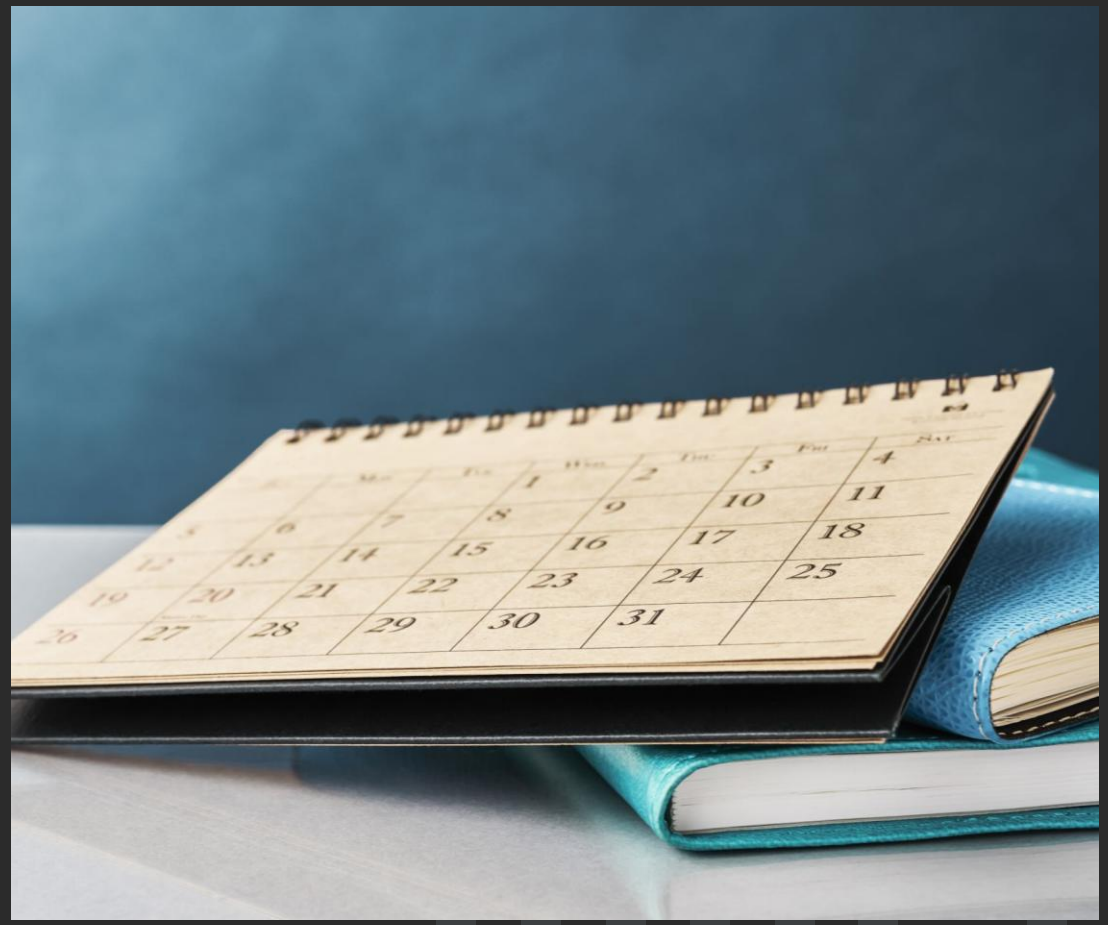
COLLÈGE DE
BOIS-DE-BOULOGNE

420-5GM-BB -Cours #6 Scaffold

Pierre Prades et Mathieu Brodeur-Béliveau

Agenda de la séance

- Présences
- Terminer et corriger les exercices de la séance précédente
- Scaffold
- TopAppBar avec menu



Qu'est-ce qu'un Scaffold ?

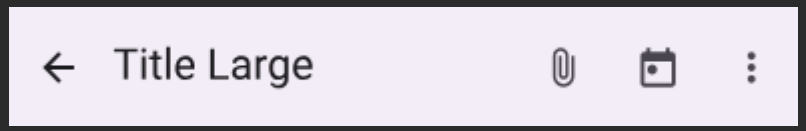
- Scaffold est un composant qui fournit une structure de base pour une application Jetpack Compose en suivant les directives Material Design.
- Il organise les principaux éléments d'interface :
  - TopAppBar → barre d'app en haut.
  - BottomBar → barre de navigation en bas (optionnel).
  - FloatingActionButton (FAB) → bouton flottant.
  - Drawer → menu latéral (optionnel).
  - Content → zone principale.
- 👉 Ça permet d'uniformiser l'apparence de l'app.

Scaffold

```
@Composable
fun Scaffold(
    modifier: Modifier = Modifier,
    topBar: @Composable () -> Unit = {},
    bottomBar: @Composable () -> Unit = {},
    snackbarHost: @Composable () -> Unit = {},
    floatingActionButton: @Composable () -> Unit = {},
    floatingActionButtonPosition: FabPosition = FabPosition.End,
    containerColor: Color = MaterialTheme.colorScheme.background,
    contentColor: Color = contentColorFor(containerColor),
    contentWindowInsets: WindowInsets = ScaffoldDefaults.contentWindowInsets,
    content: @Composable (PaddingValues) -> Unit
)
```

TopAppBar

- De manière générale le TopAppBar contient un titre (nom de l'application)
- On ajoute aussi des boutons et/ou menus:



TopAppBar – code source

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ScaffoldExemple() {
    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Mon Application") }
            )
        }
    ) { padding ->
        // Contenu principal
        Box(
            modifier = Modifier
                .fillMaxSize()
                .padding(padding),
            contentAlignment = Alignment.Center
        ) {
            Text("Bonjour, ceci est le contenu principal !")
        }
    }
}
```

Ajouter un menu dans la TopAppBar

- On peut ajouter des actions (icônes ou menus déroulants) dans la barre du haut.
- Exemple : un menu avec 2 options.
- Expanded est une variable observable:
  - var expanded by remember { *mutableStateOf*(false) }
- Introduction de l'événement onDismissRequest

```
TopAppBar(
    title = { Text("App avec menu") },
    actions = {
        IconButton(onClick = { expanded = true }) {
            Icon(Icons.Default.MoreVert, contentDescription = "Menu")
        }
        DropdownMenu(
            expanded = expanded,
            onDismissRequest = { expanded = false }
        ) {
            DropdownMenuItem(
                text = { Text("Option 1") },
                onClick = { expanded = false }
            )
            DropdownMenuItem(
                text = { Text("Option 2") },
                onClick = { expanded = false }
            )
        }
    }
)
```

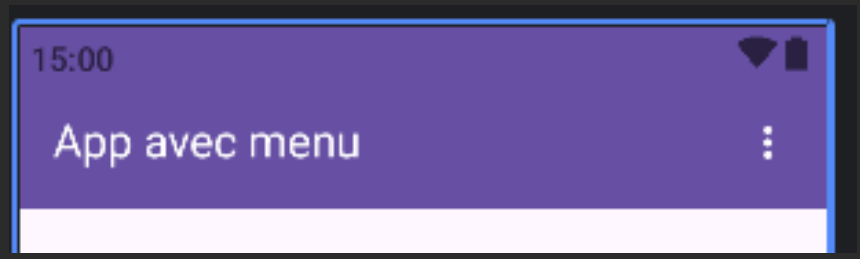
Explications

- L'icône IconButton sert à ouvrir le menu.
- Le menu, DropdownMenu, est composé de plusieurs DropdownMenuItem.
- Chaque élément du menu utilise l'événement onClick pour gérer le clic.

Ajouter de la couleur

- Modifier le paramètre color de TopAppBar:

```
,
colors = TopAppBarDefaults.topAppBarColors(
    containerColor = MaterialTheme.colorScheme.primary,
    titleContentColor = MaterialTheme.colorScheme.onPrimary,
    actionIconContentColor = MaterialTheme.colorScheme.onPrimary
)
```



Exercice

- Créez une application avec un Scaffold qui contient :
- Une TopAppBar avec un titre.
- Un bouton de menu (MoreVert) qui affiche un DropdownMenu avec :
  - "Paramètres"
  - "À propos"
  - "Quitter"
- Dans le **contenu principal**, affichez le texte "Zone principale de l'app".
- Lorsqu'on clique sur une option du menu, le texte est affiché dans la console.

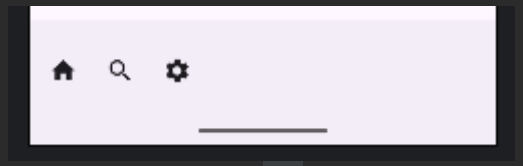
Points à retenir

- Scaffold = squelette de l'application.
- TopAppBar = titre + actions (menu, boutons).
- DropdownMenu = menu déroulant lié à un bouton d'action.
- Utiliser padding(padding) pour éviter que le contenu ne chevauche la barre du haut.

BottomAppBar

- Un autre élément important est le BottomAppBar, celui-ci est situé en bas de l'écran.

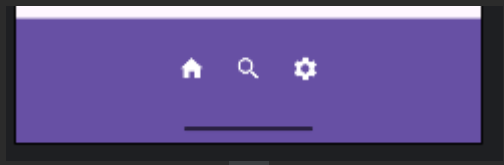
```
,
bottomBar = {
    BottomAppBar {
        IconButton(onClick = { /*Faire quelque chose*/ }) {
            Icon(Icons.Default.Home, contentDescription = "Accueil")
        }
        IconButton(onClick = { /*Faire quelque chose*/ }) {
            Icon(Icons.Default.Search, contentDescription = "Recherche")
        }
        IconButton(onClick = { /*Faire quelque chose*/ }) {
            Icon(Icons.Default.Settings, contentDescription = "Paramètres")
        }
    }
}
```



Ajouter de la couleur et modifier l'alignement

- On ajoute des paramètres pour la couleur et on modifie le contenu pour changer l'alignement:

```
,
bottomBar = {
    BottomAppBar(containerColor = MaterialTheme.colorScheme.primary,
        contentColor = MaterialTheme.colorScheme.onPrimary) {
        Row(modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.Center) {
            IconButton(onClick = { /*Faire quelque chose*/ }) {
                Icon(Icons.Default.Home, contentDescription = "Accueil")
            }
            IconButton(onClick = { /*Faire quelque chose*/ }) {
                Icon(Icons.Default.Search, contentDescription = "Recherche")
            }
            IconButton(onClick = { /*Faire quelque chose*/ }) {
                Icon(Icons.Default.Settings, contentDescription = "Paramètres")
            }
        }
    }
}
```



Hisser un état

- But: Le hissage d'état permet :
  - la réutilisation des composables en les rendant « stateless »
  - Le partage d'état entre plusieurs composables
  - Une meilleure organisation du code en ayant une gestion d'états centralisée.

- Mise en place :
  - Le Composable ne garde plus l'état en interne, mais en reçoit la valeur et une fonction pour le mettre à jour.

Exemple sans hissage : Mauvaise pratique

```
@Composable
fun Compteur() {
    var compte by remember { mutableStateOf(0) }

    Button(onClick = { compte++ }) {
        Text("Compteur : $compte")
    }
}
```

- Ici, le Composable gère lui-même son état.
- Résultat : impossible de réinitialiser ce compteur depuis l'extérieur, ni de partager compte.

Exemple avec hissage (bonne pratique)

```
@Composable
fun Compteur(
    compte: Int,
    onChangeCompteur: (Int) -> Unit
) {
    Button(onClick = { onChangeCompteur(compte+1) }) {
        Text("Compteur : $compte")
    }
}
```

- Ici, le Composable Compteur n'a plus de logique d'état interne :
  - Il reçoit compte (valeur).
  - Il appelle onChangeCompteur (mise à jour de la valeur).

Passage d'état

```
@Composable
fun CompteurApp() {
    var compte by remember { mutableStateOf(0) }

    Column(horizontalAlignment = Alignment.CenterHorizontally) {
        Compteur(compte = compte, onChangeCompteur = { compte = it })

        Button(onClick = { compte = 0 }) {
            Text("Réinitialiser")
        }
    }
}
```

- L'état (compte) est géré par CompteurApp, pas par Compteur.
- Compteur devient réutilisable

Règles générales du hissage d'état

- Toujours hisser l'état quand :
  - Le Composable doit être réutilisable.
  - Plusieurs Composables ont besoin d'accéder à la même donnée.
  - L'état doit survivre plus longtemps que le Composable qui l'utilise.
- Un Composable UI pur devrait être stateless.
- L'état est géré plus haut dans la hiérarchie, souvent dans un ViewModel en MVVM.

Exercice

- Modifier l'application jeu de dé:
  - en utilisant des composables stateless.
  - Ajouter un TopAppBar avec une icone qui permet de lancer une nouvelle partie