



TP guidé : Créer un écran dynamique simple avec Compose

1. Objectif du TP

Dans ce TP, vous allez construire progressivement un petit écran Compose permettant d'afficher une information qui change selon des données.

L'objectif n'est pas encore de construire un projet complet mais de pratiquer les notions de la séance :

- créer plusieurs composables ;
- construire un arbre d'interface ;
- passer des paramètres ;
- modifier l'affichage selon une valeur ;
- séparer données et événements ;
- commencer à appliquer le principe "*state down, events up*".

Nous allons travailler sur un écran simple de type « fiche produit ».

Ce travail prépare le futur projet **Shop Explorer**, mais il peut être réalisé aussi dans une application Compose de découverte.

2. Point de départ

Vous pouvez repartir d'un projet Compose simple, par exemple celui utilisé lors du premier TP ou en créer un nouveau.

Dans `MainActivity.kt`, gardez la structure générale générée par Android Studio :

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            GreetingCardTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    ProductDemoScreen(
                        modifier = Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }
}
```

Selon le nom de votre projet, le thème ne s'appellera pas forcément `GreetingCardTheme`.

Utilisez le nom réellement généré dans votre projet, par exemple : `MyApplicationTheme`, `GreetingCardTheme` ou un autre nom selon votre projet.

3. Étape 1 – Créer un premier écran

Nous allons commencer par créer un écran très simple.

Ajoutez ce composable (à la place de la fonction `Greeting()` si vous partez du projet **GreetingCard** du dernier TP) :



```
@Composable
fun ProductDemoScreen(modifier: Modifier = Modifier) {
    Text(
        text = "Produit de démonstration",
        modifier = modifier.padding(24.dp)
    )
}
```

Ajoutez aussi une **Preview** :

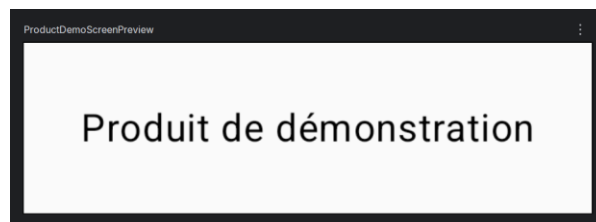
```
@Preview(showBackground = true)
@Composable
fun ProductDemoScreenPreview() {
    GreetingCardTheme {
        ProductDemoScreen()
    }
}
```

L'objectif ici est simplement de vérifier que votre écran s'affiche.

À ce stade, l'arbre est très simple :

```
ProductDemoScreen
└─ Text
```

Vous avez un composable qui affiche un texte.



4. Étape 2 – Créer un composable ProductTitle

Nous allons maintenant découper l'interface.

Au lieu d'écrire directement le **Text** dans **ProductDemoScreen**, créons un composable dédié au titre du produit.

Ajoutez :

```
@Composable
fun ProductTitle(name: String) {
    Text(text = name)
}
```

Puis modifiez l'écran :

```
@Composable
fun ProductDemoScreen(modifier: Modifier = Modifier) {
    ProductTitle(
        name = "Sac à dos",
    )
}
```

Pour garder une marge, on peut placer le **modifier** sur un **conteneur**. Nous allons utiliser **Surface** :



```
@Composable
fun ProductDemoScreen(modifier: Modifier = Modifier) {
    Surface(modifier = modifier.padding(24.dp)) {
        ProductTitle(
            name = "Sac à dos"
        )
    }
}
```

Ce que vous venez de faire est **important** : vous avez créé un **composant réutilisable**.

ProductTitle ne connaît pas le produit « *Sac à dos* ». Il reçoit un paramètre **name**.

Cela signifie qu'on peut maintenant l'utiliser avec plusieurs valeurs, par exemple :

```
ProductTitle(name = "Sac à dos")
```

```
ProductTitle(name = "Casque audio")
```

```
ProductTitle(name = "Clavier mécanique")
```

5. Étape 3 – Ajouter un prix

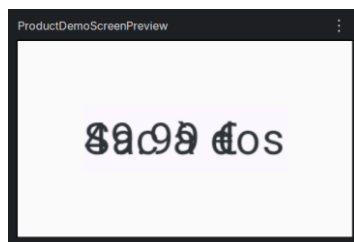
Ajoutons maintenant un deuxième composable :

```
@Composable
fun ProductPrice(price: Double) {
    Text(text = "$price €")
}
```

Puis modifions l'écran :

```
@Composable
fun ProductDemoScreen(modifier: Modifier = Modifier) {
    Surface(modifier = modifier.padding(24.dp)) {
        ProductTitle(name = "Sac à dos")
        ProductPrice(price = 49.99)
    }
}
```

Ce code pose un problème : **Surface** contient deux composables côte à côte sans structure claire :



Pour organiser **verticalement** les éléments, on utilise **Column**. Ajoutez l'import si nécessaire (ou laissez IDE de s'en charger plus tard) : **import androidx.compose.foundation.layout.Column**.

Puis écrivez :

Code

Résultat



```
@Composable
fun ProductDemoScreen(modifier: Modifier = Modifier) {
    Surface(modifier = modifier.padding(24.dp)) {
        Column {
            ProductTitle(name = "Sac à dos")
            ProductPrice(price = 49.99)
        }
    }
}
```



Rappel : Pour entourer automatiquement les 2 instructions / lignes du code **ProductTitle** et **ProductPrice** par un centenaire dans Android Studio : sélectionner le code, puis utiliser l'action rapide :

- **Windows / Linux :** **Alt + Enter**
- **macOS :** **Option + Enter**

Puis choisir l'action proposée du type **Surround with widget** → **Surround with Column**.

L'arbre devient :

```
ProductDemoScreen
├── Surface
│   └── Column
│       ├── ProductTitle
│       │   └── Text
│       └── ProductPrice
│           └── Text
```

Vous voyez ici la notion d'arbre de composition.

6. Étape 4 – Créer un composable **ProductCard**

Pour rendre le code plus propre, nous allons créer un composable **ProductCard**. Ajoutez :

```
@Composable
fun ProductCard(
    name: String,
    price: Double,
    modifier: Modifier = Modifier
) {
    Surface(modifier = modifier) {
        Column(
            modifier = Modifier.padding(16.dp)
        ) {
            ProductTitle(name = name)
            ProductPrice(price = price)
        }
    }
}
```

Puis simplifiez **ProductDemoScreen** :

```
@Composable
fun ProductDemoScreen(modifier: Modifier = Modifier) {
    ProductCard(
        name = "Sac à dos",
        price = 49.99,
        modifier = modifier.padding(24.dp)
    )
}
```

Ce découpage est plus propre.

- **ProductDemoScreen** décide quoi afficher.



- **ProductCard** décrit la carte produit.
- **ProductTitle** affiche le nom.
- **ProductPrice** affiche le prix.

Chaque composable a une responsabilité plus claire.

7. Étape 5 – Ajouter un état visuel : disponible ou indisponible

Ajoutons maintenant une donnée **booléenne** : le produit est-il disponible ?

Créez ce composable :

```
@Composable
fun ProductAvailability(isAvailable: Boolean) {
    Text(
        text = if (isAvailable) {
            "Produit disponible"
        } else {
            "Produit indisponible"
        }
    )
}
```

Puis modifiez **ProductCard** :

```
@Composable
fun ProductCard(
    name: String,
    price: Double,
    isAvailable: Boolean,
    modifier: Modifier = Modifier
) {
    Surface(modifier = modifier) {
        Column(
            modifier = Modifier.padding(16.dp)
        ) {
            ProductTitle(name = name)
            ProductPrice(price = price)
            ProductAvailability(isAvailable = isAvailable)
        }
    }
}
```

Modifiez aussi l'appel :

```
@Composable
fun ProductDemoScreen(modifier: Modifier = Modifier) {
    ProductCard(
        name = "Sac à dos",
        price = 49.99,
        isAvailable = true,
        modifier = modifier.padding(24.dp)
    )
}
```

Testez ensuite avec : **isAvailable = false**. Observez que l'interface change simplement parce que la donnée passée au composable change.

isAvailable = true

isAvailable = false



C'est exactement le modèle : **Donnée différente → Interface différente.**

8. Étape 6 – Ajouter un bouton de favori

Nous allons ajouter un bouton.

Ajoutez l'import si nécessaire : `import androidx.compose.material3.Button` et créez ce composable :

```
@Composable
fun FavoriteButton(
    isFavorite: Boolean,
    onFavoriteClick: () -> Unit
) {
    Button(onClick = onFavoriteClick) {
        Text(
            text = if (isFavorite) {
                "★ Favori"
            } else {
                "☆ Ajouter aux favoris"
            }
        )
    }
}
```

Analysez bien ce code.

- `isFavorite` est une donnée.
- `onFavoriteClick` est un événement.

Le bouton ne décide pas lui-même comment gérer le favori. Il reçoit l'état et signale le clic.

Modifiez `ProductCard` :

```
@Composable
fun ProductCard(
    name: String,
    price: Double,
    isAvailable: Boolean,
    isFavorite: Boolean,
    onFavoriteClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Surface(modifier = modifier) {
        Column(
            modifier = Modifier.padding(16.dp)
        ) {
            ProductTitle(name = name)
            ProductPrice(price = price)
            ProductAvailability(isAvailable = isAvailable)

            FavoriteButton(
                isFavorite = isFavorite,
                onFavoriteClick = onFavoriteClick
            )
        }
    }
}
```



Puis modifiez l'écran :

```
@Composable
fun ProductDemoScreen(modifier: Modifier = Modifier) {
    ProductCard(
        name = "Sac à dos",
        price = 49.99,
        isAvailable = true,
        isFavorite = false,
        onFavoriteClick = {
            // Pour l'instant, nous ne modifions pas encore l'état.
            // Nous verrons cela dans la séance sur la gestion d'état.
        },
        modifier = modifier.padding(24.dp)
    )
}
```

À ce stade, le bouton s'affiche, mais cliquer dessus ne change pas encore l'interface - c'est normal.

La séance d'aujourd'hui porte sur **le modèle mental** : données, paramètres, événements, arbre de composition.

La modification réelle de l'état sera étudiée dans la suite du bloc **State Management**.

9. Étape 7 – Prévisualiser plusieurs états

Les **previews** sont très utiles pour observer plusieurs cas. Créez une **preview** pour le produit disponible :

```
@Preview(showBackground = true)
@Composable
fun ProductCardAvailablePreview() {
    GreetingCardTheme {
        ProductCard(
            name = "Sac à dos",
            price = 49.99,
            isAvailable = true,
            isFavorite = false,
            onFavoriteClick = {},
            modifier = Modifier.padding(24.dp)
        )
    }
}
```

Puis une **preview** pour le produit indisponible et favori :

```
@Preview(showBackground = true)
@Composable
fun ProductCardUnavailableFavoritePreview() {
    GreetingCardTheme {
        ProductCard(
            name = "Casque audio",
            price = 89.99,
            isAvailable = false,
            isFavorite = true,
            onFavoriteClick = {},
            modifier = Modifier.padding(24.dp)
        )
    }
}
```

Ces **previews** permettent de voir rapidement plusieurs états de votre interface.



Dans une vraie application, c'est une excellente habitude : ne testez pas seulement l'état idéal. Testez aussi les variantes.

10. Étape 8 – Observer l'UDF (flux de données unidirectionnel) dans votre code

Regardons la signature de **ProductCard** :

```
@Composable
fun ProductCard(
    name: String,
    price: Double,
    isAvailable: Boolean,
    isFavorite: Boolean,
    onFavoriteClick: () -> Unit,
    modifier: Modifier = Modifier
)
```

On peut séparer les paramètres en deux groupes.

1. Données qui descendent :

```
name: String
price: Double
isAvailable: Boolean
isFavorite: Boolean
```

2. Événement qui remonte :

```
onFavoriteClick: () -> Unit
```

C'est exactement le flux de données unidirectionnel :

- Données → UI
- UI → Événements

Même si le bouton ne change pas encore l'état, vous avez déjà construit la bonne forme de composant.

Dans une prochaine séance, nous verrons comment faire évoluer réellement **isFavorite**.

11. Code complet indicatif

Voici une version complète possible du fichier, hors imports et hors **MainActivity**.

```
@Composable
fun ProductDemoScreen(modifier: Modifier = Modifier) {
    ProductCard(
        name = "Sac à dos",
        price = 49.99,
        isAvailable = true,
        isFavorite = false,
        onFavoriteClick = {
```



```
// Pour l'instant, nous ne modifions pas encore l'état.
// Nous verrons cela dans la séance sur la gestion d'état.
    },
    modifier = modifier.padding(24.dp)
  )
}

@Composable
fun ProductCard(
    name: String,
    price: Double,
    isAvailable: Boolean,
    isFavorite: Boolean,
    onFavoriteClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Surface(modifier = modifier) {
        Column(
            modifier = Modifier.padding(16.dp)
        ) {
            ProductTitle(name = name)
            ProductPrice(price = price)
            ProductAvailability(isAvailable = isAvailable)

            FavoriteButton(
                isFavorite = isFavorite,
                onFavoriteClick = onFavoriteClick
            )
        }
    }
}

@Composable
fun ProductTitle(name: String) {
    Text(text = name)
}

@Composable
fun ProductPrice(price: Double) {
    Text(text = "$price €")
}

@Composable
fun ProductAvailability(isAvailable: Boolean) {
    Text(
        text = if (isAvailable) {
            "Produit disponible"
        } else {
            "Produit indisponible"
        }
    )
}

@Composable
fun FavoriteButton(
    isFavorite: Boolean,
    onFavoriteClick: () -> Unit
) {
    Button(onClick = onFavoriteClick) {
        Text(
            text = if (isFavorite) {
                "★ Favori"
            } else {
                "☆ Ajouter aux favoris"
            }
        )
    }
}

@Preview(showBackground = true)
@Composable
fun ProductDemoScreenPreview() {
```



```
GreetingCardTheme {
    ProductDemoScreen()
}

@Preview(showBackground = true)
@Composable
fun ProductCardAvailablePreview() {
    GreetingCardTheme {
        ProductCard(
            name = "Sac à dos",
            price = 49.99,
            isAvailable = true,
            isFavorite = false,
            onFavoriteClick = {},
            modifier = Modifier.padding(24.dp)
        )
    }
}

@Preview(showBackground = true)
@Composable
fun ProductCardUnavailableFavoritePreview() {
    GreetingCardTheme {
        ProductCard(
            name = "Casque audio",
            price = 89.99,
            isAvailable = false,
            isFavorite = true,
            onFavoriteClick = {},
            modifier = Modifier.padding(24.dp)
        )
    }
}
```

Adaptez le nom du thème selon celui de votre projet.

12. Questions de compréhension (auto-contrôle)

Répondez brièvement aux questions suivantes.

1. Pourquoi `ProductTitle` reçoit-il un paramètre `name` ?
2. Quel est l'intérêt de découper l'interface en plusieurs composables ?
3. Quel est le rôle de `ProductCard` ?
4. Que se passe-t-il visuellement si `isAvailable` passe de `true` à `false` ?
5. Pourquoi `FavoriteButton` reçoit-il `isFavorite` au lieu de le créer lui-même ?
6. Quel est le rôle de `onFavoriteClick` ?
7. Dans ce TP, quelles données descendent vers l'interface ?
8. Quel événement remonte vers le parent ?
9. Pourquoi le clic sur le bouton ne modifie-t-il pas encore l'état ?
10. En quoi les `previews` multiples sont-elles utiles ?

13. Résultat du TP

À la fin du TP, vous devez avoir :

- un écran `ProductDemoScreen` fonctionnel ;
- un composable `ProductCard` ;
- des composables séparés pour le **titre**, le **prix**, la **disponibilité** et le **bouton favori** ;
- au moins deux `previews` montrant **deux états différents** ;
- des réponses aux questions de compréhension.