Compose Comanders

Liderando a Revolução UI em Android com Kotlin

W. Galbarini

Compose Comanders

Introdução ao Jetpack Compose

Jetpack Compose é uma biblioteca moderna de UI do Android, que usa Kotlin para revolucionar a forma como criamos interfaces de usuário. Com ele, você escreve menos código em comparação com abordagens antigas, como XML, e obtém resultados visuais dinâmicos e reativos.

O Jetpack Compose simplifica a construção de UIs com Kotlin, permitindo que você descreva a interface de forma declarativa. Você define o que sua UI deve fazer e o Compose cuida do resto.

01

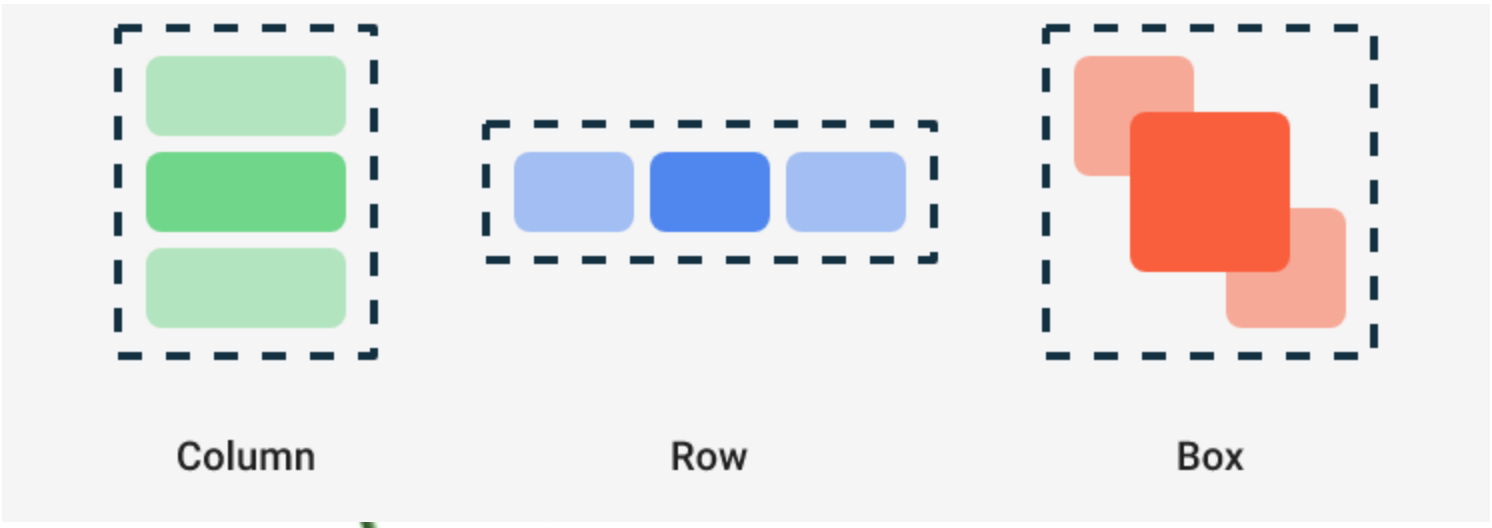
Layouts

A hierarquia da interface no Jetpack Compose é baseada em contenção, o que significa que os componentes são organizados em uma estrutura hierárquica de pais e filhos. Isso permite criar layouts complexos e responsivos, onde elementos pais contêm elementos filhos, que por sua vez podem conter outros elementos filhos. Essa abordagem é semelhante à organização de uma árvore, onde o elemento raiz é o pai de todos os outros elementos

Layouts no Compose

Layouts Padrão

- Column(): Organiza os componentes verticalmente, um abaixo do outro.
- Row(): Organiza os componentes horizontalmente, lado a lado.
- Box(): Permite sobrepor os componentes, um em cima do outro.



Esses elementos de layout são fundamentais no Jetpack Compose para criar interfaces de usuário com diferentes estruturas e disposições. Ao combiná-los adequadamente, é possível criar layouts complexos e responsivos que se adaptam às necessidades de design e funcionalidade do aplicativo Android.

02

Gerenciamento de estados

Gerenciar o estado de um aplicativo é crucial para criar uma experiência de usuário responsiva e interativa.

Gerenciamento de Estados

Estado

É qualquer valor que pode mudar ao longo do tempo, como o texto em um campo de entrada, se um botão está ativo ou não, ou se um diálogo está visível. No contexto de uma interface de usuário, o estado determina a aparência e o comportamento dos componentes na tela.

No Jetpack Compose, o estado é geralmente mantido em variáveis especiais chamadas de State, que são observáveis, o que significa que o Compose pode detectar quando os valores mudam. Quando um valor de estado muda, o Compose automaticamente reconstróimos componentes afetados pela mudança,.

```
var nome by remember { mutableStateOf("") }
```

remember é usado para lembrar o estado durante as recomposições, e mutableStateOf cria um objeto de estado mutável..

03

Composables

Os Composables são os blocos de construção fundamentais do Jetpack Compose. Eles representam componentes de UI reativos e modulares, capazes de se adaptar dinamicamente a alterações de estado e propriedades. São compostos hierarquicamente, permitindo a criação de interfaces complexas a partir de elementos simples e reutilizáveis

Funções Composable

Características dos Composables

A anotação @Composable é fundamental no Jetpack Compose, marcando uma função para indicar que ela define um componente da interface do usuário. Esta anotação sinaliza ao framework que a função em questão é encarregada de renderizar um elemento visual, podendo ser integrada com outros composables para constituir a interface completa.

. Um dos principais benefícios do uso de funções composables é a capacidade de chamar outras funções composables internamente, facilitando a criação de uma arquitetura hierárquica.

Permite que desenvolvedores construam interfaces complexas a partir de componentes modulares e reutilizáveis, otimizando tanto a organização do código quanto a manutenção dele.

Funções Composable

Conceito de Modifier

Um modifier é utilizado para alterar a aparência, o layout ou o comportamento de um componente composable, permitindo encadear várias alterações de forma concisa e legível. Alguns exemplos importantes de modifiers incluem:

- padding: Define o espaçamento interno e externo do componente.
- size: Define as dimensões do componente.
- width: Define a largura do componente.
- height: Define a altura do componente.
- background: Define o plano de fundo do componente.
- border: Adiciona uma borda ao redor do componente.

- clip: Define como o conteúdo do componente deve ser cortado ou mascarado.
- clickable: Define se o componente pode ser clicado ou não.
- scrollable: Define se o componente é rolável ou não.

3.1

TEXT( )

A função composable Text() é uma das funções mais básicas e essenciais. Ela é utilizada para exibir texto a tela de usuário. Alguns parâmetros importantes para essa função são: text, fontSize, fontStyle, color, textAlign, maxLines e modifier.

Composable Text( )

text.kt

```kotlin
@Composable
fun Saudacao(nome: String) {
    Text(
        text = nome,
        fontSize = 18.sp,
        color = Color.Blue,
        modifier = Modifier
            .background(Color.LightGray)
            .padding(16.dp)
    )
}
```

O exemplo acima, é um código de função simples que recebe um parâmetro e usa-os para renderizar um elemento de texto na tela. Isso é feito usando a função composable Text() juntamente com alguns modificadores.

3.3

TextField( )

A função composable TextField() é utilizado para criar campos de entrada de texto em interfaces de usuário desenvolvidas com Jetpack Compose. Ele permite que os usuários insiram informações e interajam com o aplicativo de forma dinâmica. Ele aceita diversos parâmetros para personalização, como texto inicial, manipuladores de eventos, estilo da fonte e muito mais

Composable TextField( )

```
@Composable
fun NomeInput() {
    var text by remember { mutableStateOf("") }
    TextField(
        value = text,
        onValueChange = { newText ->
            text = newText
        },
        label = { Text("Digite seu nome") }
    )
}
```

O exemplo acima, é um código de função simples gera uma entrada de texto, que recebe o nome e o armazena na variável text.

3.4

Image( )

O componente Image() é crucial para aplicativos que precisam mostrar conteúdo gráfico como fotos, ícones ou ilustrações. Suas capacidades de integração com outras bibliotecas como Coil ou Glide para carregamento de imagens remotas ou gerenciamento avançado de cache tornam o Image() uma ferramenta versátil em qualquer kit de ferramentas de desenvolvedor de aplicativos Android.

Composable Image( )

```
// Carrega a imagem do diretório res/drawable
val image = painterResource(id = R.drawable.my_image)

Image(
    painter = image,
    contentDescription = "Descrição da imagem",
    modifier = Modifier.size(200.dp)
)
```

O exemplo acima, é um código de utilização do composable Image() modificando o as dimensões de 200dp e uma descrição, o painterResource cria um recurso de pintor a partir de um ID de recurso drawable. Isso informa ao Compose de onde carregar a imagem.

3.5

LazyColumn( )

O composable LazyColumn() é utilizado para criar listas rolaveis verticais eficientes e dinâmicas em interfaces de usuário desenvolvidas com Jetpack Compose. Ele carrega apenas os itens visíveis na tela, proporcionando um desempenho otimizado mesmo para listas muito longas.

Composable LazyColumn( )

```
@Composable
fun ListaDeNomes() {
    val nomes = listOf("Alice", "Bruno", "Carla", "David"
    )

    LazyColumn {
        items(nomes) { nome ->
            Text(nome, modifier = Modifier.padding(8.dp))
        }
    }
}
```

Neste exemplo, a LazyColumn é usada para criar uma lista vertical scrollável de textos. .
items: É um método de alto nível usado dentro de LazyColumn para iterar sobre uma coleção de dados. Gera um composable Text para exibir cada nome na lista fornecida, com modificador de espaçamento.

3.6

CheckBox( )

O composable Checkbox() é utilizado para criar caixas de seleção em interfaces de usuário desenvolvidas com Jetpack Compose. Ele permite aos usuários selecionar ou deselecionar uma opção específica e é frequentemente usado em formulários e listas para representar estados de conclusão, preferências ou opções múltiplas.

Composable Image( )

```
var isChecked by remember { mutableStateOf(false) }

    Column(modifier = Modifier.padding(16.dp)) {
        Checkbox(
            checked = isChecked,
            onCheckedChange = { isChecked = it }
        )
        Spacer(modifier = Modifier.height(8.dp))
        if (isChecked) {
            Text("Checkbox está marcado!")
        }
    }
```

Este exemplo demonstra o uso de um Checkbox com um texto condicional que aparece somente quando o Checkbox está marcado, baseando-se no estado booleano de isChecked. onCheckedChange é uma função que atualiza isChecked quando o usuário interage com o Checkbox, invertendo seu estado atual.

04

Preview

O **@Preview** é uma anotação poderosa do Jetpack Compose que permite visualizar rapidamente como seus Composables serão exibidos na interface do usuário, sem a necessidade de executar o aplicativo completo.

Visualização

O @Preview é uma anotação especial que você pode adicionar a uma função @Composable para gerar uma visualização interativa do Composable correspondente. Ele é integrado ao Android Studio e permite que você veja instantaneamente como seus componentes ficarão na tela enquanto você os está desenvolvendo.

```
@Preview(showBackground = true)
@Composable
fun PreviewNomeEditor() {
    NomeEditor()
}
```

@Preview(showBackground = true): Indica que queremos uma visualização do NomeEditor() e que queremos que a visualização seja exibida em um fundo de cor sólida (padrão). Você pode ajustar showBackground para false se preferir uma visualização sem fundo.

05

CASOS DE USO

A seguir, alguns exemplos de criação de UI com Jetpack Compose.

- Lista de Tarefas
- Componente de aplicativo de Mídia
- Componente de Aplicativo de Comercio Eletrônico

5.1

Lista de Tarefas

Ele inclui a definição da classe de dados Tarefa, a lista de tarefas e o estado para alteração da lista, funções para adicionar tarefas, e os composables para exibir a lista de tarefas e cada item individualmente. A função ListaTarefas() é um Composable que recebe uma lista de tarefas e exibe essas tarefas em uma lista vertical, utilizando o componente LazyColumn para otimizar o desempenho ao lidar com grandes conjuntos de dados. Cada item da lista é representado pelo Composable ItemTarefa(), que exibe o título da tarefa e um checkbox para indicar se a tarefa foi concluída ou não.

Caso 1: Lista de Tarefas

O aplicativo inclui a definição da classe de dados Tarefa, a lista de tarefas e o estado para alteração da lista, funções para adicionar tarefas, e os composables para exibir a lista de tarefas e cada item individualmente. A função ListaTarefas() é um Composable que recebe uma lista de tarefas e exibe essas tarefas em uma lista vertical, utilizando o componente LazyColumn para otimizar o desempenho ao lidar com grandes conjuntos de dados. Cada item da lista é representado pelo Composable ItemTarefa(), que exibe o título da tarefa e um checkbox para indicar se a tarefa foi concluída ou não.

O AfazeresApp() é o composable principal que contém a entrada de texto e a lista de tarefas, permitindo ao usuário adicionar novas tarefas e marcar tarefas como concluídas. Foi utilizado o componente EntradaTexto() do exemplo de código anterior.

ListaTarefas.kt

```kotlin
data class Tarefa(val titulo: String, var taCompleta: Boolean = false)

val tarefas = mutableStateListOf<Tarefa>()

fun adcTarefa(tarefa: Tarefa){
    tarefas.add(tarefa)
}

@Composable
fun ListaTarefas(tarefas: List<Tarefa>) {
    LazyColumn {
        items(tarefas.size) { index ->
            ItemTarefa(tarefa = tarefas[index])
        }
    }
}
@Composable
fun ItemTarefa(tarefa: Tarefa) {
    var completa by remember { mutableStateOf(tarefa.taCompleta) }
    Row(
        modifier = Modifier
            .padding(8.dp)
            .fillMaxWidth(),
        verticalAlignment = Alignment.CenterVertically
    ) {
        Checkbox(
            checked = completa,
            onCheckedChange = { taChecado ->
                completa = taChecado
                tarefa.taCompleta = taChecado
            },
            modifier = Modifier.padding(end = 8.dp)
        )
        Text(text = tarefa.titulo)
    }
}
```

ListaTarefas.kt

```kotlin
@Composable
fun AfazeresApp(tarefas: List<Tarefa>) {
    var novaTarefa by remember { mutableStateOf("") }
    Column(
        modifier = Modifier.background(Color.Cyan)
    ) {
        Row (
            modifier = Modifier.padding(16.dp),
            verticalAlignment = Alignment.CenterVertically
        ){
            EntradaTexto(
                value = novaTarefa,
                onValueChange = { novaTarefa = it },
                label = "Insira a tarefa!",
                modifier = Modifier.weight(1f)
            )
            Button(
                onClick = {
                    if (novaTarefa.isNotBlank()) {
                        adcTarefa(Tarefa(novaTarefa))
                        novaTarefa = ""
                    }
                },
                modifier = Modifier.padding(start = 16.dp)
            ) {
                Text(text = "Adicionar")
            }
        }
        ListaTarefas(tarefas = tarefas)
    }
}

@Preview
@Composable
fun Visualizar(){
    AfazeresApp(tarefas = tarefas)
}
```

5.2

Componente de Mídia

Outro caso de uso do Jetpack Compose é a criação de aplicativos de mídia, como reprodutores de música ou vídeo. Nesse contexto, os Composables podem ser utilizados para representar elementos como controles de reprodução, listas de reprodução, informações de faixa/artista e capas de álbuns ou vídeos.

Caso 2: Componente de mídia

A função MediaPlayerControls é um componente Composable que exibe controles de reprodução de mídia, como botões de play/pause, skipPrevious e skipNext. Os estados dos botões e as ações associadas são gerenciados externamente e passados como parâmetros para o Composable, permitindo uma integração flexível e dinâmica com a lógica de reprodução do aplicativo.

ListaTarefas.kt

```kotlin
@Composable
fun MediaPlayerControls(
    isPlaying: Boolean,
    onPlayPauseClick: () -> Unit,
    onSkipPreviousClick: () -> Unit,
    onSkipNextClick: () -> Unit
) {
    Row(
        modifier = Modifier.fillMaxWidth(),
        horizontalArrangement = Arrangement.Center
    ) {
        IconButton(
            onClick = onSkipPreviousClick,
            modifier = Modifier.padding(end = 16.dp)
        ) {
            Icon(
                imageVector = Icons.Default.ArrowBack,
                contentDescription = "Skip Previous"
            )
        }
        IconButton(
            onClick = onPlayPauseClick,
            modifier = Modifier.padding(end = 16.dp)
        ) {
            Icon(
                imageVector =
                  if (isPlaying) Icons.Default.Pause else Icons.Default.PlayArrow,
                contentDescription = if (isPlaying) "Pause" else "Play"
            )
        }
        IconButton(
            onClick = onSkipNextClick
        ) {
            Icon(
                imageVector = Icons.Default.ArrowForward,
                contentDescription = "Skip Next"
            )
        }
    }
}
```

5.3

Aplicativo de Comércio

O Jetpack Compose também é adequado para o desenvolvimento de aplicativos de comércio eletrônico, que exigem interfaces intuitivas e responsivas para navegação, busca e compra de produtos. Nesse contexto, os Composables podem ser utilizados para representar elementos como categorias de produtos, listas de produtos, detalhes do produto, carrinho de compras e fluxo de checkout.

Caso 2: Comercio Eletrônico

No exemplo de aplicativo de comercio eletrônico, a função ProductList é um Composable que exibe uma grade de produtos em duas colunas, utilizando o componente LazyVerticalGrid para otimizar a exibição de grandes conjuntos de dados. Cada item da grade é representado pelo Composable ProductItem, que exibe uma imagem do produto, seu nome e preço, e permite a navegação para os detalhes do produto ao ser clicado.

Comercio.kt

```kotlin
data class Product(val name: String, val price: Double, val image: String)
@Composable
fun ProductList(products: List<Product>) {
    LazyVerticalGrid(columns = GridCells.Fixed(2)) {
        items(products) { product ->
            ProductItem(product = product)
        }
    }
}

@Composable
fun ProductItem(product: Product) {
    Column(
        modifier = Modifier
            .padding(8.dp)
            .clickable { /* Navegar para detalhes do produto */ }
    ) {
        Image(
            painter = painterResource(product.image),
            contentDescription = product.name,
            modifier = Modifier
                .size(120.dp)
                .clip(shape = RoundedCornerShape(8.dp)),
            contentScale = ContentScale.Crop
        )
        Spacer(modifier = Modifier.height(8.dp))
        Text(
            text = product.name,
            modifier = Modifier
                .fillMaxWidth()
                .wrapContentWidth(Alignment.CenterHorizontally)
        )
        Text(
            text = product.price.toString(),
            modifier = Modifier
                .fillMaxWidth()
                .wrapContentWidth(Alignment.CenterHorizontally)
        )
    }
}
```

Obrigado por ler até aqui!

Esse E-book foi criado com o propósito de colaborar com a explanação de uma palestra a fim de introduzir conceitos de Jetpack Compose, utilizando a linguagem de programação Kotlin, ao desenvolvimento de UI para aplicações em dispositivos android

https://github.com/Joannegton/ListadeComprasApp

Author

Wellington Tavares Galbarini

Github | LinkedIn | Instagram