# Bloc 2 – Compose UI

## Introduction théorique courte

### 1. Objectif du bloc : construire des interfaces Compose complètes

Dans le bloc précédent, vous avez appris à écrire des composables propres et réutilisables.

Dans ce bloc, nous allons enrichir ces composables pour construire des **interfaces plus complètes**.

L'objectif est de comprendre les grandes briques de l'interface **Compose** :

- les composants **d'affichage** ;
- les composants **d'action** ;
- les **images** et éléments **visuels** ;
- les **layouts** ;
- les **listes** ;
- le **thème** graphique de l'application.

Une interface Compose n'est pas seulement une addition de *textes* et de *boutons*. C'est un ensemble de composants **organisés**, **réutilisables** et **cohérents visuellement**.

### 2. Composants fondamentaux

Les composants fondamentaux sont les **briques visibles** de l'interface.

Ils permettent **d'afficher** une **information**, de **déclencher** une **action** ou de **structurer** une **zone** importante de l'écran.

> `Text` sert à afficher du texte : **titre**, **description**, prix, message, information secondaire.
>
> `Button` permet à l'utilisateur de **déclencher une action** claire.
>
> `Icon` sert à représenter **visuellement** une action, un état ou une information.
>
> `Image` permet d'afficher une **image locale ou distante**, par exemple une illustration ou une image produit.
>
> `Card` sert à **regrouper des informations** dans un bloc visuel identifiable.
>
> `TextField` permet de saisir ou d'afficher une **zone de texte éditable**.

**Ces composants sont souvent combinés** pour former des éléments plus riches : une fiche, une carte, un résumé, un en-tête ou une section.

### 3. Pourquoi ne pas utiliser uniquement des composants simples ?

Un écran réel contient rarement un seul composant. Il faut souvent **combiner plusieurs éléments** :

- un **titre** ;
- une **image** ;
- une **description** ;
- un **bouton** ;
- des **informations secondaires** ;
- des **indicateurs** visuels.

La difficulté n'est pas seulement de connaître chaque composant séparément.

La vraie compétence consiste à les **assembler proprement** dans des composants **réutilisables**.

Par exemple, une **fiche produit** peut combiner un **titre**, une **image**, un **prix**, une **note** et un **bouton** d'action.

## 4. Layouts Compose

Les **layouts** permettent d'organiser les composants dans l'écran. Ils répondent à une question simple : *Où placer chaque élément ?*

> `Column` organise les éléments **verticalement**.
>
> `Row` organise les éléments **horizontalement**.
>
> `Box` permet de **superposer** ou de **positionner** des éléments **dans une même zone**.
>
> `Spacer` ajoute un **espace contrôlé** entre plusieurs éléments.
>
> `Surface` permet de **créer une zone visuelle** cohérente avec le thème **Material**.

Les **layouts** sont essentiels pour transformer plusieurs composants isolés en un écran clair et lisible.

## 5. Construire une hiérarchie d'interface

Une **interface Compose** est organisée comme **une hiérarchie**.

→ Un **écran** contient des **sections**.
→ Une **section** contient des **composants**.
→ Un **composant** peut lui-même contenir **d'autres composants** plus petits.

Cette organisation permet de garder le code lisible.

Au lieu d'écrire un **grand composable difficile** à comprendre, on **découpe l'interface** en blocs plus simples :

- un **en-tête** ;
- une zone de **contenu** ;
- une **carte** ;
- une **ligne d'information** ;
- un **bouton** d'action.

Cette logique facilite la réutilisation, les **previews** et les futures évolutions de l'interface – leur scalabilité.

## 6. Listes modernes

Beaucoup d'applications affichent des collections d'éléments, par exemple :

- produits ;
- messages ;
- contacts ;
- articles ;
- compétences ;
- projets ;
- résultats de recherche.

Compose propose des composants adaptés à ces affichages.

→ `LazyColumn` permet d'afficher une **liste verticale**.
→ `LazyRow` permet d'afficher une **liste horizontale**.

Le mot **« Lazy »** - "*à la demande*" – signifie que **Compose** compose les éléments de manière **efficace**, selon ce qui doit être affiché à l'écran.

**Compose ne prépare pas toute la liste d'un coup.** Il prépare seulement les éléments *visibles à l'écran*, et quelques éléments autour.

**Par exemple :** Si la liste contient **1000 produits**, mais que l'écran n'en montre que **8**, `LazyColumn` ne compose pas les 1000 directement. Elle compose surtout les 8 visibles. Quand on scrolle, elle compose les suivants **au fur et à mesure**.

Les listes modernes ne servent pas seulement à répéter du contenu. Elles permettent aussi de **réutiliser des cartes**, de **séparer les données** de l'affichage et de **structurer des écrans** plus riches.

## 7. Cartes et éléments réutilisables dans une liste

Dans une liste, **chaque élément** doit généralement être affiché avec le **même modèle** visuel.

Par exemple :
- une carte produit ;
- une carte de compétence ;
- une carte de projet ;
- une carte de contact.

L'idée est de **créer un composant réutilisable** qui reçoit des données en **paramètres**.

La liste se contente ensuite d'appeler ce **composant** pour **chaque élément**.

Cela permet **d'éviter la duplication de code** et de garder une interface **cohérente**.

## 8. Material Design 3

**Material Design 3** est le système de design utilisé dans les applications Android modernes. C'est le système de **design d'Android :** il donne des **règles communes** pour les composants.

L'objectif est d'avoir des applications Android **plus cohérentes entre elles**, comme **Cupertino** joue ce rôle dans l'univers iOS.

Il aide à construire une interface cohérente grâce à plusieurs éléments :
- les **couleurs** ;
- la **typographie** ;
- les **formes** ;
- les **composants** ;
- le **thème clair** et **sombre**.

Avec **Material 3**, l'objectif n'est **pas seulement** de rendre l'application plus **jolie**. L'objectif est aussi de **rendre l'interface plus cohérente**, plus lisible et plus facile à maintenir.

## 9. Thème, cohérence et identité visuelle

Un **thème** permet d'appliquer des choix graphiques à **toute l'application**. Au lieu de **modifier chaque texte**, chaque **bouton** ou chaque **carte séparément**, on définit **une base commune**.

Le thème peut contrôler :

- les **couleurs principales** ;
- les couleurs de **fond** ;
- les **styles de texte** ;
- les **formes** des composants ;
- l'apparence en **mode clair** ;
- l'apparence en **mode sombre**.

Cela permet d'obtenir une **application plus homogène**.

Une interface **cohérente** donne aussi une impression plus **professionnelle**.

## 10. Ce que ce bloc va vous permettre de faire

À la fin de ce bloc, vous serez capables de construire une interface Compose plus complète.

Vous saurez :

- **choisir** les bons **composants** fondamentaux ;
- **organiser** un écran avec les **layouts** ;
- **afficher** des **collections** avec des **listes modernes** ;
- **créer** des **cartes** réutilisables ;
- **appliquer** une identité visuelle avec **Material 3** ;
- **garder** une interface lisible, structurée et **cohérente**.

Ces notions seront manipulées progressivement dans les TP du bloc.

L'objectif est de passer de petits composants isolés à une interface complète et réutilisable.