Applications Mobiles

420-5GM-BB - Cours 8

Pierre Prades & Mathieu Brodeur-Béliveau

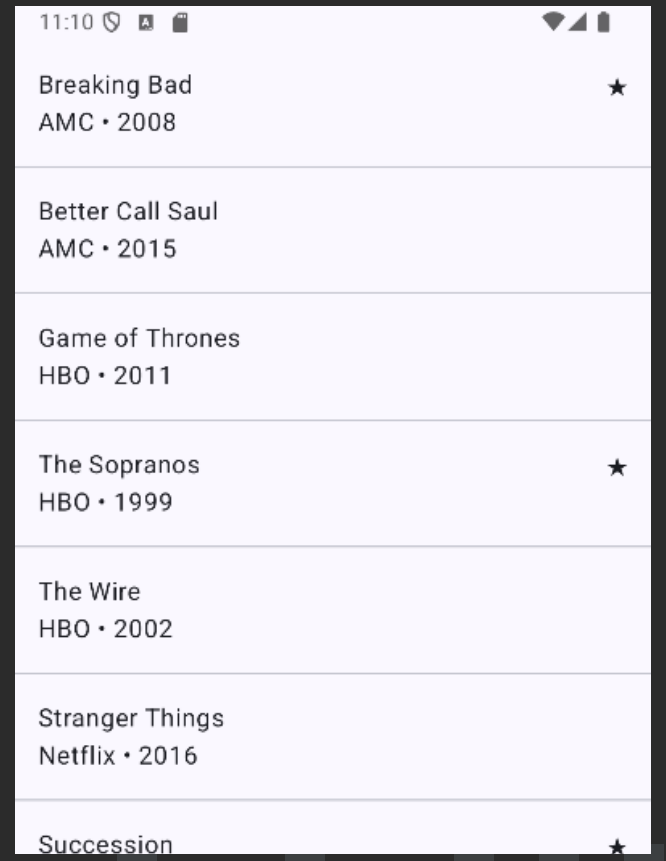
COLLÈGE DE
BOIS-DE-BOULOGNE

Agenda de la séance

- Présences
- LazyList
  - Items
  - Clickable
- Form
  - OutlinedTextField
  - Checkbox
- Exercice

LazyList – Description

- Liste défilante optimisée : juste les éléments à l'écran!
- Utilise le composant LazyList ou LazyColumn
  - Quel est la différence selon vous?
- Chaque élément va être dans un bloc
  - *item* { } pour présenter 1 élément
  - *items*(items = shows) { } pour présenter une liste



LazyList – Début exercice

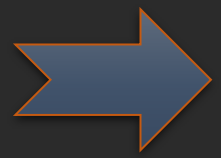
```
data class TvShow(
    val id: Int,
    val title: String,
    val year: Int,
    val network: String,
    val isFavorite: Boolean
)
```

```
val sampleShows = listOf(
    TvShow(1, "Breaking Bad", 2008, "AMC", true),
    TvShow(2, "Better Call Saul", 2015, "AMC", false),
    TvShow(3, "Game of Thrones", 2011, "HBO", false),
    TvShow(4, "The Sopranos", 1999, "HBO", true),
    TvShow(5, "The Wire", 2002, "HBO", false),
    TvShow(6, "Stranger Things", 2016, "Netflix", false),
    TvShow(7, "Succession", 2018, "HBO", true),
    TvShow(8, "Mad Men", 2007, "AMC", false),
    TvShow(9, "Dark", 2017, "Netflix", false),
    TvShow(10, "Mindhunter", 2017, "Netflix", true),
    TvShow(11, "The Crown", 2016, "Netflix", false),
    TvShow(12, "True Detective", 2014, "HBO", false)
)
```

LazyList – Suite exercice

- Pour présenter cette information :

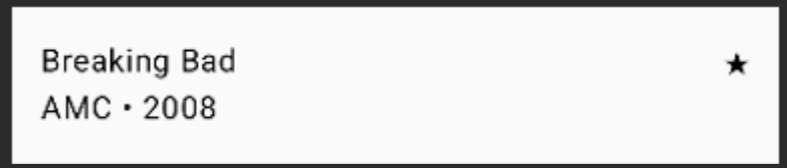
```
LazyColumn {
    items(items = shows) { show ->
        Column {
            Text(show.title)
            Text("${show.network} • ${show.year}")
        }
        if (show.isFavorite) Text("★")
        Divider()
    }
}
```



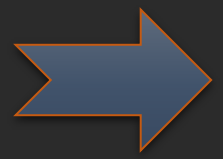
LazyList – Items

- Syntaxe :
  - *items*(items) { element -> ... }
- Chaque élément parcouru et affiché
  - Similaire à un for each
- Dans notre cas
  - items est notre liste de show
  - element est 1 show de notre liste
- Exercice
  - Affiché l'étoile à droite de l'écran

```
LazyColumn {
   items(items = shows) { show ->
      Column {
         Text(show.title)
         Text("${show.network} • ${show.year}")
      }
      if (show.isFavorite) Text("★")
      Divider()
   }
}
```



```
LazyColumn {
    items(items = shows) { show ->
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            Column {
                Text(show.title)
                Text("${show.network} • ${show.year}")
            }
            if (show.isFavorite) Text("★")
        }
        Divider()
    }
}
```



Breaking Bad ★
AMC • 2008

Better Call Saul
AMC • 2015

Game of Thrones
HBO • 2011

The Sopranos ★
HBO • 1999

The Wire
HBO • 2002

LazyList – Ajout clickable

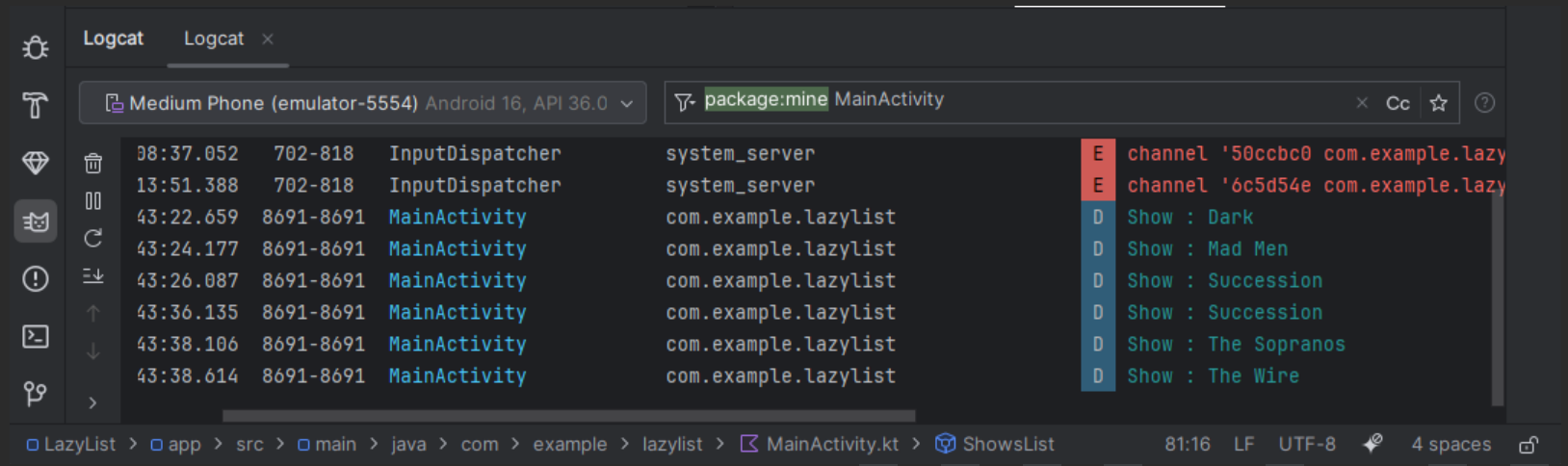
- J'aimerais être capable d'intéragir avec les éléments de ma liste
  - Ajout de la propriété modifier.*clickable* { }

- Je peux ajouter cette propritété n'importe ou
  - Dans notre cas, sur chaque élément de notre liste

```
modifier = Modifier
    .clickable {  },
```

- Exercice
  - Ajouter que lorsqu'on clic sur un élément de la liste
  - On Log son titre

```
...
Row(
    modifier = Modifier
        .fillMaxWidth()
        .padding(16.dp)
        .clickable { Log.d(TAG, "Show : ${show.title}") },
    horizontalArrangement = Arrangement.SpaceBetween
)
...
```



LazyList – Modifier information

- J'aimerais pouvoir modifier l'information de ma liste
- Pour se faire, je vais me créer un nouveau composable
  - Avec des champs de saisie et case à cocher

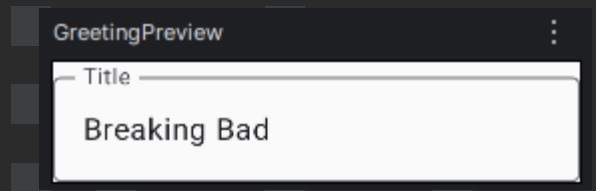
```
@Composable
fun EditShowScreen(show: TvShow) {  }
```

- Comment faire un champ de saisie?
  - Avec l'élément Compose TextField ou OutlineTextField
- Comment faire une case à cocher?
  - Avec l'élément Compose Checkbox

OutlinedTextField

- Champ de saisie avec un beau contour
- Paramètres principaux :
  - value → la valeur affiché
  - onValueChange → appelé après modification
  - label → titre de notre champ

```
OutlinedTextField(
    value = title,
    onValueChange = { title = it },
    label = { Text("Title") }
)
```



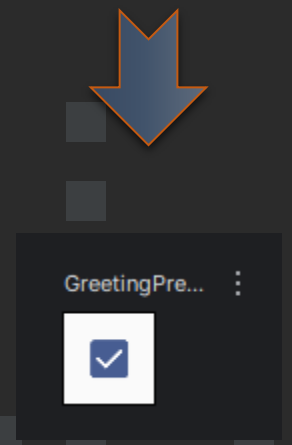
- Doit être relié à un état!

```
var title by remember { mutableStateOf(show.title) }
```

Checkbox

- Boîte à cocher qui suit MaterialDesign
- Paramètre principaux :
  - checked → état actuel (true/false)
  - onCheckChange → appelé après modification

```
Checkbox(
    checked = isFav,
    onCheckedChange = { isFav = it }
)
```

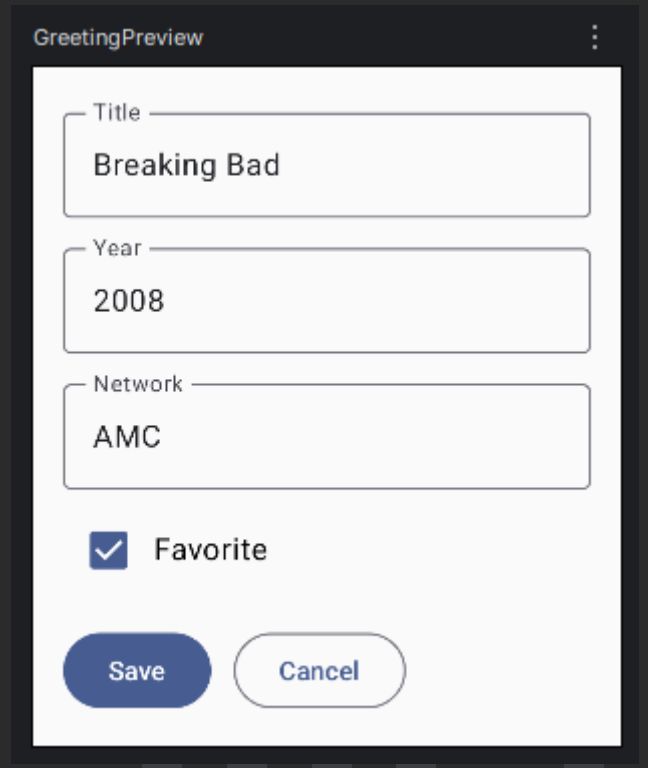


- Doit être relié à un état!

```
var isFav by remember { mutableStateOf(show.isFavorite) }
```

Exercice – Création d'un form

- Utilisé ce que vous venez d'apprendre
- Pour répliquer ce UI



- Chaque élément devrait être lié à un état
  - Devrait contenir la valeur d'un film

```
var title by remember { mutableStateOf(show.title) }
var year by remember { mutableStateOf(show.year.toString()) }
var network by remember { mutableStateOf(show.network) }
var isFav by remember { mutableStateOf(show.isFavorite) }

Column(Modifier.padding(16.dp)) {
    OutlinedTextField(value = title, onValueChange = { title = it }, label = { Text("Title") })
    Spacer(Modifier.height(8.dp))
    OutlinedTextField(value = year, onValueChange = { year = it }, label = { Text("Year") })
    Spacer(Modifier.height(8.dp))
    OutlinedTextField(value = network, onValueChange = { network = it }, label = { Text("Network") })
    Spacer(Modifier.height(8.dp))
    Row(verticalAlignment = Alignment.CenterVertically) {
        Checkbox(checked = isFav, onCheckedChange = { isFav = it })
        Text("Favorite")
    }
    Spacer(Modifier.height(16.dp))
    Row(horizontalArrangement = Arrangement.spacedBy(12.dp)) {
        Button(onClick = {}) { Text("Save") }
        OutlinedButton(onClick = {}) { Text("Cancel") }
    }
}
```

Changer l'affichage

- Pour être capable de passer d'un affichage à un autre
- Je vais me créer un nouveau Composable et un enum

```
enum class Screen { LIST, EDIT }
@Composable
fun Application() {
    var screen by remember { mutableStateOf(Screen.LIST) }
    var selectedId by remember { mutableIntStateOf(0) }

    when (screen) {
        Screen.LIST -> ShowsList(sampleShows)
        Screen.EDIT -> EditShowScreen(sampleShows[selectedId])
    }
}
```

Changer l'affichage

- Je veux que lorsque je clique sur un élément de ma liste
  - J'affiche mon form avec les valeur de cet élément
  - Pour faire cela, je vais hisser une méthode pour modifier selectedId

```
...
Screen.LIST -> {
    ShowsList(
        sampleShows,
        onShowClick = {
            selectedId = it
            screen = Screen.EDIT
        }
    )
}
...
```

```
fun ShowsList(
    shows: List<TvShow>,
    onShowClick: (Int) -> Unit
) ...
```

```
...
Row(
    modifier = Modifier
        .fillMaxWidth()
        .padding(16.dp)
        .clickable { onShowClick(show.id) },
...
```

Exercice – Changer l'affichage

- Faîtes la même chose
- Mais pour passer de Edit à List
  - Quand on clic sur le bouton Cancel

- Ensuite, créer vous un état de la liste de show pour pouvoir le modifier

```
val shows = remember { mutableStateListOf<TvShow>().apply { addAll(sampleShows) } }
```

  - Utilisé le pour le bouton onSave
  - Astuce : onSave devrait ressembler à : `onSave: (TvShow) -> Unit`
    - La mise à jour de shows se fait dans Application()

Exercice - Liste de contacts

- Vous devez implémenter une application simple :
  - Afficher une liste de contacts (nom, prénom, téléphone, image).
  - Chaque item contient un bouton « Modifier » permettant d'éditer le contact **dans la même activité**.
  - L'état des contacts (nom, prénom, téléphone, image) doit être sauvegardé dans un mutableState
- Modèle de données:

```
data class Contact(
    val id: Int,
    var prenom: String,
    var nom: String,
    var telephone: String,
    val photoId: Int
)
```

Exercice - Liste de contacts

- Votre UI devrait ressembler à ceci: