

Prog mobile

Rappel

Projet à rendre pour début mai

Livrables attendus :

- **Documentation succincte** (README)
- **Petite démonstration**
- **Quelques questions sur certaines parties du code**
 - Exemple de question : *Si vous voulez ajouter une nouvelle notification, comment feriez-vous ?*

Contrôle écrit prévu pour mai

Points importants à maîtriser

1. Les Fragments

- Bien comprendre leur fonctionnement et leur cycle de vie.
- Savoir comment les intégrer dans une application Android.

2. Layouts dans Android Studio

- **Par défaut**, Android Studio utilise `ConstraintLayout`.
- On peut modifier `ConstraintLayout` et le remplacer par `LinearLayout`.

LinearLayout

- Toujours spécifier l'**orientation** (`vertical` ou `horizontal`).
- **wrap_content** → prend la largeur minimale nécessaire pour afficher le texte.
- **match_parent** → prend toute la largeur disponible.
- Pour les tailles de texte, **privilégier les unités sp**.
- L'id d'un élément est **nécessaire** si on veut modifier son design avec Kotlin.

Propriétés importantes :

- `width = match_parent`
- `height = wrap_content`
- `orientation = horizontal`
- `gravity = center` → contrôle l'alignement du contenu interne.

- `layout_gravity` → contrôle l'alignement du layout dans son parent.
- `layout_weight` → définit l'espace relatif occupé par un élément.
 - Exemple : si un bouton doit prendre **deux fois plus d'espace** qu'un autre, il faut mettre `layout_weight = 2`.

ConstraintLayout

- Nécessite **quatre contraintes** pour une bonne mise en page.
- Une vue **non contrainte** sera indiquée par **quatre petits ronds blancs autour du texte**.



-
- `0dp` en largeur ou hauteur signifie « s'adapte à la contrainte ».

3. RecyclerView

RecyclerView

Layout manager

- RecyclerView library provides three layout managers:
 - LinearLayoutManager
 - GridLayoutManager
 - StaggeredGridLayoutManager

4

- **Contrairement à ListView**, RecyclerView n'affiche que les vues visibles à l'écran et recycle les anciennes vues pour économiser les ressources.

- ViewHolder définit la vue à afficher.
- Les données peuvent être **stockées dans un repository**, une **base de données distante**, etc.
- Une **vue** est constituée d'une **image**, d'un **texte** et d'**autres données associées**.
- Pour créer un RecyclerView, il faut définir un **layout ressource file** dans /res/layout.

Étapes pour implémenter un RecyclerView :

1. Créer une ressource de mise en page (**layout_resource_file**)
2. Définir une classe **ViewHolder** pour gérer la vue d'un élément.
3. Créer un **RecyclerView.Adapter** pour connecter les données à l'interface.
4. Ajouter un **RecyclerView** dans l'Activity ou le Fragment.

Exemple : Création de `ExampleViewHolder.kt`

```
class ExampleViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    // Définir ici les vues à manipuler
}
```

Création de `ExampleViewAdapter.kt`

Cette classe dérive de `RecyclerView.Adapter` et implémente trois méthodes essentielles :

```
class ExampleViewAdapter : RecyclerView.Adapter<ExampleViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ExampleViewHolder {
        val v = LayoutInflater.from(parent.context).inflate(R.layout.item_example, parent, false)
        return ExampleViewHolder(v)
    }

    override fun onBindViewHolder(holder: ExampleViewHolder, position: Int) {
        // Ici, on lie les données à la vue pour une position donnée
    }

    override fun getItemCount(): Int {
        return 50
    }
}
```