



TP 6 – Affichage d'un catalogue local de produits

Listes modernes

1. Objectif du TP

Dans le TP précédent, vous avez construit un écran d'accueil pour **Product Explorer**.

Dans ce TP, vous allez faire évoluer cette application pour afficher un **catalogue local de produits**.

Vous allez utiliser :

- **LazyColumn** ;
- **LazyRow** ;
- une liste locale de produits ;
- des cartes réutilisables.

À la fin du TP, l'application affichera :

- un **titre** de catalogue ;
- une **ligne horizontale** de catégories ;
- une **liste verticale** de produits ;
- une carte réutilisable pour chaque produit.

2. Point de départ

Vous repartez du code du TP5 ou vous avez déjà :

- **ProductUi** ;
- **sampleProduct()** ;
- **ProductHomeScreen** ;
- plusieurs composants d'interface ;
- le thème **ProductExplorerTheme**.

Dans ce TP, nous allons ajouter un nouvel écran : **ProductCatalogScreen** qui affichera plusieurs produits locaux.

3. Étape 1 – Ajouter un identifiant au produit

Pour afficher une liste proprement, chaque produit doit pouvoir être identifié.

Nous allons donc ajouter un champ **id** dans **ProductUi**. Modifiez votre classe **ProductUi** :

```
data class ProductUi(  
    val id: Int,  
    val title: String,  
    val brand: String,  
    val category: String,  
    val description: String,  
    val price: Double,  
    val discountPercentage: Double,  
    val rating: Double,  
    val stock: Int,  
    val warrantyInformation: String,  
    val shippingInformation: String  
)
```

Comme vous avez ajouté **id**, vous devez aussi modifier vos produits d'exemple en ajoutant ce champs, par exemple :



```
fun sampleProduct(): ProductUi {  
    return ProductUi(  
        id = 1,  
        // le reste du code ne change pas  
    )  
}
```

Faites la même chose pour les autres produits déjà créés.

4. Étape 2 – Créer une liste locale de produits

Pour afficher un catalogue, nous avons besoin de plusieurs produits.

Nous allons créer une fonction qui retourne une liste :

```
fun sampleProducts(): List<ProductUi> {  
    return listOf(  
        sampleProduct(),  
        ProductUi(  
            id = 2,  
            title = "Casque Audio Pulse",  
            brand = "SoundPeak",  
            category = "Audio",  
            description = "Un casque confortable pour écouter de la musique et travailler dans de bonnes  
conditions.",  
            price = 129.99,  
            discountPercentage = 8.0,  
            rating = 4.2,  
            stock = 0,  
            warrantyInformation = "Garantie constructeur : 1 an",  
            shippingInformation = "Produit temporairement indisponible"  
        ),  
        ProductUi(  
            id = 3,  
            title = "Montre Connectée FitTime",  
            brand = "FitTime",  
            category = "Wearables",  
            description = "Une montre connectée simple pour suivre l'activité quotidienne.",  
            price = 89.99,  
            discountPercentage = 15.0,  
            rating = 4.4,  
            stock = 18,  
            warrantyInformation = "Garantie constructeur : 2 ans",  
            shippingInformation = "Livraison estimée : 2 à 4 jours ouvrés"  
        ),  
        ProductUi(  
            id = 4,  
            title = "Enceinte Mini Boom",  
            brand = "BoomSound",  
            category = "Audio",  
            description = "Une enceinte compacte pour écouter de la musique à la maison ou en déplacement.",  
            price = 59.99,  
            discountPercentage = 5.0,  
            rating = 4.1,  
            stock = 52,  
            warrantyInformation = "Garantie constructeur : 1 an",  
            shippingInformation = "Livraison estimée : 3 jours ouvrés"  
        )  
    )  
}
```

Cette fonction simule une source de données. Pour le moment, la liste est locale.

L'objectif est uniquement d'apprendre à afficher plusieurs éléments.

5. Étape 3 – Afficher le catalogue depuis *MainActivity*

Nous allons remplacer l'écran d'accueil par l'écran catalogue.

L'écran d'accueil du TP5 reste dans le code, mais l'application affichera maintenant le catalogue.

Dans **MainActivity**, remplacez l'appel à **ProductHomeScreen** par :



```
Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
    ProductCatalogScreen(
        products = sampleProducts(),
        onProductClick = {
            // Plus tard : ouvrir le détail du produit
        },
        modifier = Modifier.padding(innerPadding)
    )
}
```

ProductCatalogScreen reçoit une liste de produits. Le clic sur un produit est prévu, mais il ne fait pas encore de navigation.

6. Étape 4 – Créer la structure avec **LazyColumn**

Un catalogue peut contenir beaucoup de produits. Pour une liste **verticale**, on utilise **LazyColumn**.

Contrairement à une **Column**, **LazyColumn** est pensée pour afficher une collection d'éléments.

Créez ce composable :

```
@Composable
fun ProductCatalogScreen(
    products: List<ProductUi>,
    onProductClick: (ProductUi) -> Unit,
    modifier: Modifier = Modifier
) {
    LazyColumn(
        modifier = modifier.fillMaxSize(),
        contentPadding = PaddingValues(16.dp),
        // Ajoute une marge intérieure autour du contenu de la liste
        verticalArrangement = Arrangement.spacedBy(16.dp)
        // Ajoute un espace vertical entre les éléments de la liste.
    ) {
        item {
            Text(
                text = "Catalogue produits",
                style = MaterialTheme.typography.headlineMedium
            )
        }
    }
}
```

LazyColumn contient des blocs.

Ici, **item** ajoute un seul élément : le **titre** du catalogue.

Ensuite, nous ajouterons les produits avec **items**.

7. Étape 5 – Créer une carte produit réutilisable

Chaque produit de la liste doit être affiché de la même façon.

Nous allons créer alors une carte réutilisable.

Elle recevra :

- un **produit**,
- une **action de clic**
- et un **modifier**.

Ajoutez ce composable :



```
@Composable
fun ProductListItem(
    product: ProductUi,
    onClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Card(modifier = modifier.fillMaxWidth()) {
        Column(
            modifier = Modifier.padding(16.dp)
        ) {
            Text(
                text = product.title,
                style = MaterialTheme.typography.titleMedium
            )
            Text(
                text = product.brand,
                style = MaterialTheme.typography.bodyMedium
            )
            Spacer(modifier = Modifier.height(8.dp))
            Text(
                text = "${product.price} € • ★ ${product.rating}",
                style = MaterialTheme.typography.bodyMedium
            )
            Spacer(modifier = Modifier.height(8.dp))
            Button(onClick = onClick) {
                Text(text = "Voir le produit")
            }
        }
    }
}
```

Cette carte ne connaît pas toute la liste. Elle affiche **un seul produit**.

C'est **ProductCatalogScreen** qui va l'appeler pour chaque élément.

8. Étape 6 – Afficher la liste de produits avec *items*

Maintenant que nous avons **une carte produit**, nous pouvons afficher toute la liste.

Dans une **LazyColumn**, **items** permet de parcourir une collection.

Modifiez **ProductCatalogScreen** :

```
@Composable
fun ProductCatalogScreen(
    products: List<ProductUi>,
    onProductClick: (ProductUi) -> Unit,
    modifier: Modifier = Modifier
) {
    LazyColumn(
        modifier = modifier.fillMaxSize(),
        contentPadding = PaddingValues(16.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        item {
            Text(
                text = "Catalogue produits",
                style = MaterialTheme.typography.headlineMedium
            )
        }

        items(
            items = products,
            key = { product -> product.id }
        ) { product ->
            ProductListItem(
                product = product,
                onClick = {
                    onProductClick(product)
                }
            )
        }
    }
}
```



Si Android Studio affiche l'erreur « **No value passed for parameter 'count'** » sur `items`, ajoutez l'import suivant : `import androidx.compose.foundation.lazy.items`.

- `items` reçoit la liste `products`.
- Pour chaque produit, Compose appelle `ProductListItem`.
- Le paramètre `key` donne une identité stable à chaque élément.

9. Étape 7 – Ajouter une liste horizontale de catégories avec `LazyRow`

Un catalogue peut aussi afficher des catégories.

Pour une **liste horizontale**, on utilise `LazyRow`.

Nous allons afficher les **catégories** présentes dans nos produits.

Ajoutez cette fonction simple :

```
fun sampleCategories(): List<String> {  
    return listOf("Smartphones", "Audio", "Wearables", "Maison")  
}
```

Puis créez ce composable :

```
@Composable  
fun CategoryRow(  
    categories: List<String>,  
    modifier: Modifier = Modifier  
) {  
    LazyRow(  
        modifier = modifier,  
        horizontalArrangement = Arrangement.spacedBy(8.dp)  
    ) {  
        items(categories) { category ->  
            CategoryChip(label = category)  
        }  
    }  
}
```

`LazyRow` fonctionne comme `LazyColumn`, mais horizontalement.

Ici, chaque catégorie est affichée avec le composant `CategoryChip` créé dans le TP5.

10. Étape 8 – Ajouter les catégories dans le catalogue

Nous allons placer la ligne de catégories juste sous le titre.

Cela donne à l'écran une structure plus proche d'un vrai catalogue.

```
@Composable  
fun CategoryRow(  
    categories: List<String>,  
    modifier: Modifier = Modifier  
) {  
    LazyRow(  
        modifier = modifier,  
        horizontalArrangement = Arrangement.spacedBy(8.dp)  
    ) {  
        items(categories) { category ->  
            CategoryChip(label = category)  
        }  
    }  
}
```

Ajoutez un `item` avec `CategoryRow` dans `ProductCatalogScreen` après l'`item` de titre :



```
item {
    CategoryRow(categories = sampleCategories())
}
```

Le reste de la liste de produits reste en dessous.

Vous avez compris qu'une **LazyColumn** peut contenir :

- des éléments uniques avec **item** ;
- des collections avec **items**.

Cela permet de construire un écran complet dans une seule **liste verticale**.

11. Étape 9 – Ajouter une petite description du catalogue

Ajoutons une courte phrase pour expliquer ce que l'écran affiche. Ce n'est pas obligatoire techniquement, mais cela rend l'interface plus claire.

Entre le **titre** et les **catégories**, ajoutez :

```
item {
    Text(
        text = "Explorez une sélection de produits disponibles localement.",
        style = MaterialTheme.typography.bodyMedium
    )
}
```

La **LazyColumn** peut donc contenir plusieurs éléments avant la liste. Elle ne sert pas uniquement à répéter des cartes.

12. Étape 10 – Ajouter une *preview*

Ajoutons une **preview** pour vérifier rapidement le catalogue.

```
@Preview(showBackground = true)
@Composable
fun ProductCatalogScreenPreview() {
    ProductExplorerTheme {
        ProductCatalogScreen(
            products = sampleProducts(),
            onProductClick = {}
        )
    }
}
```

La **preview** permet de voir :

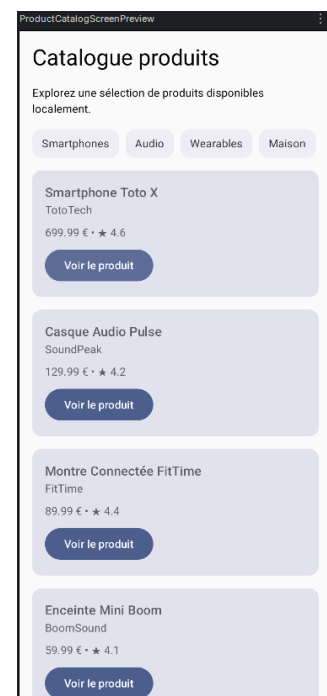
- le **titre** ;
- les **catégories** ;
- la **liste verticale** ;
- les **cartes produit**.

13. Nouveaux imports utiles

Ajoutez ces imports si Android Studio ne les ajoute pas automatiquement :

```
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.LazyRow
import androidx.compose.foundation.lazy.items
```

Gardez aussi les imports déjà utilisés dans les TP précédents.





14. Version finale de ProductCatalogScreen

Voici la version finale de l'écran catalogue (**MainActivity.kt**) :

<https://gist.github.com/ariatsi/6948f2192e30229a4b76734fc6112ed3> .

Adaptez comme d'habitude le nom du package et le nom du thème à votre projet.

Et voici le répertoire GitHub avec cette version :

<https://github.com/ariatsi/ProductExplorer/releases/tag/tp6-final> .

15. Ce qu'il faut retenir

Dans ce TP, vous avez transformé **Product Explorer** en catalogue local.

Vous avez utilisé :

- **LazyColumn** pour afficher une **liste verticale** ;
- **LazyRow** pour afficher une **liste horizontale** ;
- **items** pour parcourir une **collection** ;
- des données locales Kotlin ;
- une **carte** produit réutilisable.

Vous avez aussi continué à appliquer les bonnes pratiques vues précédemment :

- composants courts ;
- paramètres clairs ;
- **modifier: Modifier = Modifier** ;
- **callbacks** pour les actions ;
- **previews**.

Dans le prochain TP, vous personnaliserez l'apparence de **Product Explorer** avec **Material Design 3**.