Sujet de TP N°4 - Création d'une application To-Do avec Jetpack Compose

VUE D'ENSEMBLE

Dans ce TP, vous allez développer une application Android simple utilisant Jetpack Compose pour concevoir une liste de tâches (to-do list). Ce projet vous permettra de vous familiariser avec les composants de base de Compose, la gestion de l'état et les modificateurs, tout en explorant la prévisualisation des éléments de l'interface dans Android Studio. Ce TP vous servira également de base pour comprendre comment gérer des données dynamiques et actualiser l'affichage en fonction des interactions de l'utilisateur.

OBJECTIFS

- Créer une application Android en utilisant Jetpack Compose.
- Utiliser la prévisualisation pour voir et ajuster les composables directement dans Android Studio.
- Concevoir une liste de tâches to-do simple et interactive.
- Gérer l'état des éléments dans la liste et apprendre la recomposition dans Compose.
- Appliquer des Modificateurs pour ajuster l'apparence et le comportement des éléments de l'interface utilisateur.

Fonctionnalités à mettre en œuvre :

- Une liste de tâches (to-do list) interactive où chaque tâche est représentée par un élément contenant un texte descriptif, une case à cocher, et une icône.
- Affichage en temps réel des modifications apportées aux tâches grâce à la gestion de l'état.
- Application de modificateurs pour personnaliser la présentation des éléments de la liste.

Étapes du TP

Étape 1 : Création de l'application et premier composable

1. **Créer un nouveau projet Android avec Jetpack Compose**

- Ouvrez Android Studio et créez un nouveau projet en sélectionnant le template **Empty Activity**.
- Nommez votre application `TP_04_name_name`.
- Choisissez le langage **Kotlin** et sélectionnez **API 29** comme minimum SDK.

2. **Configuration du premier composable**

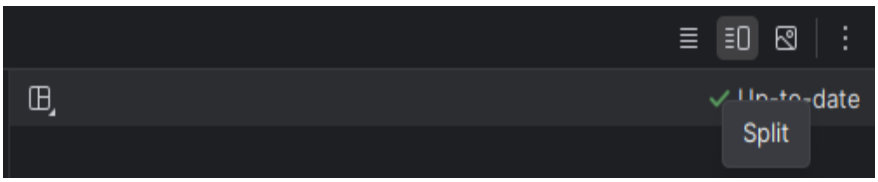
- Dans le fichier principal de l'activité (par défaut `MainActivity.kt`), créez un composable nommé `Todo`.
- Le composable `Todo` prendra deux paramètres :
  - `title : String` — pour afficher le titre de la tâche.
  - `modifier : Modifier` — pour ajuster l'apparence et le comportement du composable (nous expliquerons plus en détail les modificateurs dans les étapes suivantes).
- Pour cette première étape, faites en sorte que `Todo` n'affiche que le texte passé en paramètre `title`.

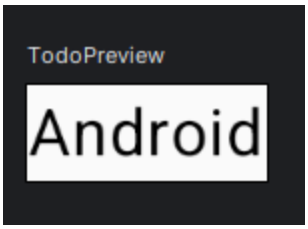
  Conseil : *Placez le composable `Todo` directement dans le fichier `MainActivity.kt` pour simplifier la gestion des fichiers dans cette première version.*

3. **Prévisualisation du composable**

- Dans Android Studio, vous pouvez facilement prévisualiser les composables que vous êtes en train de concevoir. La prévisualisation vous permet de voir directement les résultats de votre code sans avoir à exécuter l'application, ce qui accélère le processus de développement visuel.
- Pour activer la prévisualisation :
  - Créez une fonction `@Composable` dédiée à l'aperçu (par exemple, `TodoPreview`).
  - **Ajoutez l'annotation `@Preview`** au-dessus de cette fonction. Cela indique à Android Studio que vous souhaitez voir un aperçu du composable.
  - Appelez le composable `Todo` à l'intérieur de cette fonction pour visualiser un exemple de rendu.

```
@Preview(showBackground = true)
@Composable
fun TodoPreview() {
    TP_04Theme {
        Todo(title = "Android")
    }
}
```





Étape 2 : Ajout d'images au projet et gestion des ressources

1. **Ajout des images au projet**

   Pour cette étape, vous allez inclure dans votre projet des images qui se trouvent dans le dossier **TP - 4 - Images**. Ces images représentent différentes catégories et seront utilisées dans l'interface de l'application.

   Méthode 1 (Recommandée) : Depuis Android Studio

   - Ouvrez Android Studio et accédez au **Resource Manager**.
   - Cliquez sur **Add resource** puis sur **Import drawable**.
   - Sélectionnez les fichiers d'image et importez-les directement dans le dossier `drawable`.

   Méthode 2 : Ajouter les fichiers directement dans le dossier `drawable`

   - Dans l'explorateur de fichiers de votre système, copiez les fichiers d'image depuis le dossier **TP - 4 - Images**.
   - Collez-les directement dans le dossier `res/drawable` de votre projet.

   Conseil : *Utiliser le* ***Resource Manager*** *d'Android Studio facilite la gestion des ressources, car il gère également l'optimisation et l'organisation des fichiers d'images.*

2. Structure des ressources Android

Dans Android, les ressources (images, chaînes de texte, couleurs, etc.) sont placées dans le répertoire `res/`, qui contient des sous-répertoires pour chaque type de ressource. Voici quelques-uns des sous-répertoires les plus courants :

- `drawable/` — Contient les images et les graphiques (icônes, illustrations, etc.).
- `values/` — Contient les ressources de texte (`strings.xml`), couleurs (`colors.xml`), styles, etc.
- `layout/` — Contient les fichiers de mise en page XML (non utilisés ici car nous travaillons avec Jetpack Compose).

3. Tailles d’images

Lorsque vous ajoutez des images, il est important de comprendre comment Android gère les différentes tailles d'images en fonction de la densité d'écran. Android utilise des catégories de densité, appelées **DPI (dots per inch)**, pour garantir que les images s'affichent correctement sur différents appareils.

Voici les catégories de densité d'écran et leurs DPI associés :

- **ldpi (Low DPI)** : 120 dpi
- **mdpi (Medium DPI)** : 160 dpi (densité de référence)
- **hdpi (High DPI)** : 240 dpi
- **xhdpi (Extra High DPI)** : 320 dpi
- **xxhdpi (Extra Extra High DPI)** : 480 dpi
- **xxxhdpi (Extra Extra Extra High DPI)** : 640 dpi

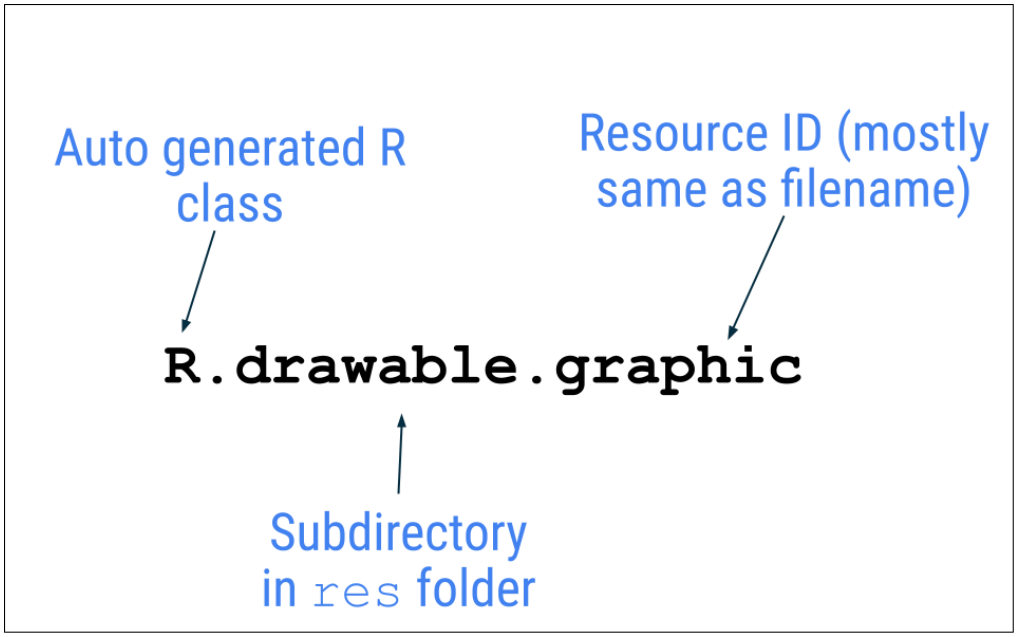
Chaque catégorie est conçue pour s'adapter à des types d'appareils spécifiques, permettant aux développeurs de fournir des images optimisées pour garantir une qualité d'affichage optimale. Il est recommandé de créer des versions de vos images pour chaque densité afin de maintenir une bonne qualité visuelle sur tous les écrans.

4. Accéder aux ressources

Jetpack Compose permet d'accéder aux ressources du projet grâce aux ID générés dans la classe R. Android génère automatiquement cette classe pour contenir les ID de toutes

les ressources du projet. Dans la plupart des cas, l'ID de la ressource est identique au nom du fichier.

Par exemple, une image nommée `cat_1.png` dans le répertoire `drawable/` est accessible avec `R.drawable.cat_1`.



Pour afficher cette image dans un composable, vous pouvez utiliser `painterResource` en combinaison avec `R.drawable.nom_image`, comme dans l'exemple suivant :

```
Image(
    painter = painterResource(id = R.drawable.cat_1),
    contentDescription = "Category 1"
)
```

5. **Fonction composable pour afficher l'image**

- Créez une nouvelle fonction composable nommée `CategoryImage`. Cette fonction prendra deux paramètres :
    - `resId: Int` — pour identifier la ressource image à afficher.
    - `modifier: Modifier` — pour ajuster l'apparence et le comportement de l'image.
- Pour afficher l'image, utilisez le composant `Image` de Jetpack Compose. Vous devrez récupérer l'image à partir des ressources en utilisant l'ID que vous aurez passé en paramètre.

Étape 3 : Création de la data class et du jeu de données

Dans cette étape, vous allez définir la structure des éléments de votre liste de tâches en utilisant une `data class` et créer un jeu de données qui sera utilisé dans l'application.

1. **Création de la data class**

- Créez une nouvelle `data class` nommée **TodoItem** avec les trois champs suivants :
  - `categoryResId: Int` — pour identifier la ressource d'image de la catégorie.
  - `title: String` — pour stocker le titre de la tâche.
  - `isCompleted: Boolean` — pour indiquer si la tâche est terminée ou non.

2. **Création d'un jeu de données**

- Créez ensuite un jeu de données en définissant une liste d'une dizaine d'éléments de type **TodoItem**. Ce jeu de données sera utilisé pour remplir votre interface utilisateur plus tard dans le projet.

  Voici un exemple de ce à quoi pourrait ressembler votre liste de données :

```
val todoItems = listOf(
    TodoItem(R.drawable.cat_1, "Acheter du pain", false),
    TodoItem(R.drawable.cat_2, "Faire du sport", true),
    TodoItem(R.drawable.cat_3, "Étudier pour l'examen", false),
)
```

Étape 4 : Création d'un composable pour afficher un élément TodoItem

Dans cette étape, vous allez créer un composable qui affichera un élément de type **TodoItem** sur une ligne. Ce composable utilisera les composants que vous avez définis précédemment, à savoir **Todo** et **CategoryImage**.

1. **Création du composable**

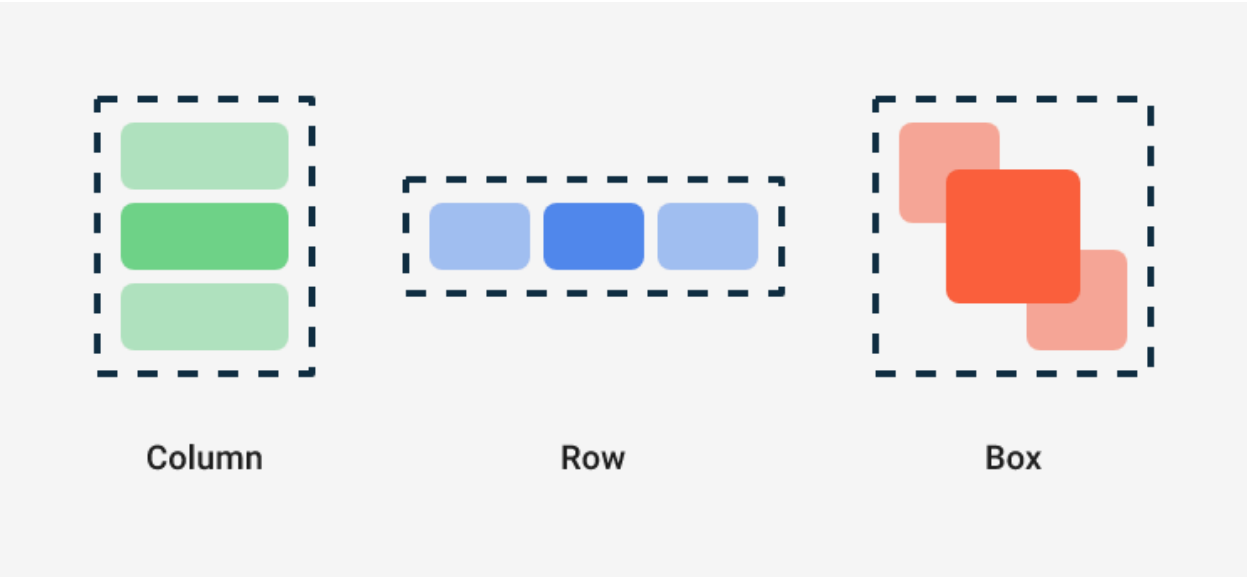
- Créez une fonction composable nommée **TodoItemRow** qui prendra les paramètres suivants :
    - `todoItem: TodoItem` — l'élément de tâche à afficher.
    - `modifier: Modifier` — pour ajuster l'apparence et le comportement du composable.

```
@Composable
fun TodoItemRow(todoItem: TodoItem, modifier: Modifier) {
    // À compléter
}
```

2. **Gérer la mise en page**

Dans Jetpack Compose, les trois éléments de mise en page standards de base sont **Column**, **Row**, et **Box**. Ces fonctions composables permettent de structurer l'affichage des éléments en organisant leur placement :

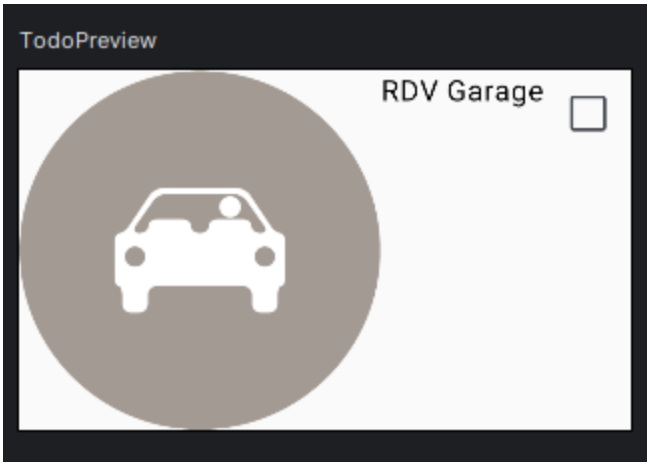
- **Column** : Utilisé pour empiler des éléments verticalement. Chaque enfant à l'intérieur d'une **Column** sera placé les uns au-dessus des autres.
- **Row** : Utilisé pour afficher des éléments sur la même ligne. C'est le composant idéal pour notre cas, car nous souhaitons afficher le **TodoItem** en ligne, avec l'image de la catégorie et le titre de la tâche.
- **Box** : Utilisé pour superposer des éléments. Ce composant est moins courant pour des mises en page simples, mais peut être utile pour des scénarios plus complexes où vous souhaitez superposer des éléments.



Dans notre cas, nous allons utiliser **Row** pour afficher nos composables en ligne. Voici un exemple d'utilisation :

```
Row(modifier = modifier) {
    // Contenu ici (CategoryImage, To-do, CheckBox)
}
```

Après avoir placé les éléments en ligne et mis à jour votre prévisualisation, vous devriez pouvoir constater que les éléments s'affichent correctement en ligne. Cependant, l'interface utilisateur pourrait encore nécessiter des améliorations pour être plus esthétique et fonctionnelle.



Étape 5 : Personnaliser les Composable avec l’objet Modifier

Dans cette étape, vous allez apprendre à utiliser l’objet `Modifier` pour personnaliser l'apparence de vos Composables. Les modificateurs vous permettent d'ajuster l'alignement, la taille, le remplissage, et d'autres aspects des éléments de votre interface utilisateur.

Ajustement du Modifier sur le composable Row

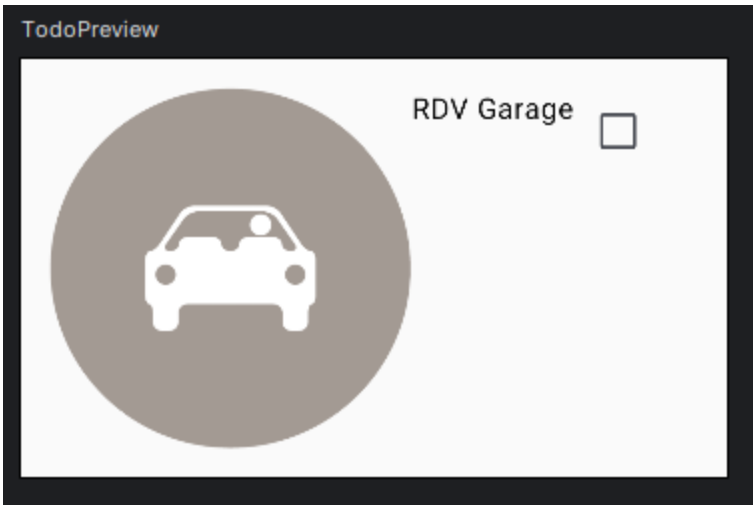
1. **Étendre le Row au maximum en largeur :**

- Trouvez et appliquez le modificateur qui permet à votre composable `Row` de s’étendre au maximum en largeur. Cela garantira qu'il utilise toute la largeur disponible de l'écran.

2. **Ajouter un padding :**

- Toujours dans le `Modifier`, ajoutez un padding de 16 dp pour éviter que les éléments ne collent aux bords de l’écran. Cela améliorera l'esthétique et la lisibilité de votre interface.

Voici le résultat après avoir ajouté les deux propriétés :

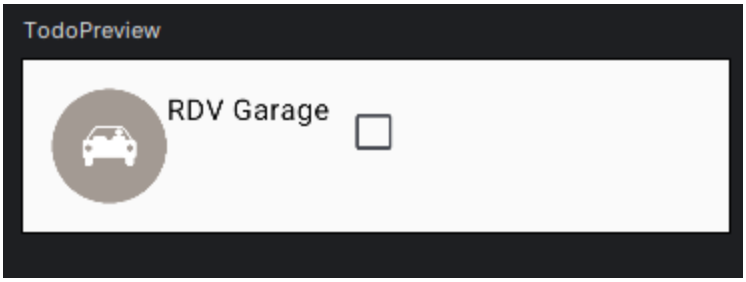


Ajustement du Modifier sur le composable Image

1. **Définir une taille fixe pour l'image :**

- Dans votre fonction composable, appliquez un modificateur à votre composable `Image` pour lui attribuer une taille fixe de 64.dp. Cela permettra de garantir que toutes les images dans votre liste de tâches auront une taille cohérente, ce qui est essentiel pour une présentation harmonieuse.

Voici le résultat après avoir ajouté cette propriété :

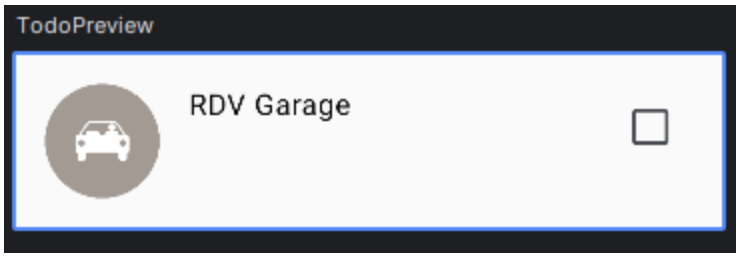


Ajustement du Modifier sur le composable Text

1. **Personnaliser le texte pour occuper toute la largeur**

- Pour que le texte prenne toute la largeur restante après l'image, vous allez utiliser la propriété `weight` dans le modificateur du composable `Text`.
- Ajoutez un padding horizontal au texte pour le décoller de l'image et de la checkbox, ce qui améliorera l'esthétique et la lisibilité de votre interface utilisateur.

Voici le résultat après avoir ajouté cette propriété :

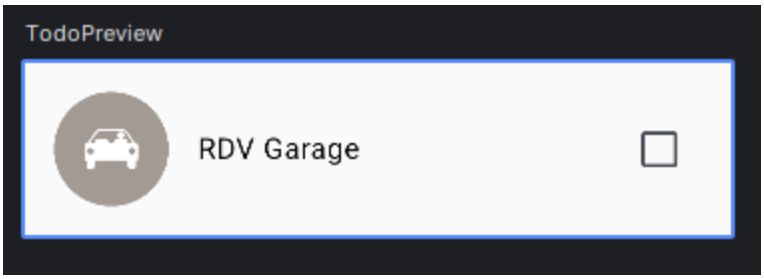


Centrer verticalement les éléments dans la Row

1. **Examinez la déclaration de la fonction `Row` :**

- Recherchez un paramètre dans la fonction `Row` qui permet de centrer verticalement les enfants de ce composable. Utilisez ce paramètre pour centrer en hauteur l'image, le texte et la case à cocher.

Voici le résultat après avoir ajouté cette propriété :



Étape 6 : Afficher la liste de TodoItems dans une Column scrollable

L'objectif de cette étape est d'afficher tous les éléments `TodoItem` dans une liste que l'on peut faire défiler.

Mise en place dans `onCreate`

Dans la méthode `onCreate` de l'activité, vous verrez un composable `Scaffold`. C'est un composable souvent utilisé pour structurer l'interface utilisateur d'un écran Android en Jetpack Compose, en facilitant la gestion de la mise en page.

À l'intérieur de `Scaffold`, vous allez intégrer vos composables créés précédemment et organiser l'affichage de la liste de tâches.

Affichage en colonne

Contrairement à l'affichage en ligne des éléments dans `TodoItemRow`, ici nous cherchons à empiler chaque `TodoItem` verticalement. Pour cela :

1. Utilisez une `Column` pour contenir les éléments.
   Astuce : utilisez le langage Kotlin pour parcourir la liste des tâches avec une boucle et afficher chaque `TodoItem` en appelant votre composable `TodoItemRow`.

2. Recherchez dans le `Modifier` un moyen de rendre la `Column` scrollable pour permettre le défilement vertical, indispensable lorsque la liste est plus longue que la hauteur de l'écran.

Avertissement : `Column` affiche tous les éléments passés en paramètres sans optimisation pour les grandes quantités de données. Pour les longues listes, cela peut devenir coûteux en mémoire et en performances. Dans ce cas, un composable optimisé comme `LazyColumn` serait préférable. Ici, nous utilisons `Column` pour la simplicité, mais gardez cette limitation en tête pour des projets réels.

À ce stade, vous devriez avoir une application fonctionnelle qui affiche toute la liste de tâches. N'hésitez pas à tester votre application pour voir le rendu final et vérifier que chaque élément de votre liste s'affiche correctement.

Étape 7 : Interactions utilisateur

Dans cette étape, vous allez ajouter des interactions utilisateur pour rendre votre application plus dynamique. L'objectif est de permettre aux utilisateurs de marquer les tâches comme terminées, soit en cochant la case, soit en appuyant n'importe où sur la ligne d'un élément `TodoItem`.

Gérer l'événement `onCheckedChange` de la CheckBox

Pour commencer, vous allez gérer l'événement `onCheckedChange` de la CheckBox. Cela vous permettra de mettre à jour l'état de l'élément lorsqu'il est coché ou décoché.

1. **Ajouter une fonction de rappel (callback)**
   Dans votre composable `TodoItemRow`, ajoutez un paramètre supplémentaire :
   `toggleTodoItem: (TodoItem) -> Unit`

   Cette lambda sera appelée chaque fois que l'utilisateur modifie l'état de la case ou clique sur l'élément entier.

2. **Rendre la Row cliquable**

   Pour améliorer l'expérience utilisateur, faites en sorte que toute la ligne (la `Row`) soit cliquable. Ajoutez la propriété `clickable` au `Modifier` de la Row. De cette manière, cliquer sur n'importe quelle partie de l'élément `TodoItem` changera son état.

3. **Utiliser la lambda**

   Dans la CheckBox, appelez la lambda `toggleTodoItem` dans `onCheckedChange`.

   Dans la Row, ajoutez la propriété `clickable` au `Modifier` et appelez également `toggleTodoItem` pour rendre l'ensemble de la ligne interactif.

Si vous essayez votre application maintenant, vous allez voir que la ligne et la case à cocher sont bien cliquables (avec un effet de ripple), mais que rien ne se passe. L'état dans la liste n'est pas encore mis à jour. Dans la prochaine étape, vous allez apprendre à transformer votre liste en `mutableStateListOf<TodoItem>` pour que les changements d'état soient correctement reflétés dans l'interface utilisateur.

4. **Conversion de la liste en mutableStateListOf**

   Convertissez votre `listOf<TodoItem>` en une `mutableStateListOf<TodoItem>` pour que Compose détecte les changements dans la liste.

5. **Mise à jour de la liste avec forEachIndexed**

   Modifiez votre boucle `forEach` pour utiliser `forEachIndexed`. Cela permet d'accéder à l'index de chaque élément, ce qui facilitera la mise à jour de la liste.

6. **Modifier l'élément dans la liste**

   Dans la lambda `toggleTodoItem`, mettez à jour l'élément correspondant en créant une copie de l'objet `TodoItem` à l'index actuel, avec la propriété `isCompleted` inversée.

   Cela devrait ressembler à quelque chose comme :

```
val item = todoItems[index]
todoItems[index] = item.copy(isCompleted = !item.isCompleted)
```

Si vous testez votre application maintenant, vous pouvez voir que vous pouvez cocher ou décocher des éléments de votre liste, ce qui rend l'application beaucoup plus interactive.

Pour aller plus loin...

Pour bien maîtriser Compose, la pratique est essentielle. Maintenant que votre application est fonctionnelle, vous pouvez essayer de lui donner une interface utilisateur plus attrayante. Vous pouvez tenter de reproduire la mise en forme suivante ou laisser libre cours à votre créativité.

Dans cette mise en page, voici quelques nouveautés que vous pourriez intégrer :

- **Un composable Surface** autour de la ligne pour offrir une meilleure gestion des ombres et des coins arrondis.
- **Une couleur de fond** qui change selon l'état `isCompleted` pour donner un feedback visuel immédiat.
- **Une bordure** pour délimiter clairement chaque élément de votre liste.

Amusez-vous à personnaliser votre application et à expérimenter avec les différentes fonctionnalités offertes par Compose !