Développement Android avec Kotlin

Cours - 04 - Jetpack Compose Kit d'interface utilisateur

Jordan Hiertz

Contact
hiertzjordan@gmail.com
jordan.hiertz@al-enterprise.com

Jetpack Compose - Kit d'interface utilisateur

Compose simplifie la création d'applications au design cohérent en intégrant nativement les principes de Material Design.

Material Design

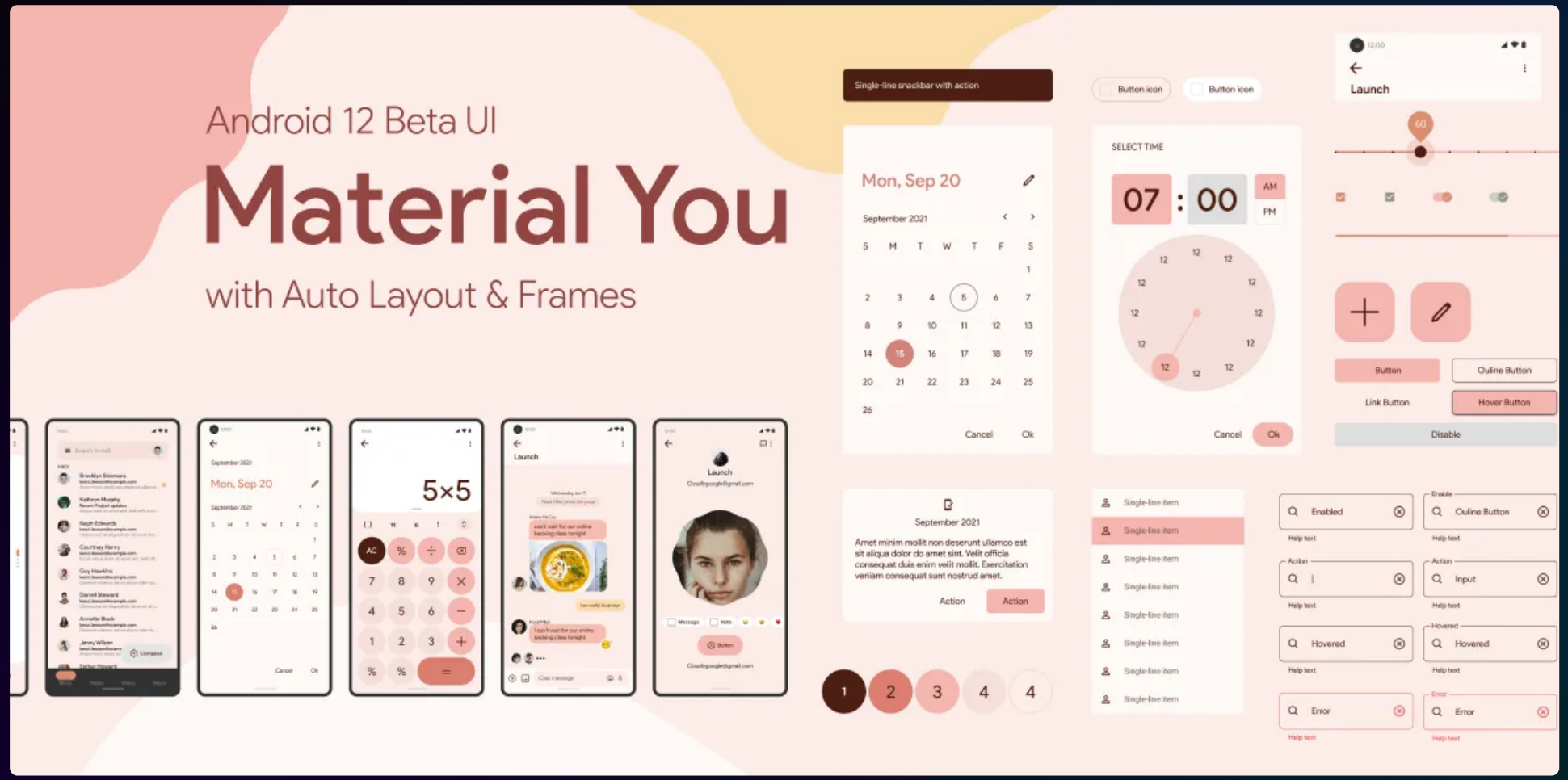
- Material Design est un système adaptable qui propose des lignes directrices, des composants et des outils pour créer des interfaces utilisateur respectant les meilleures pratiques.

- Compose prend en charge Material Design 2 et 3 (la version la plus récente), ainsi que Material You, une fonctionnalité de personnalisation conçue pour s'adapter à l'apparence et à l'ergonomie d'Android 12 et des versions ultérieures.

Android 12 Beta UI

Material You

with Auto Layout & Frames



Avant d'approfondir les thèmes...

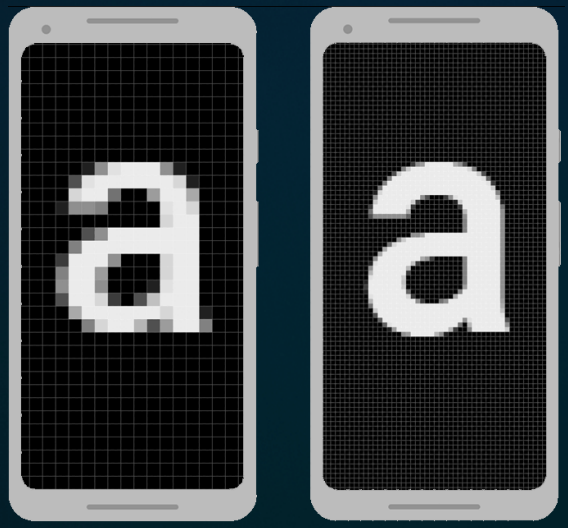
👉 Un concept clé pour le design Android : la densité de pixels

Densités de pixels

Les appareils Android ont des tailles d'écran variées : **téléphones, tablettes, téléviseurs.**

Les écrans ont aussi des tailles de pixels différentes :

- Exemple : **160 pixels par pouce** vs **480 pixels par pouce** pour le même espace.



Comment préserver des tailles cohérentes ?

- Utilisez les **density-independent pixels (dp)**.
- Pour les textes, préférez les **scalable pixels (sp)**.

Pourquoi créer un thème ?

Identité visuelle :

- Reflète la **personnalité du développeur** ou de l'**entreprise**.
- Donne une **cohérence visuelle** à l'application.

Expérience utilisateur :

- Améliore la **lisibilité** et la **navigation**.
- Garantit une adaptation fluide aux **différents écrans**.

Efficacité :

- Simplifie les modifications globales (couleurs, typographie, etc.).
- Évite la duplication de styles dans le code.
- Permet de **réutiliser le style** dans d'autres applications ou projets similaires.

https://material-foundation.github.io/material-theme-builder/

Thème

- Material Design permet de thématiser l'application pour qu'elle reflète l'identité visuelle d'une marque, en personnalisant les couleurs, les typographies et les formes.

```
MaterialTheme(
    colorScheme = MyAppsColorScheme,
    typography = MyAppsTypography,
    shapes = MyAppsShapes
) {
    // Contenu de l'application
}
```

Thème

```
@Composable
fun TP_05_CorrectionTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    // Dynamic color is available on Android 12+
    dynamicColor: Boolean = true,
    content: @Composable () -> Unit
) {
    val colors = if (darkTheme) {
        DarkColors
    } else {
        LightColors
    }

    MaterialTheme(
        colorScheme = colors,
        shapes = Shapes,
        typography = Typography,
        content = content
    )
}
```

Typographies

```
val Typography = Typography(
    bodyLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp,
        lineHeight = 24.sp,
        letterSpacing = 0.5.sp
    ),
    titleLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 22.sp,
        lineHeight = 28.sp,
        letterSpacing = 0.sp
    ),
    labelSmall = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Medium,
        fontSize = 11.sp,
        lineHeight = 16.sp,
        letterSpacing = 0.5.sp
    )
)
```

Couleurs

```
1  private val LightColors = lightColorScheme(
2      primary = md_theme_light_primary,
3      onPrimary = md_theme_light_onPrimary,
4      primaryContainer = md_theme_light_primaryContainer,
5      onPrimaryContainer = md_theme_light_onPrimaryContainer,
6      secondary = md_theme_light_secondary,
7      onSecondary = md_theme_light_onSecondary,
8      secondaryContainer = md_theme_light_secondaryContainer,
9      onSecondaryContainer = md_theme_light_onSecondaryContainer,
10     tertiary = md_theme_light_tertiary,
11     ...
12 )
13
14
15 val md_theme_light_primary = Color(0xFF6B5C4D)
```

Formes

```
val Shapes = Shapes(
    extraSmall = RoundedCornerShape(12.dp),
    small = RoundedCornerShape(10.dp)
)
```

Appliquer un thème dans Compose

```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()

    setContent {
        TP_05_CorrectionTheme {
            MyGameApp(windowSizeClass)
        }
    }
}
```

Le thème doit être appliqué au niveau le plus externe. Cela garantit que tous les composables enfants utilisent :

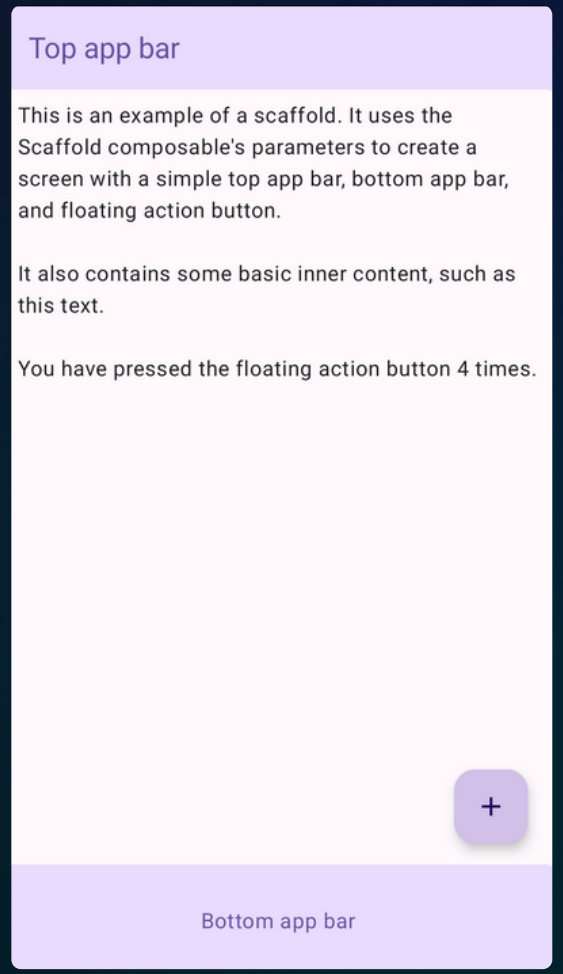
- Les **formes** définies.
- Les **typographies** personnalisées.
- Les **couleurs** du thème.

Construire une interface fluide avec les bons composables ! 🛠️💡

Scaffold : Un layout de base pour vos écrans

- Un composable fondamental qui permet d'arranger facilement les composants Material.
- Utilisé pour créer des écrans avec des éléments communs :
  - **TopAppBar** en haut.
  - **FloatingActionButton** en bas à droite.
  - **BottomBar** en bas.
  - **Contenu centré** au milieu de l'écran.

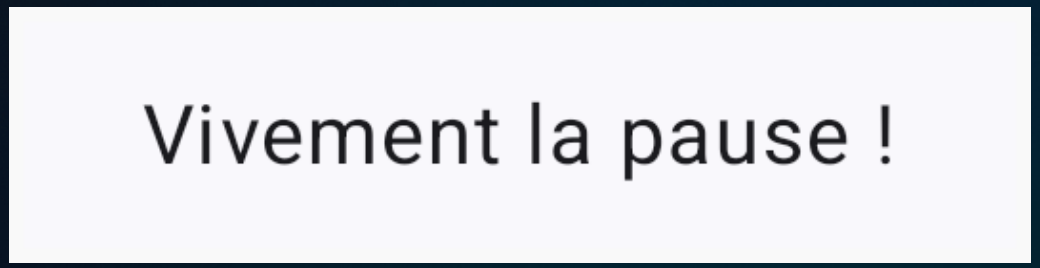
```
1 Scaffold(
2     topBar = { AppTopBar() },
3     bottomBar = { /* ... */ },
4     floatingActionButton = { /* ... */ }
5 ) { padding ->
6
7     // Le contenu de l'écran
8
9 }
```



Surface : Élément clé dans Material Design

- **Surface** est un élément clé de Material Design, servant de support pour le contenu.

```
1 Surface {
2   Text("Vivement la pause")
3 }
```



Surface : Fondement visuel dans Material Design

- **Surface** est un élément clé de Material Design, servant de support pour le contenu.

```
1 Surface (
2   color = MaterialTheme.colorScheme.primary,
3 ) {
4   Text("Vivement la pause")
5 }
```

Surface : Fondement visuel dans Material Design

- **Surface** est un élément clé de Material Design, servant de support pour le contenu.

```
1 Surface (
2   color = MaterialTheme.colorScheme.primary,
3   shape = RoundedCornerShape(percent = 10)
4 ) {
5   Text("Vivement la pause")
6 }
```

Surface : Fondement visuel dans Material Design

- **Surface** est un élément clé de Material Design, servant de support pour le contenu.

```
1  Surface (
2    color = MaterialTheme.colorScheme.primary,
3    shape = RoundedCornerShape(percent = 10),
4    border = BorderStroke(
5      width = 2.dp,
6      color = Color.Red
7    )
8  ) {
9    Text("Vivement la pause")
10 }
```

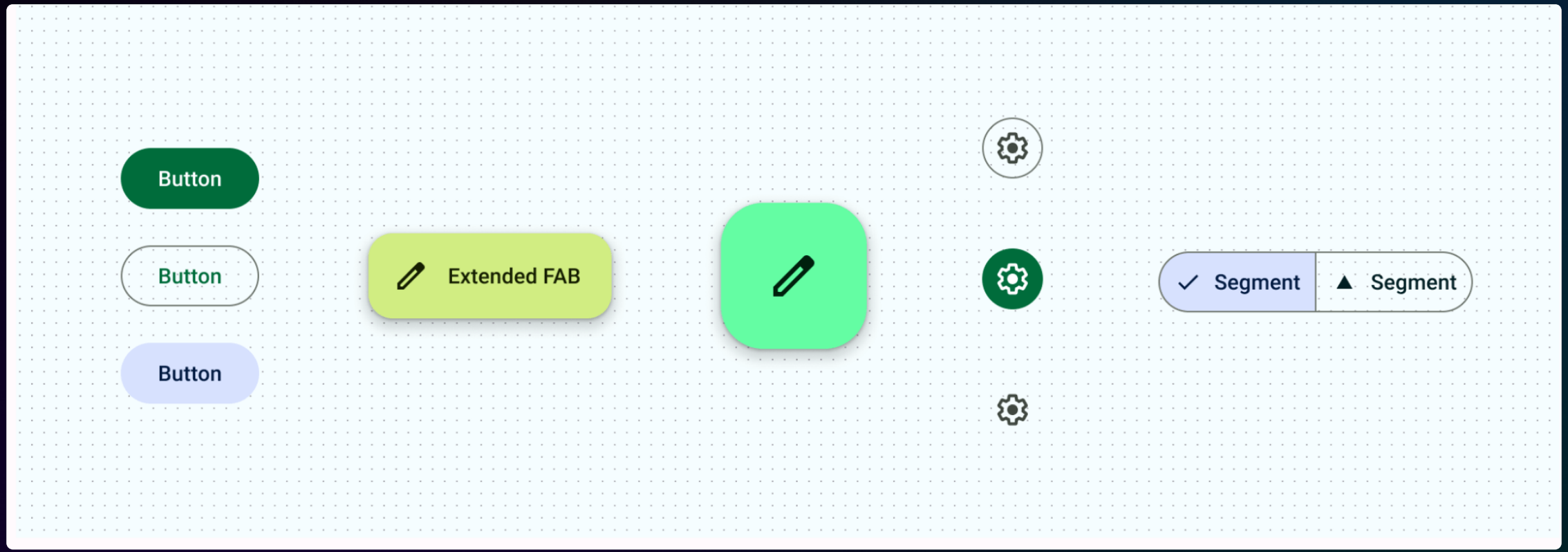
Surface : Fondement visuel dans Material Design

- **Surface** est un élément clé de Material Design, servant de support pour le contenu.

```
1 Surface (
2   color = MaterialTheme.colorScheme.primary,
3   shape = RoundedCornerShape(percent = 10),
4   shadowElevation = 8.dp,
5   tonalElevation = 8.dp
6 ) {
7   Text("Vivement la pause")
8 }
```

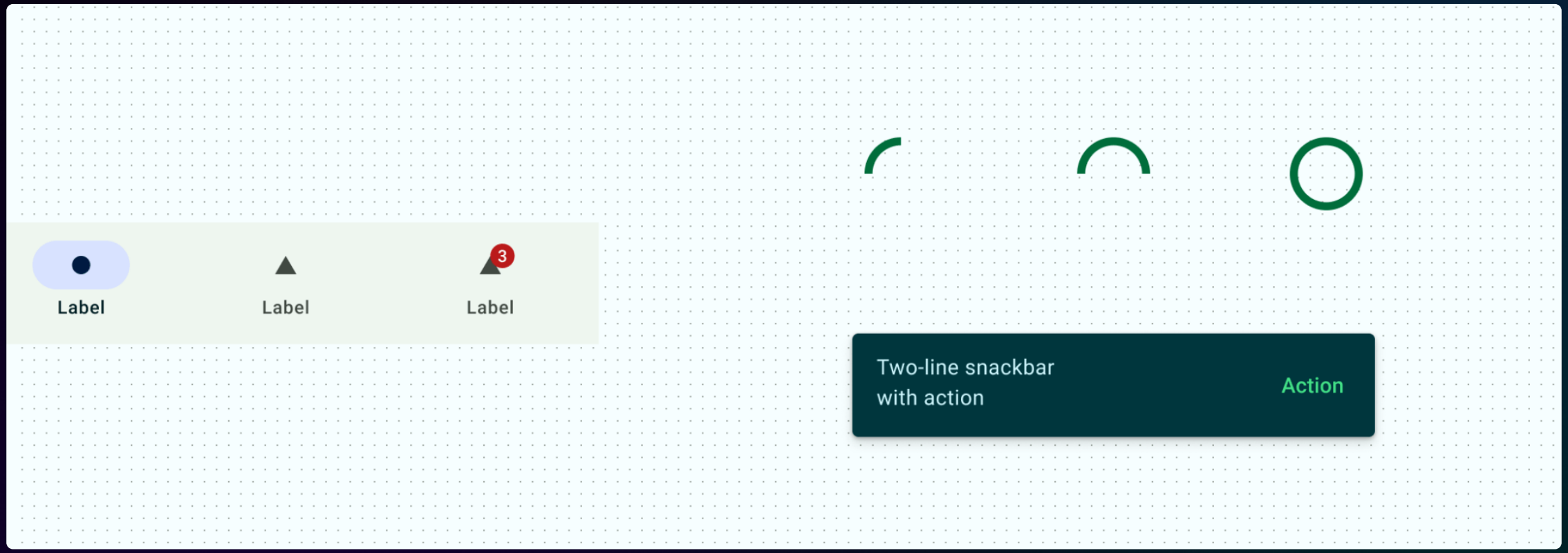
Et bien plus encore avec Compose…

- **Composants d’action :** Buttons, FAB, Icon Buttons.



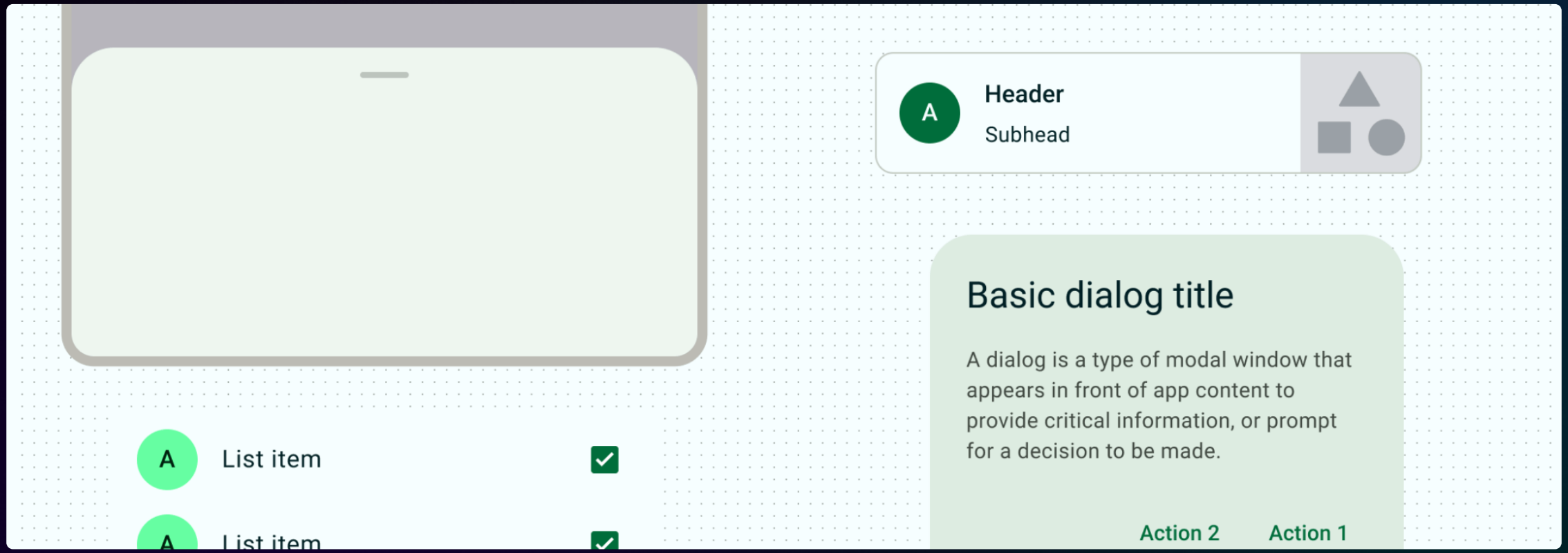
Et bien plus encore avec Compose...

- **Composants de communication :** Badges, Progress Indicators, Snackbars.



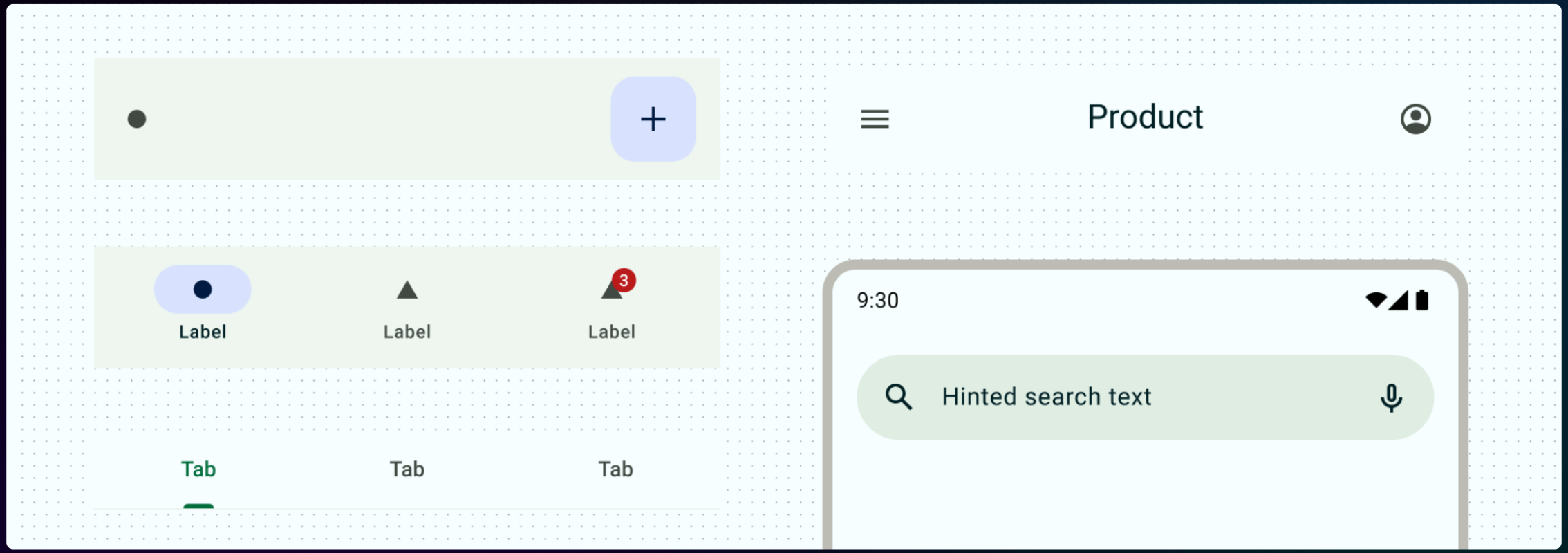
Et bien plus encore avec Compose...

- **Composants de conteneurs :** Cards, Dialogs, Side Sheets, Carousels, Lists...



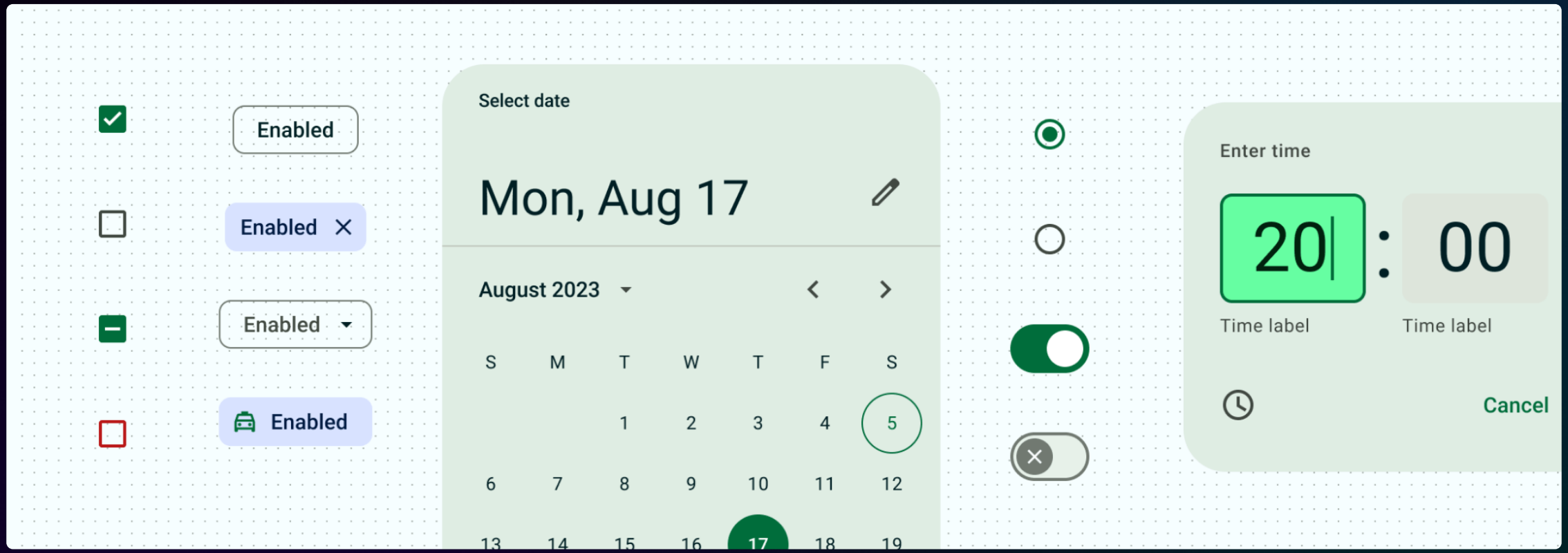
Et bien plus encore avec Compose...

- **Composants de navigation :** Navigation Bar, Navigation Drawer, Tabs, Top app bar.



Et bien plus encore avec Compose...

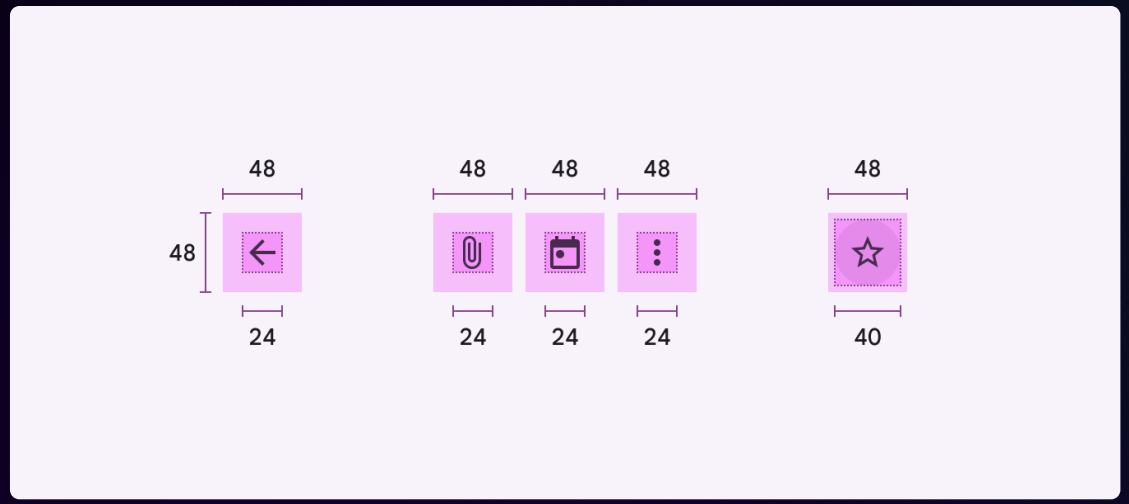
- **Composants de sélection :** Checkboxes, Radio Buttons, Chips, Switches.

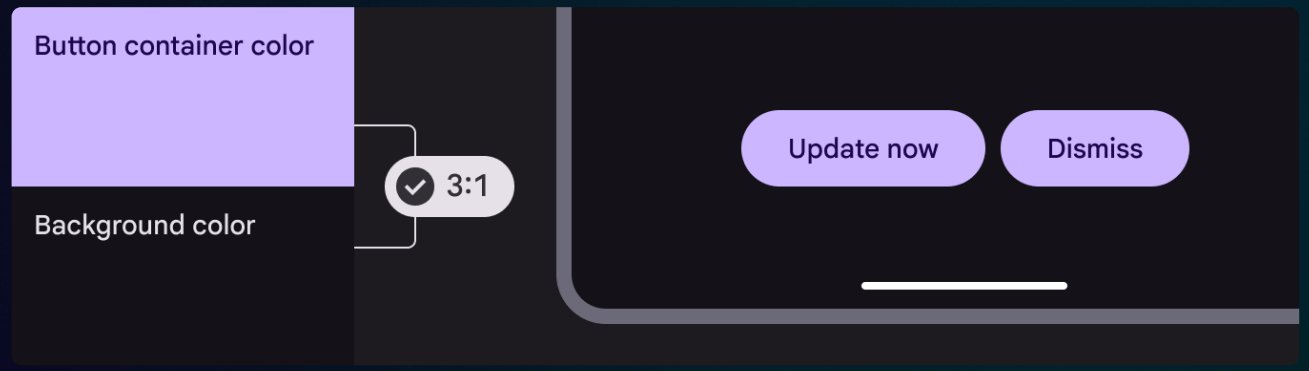


Accessibilité avec Material Design

Les composants Material Design sont accessibles par défaut :

- **Tailles minimales** adaptées pour une interaction tactile confortable (48dp x 48dp).
- **Contraste des couleurs** conforme pour une meilleure lisibilité.



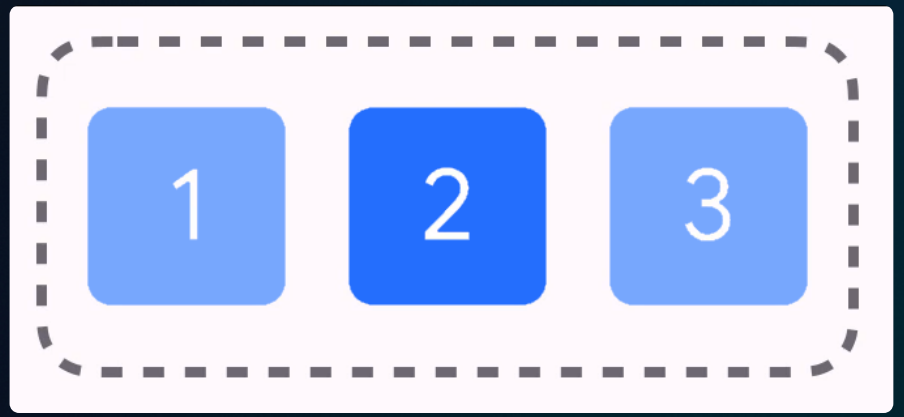


Organiser votre UI dans Compose grâce aux principaux layouts

Row : Disposer les éléments horizontalement

- Utilisé pour organiser les éléments de façon **horizontale**.
- Les éléments s'affichent de gauche à droite (ou droite à gauche selon la langue).

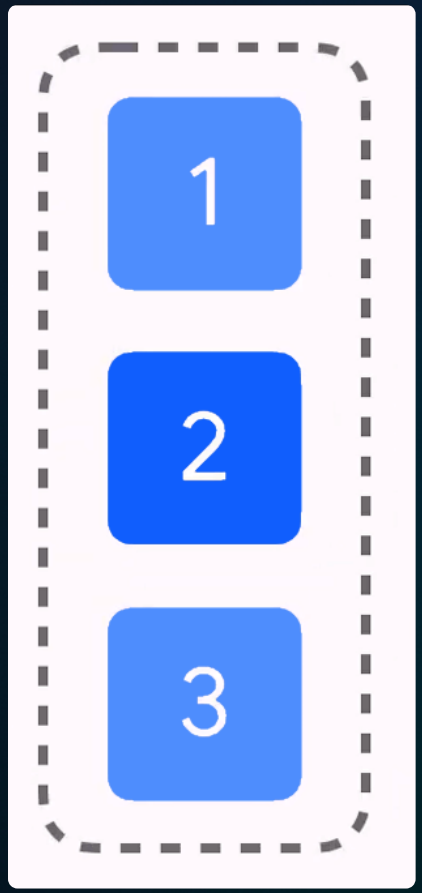
```
1 Row {
2     Component1()
3     Component2()
4     Component3()
5 }
```



Column : Disposer les éléments verticalement

- Utilisé pour organiser les éléments de façon **verticale**.
- Les éléments s'affichent de haut en bas.

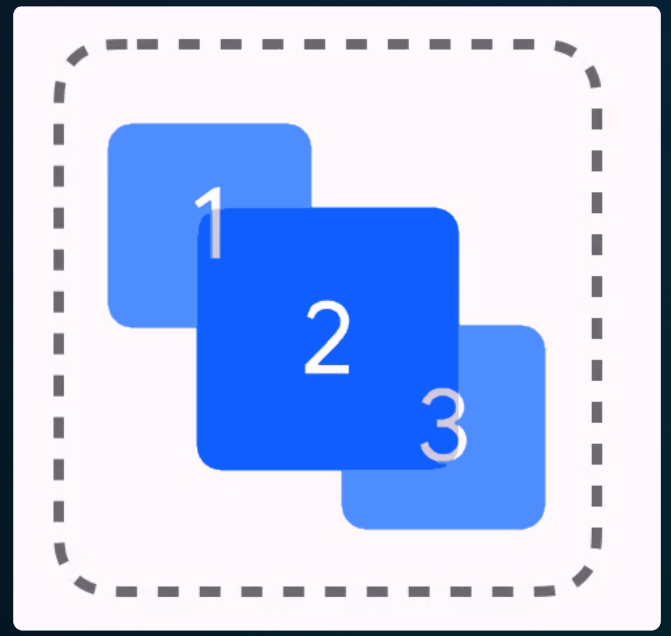
```
1 Column {
2     Component1()
3     Component2()
4     Component3()
5 }
```



Box : Superposer les éléments

- Permet de positionner des éléments les uns sur les autres.
- Idéal pour les arrière-plans, icônes superposées ou vues chevauchées.

```
1 Box {
2     Component1()
3     Component2()
4     Component3()
5 }
```



L'alignement et l'arrangement

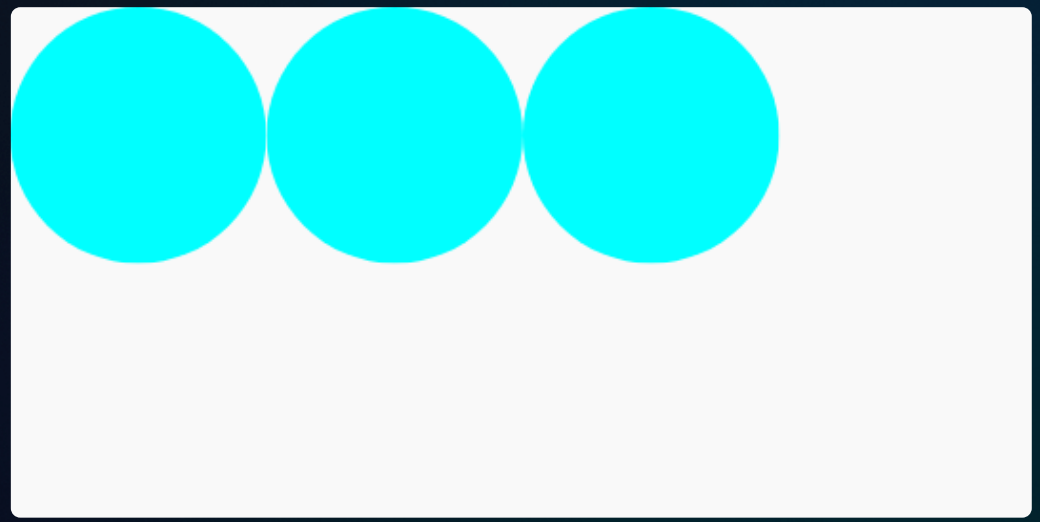
- Les Composables **Row** et **Column** offrent deux paramètres clés pour gérer la position des éléments enfants :
  - **alignment** : contrôle l'alignement des éléments sur l'axe **vertical** pour **Row** et sur l'axe **horizontal** pour **Column**.
  - **arrangement** : définit l'espacement entre les éléments sur l'axe principal.

```
1  @Composable
2  inline fun Row(
3      modifier: Modifier = Modifier,
4      horizontalArrangement: Arrangement.Horizontal = Arrangement.Start,
5      verticalAlignment: Alignment.Vertical = Alignment.Top,
6      content: @Composable RowScope.() -> Unit
7  ) {
8      val measurePolicy = rowMeasurePolicy(horizontalArrangement, verticalAlignment)
9      Layout(
10         content = { RowScopeInstance.content() },
11         measurePolicy = measurePolicy,
12         modifier = modifier
13     )
14 }
```

L'alignement

- **Alignement.Top** : Place les composables enfants en haut de la **Row**.

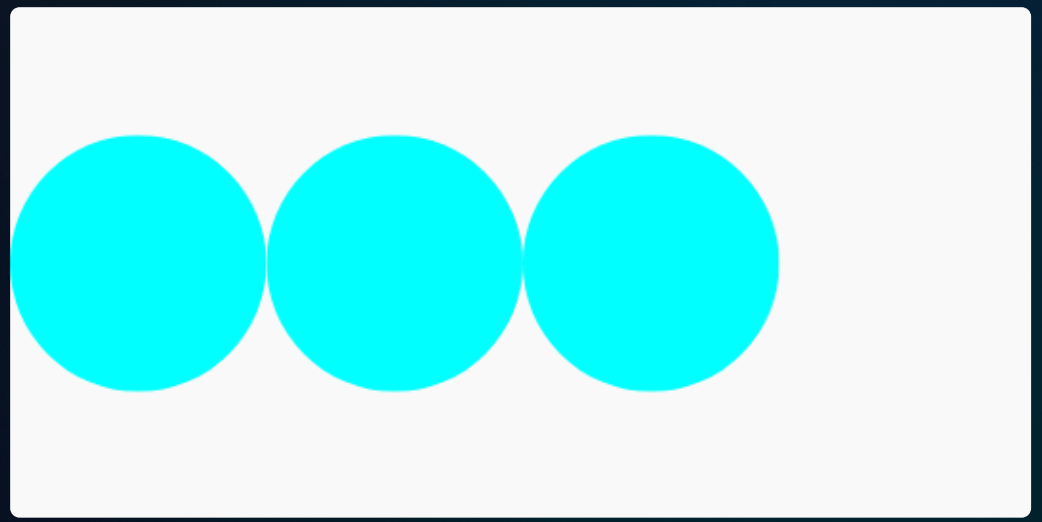
```
1 Row(
2     verticalAlignment = Alignment.Top
3 ) {
4     Component1()
5     Component2()
6     Component3()
7 }
```



L'alignement

- **Alignement.CenterVertically** : Centre les composables enfants verticalement dans la **Row**.

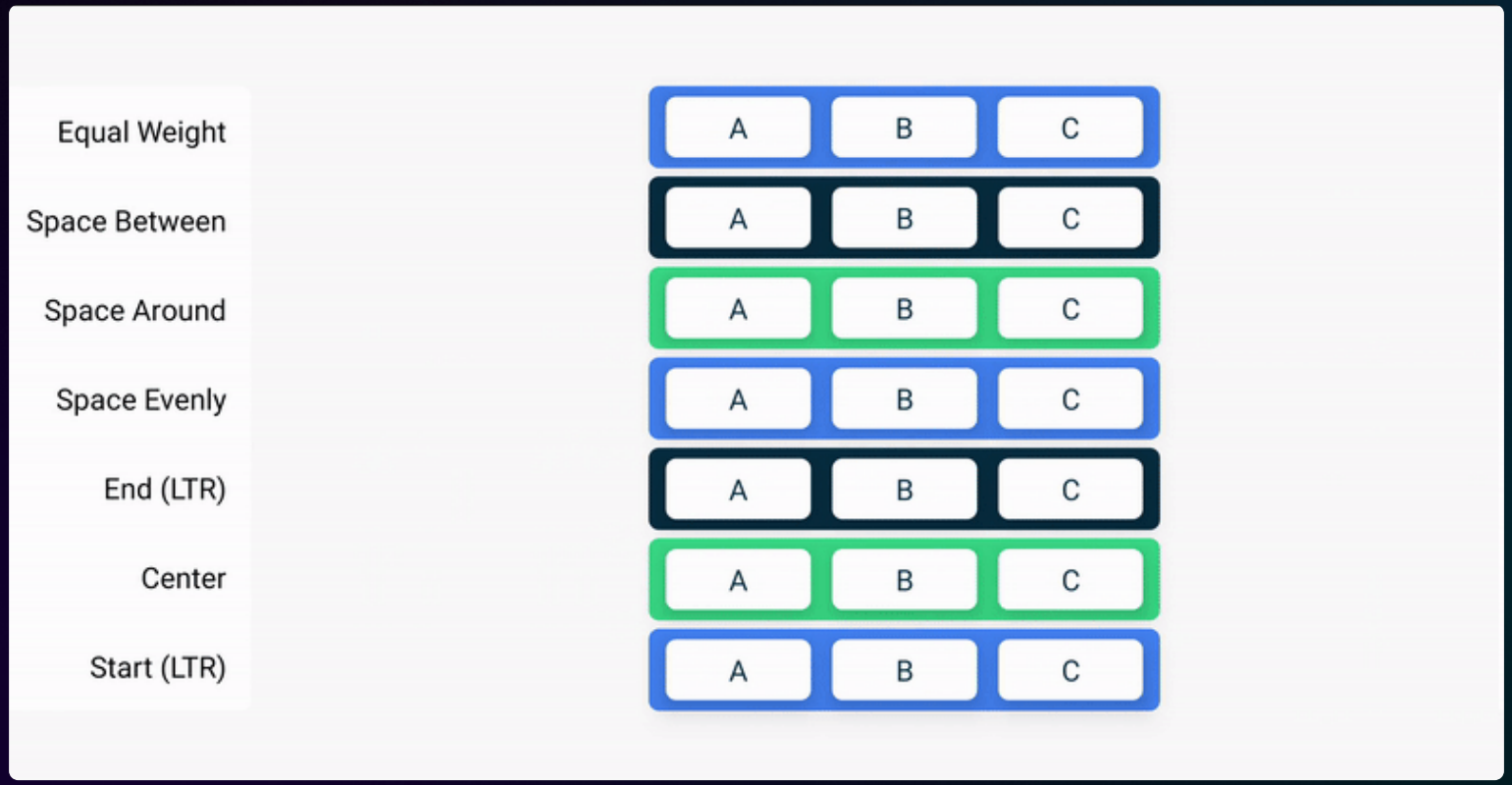
```
1 Row(
2     verticalAlignment = Alignment.CenterVertically
3 ) {
4     Component1()
5     Component2()
6     Component3()
7 }
```



L'arrangement

L'arrangement définit l'espacement entre les éléments d'une **Row** ou d'une **Column**.

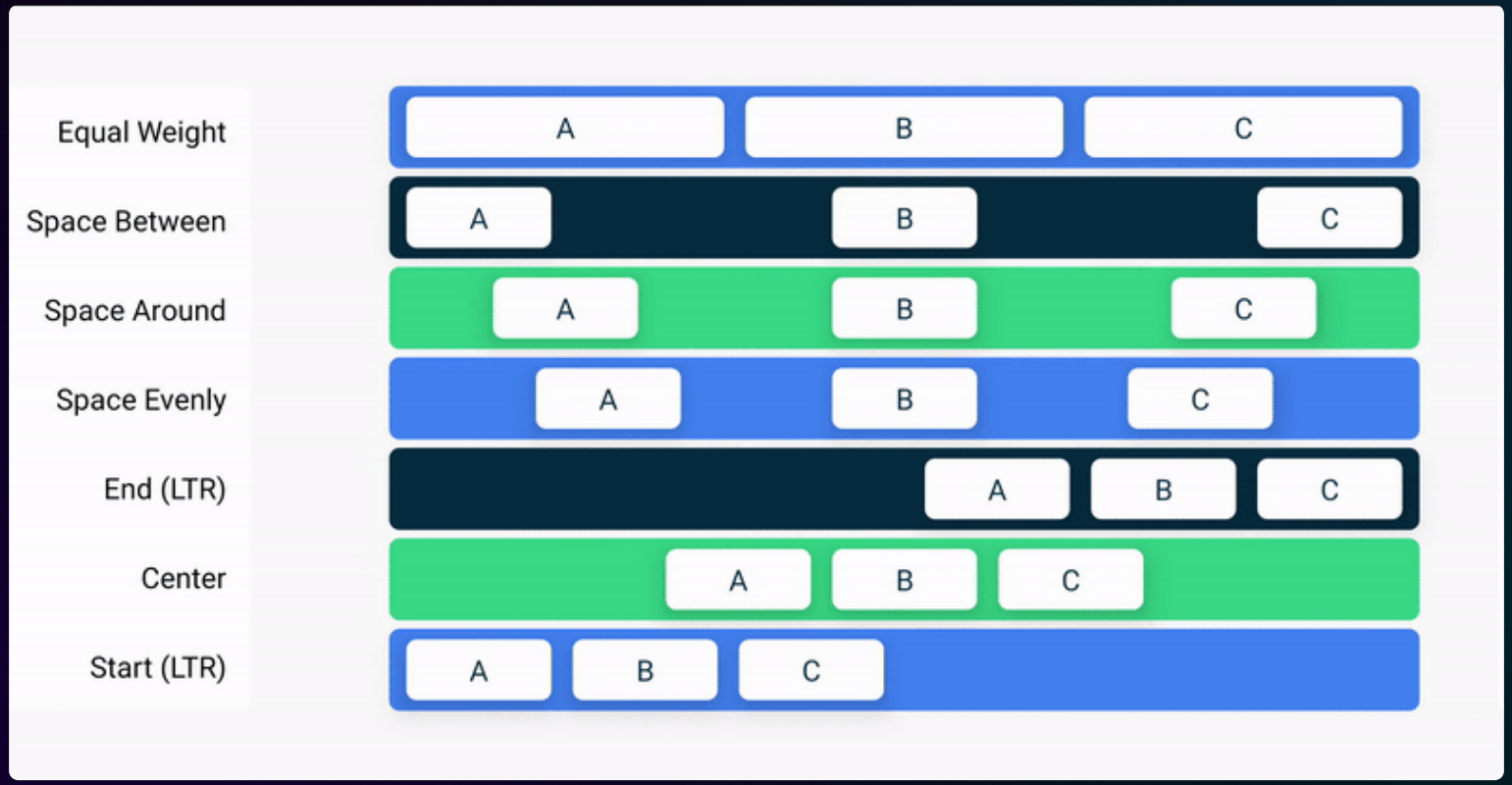
Note : Il n'a aucun effet si la taille de la **Row** ou **Column** est en **wrap** ou plus petite que celle des éléments enfants.



L'arrangement

L'arrangement définit l'espacement entre les éléments d'une **Row** ou d'une **Column**.

Note : Il n'a aucun effet si la taille de la **Row** ou **Column** est en **wrap** ou plus petite que celle des éléments enfants.

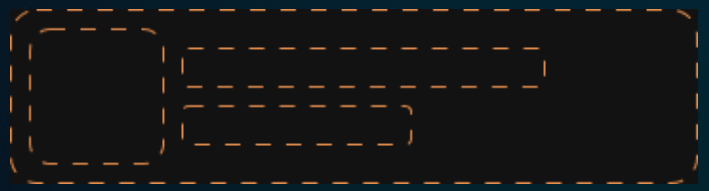


Les layouts : Sous le capot

- Les layouts sont des composables qui déterminent la disposition et l'alignement des éléments enfants.

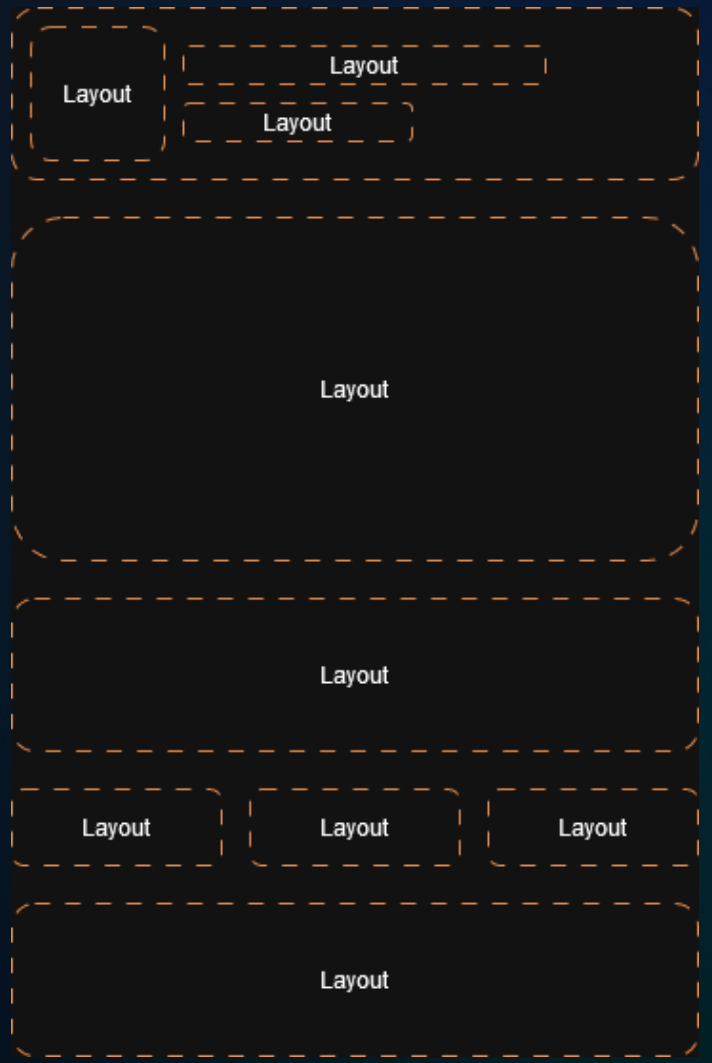
- On peut les voir comme des coordinateurs définissant la structure des autres composables.

```
Layout {
    Component1()
    Component2()
    Component3()
}
```



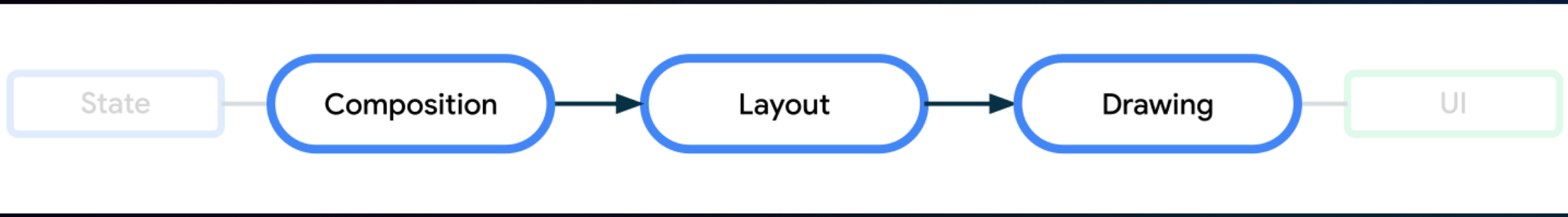
Les layouts : Sous le capot

Presque tous les composables dans **Compose** utilisent un composable **Layout()** sous-jacent pour gérer leur disposition et leur alignement.



Mais alors, comment Compose dessine un élément ? 🤔

Les trois phases de Compose



1. **Composition** : *quels* éléments d'interface utilisateur afficher. Compose exécute des fonctions et crée une description de l'UI.

2. **Layout** : *où* positionner les éléments d'interface utilisateur. Cette phase comprend :
    1. **Mesure** : Les éléments sont mesurés selon les contraintes.
    2. **Positionnement** : Les éléments et leurs enfants sont placés en coordonnées 2D.

3. **Drawing** : *comment* effectuer le rendu de l'interface utilisateur. Les éléments de l'UI apparaissent dans un canevas, généralement l'écran d'un d'appareil.

Récapitulations

- Thèmes
- Composants
- Layouts

Et maintenant ? Personnalisons la taille, les marges, les couleurs et les autres aspects visuels...

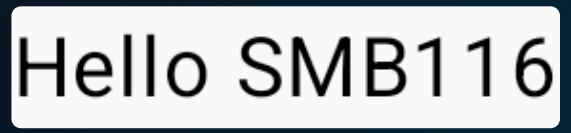
Modifier

- Permet de **décorer** ou personnaliser un composable.
- Tous les composables du kit Compose acceptent un `Modifier` en paramètre.
- Les modifiers peuvent être **enchaînés** pour obtenir la personnalisation souhaitée.

Modifier

- Les modifiers permettent de faire toutes sortes de choses

```
1 Text("Hello SMB116")
```



Modifier

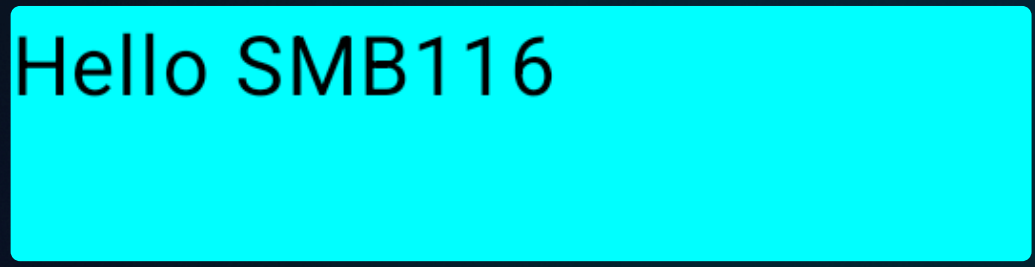
- Les modifiers permettent de faire toutes sortes de choses

```
1 Text(
2   "Hello SMB116",
3   Modifier
4     .background(Color.Cyan)
5 )
```

Modifier

- Les modifiers permettent de faire toutes sortes de choses

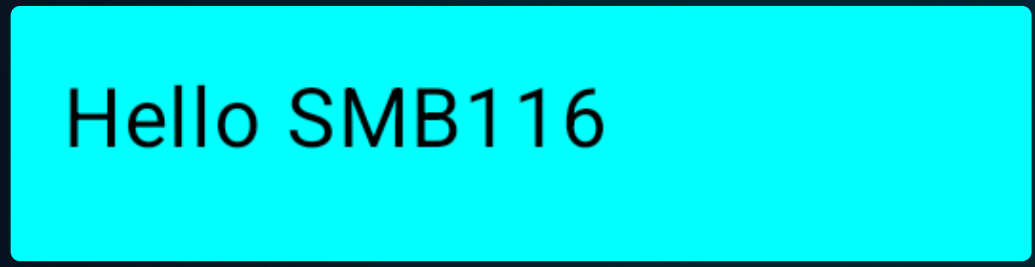
```
1 Text(
2   "Hello SMB116",
3   Modifier
4     .background(Color.Cyan)
5     .size(200.dp, 50.dp)
6 )
```



Modifier

- Les modifiers permettent de faire toutes sortes de choses

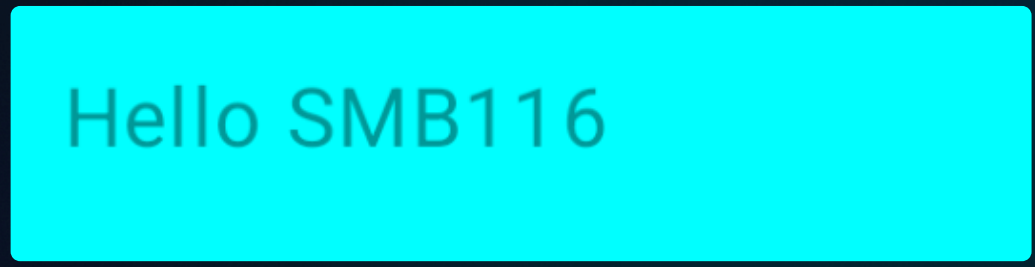
```
1 Text(
2   "Hello SMB116",
3   Modifier
4     .background(Color.Cyan)
5     .size(200.dp, 50.dp)
6     .padding(10.dp)
7 )
```



Modifier

- Les modifiers permettent de faire toutes sortes de choses

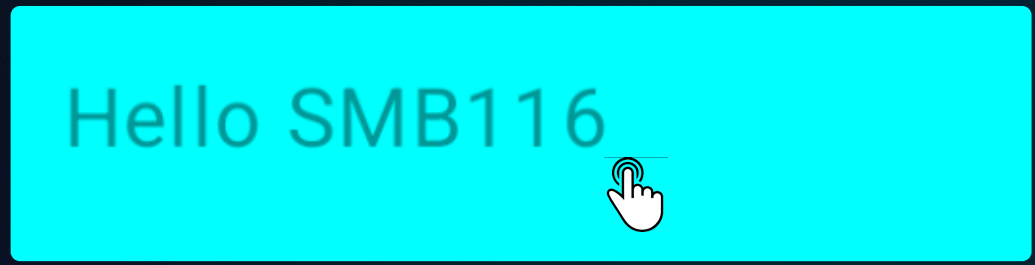
```
1 Text(
2   "Hello SMB116",
3   Modifier
4     .background(Color.Cyan)
5     .size(200.dp, 50.dp)
6     .padding(10.dp)
7     .alpha(0.4f)
8 )
```



Modifier

- Les modifiers permettent de faire toutes sortes de choses

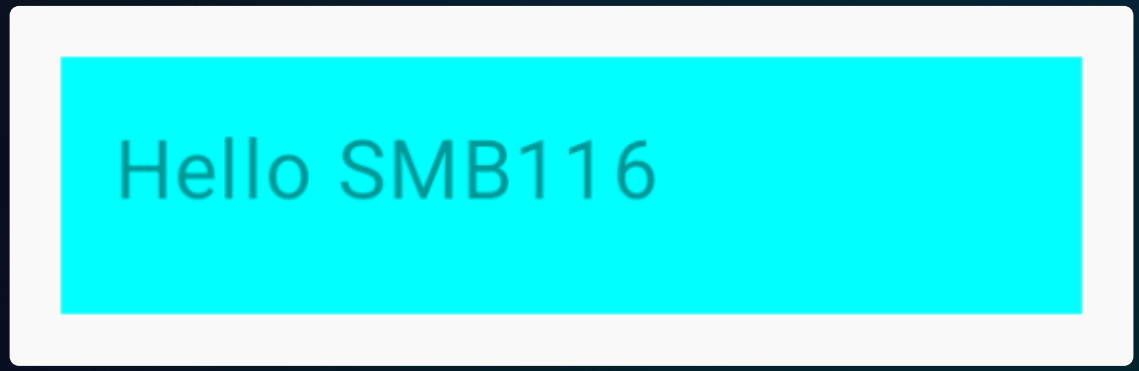
```
1  Text(
2    "Hello SMB116",
3    Modifier
4      .background(Color.Cyan)
5      .size(200.dp, 50.dp)
6      .padding(10.dp)
7      .alpha(0.4f)
8      .clickable {
9          // Action quand le texte est cliqué
10     }
11 )
```



Modifier

- L'ordre des modifiers est important

```
Text(
  "Hello SMB116",
  Modifier
    .padding(10.dp)
    .background(Color.Cyan)
    .size(200.dp, 50.dp)
    .padding(10.dp)
    .alpha(0.4f)
    .clickable {
        // Action quand le texte est cliqué
    }
)
```



Modifier

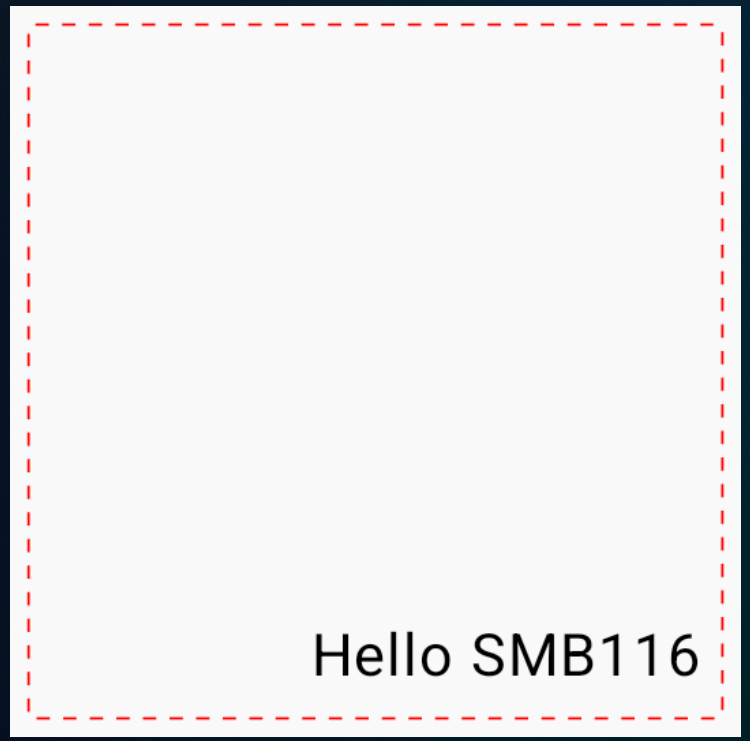
- Certains modifiers supplémentaires sont disponibles en fonction du scope de la fonction composable où vous travaillez.

```
1  Box(
2    modifier = Modifier
3      .size(200.dp)
4      .padding(5.dp)
5  ) {
6    Text(
7      text = "Hello SMB116",
8      modifier = Modifier.padding(5.dp)
9    )
10 }
```

Modifier

- Certains modifiers supplémentaires sont disponibles en fonction du scope de la fonction composable où vous travaillez.

```
1  Box(
2    modifier = Modifier
3      .size(200.dp)
4      .padding(5.dp)
5  ) {
6    Text(
7      text = "Hello SMB116",
8      modifier = Modifier.padding(5.dp)
9        .align(Alignment.BottomEnd)
10   )
11 }
```



Conclusion

📱 **Material Design**

🎚️ **Densité de pixels (dp)**

🎨 **Thèmes** : typographies, couleurs, formes

🧩 **Composables du kit de développement**

♿ **Accessibilité**

🛠️ **Layouts** : Row, Column, Box

⚙️ **Les 3 phases de Compose** : Composition, Layout, Drawing

🔧 **Modifiers**