



TP 7 – Personnalisation graphique de Product Explorer

Material Design 3 et thème

1. Objectif du TP

Dans le TP précédent, vous avez affiché un catalogue local de produits avec `LazyColumn`, `LazyRow` et des cartes réutilisables.

Dans ce TP, vous allez améliorer l'apparence de **Product Explorer** avec **Material Design 3**.

Vous allez travailler :

- `ColorScheme` ;
- `Typography` ;
- `Shapes` ;
- **Material You** ;
- thème **clair** ;
- thème **sombre**.

L'objectif n'est pas de tout redessiner composant par composant mais de comprendre comment le thème permet d'harmoniser toute l'application.

2. Point de départ

Vous repartez du code du TP6 et vous avez normalement :

- `ProductUi` ;
- `sampleProducts()` ;
- `ProductCatalogScreen` ;
- `ProductListItem` ;
- `CategoryRow` ;
- `CategoryChip` ;
- le **thème généré** par Android Studio.

Le thème se trouve généralement dans un fichier proche de : **ui.theme/Theme.kt**

Le nom exact dépend du projet.

3. Étape 1 – Observer le thème existant

Android Studio génère déjà un thème Compose qui enveloppe votre application avec `MaterialTheme`.

C'est pour cela que vous pouvez utiliser :

```
MaterialTheme.typography,  
MaterialTheme.colorScheme,  
MaterialTheme.shapes.
```

Avant de modifier les composants, il faut comprendre que beaucoup de décisions visuelles viennent déjà du thème.

Commencer par ouvrir le fichier **Theme.kt**.

Vous devriez voir une fonction proche de :



```
@Composable
fun ProductExplorerTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    // Dynamic color is available on Android 12+
    dynamicColor: Boolean = true,
    content: @Composable () -> Unit
) {
    val colorScheme = when {
        dynamicColor && Build.VERSION.SDK_INT >= Build.VERSION_CODES.S -> {
            val context = LocalContext.current
            if (darkTheme) dynamicDarkColorScheme(context) else dynamicLightColorScheme(context)
        }

        darkTheme -> DarkColorScheme
        else -> LightColorScheme
    }

    MaterialTheme(
        colorScheme = colorScheme,
        typography = Typography,
        content = content
    )
}
```

Le code exact peut varier selon la version d'Android Studio.

Le **thème** est le **point d'entrée graphique** de l'application.

Il donne aux composants **Material** leurs **couleurs**, leurs **styles** de texte et leurs **formes**.

4. Étape 2 – Créer deux palettes de couleurs

Un **ColorScheme** définit les couleurs principales de l'application.

Nous allons créer une **palette claire** et une **palette sombre**.

Pour rester simple, nous allons utiliser quelques rôles **Material** importants :

- **primary** ;
- **secondary** ;
- **background** ;
- **surface** ;
- **onPrimary** ;
- **onBackground** ;
- **onSurface**.

Les couleurs **on...** servent à afficher du contenu lisible sur la couleur correspondante.

Dans **Theme.kt**, créez ou remplacez les palettes **light** et **dark** par :

```
private val LightColorScheme = lightColorScheme(
    primary = Color(0xFF006C4C),
    onPrimary = Color.White,
    secondary = Color(0xFF4F6357),
    background = Color(0xFFF7F8F6),
    onBackground = Color(0xFF191C1A),
    surface = Color(0xFFFFFFFF),
    onSurface = Color(0xFF191C1A)
)
```

```
private val DarkColorScheme = darkColorScheme(
    primary = Color(0xFF7DDBB0),
    onPrimary = Color(0xFF03826),
    secondary = Color(0xFFB7CCBC),
    background = Color(0xFF101411),
    onBackground = Color(0xFFE0E4DF),
    surface = Color(0xFF1C211D),
    onSurface = Color(0xFFE0E4DF)
)
```



Nous ne choisissons pas simplement des couleurs décoratives – nous définissons des rôles.

Par exemple, **primary** sera utilisé pour des éléments **importants** comme certains boutons ou zones principales.

5. Étape 3 – Utiliser la bonne palette selon le thème

L'application doit pouvoir fonctionner en **thème clair** et en **thème sombre**.

Compose peut savoir si le système est en **dark theme** grâce à `isSystemInDarkTheme()`.

Dans `ProductExplorerTheme`, choisissez la bonne palette en simplifiant à l'occasion le code de base :

```
@Composable
fun ProductExplorerTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    content: @Composable () -> Unit
) {
    val colorScheme = if (darkTheme) {
        DarkColorScheme
    } else {
        LightColorScheme
    }

    MaterialTheme(
        colorScheme = colorScheme,
        typography = Typography,
        content = content
    )
}
```

Le contenu de l'application ne change pas.

C'est le **thème** qui fournit des couleurs différentes selon le **mode clair** ou **sombre**.

6. Étape 4 – Appliquer une couleur de fond à l'écran catalogue

Maintenant que le thème fournit des couleurs, nous allons les utiliser dans l'écran.

Le fond général doit venir du thème.

Dans `ProductCatalogScreen`, modifiez le modifier de `LazyColumn` :

```
LazyColumn(
    modifier = modifier
        .fillMaxSize()
        .background(MaterialTheme.colorScheme.background),
    contentPadding = PaddingValues(16.dp),
    verticalArrangement = Arrangement.spacedBy(16.dp)
) {
    // contenu
}
```

Import si nécessaire

```
import androidx.compose.foundation.background
```

On évite de mettre une couleur fixe directement dans l'écran.

On utilise `MaterialTheme.colorScheme.background`, donc l'écran s'adapte au **thème clair** ou **sombre**.

7. Étape 5 – Améliorer le titre du catalogue

Un **titre** important doit utiliser une **typographie** adaptée et une **couleur du thème**.

Nous allons utiliser :



- **headlineMedium** pour l'importance du titre ;
- **primary** pour le mettre en valeur.

Dans **ProductCatalogScreen**, modifiez le titre pour ajouter la couleur **primary** :

```
Text(  
    text = "Catalogue produits",  
    style = MaterialTheme.typography.headlineMedium,  
    color = MaterialTheme.colorScheme.primary  
)
```

La **typographie** gère la **hiérarchie** du texte.

La **couleur** aide à identifier les éléments **importants**.

8. Étape 6 – Personnaliser la carte produit

Dans le TP6, **ProductListItem** utilisait une **Card** simple.

Nous allons lui donner une apparence plus cohérente avec le **thème**.

Une carte peut utiliser :

- une **couleur de conteneur** ;
- une **forme** ;
- une **élévation**.

Modifiez le début de **ProductListItem** :

```
Card(  
    modifier = modifier.fillMaxWidth(),  
    shape = MaterialTheme.shapes.large,  
    colors = CardDefaults.cardColors(  
        containerColor = MaterialTheme.colorScheme.surface  
    ),  
    elevation = CardDefaults.cardElevation(  
        defaultElevation = 3.dp  
    )  
) {  
    // contenu  
}
```

Imports si nécessaire

```
import androidx.compose.material3.CardDefaults
```

La **carte** utilise maintenant les **formes** et les **couleurs** du thème.

Si le thème change, l'apparence des cartes peut rester cohérente.

9. Étape 7 – Mettre en valeur les informations produit

Dans une carte produit, toutes les informations n'ont pas la même importance.

Nous allons différencier :

- le **titre** ;
- la **marque** ;
- le **prix** et la **note**.

Dans **ProductListItem**, adaptez les textes principaux :



```
Text(  
    text = product.title,  
    style = MaterialTheme.typography.titleLarge,  
    color = MaterialTheme.colorScheme.onSurface  
)
```

```
Text(  
    text = product.brand,  
    style = MaterialTheme.typography.bodyMedium,  
    color = MaterialTheme.colorScheme.secondary  
)
```

Puis pour le **prix** et la **note** :

```
Text(  
    text = "${product.price} € • ★ ${product.rating}",  
    style = MaterialTheme.typography.titleMedium,  
    color = MaterialTheme.colorScheme.primary  
)
```

- On ne change pas les données.
- On améliore leur lisibilité.
- Le thème nous aide à donner une hiérarchie visuelle.

10. Étape 8 – Améliorer les catégories

Les catégories doivent ressembler à de petits éléments visuels cohérents.

Nous allons améliorer **CategoryChip** :

```
@Composable  
fun CategoryChip(  
    label: String,  
    modifier: Modifier = Modifier  
) {  
    Surface(  
        modifier = modifier,  
        shape = MaterialTheme.shapes.extraLarge,  
        color = MaterialTheme.colorScheme.secondary,  
        contentColor = MaterialTheme.colorScheme.onPrimary  
    ) {  
        Text(  
            text = label,  
            modifier = Modifier.padding(horizontal = 14.dp, vertical = 8.dp),  
            style = MaterialTheme.typography.labelLarge  
        )  
    }  
}
```

Surface peut fournir une **couleur de fond** et une **couleur de contenu**.

Le texte à l'intérieur utilise automatiquement la couleur de contenu de la **Surface**.

11. Étape 9 – Définir des formes personnalisées

Shapes permet de contrôler les **arrondis** des composants. Cela donne une identité visuelle à l'application.

Nous allons définir des formes simples.

Dans **Theme.kt**, ajoutez :

```
private val ProductShapes = Shapes(  
    small = RoundedCornerShape(8.dp),  
    medium = RoundedCornerShape(16.dp),  
    large = RoundedCornerShape(24.dp),  
    extraLarge = RoundedCornerShape(32.dp)  
)
```



Puis passez ces formes à **MaterialTheme** qui deviens :

```
MaterialTheme(  
    colorScheme = colorScheme,  
    typography = Typography,  
    shapes = ProductShapes,  
    content = content  
)
```

Imports si nécessaire

```
import androidx.compose.foundation.shape.RoundedCornerShape  
import androidx.compose.material3.Shapes  
import androidx.compose.ui.unit.dp
```

Quand un composant utilise **MaterialTheme.shapes.large**, il utilise maintenant votre définition.

Cela **évite de répéter** les mêmes arrondis partout.

12. Étape 10 – Définir une typographie simple

La **typographie** permet de donner une hiérarchie claire aux textes. Nous allons personnaliser quelques **styles** seulement. Il n'est pas nécessaire de modifier toute l'échelle typographique.

Dans le fichier **ui.theme/Type.kt**, ou directement dans **Theme.kt** si votre projet est simple, créez :

```
val ProductTypography = Typography(  
    headlineMedium = TextStyle(  
        fontSize = 28.sp,  
        lineHeight = 34.sp,  
        fontWeight = FontWeight.Bold  
    ),  
    titleLarge = TextStyle(  
        fontSize = 20.sp,  
        lineHeight = 26.sp,  
        fontWeight = FontWeight.SemiBold  
    ),  
    titleMedium = TextStyle(  
        fontSize = 16.sp,  
        lineHeight = 22.sp,  
        fontWeight = FontWeight.SemiBold  
    ),  
    bodyMedium = TextStyle(  
        fontSize = 14.sp,  
        lineHeight = 20.sp,  
        fontWeight = FontWeight.Normal  
    ),  
    labelLarge = TextStyle(  
        fontSize = 14.sp,  
        lineHeight = 18.sp,  
        fontWeight = FontWeight.Medium  
    )  
)
```

Puis utilisez-la dans **MaterialTheme** :

```
MaterialTheme(  
    colorScheme = colorScheme,  
    typography = ProductTypography,  
    shapes = ProductShapes,  
    content = content  
)
```

Imports si nécessaire

```
import androidx.compose.material3.Typography  
import androidx.compose.ui.text.TextStyle  
import androidx.compose.ui.text.font.FontWeight  
import androidx.compose.ui.unit.sp
```



Les textes qui utilisent `MaterialTheme.typography.headlineMedium`, `titleLarge`, etc. prennent maintenant votre **style**.

Cela améliore l'identité visuelle sans modifier tous les `Text` un par un.

13. Étape 11 – Ajouter une preview sombre

Il faut vérifier que l'interface fonctionne aussi en **thème sombre**.

Nous allons créer une **preview** qui force `darkTheme = true` :

```
@Preview(
    showBackground = true,
    name = "Catalogue - Dark Theme"
)
@Composable
fun ProductCatalogScreenDarkPreview() {
    ProductExplorerTheme(darkTheme = true) {
        ProductCatalogScreen(
            products = sampleProducts(),
            onProductClick = {}
        )
    }
}
```

Cette **preview** permet de voir rapidement le **rendu sombre** sans changer les paramètres du téléphone.

14. Étape 12 – Ajouter une preview claire explicite

Ajoutons aussi une **preview claire** si vous n'en avez pas déjà, pour comparer les deux.

```
@Preview(
    showBackground = true,
    name = "Catalogue - Light Theme"
)
@Composable
fun ProductCatalogScreenLightPreview() {
    ProductExplorerTheme(darkTheme = false) {
        ProductCatalogScreen(
            products = sampleProducts(),
            onProductClick = {}
        )
    }
}
```

Les deux **previews** affichent le même écran. Seul le thème change. C'est exactement l'intérêt de `MaterialTheme`.

15. Étape 13 – Vérifier le contraste

Une interface plus jolie n'est pas forcément plus lisible.

Vérifiez que :

- le texte est lisible sur les cartes ;
- les boutons restent visibles ;
- les catégories sont compréhensibles ;
- le mode sombre ne rend pas certains textes invisibles.

Lancez les deux **previews** :

- `ProductCatalogScreenLightPreview` ;
- `ProductCatalogScreenDarkPreview`.



Observez les différences.

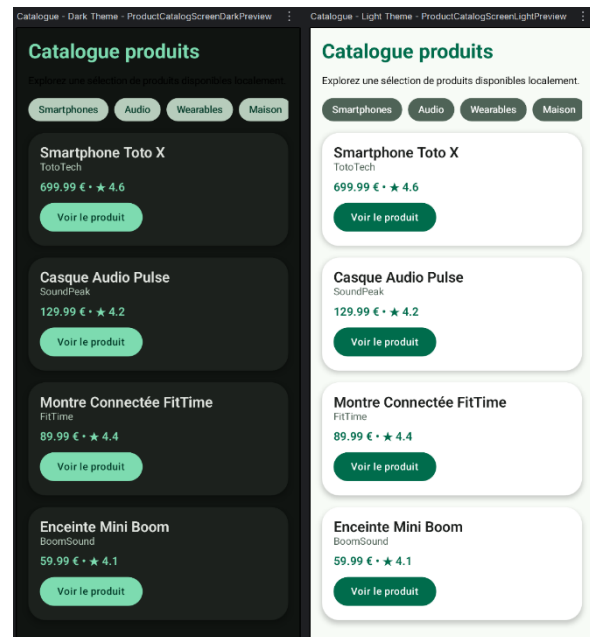
Si une couleur est difficile à lire, changez la palette plutôt que de corriger chaque composant séparément.

Un bon thème doit fonctionner partout.

Si vous devez corriger chaque composant individuellement, c'est souvent que le thème n'est pas assez cohérent.

16. Version complète avec Material Design 3 et thème

Voici la version finale du code (**MainActivity.kt**, **Theme.kt**, **Type.kt**) :



<https://gist.github.com/ariatsi/18d5374da2171e822237ada5244ff734> .

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/tp7-final> .

17. Ce qu'il faut retenir

Dans ce TP, vous avez personnalisé graphiquement **Product Explorer** avec **Material Design 3**.

Vous avez vu que le thème permet de centraliser :

- les couleurs avec **ColorScheme** ;
- les textes avec **Typography** ;
- les formes avec **Shapes** ;
- l'adaptation au **thème clair** ou **sombre**.

Vous avez aussi amélioré l'apparence de composants déjà créés :

- **titre** du catalogue ;
- **cartes** produit ;
- **catégories** ;
- écran **catalogue**.

Le point important est le suivant : Le thème permet d'obtenir une interface cohérente sans modifier chaque composant séparément.

À la fin de ce bloc, vous disposez maintenant d'une **mini-application Compose** avec :

- des **composants fondamentaux** ;
- des **layouts** ;
- des **listes** modernes ;
- une personnalisation **Material 3**.