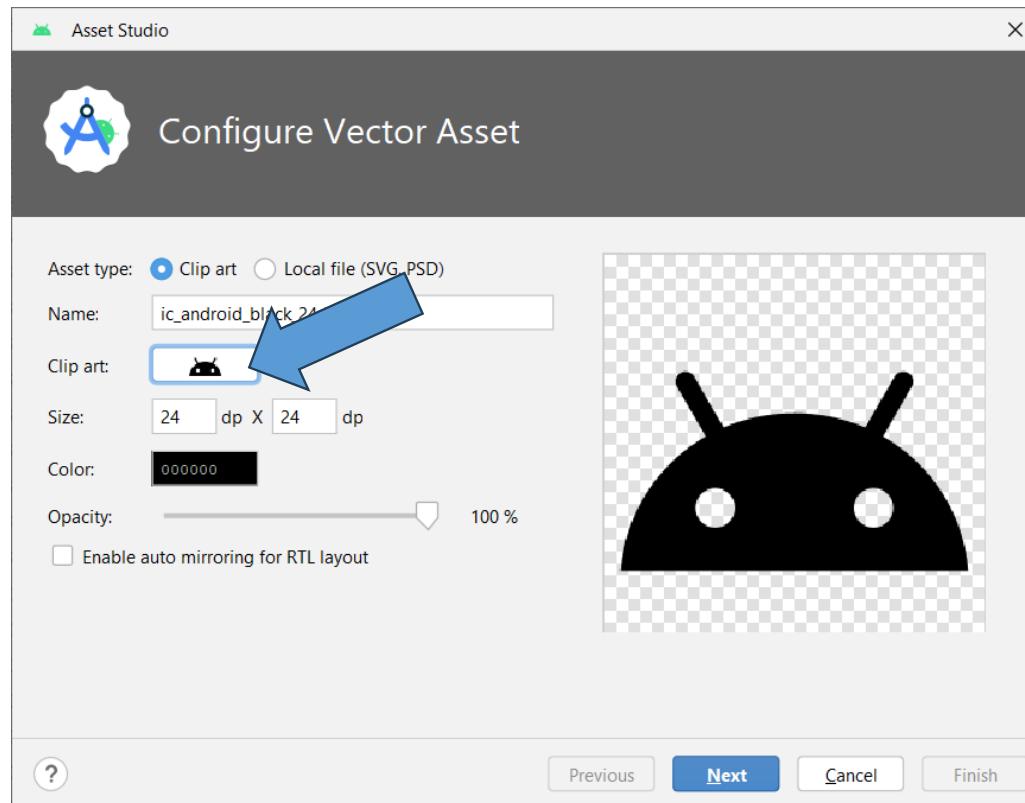
Ajouter des icônes (internes à Android Studio)

- Vous pouvez ajouter des icônes depuis Android Studio
 - Icônes fournies avec AS

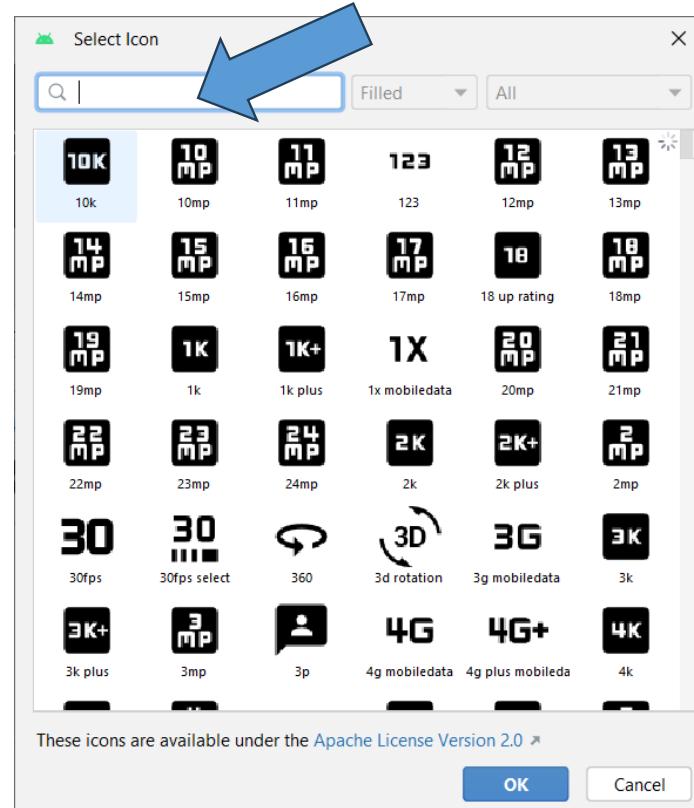
Ajouter des icônes (internes à Android Studio)



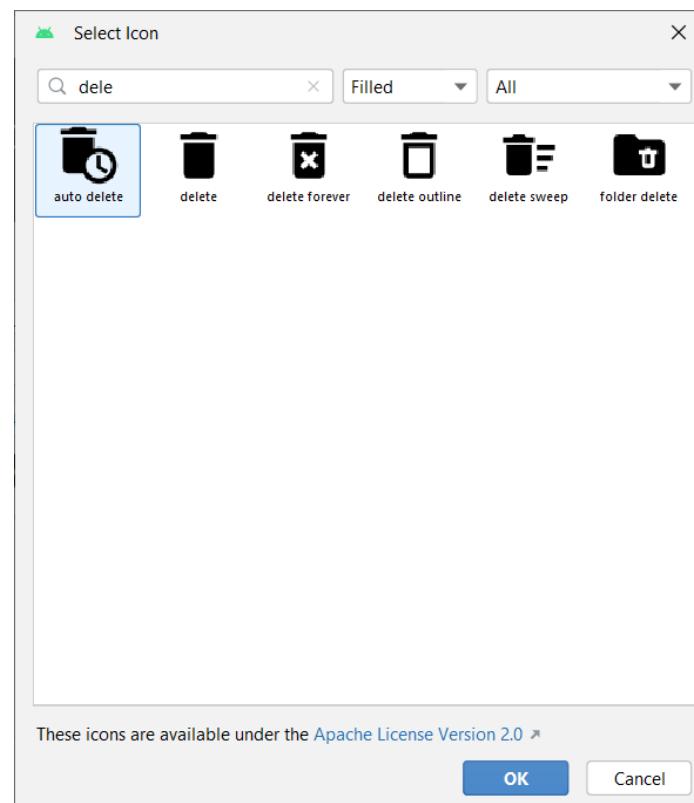
Ajouter des icônes (internes à Android Studio)



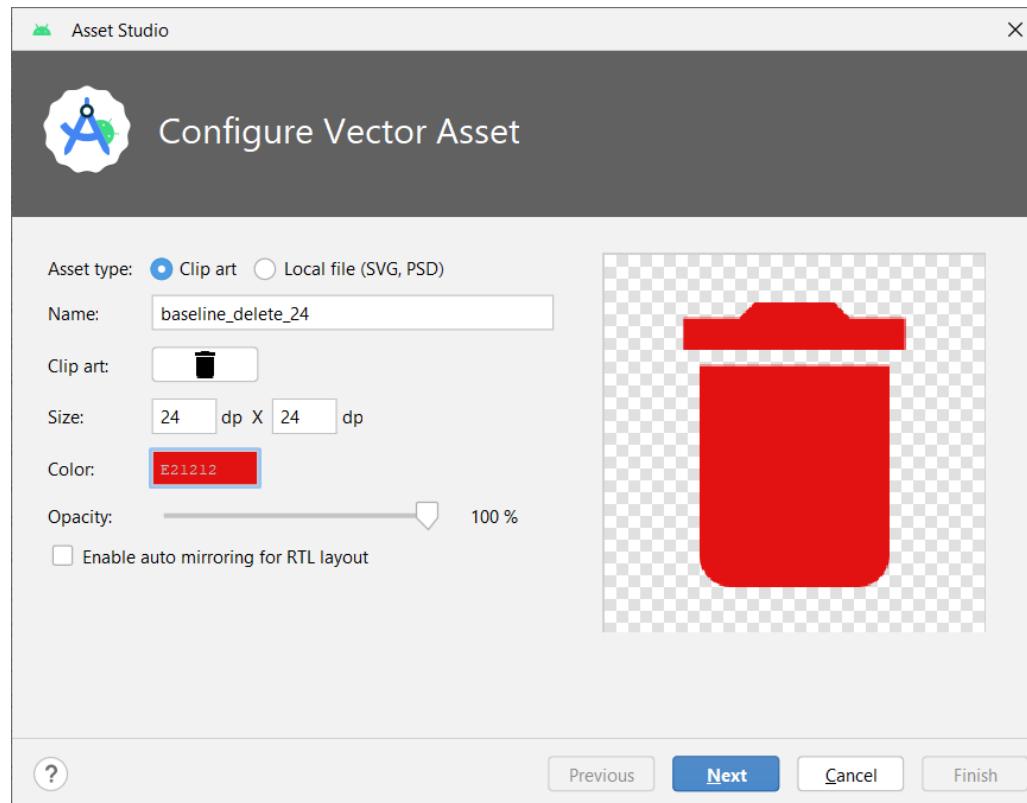
Ajouter des icônes (internes à Android Studio)



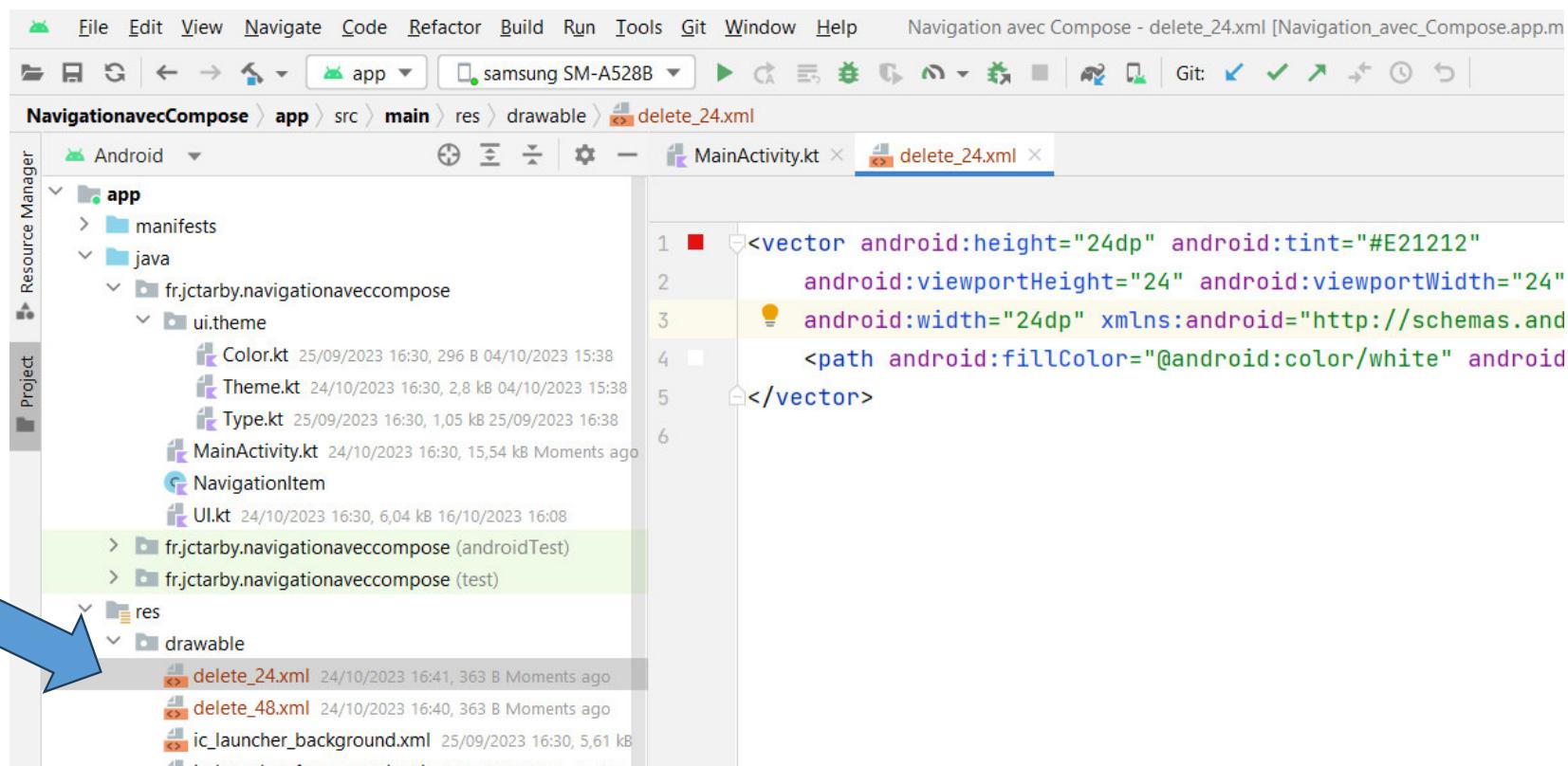
Ajouter des icônes (internes à Android Studio)



Ajouter des icônes (internes à Android Studio)



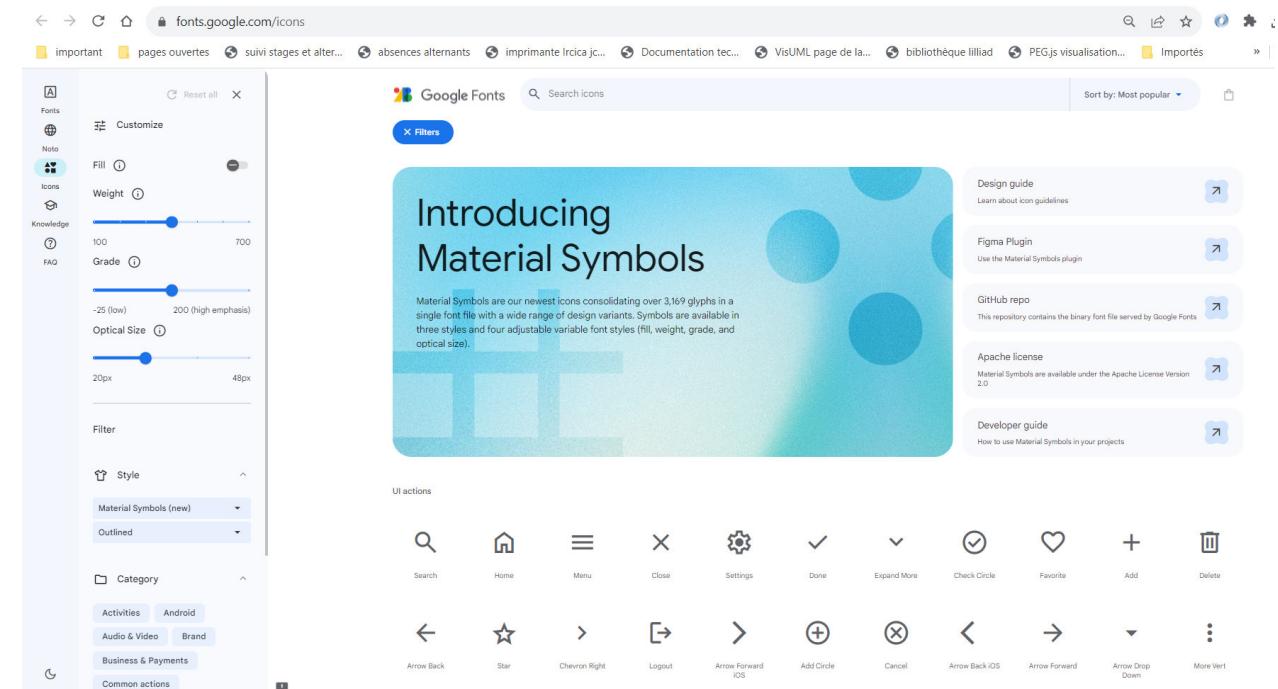
Ajouter des icônes (internes à Android Studio)



Ajouter des icônes (Material 3)

Ajouter des icônes (Material 3)

- Icônes fournies par Material3
 - Cf. <https://m3.material.io/styles/icons/overview> et <https://fonts.google.com/icons>



Ajouter des icônes (Material 3)

```
73 Icons.Filled.  
74 // f runCatching {...} (block: Icons.Filled.(... Result<R>  
75 // f takeIf {...} (predicate: (Icons.Fill... Icons.Filled?  
76 // f takeUnless {...} (predicate: (Icons... Icons.Filled?  
77 // Settings for Icons.Filled in androidx... ImageVector  
78 // ShoppingCart for Icons.Filled in andro... ImageVector  
79 // Person for Icons.Filled in androidx.co... ImageVector  
80 // Home for Icons.Filled in androidx.comp... ImageVector  
81 // Menu for Icons.Filled in androidx.comp... ImageVector  
82 // Favorite for Icons.Filled in androidx... ImageVector  
83 // Info for Icons.Filled in androidx.comp... ImageVector  
84 // ThumbUp for Icons.Filled in androidx.c... ImageVector  
85 // Warning for Icons.Filled in androidx.c... ImageVector  
86 // Share for Icons.Filled in androidx.com... ImageVector  
87 // DateRange for Icons.Filled in androidx... ImageVector  
88 // + Add for Icons.Filled in androidx.compo... ImageVector  
89 // : MoreVert for Icons.Filled in androidx... ImageVector  
90 // : Email for Icons.Filled in androidx.com... ImageVector  
91 // : Face for Icons.Filled in androidx.comp... ImageVector  
92 // < ArrowBack for Icons.Filled in androidx... ImageVector  
93 // Delete for Icons.Filled in androidx.co... ImageVector  
94 // AccountBox for Icons.Filled in android... ImageVector  
95 // AccountCircle for Icons.Filled in andr... ImageVector  
96 // AddCircle for Icons.Filled in androidx... ImageVector  
97 // ArrowDropDown for Icons.Filled in andr... ImageVector  
98 // ArrowForward for Icons.Filled in andro... ImageVector  
99 // Build for Icons.Filled in androidx.com... ImageVector
```

Icons.Filled.

Icons.Outlined.

Icons.Rounded.

Icons.TwoTone.

16

```
73 Icons.Filled.  
74 // f runCatching {...} (block: Icons.Filled.(... Result<R>  
75 // f takeIf {...} (predicate: (Icons.Fill... Icons.Filled?  
76 // f takeUnless {...} (predicate: (Icons... Icons.Filled?  
77 // Info for Icon  
78 // AccountBox for  
79 // AccountCircle for  
80 // + Add for I  
81 // AddCircle for  
82 // ArrowBack for  
83 // AccountBo  
84 // AccountCi  
85 // + Add for I  
86 // AddCircle for  
87 // ArrowBack for  
88 // AccountBo  
89 // AccountCi  
90 // + Add for I  
91 // AddCircle for  
92 // ArrowBack for  
93 // AccountBo  
94 // AccountCi  
95 // + Add for I  
96 // AddCircle for  
97 // ArrowBack for  
98 // AccountBo  
99 // AccountCi
```

Icons.Filled.

Icons.Outlined.

Icons.Rounded.

Icons.TwoTone.

16

TD 2 : Navigation

Exercice en préambule

- A partir d'un projet « vide », faites afficher dans des Toasts les différentes étapes du cycle de vie d'une activité (la MainActivity)
 - Objectif : voir que l'activité est bien détruite et recréée (?) en cas de rotation de l'écran
 - Testez déjà simplement avec le projet vide de départ
 - Puis essayez avec « **onConfigurationChanged** »
 - Et enfin avec « **resources.configuration.orientation** »
- Cf. détails ci-après

Gestion de la rotation d'écran

- Plusieurs mécanismes possibles
 - `onConfigurationChanged`
 - `resources.configuration.orientation`
 - une rotation d'écran sans rien de spécial

Solution 1 :

onConfigurationChanged

- Callback du cycle de vie Android classique (hérité d'Activity ou de ComponentActivity).
- **Quand la configuration du téléphone change** (rotation, changement de langue, passage en mode nuit, changement de taille de police...), **Android recrée l'activité**.
- Mais si modification du Manifest avec

```
<activity  
    android:name=".MainActivity"  
    android:configChanges="orientation|screenSize">
```

Alors l'Activity ne sera pas recréée, et la méthode onConfigurationChanged(newConfig: Configuration) sera appelée.

C'est l'ancienne manière de gérer les rotations. Aujourd'hui, en Compose, on évite généralement cette approche, sauf cas très particulier (ex. lecteur vidéo qui doit garder l'état inchangé).

Solution 2 :

resources.configuration.orientation

- Façon simple d'obtenir l'orientation actuelle à un instant donné.
- Pas besoin de gérer `onConfigurationChanged`
 - Compose observe automatiquement la configuration et recompose l'IHM quand l'orientation change.
- **C'est la bonne pratique moderne avec Compose**

```
val orientation = LocalConfiguration.current.orientation

if (orientation == Configuration.ORIENTATION_LANDSCAPE) {
    // Mode paysage
} else {
    // Mode portrait
}
```

Solution 3 : Rotation sans rien de spécial

- Par défaut :
 - Quand on tourne l'écran, Android détruit et recrée l'activité
 - Compose redessine l'IHM avec la nouvelle orientation.
- Les données dans remember { ... } disparaissent, car elles sont liées à l'Activity.
 - Pour conserver l'état, rememberSaveable { ... } ou bien un ViewModel (survit à la recréation de l'activité)

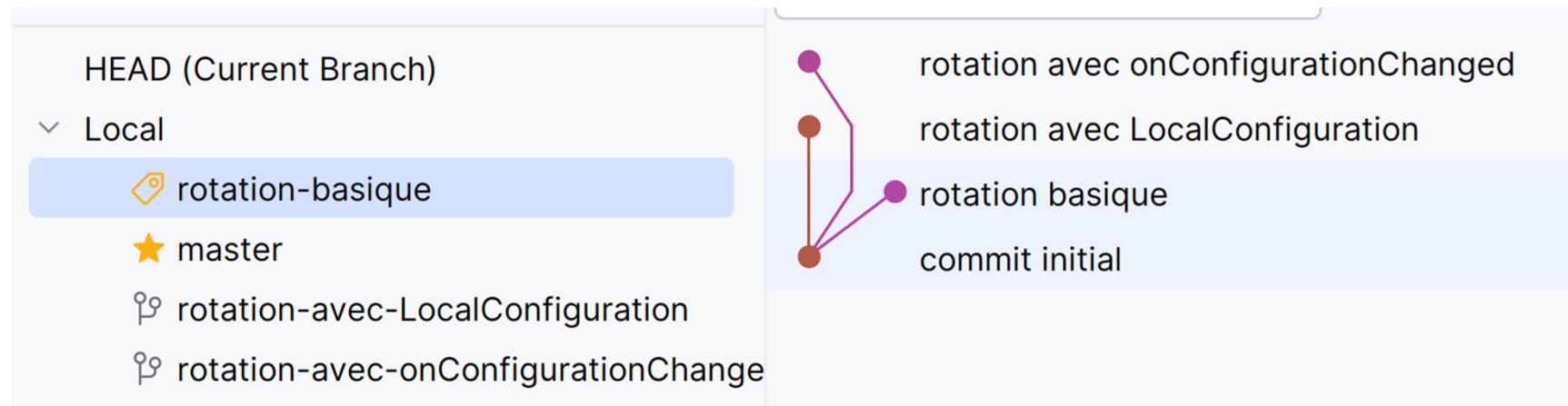
```
var text by rememberSaveable { mutableStateOf("") }
TextField(value = text, onValueChange = { text = it })
```

En résumé

- **onConfigurationChanged**
 - Ancienne méthode.
 - Gestion manuelle des changements, souvent **déconseillé aujourd'hui en Compose**.
- **resources.configuration.orientation (ou LocalConfiguration)**
 - **Méthode moderne**
 - Compose observe automatiquement et recompose.
- Rotation sans rien de spécial
 - L'activité est recréée, l'IHM est redessinée.
 - Pour garder l'état, on utilise **rememberSaveable ou ViewModel**.
- En Jetpack **Compose** :
 - Utilise **LocalConfiguration.current.orientation** pour adapter ton UI.
 - Utilise **rememberSaveable ou ViewModel** pour conserver les données à travers les rotations

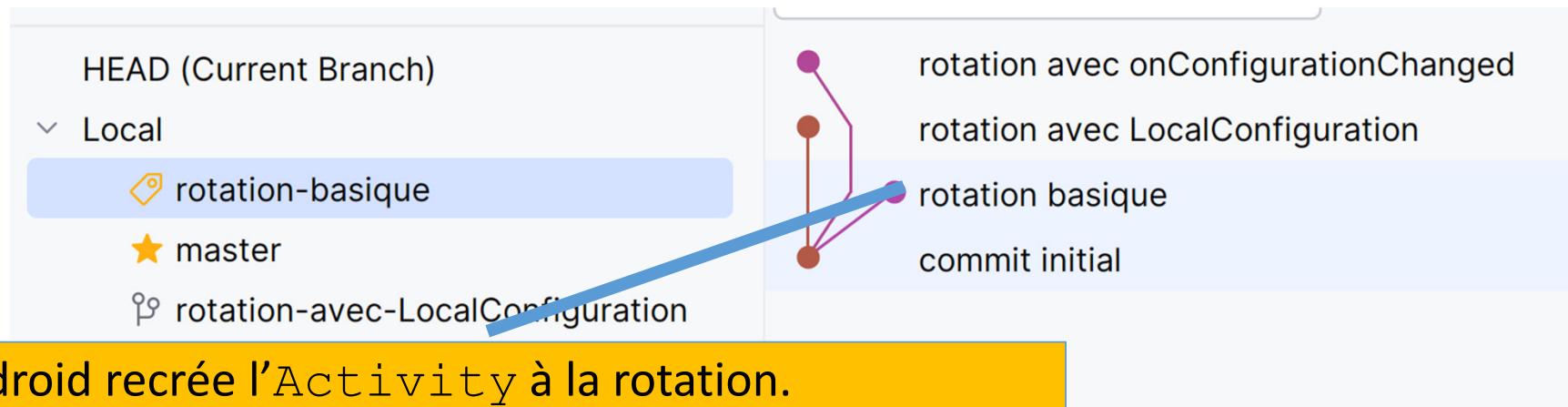
Exemples de gestion de rotation

- <https://gitlab.com/m1-ue-tac/compose/gestion-rotation-ecran.git>



Exemples

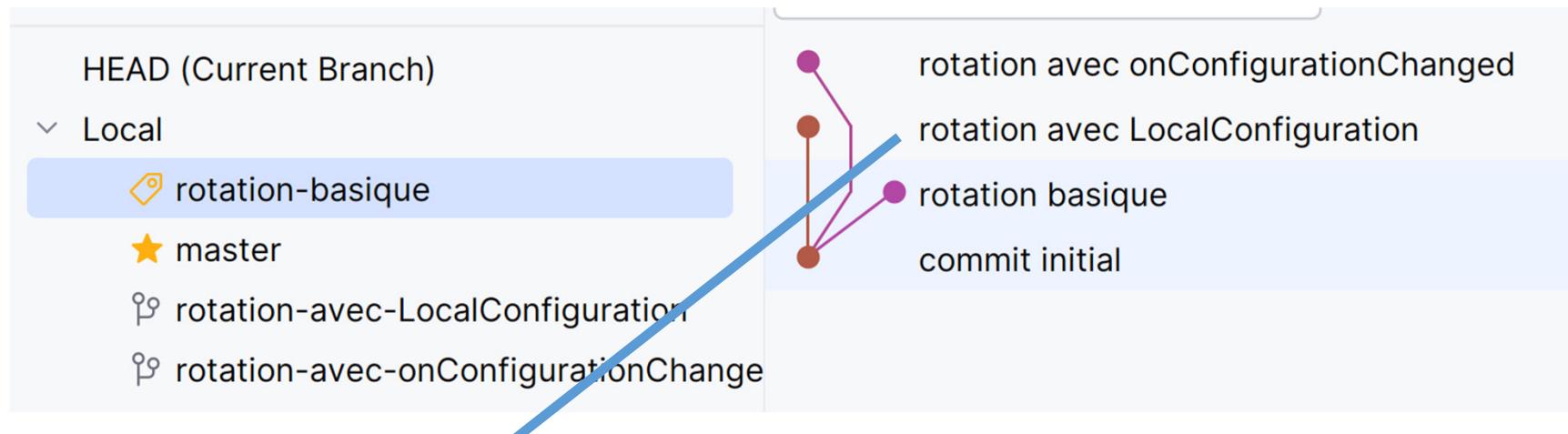
- <https://gitlab.com/m1-ue-tac/compose/gestion-rotation-ecran.git>



Android recrée l'Activity à la rotation.
On **préserve l'état** avec rememberSaveable (ou un ViewModel).

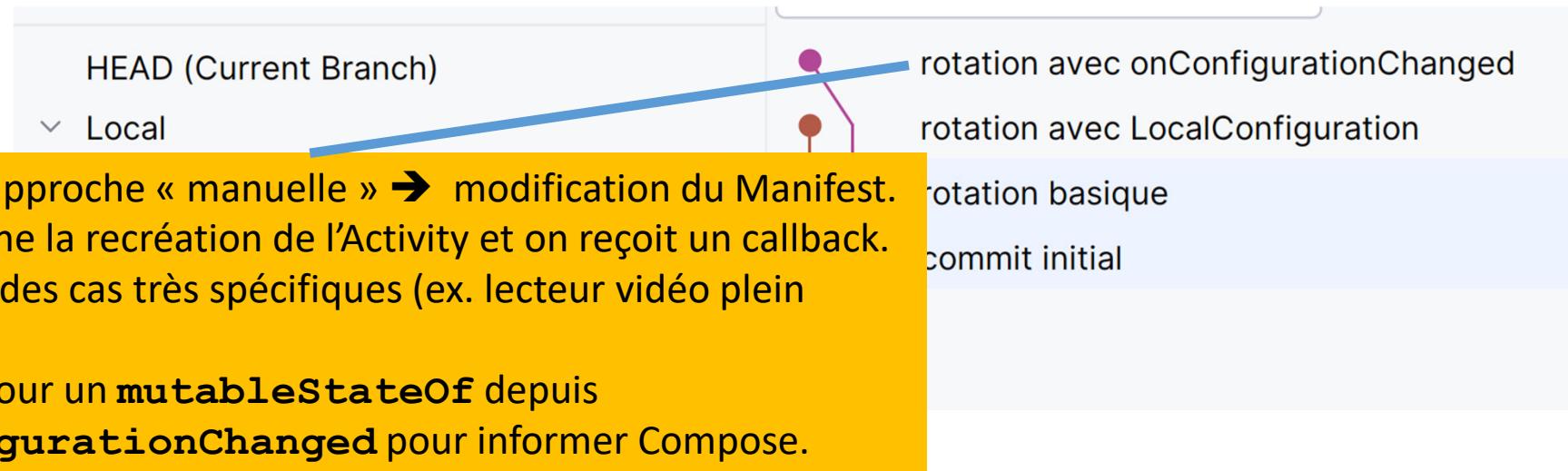
Exemples

- <https://gitlab.com/m1-ue-tac/compose/gestion-rotation-ecran.git>



Exemples

- <https://gitlab.com/m1-ue-tac/compose/gestion-rotation-ecran.git>



Exemples

- <https://gitlab.com/m1-ue-tac/compose/gestion-rotation-ecran.git>

Remarques importantes pour ce cas :

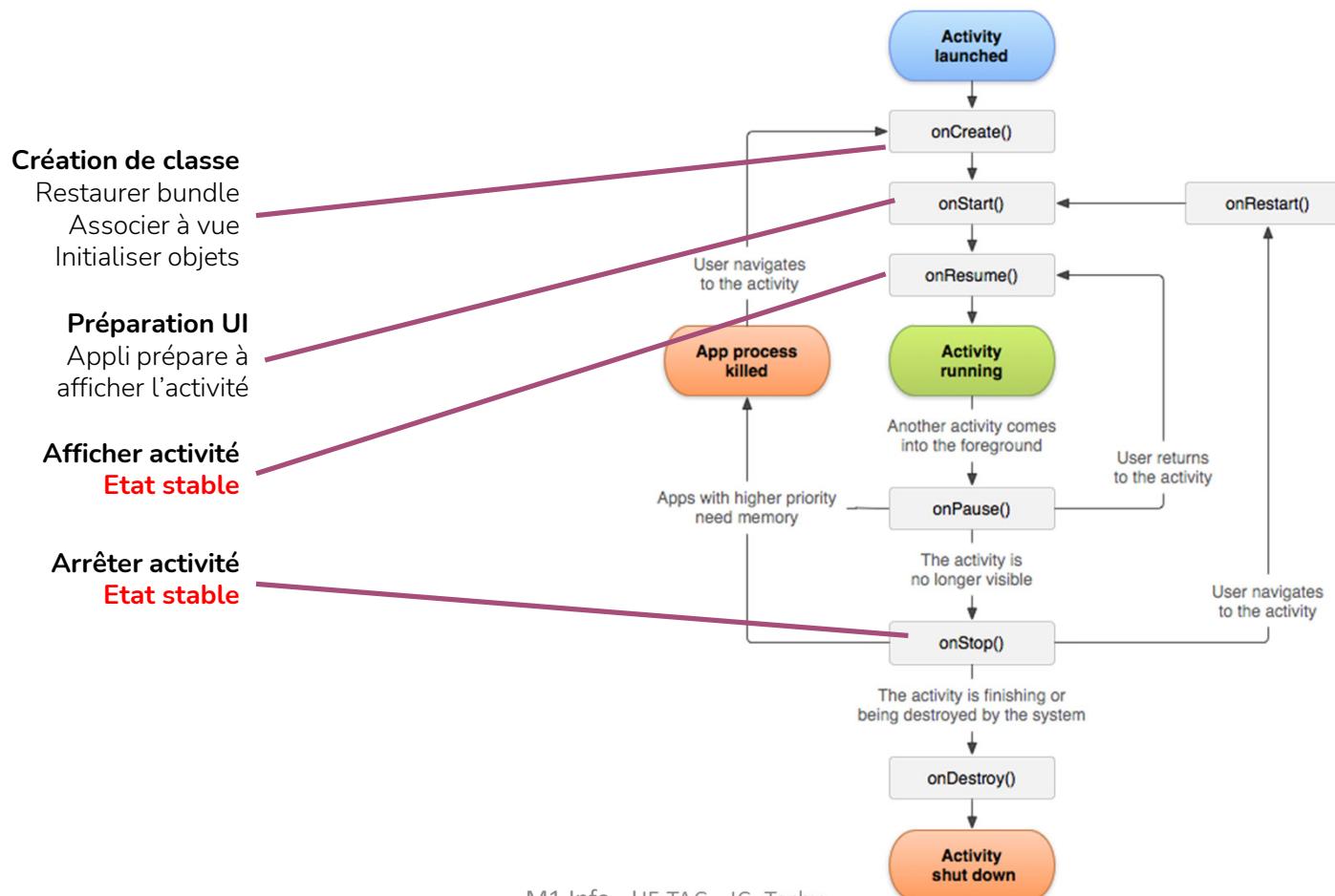
- En empêchant la recréation, **on assume la gestion de beaucoup d'autres changements** (taille d'écran, multifenêtre, langue, mode nuit...).
- On doit gérer l'IHM soi-même en grande partie.

ancienne approche « manuelle » → modification du Manifest.
On empêche la recréation de l'Activity et on reçoit un callback.
Utile pour des cas très spécifiques (ex. lecteur vidéo plein écran).
On met à jour un **mutableStateOf** depuis
onConfigurationChanged pour informer Compose.

Quel cas utiliser et quand ?

- Cas 1 (par défaut) :
 - le plus simple → ok si on sauvegarde l'état avec `rememberSaveable` ou un `ViewModel`.
- Cas 2 (recommandé en Compose) :
 - on laisse Android gérer la rotation et on adapte l'IHM avec `LocalConfiguration`.
- Cas 3 (`onConfigurationChanged`) :
 - seulement si on a une bonne raison d'éviter la recréation (lecture vidéo, perf ultra-fine, etc.).

Rappel : cycle de vie d'une activité



Solutions possibles

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Toast.makeText(this, "onCreate called", Toast.LENGTH_SHORT).show()
        ...
    }

    override fun onStart() {
        super.onStart()
        Toast.makeText(this, "onStart called", Toast.LENGTH_SHORT).show()
    }

    override fun onResume() {
        super.onResume()
        Toast.makeText(this, "onResume called", Toast.LENGTH_SHORT).show()
    }

    override fun onPause() {
        super.onPause()
        Toast.makeText(this, "onPause called", Toast.LENGTH_SHORT).show()
    }
    ...
}
```

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val context = LocalContext.current

            // Observer pour le cycle de vie
            val lifecycle = LocalLifecycleOwner.current.lifecycle
            DisposableEffect(lifecycle) {
                val observer = LifecycleEventObserver { _, event ->
                    when (event) {
                        Lifecycle.Event.ON_CREATE -> context.toast("onCreate called")
                        Lifecycle.Event.ON_START -> context.toast("onStart called")
                        Lifecycle.Event.ON_RESUME -> context.toast("onResume called")
                        Lifecycle.Event.ON_PAUSE -> context.toast("onPause called")
                        Lifecycle.Event.ON_STOP -> context.toast("onStop called")
                        Lifecycle.Event.ON_DESTROY -> context.toast("onDestroy called")
                        else -> /* ignore all other events */
                    }
                }
                lifecycle.addObserver(observer)
            }
            // Supprimer l'observer lorsqu'il n'est plus nécessaire
            onDispose {
                lifecycle.removeObserver(observer)
            }
        }
    }
}
```

Solutions possibles

```
class MainActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        Toast.makeText(this, "onCreate called", Toast.LENGTH_SHORT).show()  
        ...  
    }  
  
    override fun onStart() {  
        super.onStart()  
        Toast.makeText(this, "onStart called", Toast.LENGTH_SHORT).show()  
    }  
  
    override fun onResume() {  
        super.onResume()  
        Toast.makeText(this, "onResume called", Toast.LENGTH_SHORT).show()  
    }  
  
    override fun onPause() {  
        super.onPause()  
        Toast.makeText(this, "onPause called", Toast.LENGTH_SHORT).show()  
    }  
    ...  
}
```

```
class MainActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContent {  
            val context = LocalContext.current  
  
            // Observer pour le cycle de vie  
            val lifecycle = LocalLifecycleOwner.current.lifecycle  
            DisposableEffect(lifecycle) {  
                val observer = LifecycleEventObserver { _, event ->  
                    when (event) {  
                        Lifecycle.Event.ON_CREATE -> context.toast("onCreate called")  
                        Lifecycle.Event.ON_START -> context.toast("onStart called")  
                        Lifecycle.Event.ON_RESUME -> context.toast("onResume called")  
                        Lifecycle.Event.ON_PAUSE -> context.toast("onPause called")  
                        Lifecycle.Event.ON_STOP -> context.toast("onStop called")  
                        Lifecycle.Event.ON_DESTROY -> context.toast("onDestroy called")  
                        else -> /* ignore all other events */  
                    }  
                }  
                lifecycle.addObserver(observer)  
  
                // Supprimer l'observer lorsqu'il n'est plus nécessaire  
                onDispose {  
                    lifecycle.removeObserver(observer)  
                }  
            }  
        }  
    }  
}
```

Solutions possibles

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Toast.makeText(this, "onCreate called", Toast.LENGTH_SHORT).show()
        ...
    }

    override fun onStart() {
        super.onStart()
        Toast.makeText(this, "onStart called", Toast.LENGTH_SHORT).show()
    }

    override fun onResume() {
        super.onResume()
        Toast.makeText(this, "onResume called", Toast.LENGTH_SHORT).show()
    }

    override fun onPause() {
        super.onPause()
        Toast.makeText(this, "onPause called", Toast.LENGTH_SHORT).show()
    }
    ...
}
```

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val context = LocalContext.current

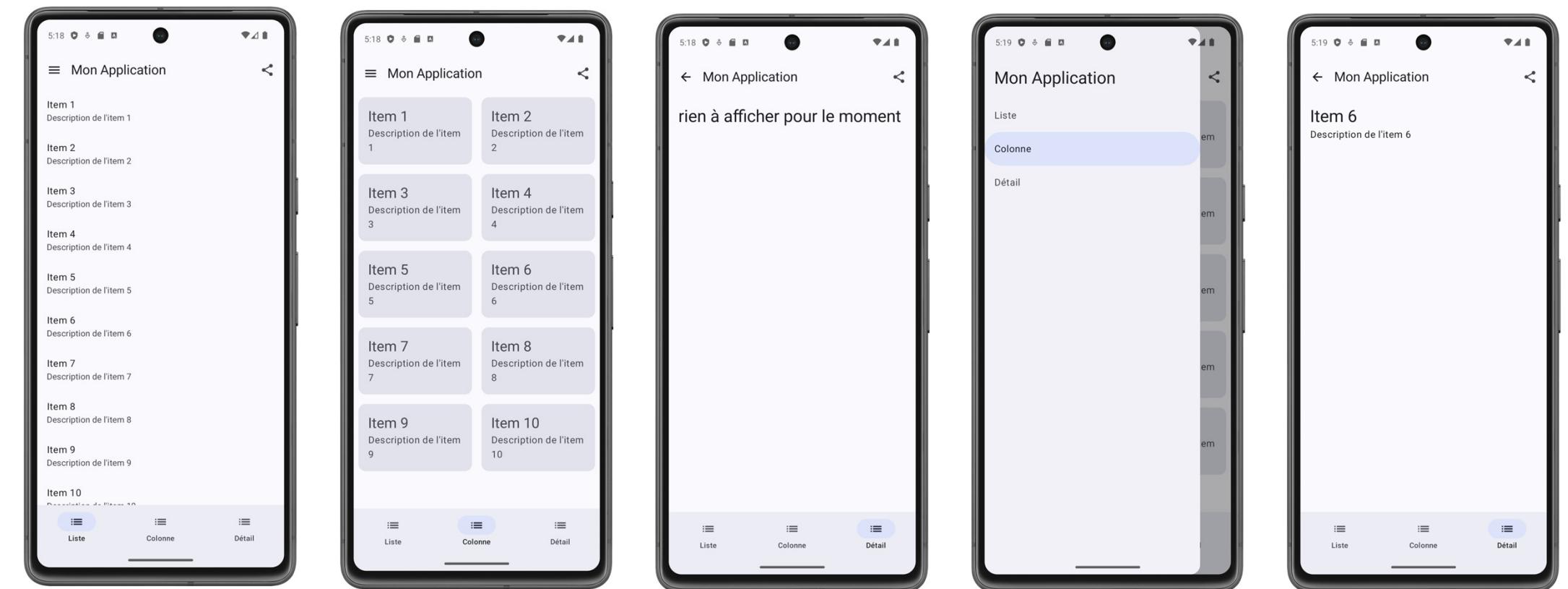
            // Observer pour le cycle de vie
            val lifecycle = LocalLifecycleOwner.current.lifecycle
            DisposableEffect(lifecycle) {
                val observer = LifecycleEventObserver { _, event ->
                    when (event) {
                        Lifecycle.Event.ON_CREATE -> context.toast("onCreate called")
                        Lifecycle.Event.ON_START -> context.toast("onStart called")
                        Lifecycle.Event.ON_RESUME -> context.toast("onResume called")
                        Lifecycle.Event.ON_PAUSE -> context.toast("onPause called")
                        Lifecycle.Event.ON_STOP -> context.toast("onStop called")
                        Lifecycle.Event.ON_DESTROY -> context.toast("onDestroy called")
                        else -> /* ignore all other events */
                    }
                }
                lifecycle.addObserver(observer)
            }
            // Supprimer l'observer lorsqu'il n'est plus nécessaire
            onDispose {
                lifecycle.removeObserver(observer)
            }
        }
    }
}

// extension pour les Toasts (merci Kotlin)
fun Context.toast(message: String, duration: Int = Toast.LENGTH_SHORT) {
    Toast.makeText(this, message, duration).show()
}
```

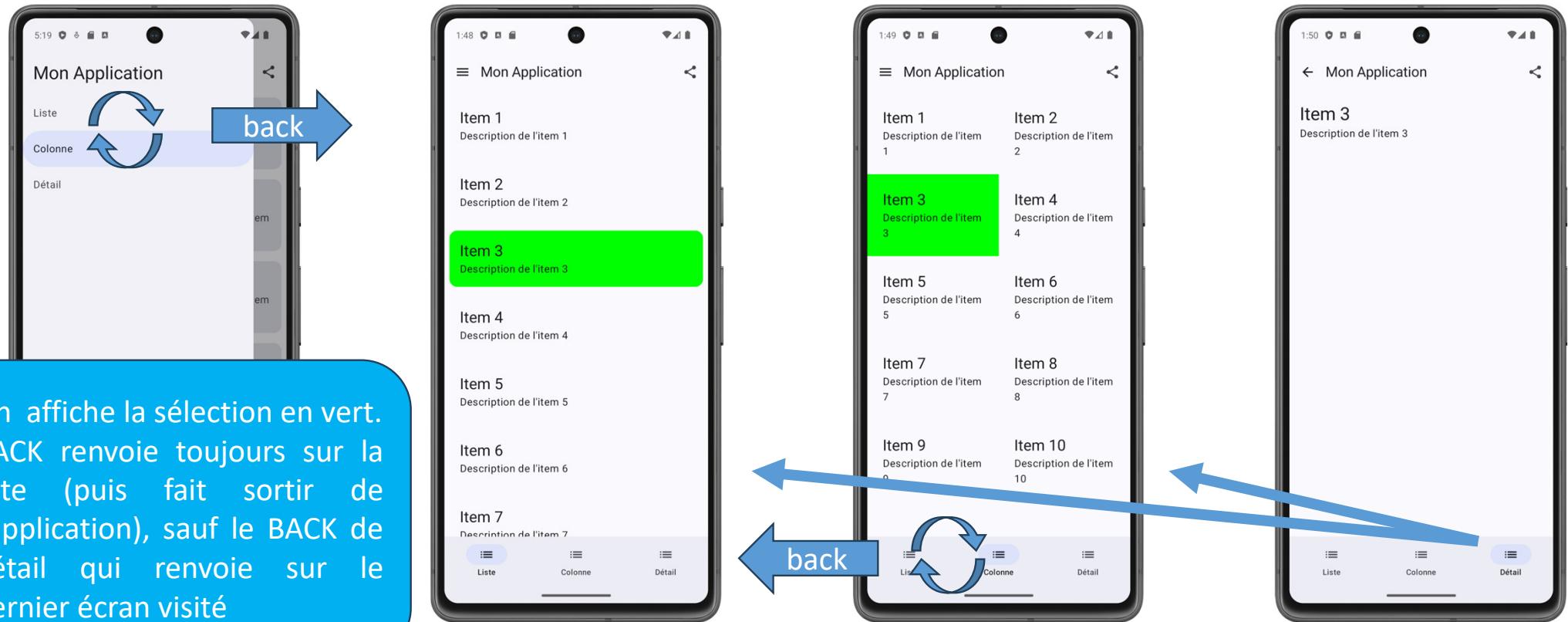
Exercice

- Recréer l'application montrée dans le cours sur la navigation, c'est-à-dire...

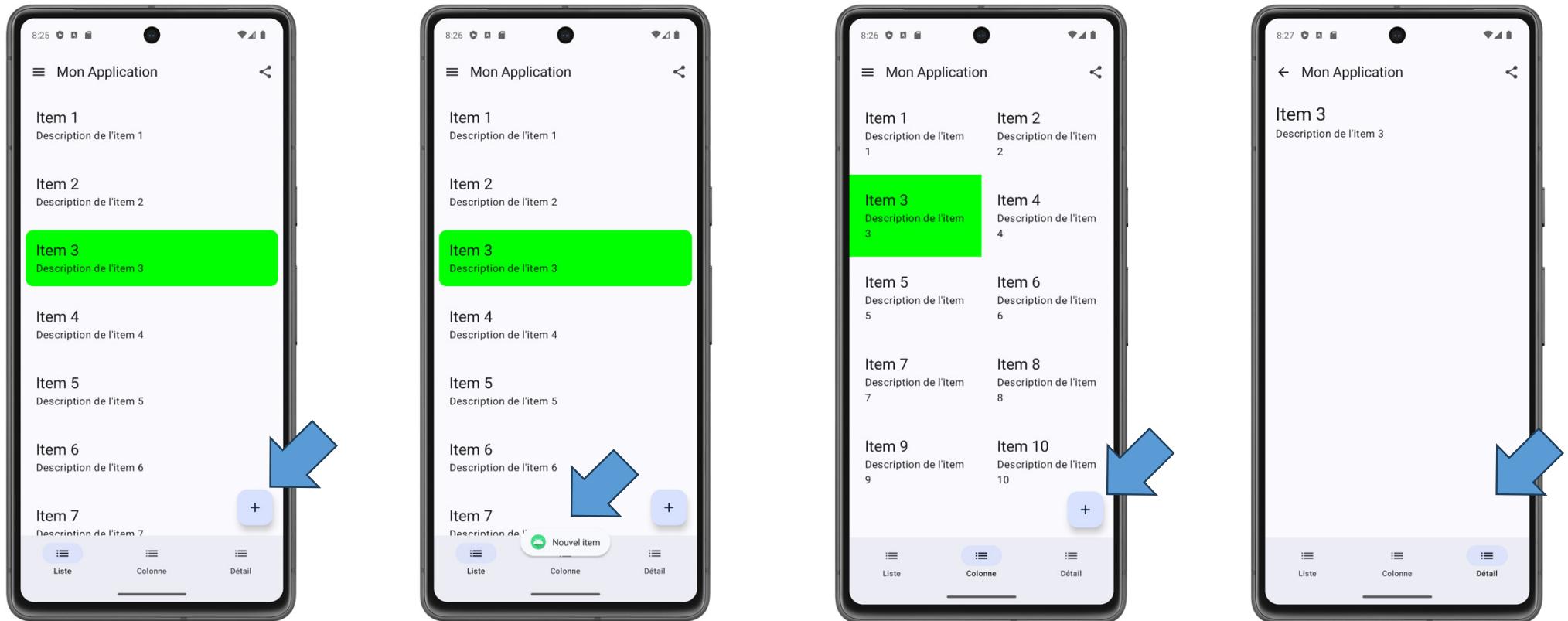
étape 1 : navigation sans gestion du bouton BACK



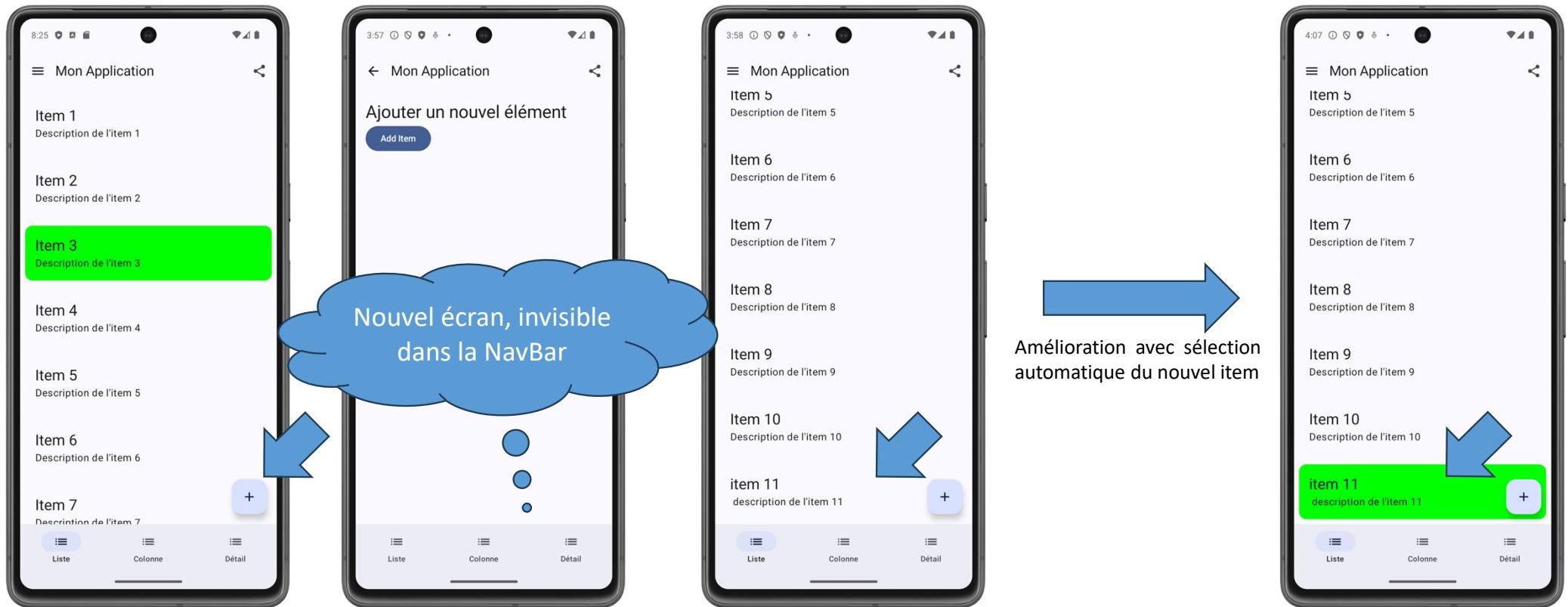
étape 2 : sélection visible + navigation avec gestion du bouton BACK



étape 3 : ajouter un Floating Button sauf sur détail



étape 4 : l'ajout est réel... et visible



étape 5 : utiliser Sealed Class (si pas encore fait)

- Pour une navigation :
 - + « sécurisée » avec usage du « when »
 - + centralisée avec une navigation centralisée dans une fonction

Conseil :

Deux méthodes pour arriver à l'étape 1 de cet exercice

1. Approche bottom-up

- a) Créez des écrans « vides » Liste, Grille et Détail, juste avec « Hello nom_écran »
- b) Dans MainActivity, ajoutez un Scaffold « vide »
- c) Ajoutez une « top bar » simple au Scaffold pour voir le résultat
 - Ajoutez éventuellement des icônes et des comportements (affichage de Toast/logs)
- d) Ajoutez une « bottom bar » au Scaffold avec 2 icônes (→ les icônes seront alignées à gauche par défaut)
- e) Remplacez la « bottom bar » par une « navigation bar » (→ les icônes seront automatiquement bien alignées)
- f) *Ajoutez la navigation sur la navigation bar (juste passage entre les écrans et sans gestion du back)*
- g) *Complétez vos écrans Liste, Grille et Détail pour avoir le vrai contenu*
- h) *Ajoutez la navigation à partir des écrans (clic sur item → détail, etc.)*
- i) *Ajoutez le menu glissant et sa navigation*
- j) *Ajoutez la gestion du back*
- k) *Optimisez si besoin votre code (navigation centralisée, sealed class...)*

Conseil :

Deux méthodes pour arriver à l'étape 1 de cet exercice

2. Approche top-down

- a) Dans MainActivity, ajoutez un Scaffold « vide »
- b) Ajoutez une « top bar » simple au Scaffold pour voir le résultat
 - Ajoutez éventuellement des icônes et des comportements (affichage de Toast/logs)
- c) Ajoutez une « bottom bar » au Scaffold avec 2 icônes (→ les icônes seront alignées à gauche par défaut)
- d) Remplacez la « bottom bar » par une « navigation bar » (→ les icônes seront automatiquement bien alignées)
- e) Créez des écrans « vides » Liste, Grille et Détail, juste avec « Hello nom_écran »
- f) *Ajoutez la navigation sur la navigation bar (juste passage entre les écrans et sans gestion du back)*
- g) *Complétez vos écrans Liste, Grille et Détail pour avoir le vrai contenu*
- h) *Ajoutez la navigation à partir des écrans (clic sur item → détail, etc.)*
- i) *Ajoutez le menu glissant et sa navigation*
- j) *Ajoutez la gestion du back*
- k) *Optimisez si besoin votre code (navigation centralisée, sealed class...)*

Pour le reste, tout est dans le cours...

- A vous !

S'il vous reste du temps...

- Créez la navigation de votre projet TAC