Simplificando o Desenvolvimento de Interfaces de Usuário Android com Jetpack Compose

Wellington Tavares Galbarini

Introdução

No cenário atual de desenvolvimento de aplicativos móveis, a criação de interfaces de usuário (UI) eficazes desempenha um papel fundamental no sucesso e na aceitação dos aplicativos pelos usuários. Com a evolução contínua da tecnologia, novas abordagens e ferramentas têm surgido para simplificar e aprimorar o processo de criação de UIs, visando oferecer experiências de usuário mais intuitivas, dinâmicas e visualmente atraentes. Neste contexto, o Jetpack Compose, uma biblioteca moderna de UI para o desenvolvimento de aplicativos Android, surge como uma solução revolucionária que tem impactado positivamente o panorama do desenvolvimento móvel. Abordaremos ao longo desse artigo os principais conceitos da criação de UI utilizando o Jetpack Compose, e suas principais funções composables, como Text, Button, Image, Column, Row, Box, TextField e LazyColumn.

Fundamentação Teórica

Foi realizada uma busca abrangente em bases de dados acadêmicas, como Scielo e Google Acadêmico, para identificar artigos relevantes que abordem o Jetpack Compose e seu impacto no desenvolvimento de aplicativos Android, com estudos de caso que demonstrem a aplicação bem-sucedida do Jetpack Compose em projetos reais. Além disso, foi realizada uma análise da documentação oficial do Jetpack Compose e do Android Developers, a fim de compreender os principais recursos, funcionalidades, tutoriais e exemplos dos benefícios do framework. Os resultados desta revisão sistemática da literatura e das análises realizadas reforçam a fundamentação teórica deste estudo. Vale salientar que, em todos os exemplos citados nesse breve artigo requer a importação de bibliotecas para o correto funcionamento.

Princípios e origens do Jetpack Compose

O Jetpack Compose é uma biblioteca de UI lançada pela Google em 2020, com o objetivo de simplificar e modernizar o desenvolvimento de interfaces de usuário para aplicativos Android com menos código. Desenvolvida em Kotlin, a linguagem oficial recomendada pela Google para o desenvolvimento Android, o Compose introduz uma abordagem declarativa e reativa para a construção de UIs, afastando-se do paradigma tradicional baseado em XML e ViewGroups.

Os principais princípios orientadores do Jetpack Compose são a simplicidade, a flexibilidade e a produtividade do desenvolvedor. A biblioteca foi projetada para oferecer uma experiência de codificação mais intuitiva e eficiente, reduzindo a

complexidade e o boilerplate code comuns em desenvolvimentos anteriores. Além disso, o Compose adota uma abordagem reativa, em que as alterações na UI são reflexos diretos das mudanças nos estados dos componentes.

A estrutura básica de uma atividade no Jetpack Compose inclui a definição da atividade principal, a aplicação do tema, a definição do conteúdo da interface do usuário usando Composables e a configuração do contêiner visual principal (Surface). Essas são etapas fundamentais para iniciar o desenvolvimento de um aplicativo com Jetpack Compose.

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. As funções combináveis são o elemento básico de uma interface no Compose.

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

O Jetpack Compose oferece uma variedade de funções composables essenciais para criar interfaces de usuário modernas, responsivas e altamente interativas em aplicativos Android. Antes de explorarmos os composables, é fundamental entender o 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.

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.

O exemplo abaixo, 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.

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

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.

Os três elementos básicos de layout padrão no Jetpack Compose são:

- **Column()**: Usado para organizar os componentes verticalmente, um abaixo do outro. Útil para criar layouts como formulários, listas verticais, ou qualquer estrutura em que os elementos sigam uma ordem vertical. Por exemplo, um formulário de inscrição onde os campos de entrada estão dispostos verticalmente.
- **Row()**: Organiza os componentes horizontalmente, lado a lado. Ideal para criar barras de navegação, listas horizontais, botões de ação alinhados horizontalmente, entre outros. Por exemplo, uma barra de navegação com botões de navegação ou botões de ação.
- **Box()**: Permite sobrepor os componentes, um em cima do outro. Útil para criar sobreposições de elementos, como texto sobre uma imagem, ou camadas de conteúdo. Por exemplo, colocar um texto descritivo sobre uma imagem ilustrativa.

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.

Para que uma imagem seja mostrada no app é necessário utilizar o composable **Image().** Ele é uma parte essencial do toolkit, permitindo que desenvolvedores incorporem recursos visuais nos aplicativos de forma eficiente e flexível. Geralmente, as imagens são armazenadas nos recursos do projeto Android, como res/drawable ou res/mipmap. O caminho da imagem é passado como parâmetro para a função painterResource(), que é utilizada como argumento para o parâmetro painter do Image() composable. As principais propriedades são:

- painter: define como a imagem será carregada e renderizada. Pode ser um recurso estático, uma URL da internet, ou outra fonte.
- contentDescription: Uma string que descreve a imagem, crucial para acessibilidade.
- modifier: Permite adicionar modificações ao composable, como tamanho, padding, ou efeitos visuais como sombras ou bordas.

O composable Button é uma parte fundamental do Jetpack Compose para a criação de botões interativos em interfaces de usuário Android. Ele é utilizado para adicionar funcionalidades de clique e interatividade aos aplicativos, permitindo que os usuários realizem ações ao tocar no botão. As principais propriedades do Button são:

- onClick: Uma função lambda que define a ação a ser realizada quando o botão é clicado. Por exemplo, abrir uma nova tela, realizar uma operação etc.
- enabled: Indica se o botão está habilitado para interação. Por padrão, é verdadeiro (habilitado).
- colors: Define as cores do botão em diferentes estados (habilitado, desabilitado, pressionado etc.), incluindo cores de texto, fundo e sombra.

O 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. Suas principais propriedades são:

- value: Representa o texto atual no campo de texto.
- onValueChange: Callback chamado quando o valor do campo de texto é alterado pelo usuário.
- label: Rótulo exibido acima do campo de texto para indicar seu propósito.
- placeholder: Texto exibido como dica dentro do campo de texto.

- modifier: Permite adicionar modificações ao composable, como tamanho, padding, ou efeitos visuais.

O Jetpack Compose permite a criação de componentes composables personalizados, o que é uma das vantagens mais poderosas da biblioteca. Isso possibilita a reutilização de código e a criação de padrões de design consistentes em todo o projeto Android. Por exemplo, ao criar um composable EntradaTexto() personalizado, pode-se reutilizá-lo em várias partes do aplicativo sempre que precisar de uma entrada de texto padrão. Isso não apenas economiza tempo e esforço de desenvolvimento, mas também garante uma experiência de usuário consistente e coesa em todo o aplicativo.

No exemplo abaixo, a função composable **EntradaTexto**() recebe os seguintes parâmetros: value, onValueChange, label e modifier. Dentro do EntradaTexto, contém o TextField do Jetpack Compose para criar o campo de texto. Aqui estão os principais parâmetros usados:

- value e onValueChange: Representam o valor atual e a ação de mudança do valor do campo de texto, respectivamente.
- label: Define o rótulo exibido acima do campo de texto.
- keyboardOptions: Define as opções do teclado, nesse caso, o tipo de teclado é configurado como numérico.
- modifier: Modifica o layout do TextField.
- colors: Define as cores do TextField, incluindo as cores do rótulo quando o campo está focado ou não.

```
@Composable
fun EntradaTexto(
    value: String,
    onValueChange: (String) -> Unit,
    visualTransformation: VisualTransformation,
    label: String,
    modifier: Modifier = Modifier
) {
    TextField(
        value = value,
        onValueChange = onValueChange,
        label = { Text(text = label, color = Color(0xFF3C94E7))},
        keyboardOptions = KeyboardOptions.Default.copy(
            keyboardType = KeyboardType.Text
        ), // Define o tipo de teclado para textos
        modifier = modifier.fillMaxWidth(),
        colors = OutlinedTextFieldDefaults.colors(
            unfocusedLabelColor = Color(0xFF3C94E7),
            focusedLabelColor = Color(0xFF3C94E7)
        )
    )
}
```

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. As principais propriedades são:

- items: Define os itens a serem exibidos na lista.
- modifier: Permite adicionar modificações ao composable, como tamanho, padding, ou efeitos visuais.

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. Algumas características importantes são:

- checked: Indica se a caixa de seleção está marcada ou não, baseando-se em um estado booleano.
- onCheckedChange: Uma callback que é chamada quando o estado de seleção da caixa é alterado pelo usuário.
- modifier: Permite adicionar modificações visuais ao composable, como espaçamento, tamanho e efeitos visuais.

Essas propriedades tornam o Checkbox() uma ferramenta versátil para a interatividade e controle de seleção em aplicativos Compose, proporcionando uma experiência de usuário intuitiva e eficiente.

Observando a interatividade no contexto do Jetpack Compose, os "estados" referem-se aos diferentes estados que um composable pode assumir e como esses estados podem afetar sua aparência e comportamento na interface do usuário. São fundamentais para criar interfaces dinâmicas e interativas, permitindo que os componentes reajam às mudanças nos dados ou na interação do usuário de forma eficiente e declarativa.

O Jetpack Compose oferece ferramentas como remember e mutableStateOf para gerenciar estados em composables.

- remember: Usado para manter um estado entre recomposições. Por exemplo, você pode usar remember para manter o estado de um tema escuro/claro escolhido pelo usuário durante toda a sessão do aplicativo.
- mutableStateOf: Cria um estado mutável que pode ser alterado. Por exemplo, em um campo de entrada de texto, você pode usar mutableStateOf para armazenar o texto digitado pelo usuário e atualizar o estado sempre que o texto muda.

Uma das principais vantagens do gerenciamento de estados no Jetpack Compose é que a recomposição é automática quando um estado mutável muda.

Isso significa que você não precisa se preocupar em atualizar manualmente a interface do usuário quando o estado muda; o Compose lida com isso automaticamente.

A anotação @Preview é usada para criar visualizações no painel Design do Android Studio. Essas visualizações são úteis durante o desenvolvimento, pois permitem que você visualize e teste rapidamente o comportamento dos composables sem precisar executar o aplicativo no dispositivo ou emulador.

É importante observar que as mudanças feitas em visualizações usando @Preview não afetam o aplicativo real. Ou seja, mesmo que você faça alterações em um composable dentro de uma visualização @Preview, essas mudanças não serão refletidas no aplicativo em si até que você as implemente diretamente no código do aplicativo.

Caso de uso e aplicação 1: Aplicativo de Lista de Tarefas

Um exemplo prático do uso do Jetpack Compose é a criação de um aplicativo de lista de tarefas. Neste caso, os Composables podem ser utilizados para representar elementos como itens de tarefas, listas, botões de adição e remoção, campos de entrada de texto e outros componentes necessários para a interação do usuário com a 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.

```
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)
    }
}
```

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.

```
@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)
}
```

O Jetpack Compose é uma ferramenta poderosa que vai além das funções composables principais. A fim de conhecimento, vamos mergulhar em alguns dos recursos avançados oferecidos por essa plataforma.

Estilização e Temas

A estilização de UIs no Jetpack Compose altamente eficiente e intuitiva é feita de forma declarativa, permitindo a definição de estilos e temas globais que se aplicam consistentemente a toda a aplicação. Isso proporciona uma experiência visual unificada, coesa e harmoniosa para os usuários, além de facilitar a manutenção e a personalização do design da UI, isso é feito usando o arquivo Theme, que serve como um ponto central para a definição de uma ampla variedade de atributos de estilo, como cores, tipografia, tamanhos, espaçamentos e outros elementos visuais essenciais. Esses estilos podem ser configurados uma vez no arquivo Theme e aplicados globalmente a todos os componentes que seguem esse tema.

Animações e Transições

As animações e transições são aspectos fundamentais no Jetpack Compose, com suporte nativo que possibilita a criação de interações fluidas e envolventes na interface do usuário (UI). Essas animações são declarativas, o que significa que podem ser aplicadas a qualquer elemento Composable, tornando a experiência do usuário mais dinâmica e atraente. Com as animações declarativas do Jetpack Compose, é possível criar transições suaves entre diferentes estados de um componente, animações de entrada e saída, efeitos de movimento e muito mais. Isso proporciona uma experiência de usuário mais rica e dinâmica, agregando valor ao design e à usabilidade da aplicação.

Aplicações Práticas e Casos de Uso 2: Aplicativo 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.

```
@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"
            )
        }
    }
}

@Preview
@Composable
fun Visualizar(){
    MediaPlayerControls(
        isPlaying = true,
        onPlayPauseClick = { /*TODO*/ },
        onSkipPreviousClick = { /*TODO*/ }) {

    }
}
```

No exemplo acima, 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.

Aplicações Práticas e Casos de Uso 3: Aplicativo de Comércio Eletrônico

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.

```
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)
        )
    }
}
```

No exemplo acima, 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.

Conclusão

Desde o seu lançamento, o Jetpack Compose tem impactado positivamente o desenvolvimento de aplicativos Android, proporcionando uma experiência de codificação mais intuitiva, flexível e eficiente para os desenvolvedores. A adoção do Compose tem sido crescente na comunidade de desenvolvimento Android, com muitos desenvolvedores migrando gradualmente seus projetos para aproveitar os benefícios e as capacidades oferecidas pela biblioteca.

O Jetpack Compose está redefinindo a maneira como as interfaces de usuário são criadas no ambiente Android, impulsionando a inovação e melhorando a experiência do usuário em aplicativos móveis. Sua abordagem moderna, baseada em Composables, torna o desenvolvimento de UIs mais intuitivo, flexível e eficiente, permitindo aos desenvolvedores criar aplicativos Android de alta qualidade e com uma aparência visual moderna e atrativa.

A biblioteca continua evoluindo e recebendo atualizações e melhorias constantes da Google e da comunidade de desenvolvedores, consolidando-se como uma das principais escolhas para o desenvolvimento de UIs Android no presente e no futuro. Com o Jetpack Compose, os desenvolvedores têm à sua disposição uma ferramenta poderosa e versátil para criar experiências de usuário excepcionais e destacar seus aplicativos no competitivo mercado móvel.

Referencias

AWARI. Entendendo Jetpack Compose: uma introdução ao framework de UI da Google. **Awari Blog**, 2021. Disponível em: https://awari.com.br/jetpack-compose/. Acesso em: 07 abr. 2024.

GOOGLE. Documentação do Jetpack Compose. **Android Developers**, 2024. Disponível em: https://developer.android.com/develop/ui/compose/documentation. Acesso em: 07 abr. 2024.

JAYAWICKRAMA, N. Conhecendo o Jetpack Compose. **Medium**, 2021. Disponível em: https://medium.com/wearejaya/conhecendo-o-jetpack-compose-d0ca4398e5c7. Acesso em: 09 abr. 2024.

Milla, E. Radonjić, M. ANALYSIS OF DEVELOPING NATIVE ANDROID APPLICATIONS USING XML AND JETPACK COMPOSE. **Balkan Journal of Applied Mathematics and Informatics**, v. 23, n. 2, p. 1-16, 2023. Disponível em: https://js.ugd.edu.mk/index.php/bjami/article/view/6019/5057. Acesso em: 10 abr. 2024.