Sujet de TP N°5 - Conception d'une application mobile réaliste avec Jetpack Compose.

VUE D'ENSEMBLE

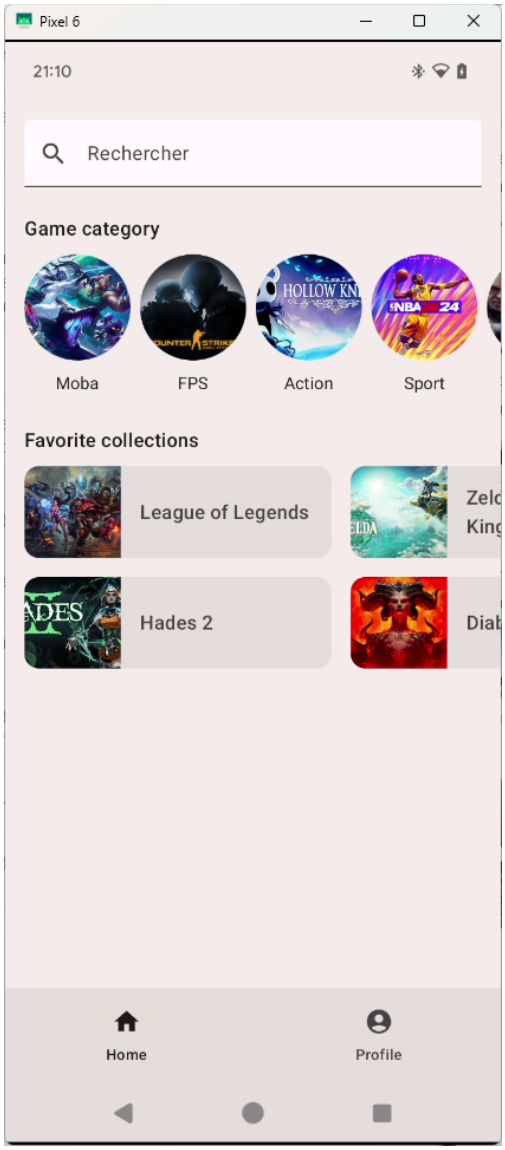
Dans ce TP, vous allez concevoir une mise en page **plus réaliste et complexe** tout en découvrant divers modificateurs et composables prêts à l'emploi. À la fin de cet atelier de programmation, vous serez capable de transformer une conception d'application simple en un code fonctionnel et efficace.

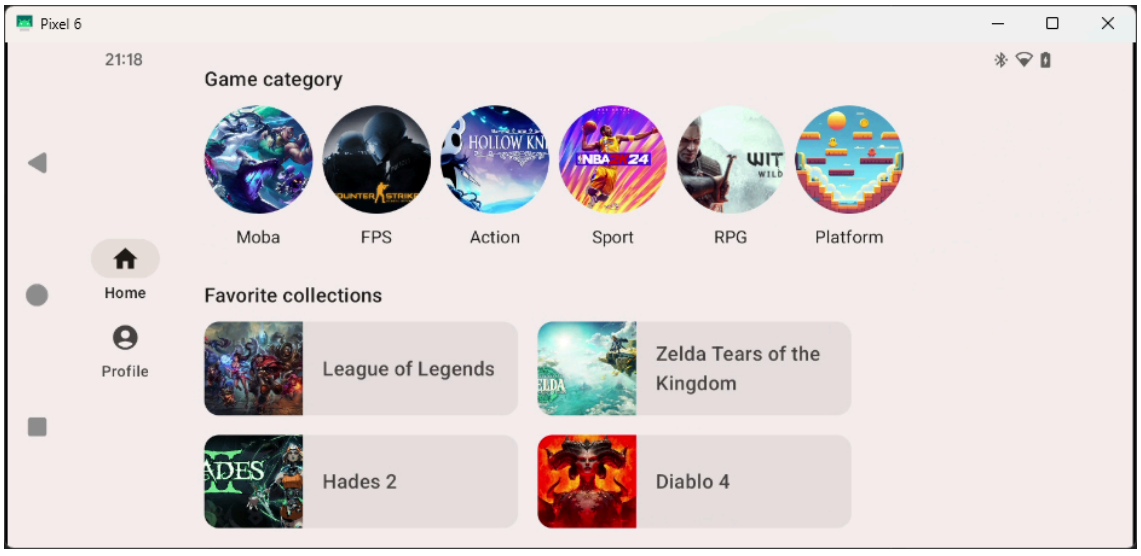
OBJECTIFS

- Comprendre comment les modificateurs enrichissent vos composables.
- Apprendre à utiliser les composants de mise en page standards, tels que `Column` et `LazyRow`, pour positionner les composables enfants.
- Découvrir comment les composables Material, tels que `Scaffold` et `BottomNavigation`, facilitent la création de mises en page complètes.
- Savoir concevoir des composables flexibles et adaptatifs.
- Apprendre à créer des mises en page adaptées à différentes configurations d'écran.
- Faire un premier pas avec les thèmes pour personnaliser l'apparence de l'application.

Fonctionnalités à mettre en œuvre :

Dans ce TP, vous allez implémenter une application réaliste en vous basant sur les maquettes fournies. L'application comportera plusieurs catégories de jeux vidéo, ainsi qu'une section dédiée à vos titres préférés. Voici une présentation de l'application :


Pixel 6
21:10
Rechercher
Game category
Moba
FPS
Action
Sport
Favorite collections
League of Legends
Hades 2
Home
Profile


Pixel 6
21:18
Game category
Moba
FPS
Action
Sport
RPG
Platform
Home
Profile
Favorite collections
League of Legends
Zelda Tears of the Kingdom
Hades 2
Diablo 4

Étapes du TP

Étape 1 : Configuration

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

- Ouvrez Android Studio et sélectionnez le template "*Empty Activity*" pour créer votre projet.
- Nommez votre application `TP_05_name_name`.
- Choisissez le langage **Kotlin** et définissez l'API 29 comme SDK minimum.

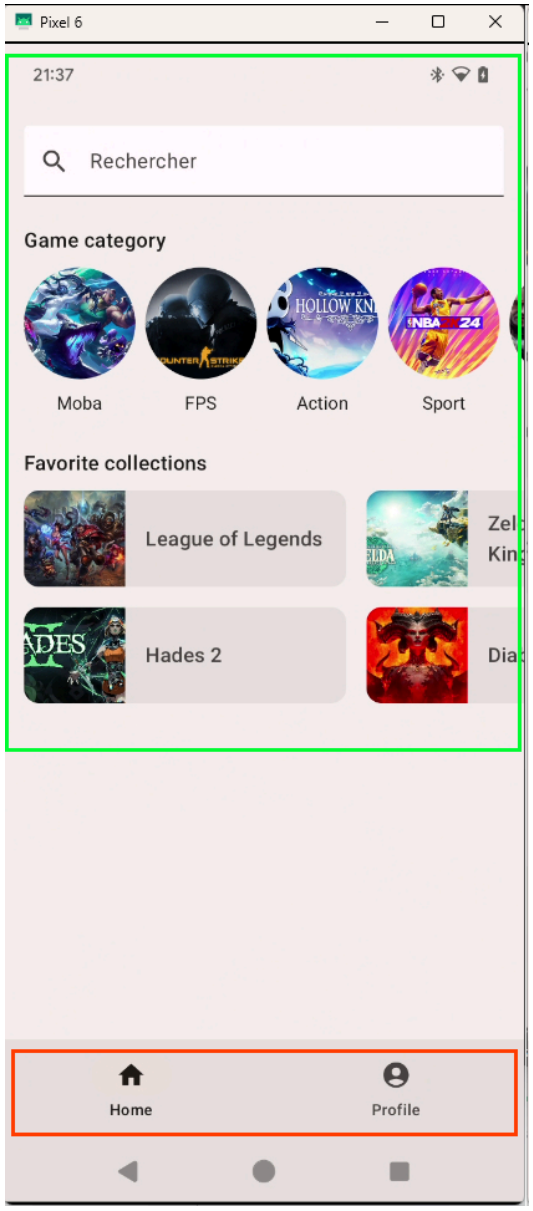
2. **Configuration des dépendances**

- Dans le fichier `build.gradle.kts` (Module :app), mettez à jour les valeurs de `compileSdk` et `targetSdk` à 35 (le template actuel n'est pas à jour et ne cible pas encore Android 15).
- Ensuite, plus bas dans la section des dépendances, ajoutez les deux lignes suivantes :
    - *implementation("com.google.android.material:material:1.12.0")*
    - *implementation("androidx.compose.material3:material3-window-size-class:1.3.1")*

  Android Studio affichera un avertissement car le reste des dépendances utilise le *version catalog*. Ce warning peut être ignoré pour le moment.

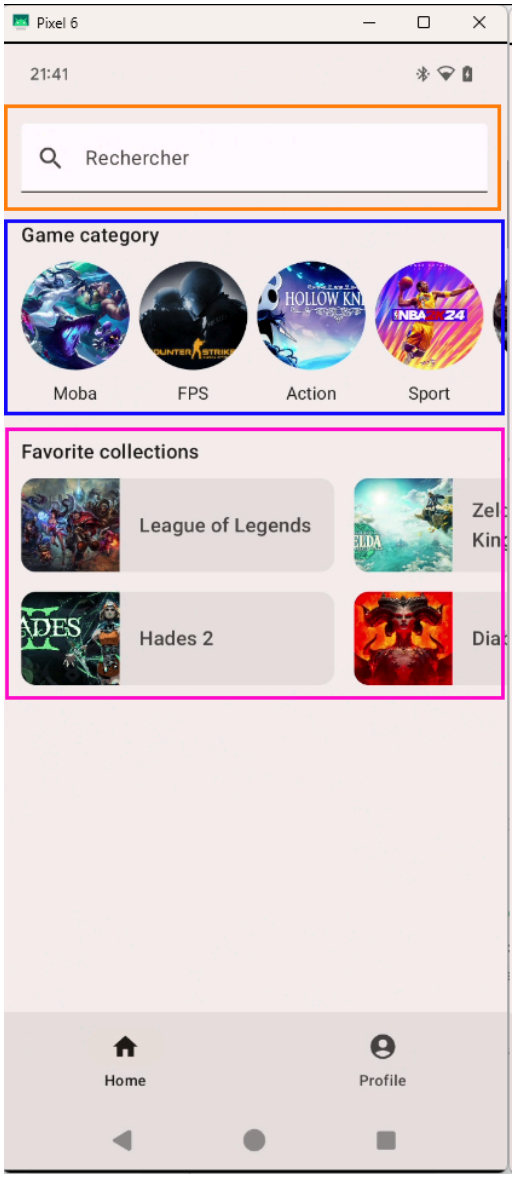
Étape 2 : Conception du mode portrait

Lorsque vous êtes invité à implémenter une conception à partir d'une maquette, il est essentiel de commencer par analyser sa structure. Identifiez les différentes sections de l'interface utilisateur afin de pouvoir les diviser en parties réutilisables. Cela facilitera l'implémentation et assurera une meilleure organisation de votre code.



Au niveau le plus élevé, cette maquette peut être décomposée en deux parties principales :

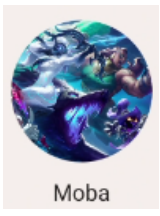
- Le contenu principal de l'écran
- La navigation située en bas de l'écran

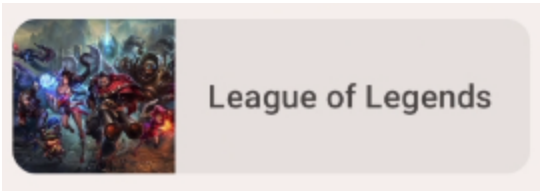


Le contenu principal peut être encore décomposé en plusieurs sous-parties :

- La barre de recherche
- Une section intitulée « Game category »
- Une section intitulée « Favorite collections »

Chaque section contient également des composants supplémentaires qui sont réutilisables :





1. **Barre de recherche**

   En règle générale, un designer fournit davantage d'informations sur la maquette, telles que les dimensions, les marges, etc.

   Dans notre cas, la barre de recherche aura une hauteur de 56.dp et occupera toute la largeur de son conteneur parent. Pour implémenter cette barre de recherche, vous utiliserez un composant Material appelé champ de texte. La bibliothèque Material de Compose propose un composable appelé `TextField`, qui est l'implémentation de ce composant.

   Créez un composable nommé `SearchBar` et n'oubliez pas de créer également sa Preview.

```
@Composable
fun SearchBar(
   modifier: Modifier = Modifier
) {
   TextField(
       value = "",
       onValueChange = {},
       modifier = modifier
   )
}

@Preview(showBackground = true, backgroundColor = 0xFFF5F0EE)
@Composable
fun SearchBarPreview() {
    AppTheme { SearchBar(Modifier.padding(8.dp)) }
}
```

Note : La fonction composable `SearchBar` accepte un paramètre `modifier`, qu'elle transmet à `TextField`. C'est une bonne pratique recommandée par Compose. Cela permet à l'appelant de la méthode de personnaliser l'apparence du composable, ce qui le rend plus flexible et réutilisable. Pensez à appliquer cette pratique à tous les composables dans ce TP.

Si vous examinez l'aperçu du composable, vous remarquerez qu'il manque certaines choses. Commençons par ajuster la taille du composable à l'aide de modificateurs.

Chaque composable que vous utilisez possède un paramètre `modifier` que vous pouvez définir pour ajuster son apparence et son comportement. En définissant le modificateur, vous

pouvez enchaîner plusieurs méthodes de modification pour créer des ajustements plus complexes. Modificateurs dans Compose.

Trouvez les méthodes de modificateur qui permettent de faire en sorte que le composable occupe toute la largeur disponible et d'ajuster sa hauteur minimale à 56dp.

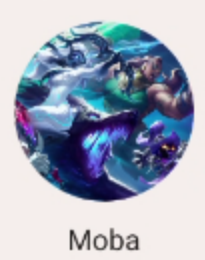
Vous devez également définir certains paramètres de `TextField`. Essayez de faire en sorte que le composable corresponde à la conception en ajustant les valeurs des paramètres.

- Ajoutez l'icône de recherche en utilisant le paramètre `leadingIcon` de `TextField`, qui accepte un autre composable.
- Vous pouvez personnaliser les couleurs du `TextField` en utilisant `TextFieldDefaults.textFieldColors` dans le paramètre `colors`.
- Enfin, ajoutez le texte « Rechercher » comme "placeholder" du composable.

Le composable devrait maintenant ressembler à ceci :

2. Game Category

Le prochain composable à implémenter est l'élément "`GameCategoryElement`".



- La **hauteur de l'image** doit être de **88 dp**.
- L'**espace** entre la ligne de base du texte et l'image doit être de **24 dp**.
- L'**espacement** entre la ligne de base du texte et le bas de l'élément doit être de **8 dp**.
- Le texte doit utiliser le **style typographique** `bodyMedium`.

Pour implémenter ce composable, utilisez un composable `Image` et un composable `Text`. Placez-les dans un composable `Column` afin de les aligner verticalement, l'un sous l'autre.

```
@Composable
fun GameCategoryElement(
    modifier: Modifier = Modifier
) {
      Column(
          modifier = modifier,
      ) {
          Image(
            painter = painterResource(R.drawable.moba_category),
            contentDescription = null,
          )
      Text(text = stringResource(R.string.moba))
      }
}
```

Pensez à créer un aperçu (`@Preview`) de votre composable afin de visualiser le rendu directement dans l'éditeur et vérifier que sa mise en page respecte les spécifications.

On constate que l’image est trop grande et qu’elle n’a pas une forme circulaire. Adaptez le composable en utilisant :

- Le modificateur `size` pour définir une taille appropriée,
- Le modificateur `clip` pour donner à l’image une forme de cercle,
- Le paramètre `contentScale` pour ajuster la manière dont l’image s’adapte à sa taille définie.

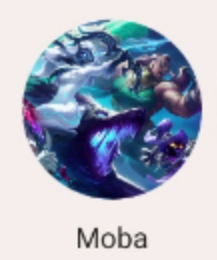
L'étape suivante consiste à aligner le texte horizontalement en configurant l'alignement du composable `Column` à l'aide du paramètre `horizontalAlignment`.

Note : En règle générale, pour aligner des composables à l'intérieur d'un conteneur parent, il est préférable de définir l'alignement au niveau du conteneur. Plutôt que de positionner chaque élément enfant individuellement, vous indiquez au parent comment aligner ses enfants.

Une fois ces éléments implémentés, il ne vous reste plus qu'à effectuer quelques ajustements pour que le composable corresponde parfaitement à la conception. Suivez les étapes suivantes :

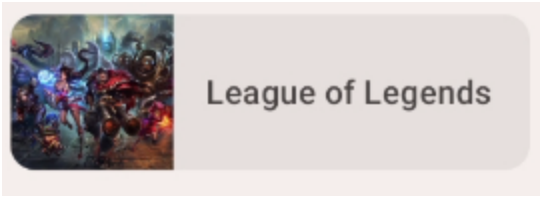
- Rendez l'image et le texte dynamiques en les passant comme arguments à la fonction composable. N'oubliez pas de mettre à jour l'aperçu en transmettant des données codées en dur.
- Mettez à jour le texte pour qu'il utilise le style typographique `bodyMedium`.
- Ajustez les espacements entre le texte et les autres éléments conformément aux dimensions demandées.

Une fois ces étapes terminées, votre composable devrait ressembler à ceci :



3. Favorite Collection Card

Le prochain composable est, d'une certaine manière, similaire à celui utilisé pour une catégorie de jeu.



- La taille réelle du composable est un rectangle de **255 dp de large** et **80 dp de hauteur**.
- Le texte doit utiliser le **style typographique** `titleMedium`.
- Le conteneur utilise la couleur d'arrière-plan **surfaceVariant** pour se différencier du reste de l'écran.

Ces éléments doivent être placés dans un composable `Surface` de Material.

Note : `Surface` est un composant de la bibliothèque Material de Compose. Il peut être configuré pour spécifier sa forme (par exemple, avec des coins arrondis) ainsi que sa couleur de fond.

```
@Composable
fun FavoriteCollectionCard(
      modifier: Modifier = Modifier
) {
      Surface(
          shape = ...,
          color = ...,
          modifier = modifier
      ) {
      Row {
            Image(
                  painter = painterResource(R.drawable.lol_game),
                  contentDescription = null,
            )
            Text(text = stringResource(R.string.lol))
      }
      }
}
```

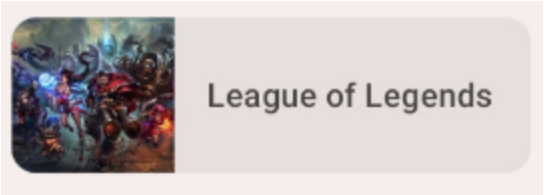
Pensez à créer l'aperçu (`@Preview`) correspondant. Vous remarquerez que les tailles et les alignements ne sont pas encore corrects.

- Définissez la largeur de `Row` et alignez ses éléments enfants verticalement.
- Définissez la taille de l'image à **80 dp** et recadrez-la dans son conteneur.

Pour terminer le composable :

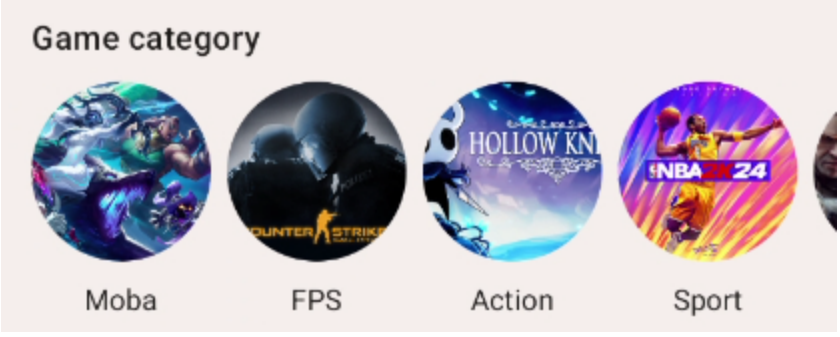
- Rendez l'image et le texte dynamiques en les passant en tant qu'arguments à la fonction composable.
- Remplacez la couleur de fond par **SurfaceVariant**.
- Mettez à jour le texte pour qu'il utilise le style typographique **titleMedium**.
- Ajoutez un espacement entre l'image et le texte.

Vous devriez obtenir un résultat comme celui-ci :



4. Game category line

Maintenant que les composables de base sont créés, vous pouvez commencer à implémenter les différentes sections. Commencez par la section intitulée **"Game category"**.



- Il y a un **espace de 16 dp** avant le premier élément et après le dernier élément de la ligne.
- Il y a un **espacement de 8 dp** entre chaque élément.

Dans Compose, vous pouvez implémenter une ligne défilante à l'aide du composable `LazyRow`. La documentation sur les listes contient des informations détaillées sur les listes différées, telles que `LazyRow` et `LazyColumn`.

Pour ce TP, il suffit de savoir que `LazyRow` ne compose que les éléments visibles à l'écran, et non tous les éléments simultanément. Cela aide à préserver les performances de votre application.

```
@Composable
fun GameCategoriesRow(
      modifier: Modifier = Modifier
) {
      LazyRow(
            modifier = modifier
      ) {
          items(gameCategoriesData) { item ->
            GameCategoryElement(item.drawable, item.text)
          }
      }
}
```

Note : Vous pouvez observer que les éléments enfants d'un `LazyRow` ne sont pas directement des composables. Au lieu de cela, vous utilisez la méthode `items` qui génère des composables sous forme d'éléments de liste. Pour chaque élément de `gameCategoriesData` fourni, vous émettez un composable `GameCategoryElement` que vous avez implémenté précédemment.

Pensez à créer l'aperçu (`@Preview`) correspondant. Vous remarquerez qu'il manque les espacements.

Le composable `LazyRow` accepte un paramètre `horizontalArrangement`. Vous pouvez utiliser la méthode `Arrangement.spacedBy()` pour ajouter un espace fixe entre chaque enfant.

Il reste à ajouter une marge intérieure sur les côtés de `LazyRow`. Ici, il n'est pas possible d'utiliser un simple modificateur de marge. Si vous essayez, vous constaterez que les premier et dernier éléments sont tronqués des deux côtés de l'écran.

`LazyRow` propose un paramètre appelé `contentPadding` pour gérer ce cas.

5. Grille de jeux favoris

La prochaine section à implémenter est la partie **"Favorite collections"** de l'écran. Elle est similaire au composant précédent, à la différence qu'elle s'affiche en grille et non sur une seule ligne.

On pourrait réutiliser une `LazyRow` avec à l'intérieur une `Column` comportant deux instances de `GameCategoryElement`. Cependant, il existe déjà un composable permettant de réaliser cela plus efficacement : il s'agit de `LazyHorizontalGrid`.

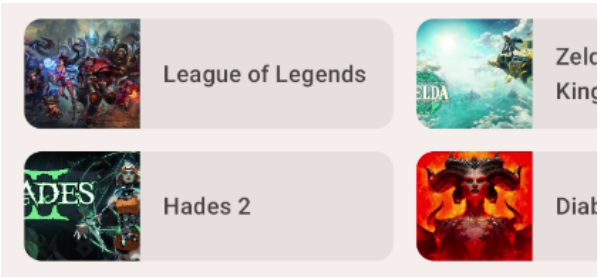
```
fun FavoriteCollectionsGrid(
    modifier: Modifier = Modifier
) {
    LazyHorizontalGrid(
        rows = GridCells.Fixed(2),
        modifier = modifier
    ) {
        items(favoriteCollectionsData) { item ->
            FavoriteCollectionCard(item.drawable, item.text)
        }
    }
}
```

Encore une fois, pensez à utiliser la `Preview`. Vous constaterez que le résultat n'est pas correct. En effet, la grille prend toute la hauteur de son élément parent, ce qui provoque un étirement des fiches des collections préférées.

Adaptez le composable pour :

- Ajouter un **padding de 16 dp** avant le premier élément et après le dernier élément.
- Ajouter un **espace de 16 dp** entre les éléments, à la fois horizontalement et verticalement.
- Définir une **hauteur fixe de 168 dp** pour la grille.
- Limiter la **hauteur du composable `FavoriteCollectionCard` à 80 dp**.

Une fois ceci fait, vous devriez obtenir le résultat suivant :



6. Construction des sections

L'écran d'accueil comporte plusieurs sections organisées selon un schéma similaire. Chaque section possède un **titre** et un contenu qui varie en fonction de la section.

Note : Cette mise en page repose sur le concept des **emplacements**. Il s'agit d'un modèle proposé par Compose permettant de personnaliser les composables en laissant des espaces réservés dans l'interface utilisateur que le développeur peut remplir librement. Ces composables utilisent généralement un paramètre de type lambda `content: @Composable () -> Unit`, qui permet de transmettre un composable à afficher.

```
fun HomeSection(
      modifier: Modifier = Modifier,
) {
      Column(modifier) {
          Text(text = stringResource(...))
      }
}
```

Adaptez le composable `HomeSection` pour :

- Recevoir en paramètres le **titre** et le **contenu** à insérer dans l'emplacement dédié.
- Afficher le composable passé en paramètre sous le titre.
- Utiliser la **typographie** **`titleMedium`** pour le titre.
- Ajouter un **espace de 40 dp** entre la ligne de base du titre et le haut de l'élément.
- Ajouter une **marge intérieure horizontale de 16 dp** au composable.

7. Écran d'accueil

Maintenant que les composants principaux sont créés, il est temps de les combiner pour construire l'interface plein écran.

L'implémentation inclura :

- La barre de recherche en haut.
- Les deux sections affichées l'une en dessous de l'autre.

Pour gérer les espacements entre ces éléments, nous utiliserons le composable `Spacer`, qui permet d'ajouter des espaces dans un `Column`.

```
@Composable
fun HomeScreen(modifier: Modifier = Modifier) {
   Column(modifier) {
       // Spacer
       // SearchBar
       // HomeSection avec GameCategoriesRow
       // HomeSection avec FavoriteCollectionsGrid
       // Spacer
   }
}
```

Bien que cette conception s'adapte à la majorité des tailles d'écran, elle doit pouvoir être scrollée verticalement si l'espace disponible est insuffisant. Pour cela, ajoutez un comportement de scroll à la `Column` parente.

Note : Vous pouvez vérifier le comportement de scroll du composable en limitant la hauteur de l'aperçu et en activant le mode aperçu interactif.

```
@Preview(showBackground = true, backgroundColor = 0xFFF5F0EE, heightDp =
180)
@Composable
fun ScreenContentPreview() {
   AppTheme { HomeScreen() }
}
```

8. Navigation inférieure : Material

Les barres de navigation sont des composants courants dans les applications Android. Il n'est donc pas nécessaire de recréer ce composant depuis zéro. Vous pouvez utiliser le composable `NavigationBar` de la bibliothèque Material de Compose. Ce composable permet d'ajouter un ou plusieurs éléments `NavigationBarItem`.

```
@Composable
private fun GameBottomNavigation(modifier: Modifier = Modifier) {
   NavigationBar(modifier = modifier) {
       NavigationBarItem(
           icon = { },
           label = { },
           selected = true,
           onClick = {}
       )
       NavigationBarItem(
           icon = { },
           label = { },
           selected = false,
           onClick = {}
       )
   }
}
```

Adaptations nécessaires :

- Modifiez la couleur d'arrière-plan de la barre de navigation en utilisant la couleur `surfaceVariant` du thème Material.
- Complétez les composables `NavigationBarItem` en définissant les paramètres `icon` (l'icône) et `label` (le libellé) pour chaque élément de la barre de navigation.

9. Échafaudage (scaffolding)

Dans cette étape, vous allez assembler tous les composables pour créer une version complète de l'application, au moins en mode portrait. Utilisez le composable `Scaffold` de Material Design pour structurer l'interface. `Scaffold` offre des emplacements configurables pour les différentes parties de l'écran, facilitant ainsi l'intégration des éléments tels que la barre de navigation, les sections de contenu et la barre de recherche.

```
@Composable
fun MyGameAppPortrait() {
   AppTheme {
       Scaffold(
           bottomBar = { . . . }
       ) { padding ->
          // Composable qui affiche le reste de l'écran
       }
   }
}
```

Note : La variable `padding` passée en paramètre au contenu du `Scaffold` permet de gérer les marges liées à la barre de statut et à la barre de navigation. Assurez-vous de l'utiliser comme padding dans votre composable, sinon le contenu risque d'être masqué.

10. Rail de navigation - Material

Lors de la création de mises en page, il est important de tenir compte de différentes configurations, y compris le mode paysage. D'après la maquette, la barre de navigation inférieure se transforme en un rail vertical à gauche du contenu. Cela permet de maximiser l'espace dédié au contenu.

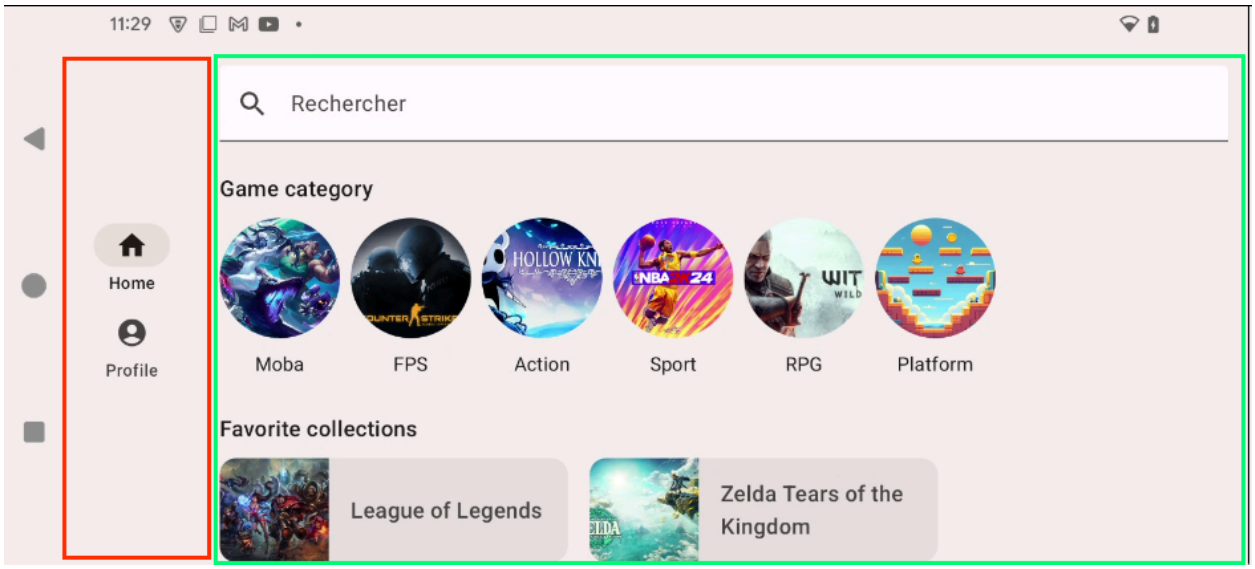
Pour implémenter cette fonctionnalité, vous pouvez utiliser le composable `NavigationRail` de la bibliothèque Compose Material. Son utilisation est similaire à celle du `NavigationBar`. Dans `NavigationRail`, vous ajouterez des éléments `NavigationRailItem` pour définir les options de navigation.

```
@Composable
private fun GameNavigationRail(modifier: Modifier = Modifier) {
   NavigationRail {
       Column() {
           NavigationRailItem(
               icon = { },
               label = { },
               selected = true,
               onClick = {}
           )
           NavigationRailItem(
               icon = { },
               label = { },
               selected = false,
               onClick = {}
           )
       }
   }
}
```

Pour adapter le `NavigationRail` :

- Ajoutez une marge intérieure de 8 dp au début et à la fin du rail.
- Modifiez la couleur d'arrière-plan du rail en utilisant la couleur `surfaceVariant` du thème Material.
- Complétez les composables `NavigationRailItem` en définissant les paramètres `icon` (l'icône) et `label` (le libellé) pour chaque élément.
- La colonne doit occuper toute la hauteur disponible, en centrant les éléments verticalement et horizontalement.
- Ajoutez un espace de 8 dp entre les éléments `NavigationRailItem`.

Dans la version paysage, vous allez ajouter le `NavigationRail` :



- Contrairement au mode portrait où vous avez utilisé `Scaffold`, en mode paysage, `Scaffold` n'est pas adapté.
- Utilisez une `Row` pour organiser les éléments, en plaçant le `NavigationRail` à gauche et le contenu de l'écran dans l'espace restant.
- Contrairement à un `Scaffold` qui définit automatiquement la couleur d'arrière-plan sur la couleur `background`, ici vous devez ajouter un composable `Surface` et définir manuellement la couleur d'arrière-plan.

```
@Composable
fun MyGameAppLandscape() {
   AppTheme {
       Surface {
           Row {
               // Rail de navigation
               // Composable qui affiche le reste de l'écran
           }
       }
   }
}
```

11. Taille de fenêtre

Si l'application est exécutée en mode paysage sur un smartphone, la version paysage ne s'affiche pas avec la configuration actuelle. Vous devez indiquer à l'application quand afficher quelle configuration, en fonction de l'orientation de l'écran.

Grâce à la bibliothèque ajoutée en début de TP, vous pouvez utiliser la fonction `calculateWindowSizeClass()` pour déterminer la configuration actuelle du téléphone.

Note : Il y a trois catégories de tailles de fenêtre : compacte, moyenne et grande. En mode portrait, l'application utilise une largeur compacte, tandis qu'en mode paysage, elle passe à une largeur grande.

```
@Composable
fun MyGameApp(windowSize: WindowSizeClass) {
   when (windowSize.widthSizeClass) {
       WindowWidthSizeClass.Compact -> {
           // Composable portrait
       }
       WindowWidthSizeClass.Expanded -> {
           // Composable paysage
       }
   }
}
```

Il ne vous reste plus qu'à utiliser le composable `MyGameApp` dans la méthode `setContent`.