Checkpoint 2 – Kotlin

Gabriel Reis Baron – RM93266

João Pedro Moura Tuneli – RM93530

Enzo Obayashi – RM95634

Abner Aragon – RM95620

Gabirel Paterra – RM93688

Enzo Mansi – RM 92955

Conceito das unidades de medidas DP e SP no código:

- DP : ou Density-independent Pixels, essa unidade é baseada na densidade de pixels da tela. Por exemplo, se a densidade da tela do usuário é de 160 dpi significa que 1dp representa 1 pixel em um total de 160. Por exemplo temos um ícone de 48 dp, ele terá 48 pixels de largura de uma tela de 160 dpi, mas em uma tela de 380 dpi esse mesmo ícone terá 96 pixels de largura. É aconselhável a utilização dessa unidade para dimensionamento de Views e objetos na tela.
- SP: ou Scale-independent Pixels, essa unidade é semelhante ao dp, mas é escalonada pelo tamanho de fonte preferida pelo usuário. Ela é utilizada principalmente para tamanhos de fonte. Se o usuário ter no seu dispositivo fontes grandes como preferencia o sp ira escalonar as fontes como grande de acordo com a preferência.

Explicação do RecycleView:

Muitos aplicativos precisam exibir coleções do mesmo tipo (como mensagens, contatos, imagens ou músicas); geralmente, essa coleção é muito grande para caber na tela, portanto, a coleção é apresentada em uma janela pequena que pode rolar suavemente por todos os itens da coleção. RecyclerView é um widget

do Android que exibe uma coleção de itens em uma lista ou em uma grade, permitindo que o usuário role pela coleção.

Porque ele é mais utilizado hoje em dia que o listview e o gridview:

O RecyclerView contem os seguintes benefícios:

Reciclagem de visualizações: O nome Recyclerview é auto sugestivo, ele recicla as visualizações. Quando um item sai da tela, o Recyclerview não descarta a visualização. Ele reutiliza a visualização para novos itens que estão entrando na tela. Isso melhora bastante o desempenho , especialmente para listas e grades grandes e complexas.

Desempenho: Ele foi projetado para lidar com grandes conjuntos de dados que podem ser rolados na tela,

Flexibilidade no Layout: Ele suporta listas verticais e horizontais, grades e até mesmo layouts personalizados, graças ao sistema de LayoutManager. Enquanto o ListView suporta apenas listas verticais, e o GridView suporta apenas grades.

Animações de Itens: O RecyclerView tem suporte embutido para animações quando você adiciona, remove ou move itens. Isso pode ser complexo de implementar no ListView e no GridView.

ViewHolder: O padrão ViewHolder é uma prática recomendada para o ListView, mas é opcional. No RecyclerView, é obrigatório. Isso força boas práticas de programação e melhora o desempenho.

Analise do código ItemsAdapter:

```
class ItemsAdapter : RecyclerView.Adapter<ItemsAdapter.ItemViewHolder>() {

    private val items = mutableListOf<ItemModel>()

    fun addItem(newItem: ItemModel) {
        if (newItem.name.isNotEmpty()) {
            items.add(newItem)
            notifyDataSetChanged()
        } else {
            println("O nome do item não pode ser nulo ou vazio!")
        }
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.item_layout, parent, attachToRoot: false)
        return ItemViewHolder(view)
    }

    override fun getItemCount(): Int = items.size

    override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
        val item = items[position]
        holder.bind(item)
    }
    class ItemViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val textView = view.findViewById<TextView>(R.id.textViewItem)
        fun bind(item: ItemModel) {
            textView.text = item.name
        }
    }

}
```

class ItemsAdapter : RecyclerView.Adapter<ItemsAdapter.ItemViewHolder>() {}

Essa linha está definindo uma nova classe chamada ItemsAdapter que estende o RecyclerView.Adapter e tem como parâmetro o ItemViewHolder que é uma classe interna.

private val items = mutableListOf<ItemModel>()

Essa linha cria uma lista mutável privada chamada “items” que armazena os objetos do tipo ItemModel que serão exibidos no RecyclerView

```
fun addItem(newItem: ItemModel) {
  if (newItem.name.isNotEmpty()) {
    items.add(newItem)
    notifyDataSetChanged()
  } else {
    println("O nome do item não pode ser nulo ou vazio!")
```

```
    }
}
```

A função “addItem” adiciona um novo item a lista ‘items’, Ele verifica se o nome do item não está vazio antes de adicioná-lo na lista. Se o nome estiver vazio, ele imprime uma mensagem de erro.

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
    val view = LayoutInflater.from(parent.context).inflate(R.layout.item_layout, parent, false)
    return ItemViewHolder(view)
  }
```

A função “onCreateViewHolder” é chamada quando o Recycler view precisa de um novo ViewHolder, ele infla (processo de transformar um layout XML, definido em um arquivo, em uma hierarquia de objetos View na memória que pode ser manipulada pelo código Kotlin ou Java) o layout do item e retorna um novo ItemViewHolder.

```
val view = LayoutInflater.from(parent.context).inflate(R.layout.item_layout, parent, false)
```

Essa linha de código está criando uma nova View a partir de um layout XML (R.layout.item_layout). Ela usa o LayoutInflater para ler o arquivo XML e criar os objetos View correspondentes na memória.

```
override fun getItemCount(): Int = items.size
```

A função getItemCount retorna o número de itens na lista “Items” tendo a função de contador.

```
override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
```

```
    val item = items[position]
    holder.bind(item)
  }
```

A função “onBindViewHolder” é chamada pelo RecyclerView para exibir dados na posição especificada da lista. Ela recupera o item da lista “Items” na posição dada e chama a função ‘bind’ do ViewHolder.

```
class ItemViewHolder(view: View) : RecyclerView.ViewHolder(view) {
    val textView = view.findViewById<TextView>(R.id.textViewItem)
    fun bind(item: ItemModel) {
      textView.text = item.name
    }
  }
```

A classe “ItemViewHolder” é um ViewHolder personalizado para exibir os itens do RecyclerView, ele herda o RecyclerView.ViewHolder.

Ela tem uma referência para a TextView no layout do item e uma função 'bind' para vincular os dados do item à TextView.