APP WORLD

NAVEGAÇÃO E **FLUXO ENTRE TELAS**

8A

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 NAVEGAÇÃO E FLUXO ENTRE TELAS

Nas abordagens anteriores ao Jetpack Compose, a construção de uma aplicação Android consistia na elaboração de diversas telas, que chamamos de "**Activity**". Cada uma dessas telas era responsável por uma funcionalidade do sistema, por exemplo: uma tela para login, outra para listar os produtos, outra para ver os detalhes de um produto etc.

Outro recurso bastante utilizado na construção de IU Android tradicional é o uso de "**Fragment**", que consiste em criar na "**Activity**" fragmentos de tela, que podem ser reutilizadas em outras telas. Isso torna o desenvolvimento da aplicação mais flexível. Esses fragmentos podem ser atualizados ou até mesmo substituídos de acordo com a interação do usuário

Com a utilização do Jetpack Compose, a abordagem que utilizamos é chamada de "**Single Activity**". Nesta abordagem, temos apenas uma Activity que será responsável por renderizar as diferentes telas ou destinos. A navegação entre as diferentes telas é feita através da criação de rotas.

1.1 Navegação

O Jetpack Compose possui uma biblioteca chamada "**Navigation**" que fornece todos os recursos necessários para a navegação entre telas em uma aplicação Android. Essa navegação ocorre através da configuração de rotas, que indicam quais funções devem ser compostas de acordo com algum evento do usuário ou da própria aplicação.

Através do "**Navigation**" é possível transportarmos dados de uma tela para outra, além de criarmos efeitos visuais durante a transição entre telas.

1.1.1 Adicionando a biblioteca do Navigation

Crie um projeto no Android Studio chamado "**Navegando entre telas**". Assim que o Android Studio finalizar a construção do projeto, apague todas as funções, mantendo apenas a classe "**MainActivity**" e a função "**onCreate()**".

Para utilizarmos a biblioteca “**Navigation**” é necessário adicionarmos uma dependência ao arquivo “**build.gradle**”. A sessão “**dependencies**” do arquivo “**build.gradle (Module :app)**” deverá se parecer com a listagem de código abaixo:

```
dependencies {
    implementation 'androidx.core:core-ktx:1.8.0'
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.5.1'
    implementation platform('androidx.compose:compose-bom:2022.10.00')
    implementation 'androidx.compose.ui:ui'
    implementation 'androidx.compose.ui:ui-graphics'
    implementation 'androidx.compose.ui:ui-tooling-preview'
    implementation 'androidx.compose.material3:material3'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    androidTestImplementation platform('androidx.compose:compose-
bom:2022.10.00')
    androidTestImplementation 'androidx.compose.ui:ui-test-junit4'
    debugImplementation 'androidx.compose.ui:ui-tooling'
    debugImplementation 'androidx.compose.ui:ui-test-manifest'

    // Dependência do Navigation
    implementation 'androidx.navigation:navigation-compose:2.6.0'
}
```

Código-fonte 1 – Adição da biblioteca Navigation ao projeto
Fonte: Elaborado pelo autor (2023)

Após a alteração, vamos clicar em “**sync now**”, conforme a figura “Sincronização do Gradle”, para que o “**Gradle**” faça o download e a configuração do “**Navigation**” em nosso projeto.

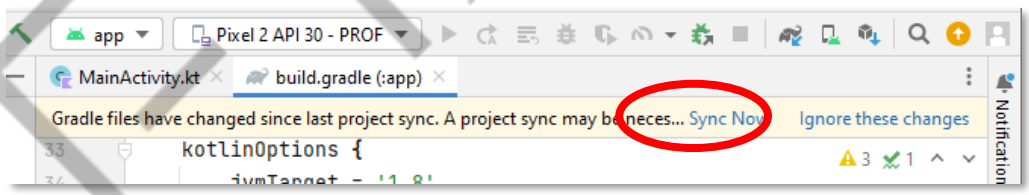

Figura 1 – Sincronização do Gradle
Fonte: Elaborado pelo autor (2023)

1.1.2 Telas do projeto

Com a biblioteca do “**Navigation**” configurada, vamos criar diferentes telas para testarmos a navegação. Ao final, o projeto deverá conter 4 telas que terão o fluxo de navegação de acordo com a figura “Esquema de navegação do aplicativo”:

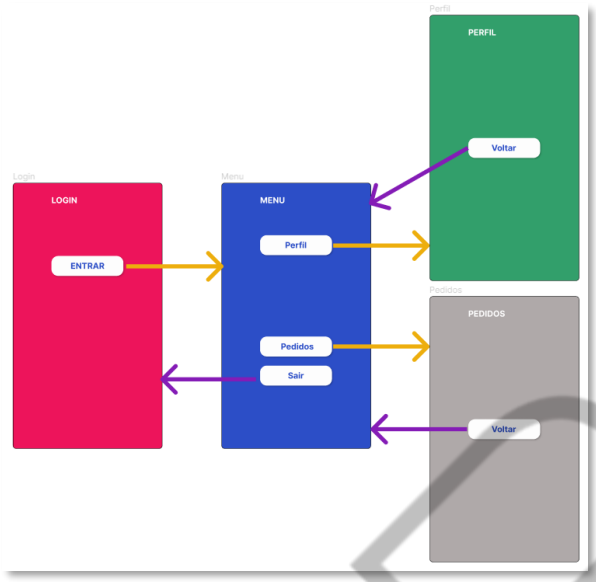

Figura 2 – Esquema de navegação do aplicativo
Fonte: Elaborado pelo autor (2023)

Vale lembrar que na abordagem tradicional de construção de aplicativos Android, precisávamos criar uma "**Activity**" para cada tela do aplicativo. Com o *Jetpack Compose*, aplicamos o conceito de "**Single Activity**", então, nossa aplicação terá apenas uma *Activity*, que é a "**MainActivity**" e teremos os composables que implementam toda a IU de cada tela.

O papel do "**Navigation**" será implementar o fluxo de navegação entre as telas, além de permitir efeitos de transição entre elas.

Por uma questão de organização e padronização, vamos criar cada um dos composables que implementam as telas em arquivos diferentes, que melhora a componentização e reuso. Vamos lá!

Clique com o botão direito do mouse no pacote "**br.com.fiap.navegandoentretelas**". Aponte para "**New**" e em seguida clique na opção "**Package**", conforme demonstra a figura "Criação de novo pacote":

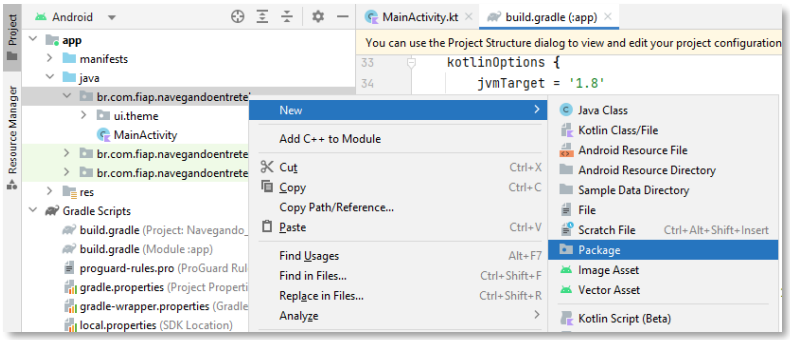

Figura 3 – Criação de novo pacote
Fonte: Elaborado pelo autor (2023)

Na janela “**New Package**”, adicione a palavra “**screens**”, conforme exibido na figura “Nome do pacote”:

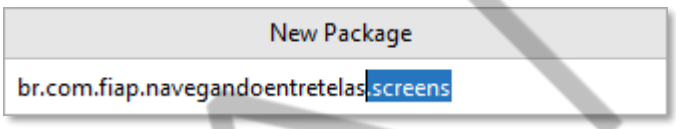

Figura 4 – Nome do pacote
Fonte: Elaborado pelo autor (2023)

Pressione “**Enter**” ao finalizar. A estrutura do seu projeto deverá se parecer com a figura “Estrutura do projeto”:

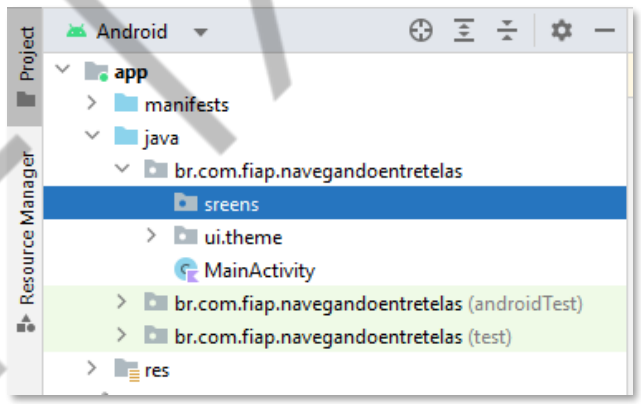

Figura 5 – Estrutura do projeto
Fonte: Elaborado pelo autor (2023)

Vamos criar a nossa primeira tela no arquivo “**LoginScreen**”. Clique com o botão direito do mouse no pacote “**screens**”, aponte para “**New**” e clique na opção “**Kotlin Class/File**”, conforme a figura “Menu criação de novo arquivo”:

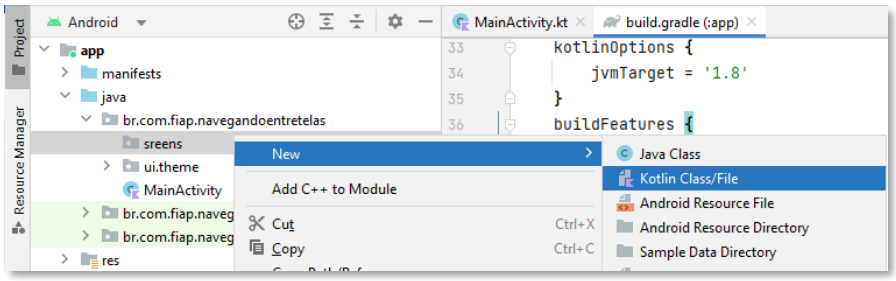

Figura 6 – Menu criação de novo arquivo
Fonte: Elaborado pelo autor (2023)

Na tela “**New Kotlin Class/File**”, digite o nome do arquivo e selecione a opção “**File**” de acordo com a figura “Criação do arquivo LoginScreen”:

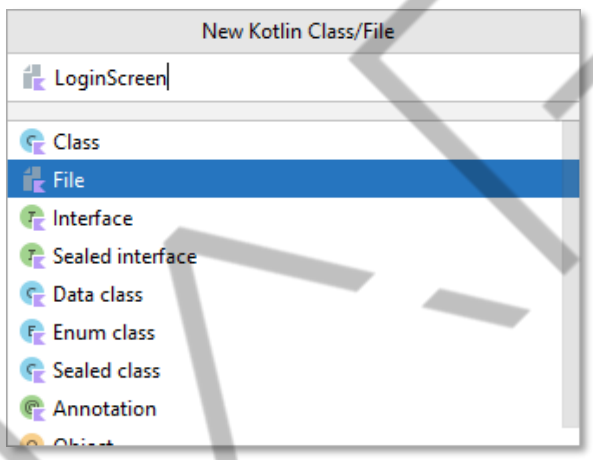

Figura 7 – Criação do arquivo LoginScreen
Fonte: Elaborado pelo autor (2023)

A estrutura do seu projeto deverá se parecer com a figura “Estrutura do pacote screens”:

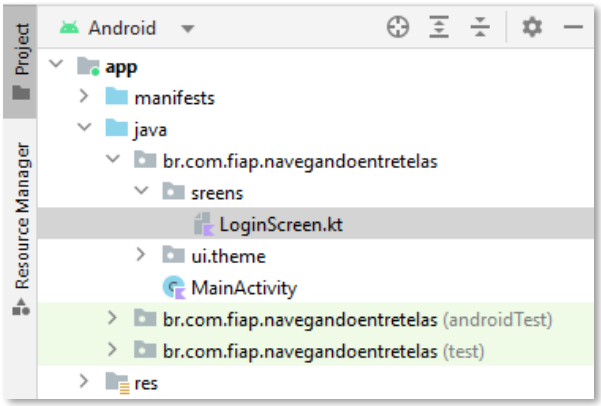

Figura 8 – Estrutura do pacote screens
Fonte: Elaborado pelo autor (2023)

Vamos construir uma tela bem simples, pois nosso foco é entender como funciona a navegação entre telas no *Jetpack Compose*.

O código da listagem abaixo será responsável por renderizar a tela "**LoginScreen**".

```
package br.com.fiap.navegandoentretelas.sreens

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

@Composable
fun LoginScreen() {
  Box(modifier = Modifier
    .fillMaxSize()
    .background(Color(0xFFED145B))
    .padding(32.dp)
  ){
    Text(
      text = "LOGIN",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White
    )
    Button(
      onClick = { /*TODO*/ },
      colors = ButtonDefaults.buttonColors(Color.White),
      modifier = Modifier.align(Alignment.Center)
    ) {
      Text(text = "ENTRAR", fontSize = 20.sp, color = Color.Blue)
    }
  }
}
```

Código-fonte 2 – Tela LoginScreen
Fonte: Elaborado pelo autor (2023)

Para executar a aplicação no emulador e ver o resultado da tela que acabamos de descrever, abra o arquivo "**MainActivity.kt**" e adicione a chamada para a função "**LoginScreen**". O código da "**MainActivity**" deverá se parecer com a listagem abaixo:

```
package br.com.fiap.navegandoentretelas

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.ui.Modifier
import br.com.fiap.navegandoentretelas.sreens.LoginScreen
import br.com.fiap.navegandoentretelas.ui.theme.NavegandoEntreTelasTheme

class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      NavegandoEntreTelasTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          LoginScreen()
        }
      }
    }
  }
}
```

Código-fonte 3 – Chamada para a função LoginScreen
Fonte: Elaborado pelo autor (2023)

Execute a aplicação no emulador e o resultado esperado deverá se parecer com a figura “Tela Login”:

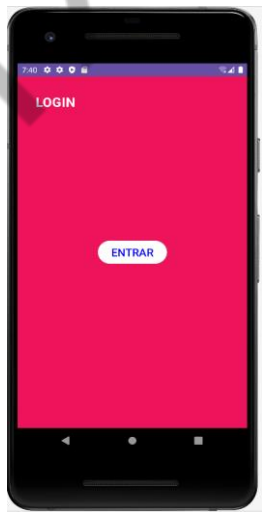

Figura 9 – Tela Login
Fonte: Elaborado pelo autor (2023)

Agora, vamos criar os outros arquivos da nossa aplicação. Na pasta “**screens**” do projeto, crie três arquivos conforme mostra a figura “Arquivos adicionais”:

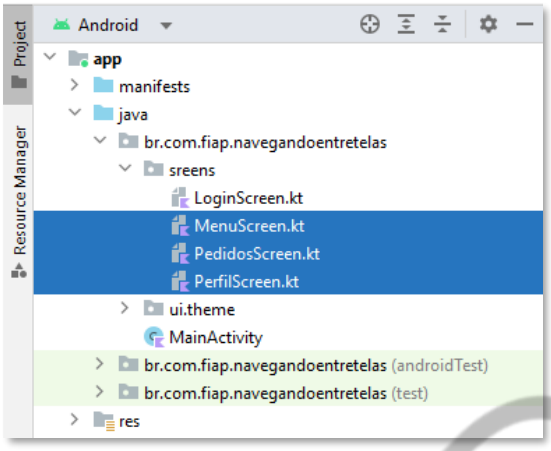

Figura 10 – Arquivos adicionais
Fonte: Elaborado pelo autor (2023)

Cada um dos arquivos deverá ter o conteúdo de acordo com as listagens abaixo:

Arquivo “**MenuScreen.kt**”:

```
@Composable
fun MenuScreen() {
  Box(modifier = Modifier
    .fillMaxSize()
    .background(Color(0xFF2C4EC7))
    .padding(32.dp)
  ){
    Text(
      text = "MENU",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White
    )
    Column(
      horizontalAlignment = Alignment.CenterHorizontally,
      modifier = Modifier
      .fillMaxWidth()
      .align(Alignment.Center)
    ) {
      Button(
        onClick = { /*TODO*/ },
        colors = ButtonDefaults.buttonColors(Color.White),
        modifier = Modifier.size(width = 200.dp, height = 48.dp)
      ) {
        Text(text = "Perfil", fontSize = 20.sp, color = Color.Blue)
      }
      Spacer(modifier = Modifier.height(16.dp))
      Button(
        onClick = { /*TODO*/ },
        colors = ButtonDefaults.buttonColors(Color.White),
        modifier = Modifier.size(width = 200.dp, height = 48.dp)
      ) {
        Text(text = "Pedidos", fontSize = 20.sp, color = Color.Blue)
      }
      Spacer(modifier = Modifier.height(16.dp))
      Button(
```

```
            onClick = { /*TODO*/ },
            colors = ButtonDefaults.buttonColors(Color.White),
            modifier = Modifier.size(width = 200.dp, height = 48.dp)
        ) {
            Text(text = "Sair", fontSize = 20.sp, color = Color.Blue)
        }
    }
  }
}
```

Código-fonte 4 – Tela MenuScreen.kt
Fonte: Elaborado pelo autor (2023)

Arquivo “**PerfilScreen.kt**”:

```
@Composable
fun PerfilScreen() {
  Box(modifier = Modifier
    .fillMaxSize()
    .background(Color(0xFF329F6B))
    .padding(32.dp)
  ){
    Text(
      text = "PERFIL",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White
    )
    Button(
      onClick = { /*TODO*/ },
      colors = ButtonDefaults.buttonColors(Color.White),
      modifier = Modifier.align(Alignment.Center)
    ) {
      Text(text = "Voltar", fontSize = 20.sp, color = Color.Blue)
    }
  }
}
```

Código-fonte 5 – Tela PerfilScreen.kt
Fonte: Elaborado pelo autor (2023)

Arquivo “**PedidosScreen.kt**”:

```
@Composable
fun PedidosScreen() {
  Box(modifier = Modifier
    .fillMaxSize()
    .background(Color(0xFFAFA9A9))
    .padding(32.dp)
  ){
    Text(
      text = "PEDIDOS",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White
    )
```

```
    Button(
      onClick = { /*TODO*/ },
      colors = ButtonDefaults.buttonColors(Color.White),
      modifier = Modifier.align(Alignment.Center)
    ) {
      Text(text = "Voltar", fontSize = 20.sp, color = Color.Blue)
    }
  }
}
```

Código-fonte 6 – Tela PedidosScreen.kt
Fonte: Elaborado pelo autor (2023)

1.1.3 Configurando o Navigation

Com as telas criadas, precisamos configurar a navegação entre elas. Para isso, vamos utilizar a "**NavController**", que é responsável por controlar o fluxo de navegação em uma aplicação Android. A "**NavController**" é uma função com estado que observa a pilha de *composables* que criam as telas. Essa pilha é chamada de "**backstack**".

Pense na *backstack* como sendo uma pilha de cartões. O cartão que está no topo é o cartão visível, ou seja, aquele que está sendo utilizado. Os outros estão em segundo plano e não estão visíveis. Veja na figura "Backstack":

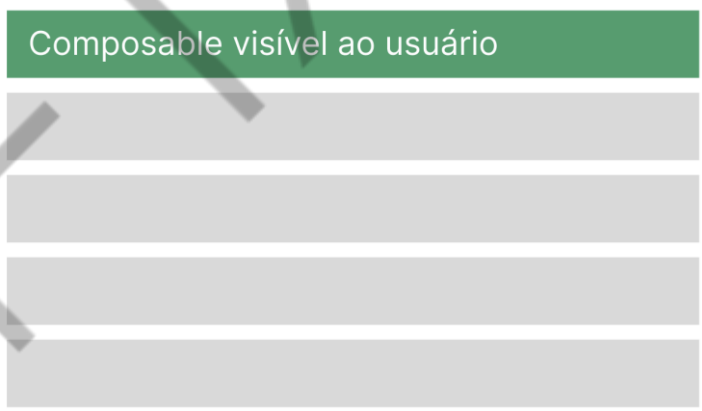

Figura 11 – Backstack
Fonte: Elaborado pelo autor (2023)

Em nosso aplicativo, podemos pensar na "**MainActivity**" como sendo a mesa onde os cartões estão sendo manipulados. O *composable* que está visível ao usuário é o que está no topo da pilha. Os outros composables estão em segundo plano esperando para serem chamados.

A navegação consiste em trazer para a frente da pilha o *composable* com o qual se deseja trabalhar. Também é possível utilizar o botão "***voltar***" do dispositivo, para retornar ao último *composable* acessado, permitindo ao usuário voltar para o ponto inicial desde o início da sua jornada na aplicação.

A configuração da "**NavController**" será na "**MainActivity**", que é o ponto de partida da nossa aplicação, então, implemente as seguintes linhas de código no método "**onCreate**" da "**MainActivity**", de modo que fique como mostrado na listagem abaixo:

```
01 class MainActivity : ComponentActivity() {
02  override fun onCreate(savedInstanceState: Bundle?) {
03    super.onCreate(savedInstanceState)
04    setContent {
05      NavegandoEntreTelasTheme {
06        Surface(
07          modifier = Modifier.fillMaxSize(),
08          color = MaterialTheme.colorScheme.background
09        ) {
10          val navController = rememberNavController()
11          NavHost(
12            navController = navController,
13            startDestination = "login"
14          ){
15            composable(route = "login"){ LoginScreen() }
16            composable(route = "menu") { MenuScreen() }
17            composable(route = "pedidos"){ PedidosScreen() }
18            composable(route = "perfil"){ PerfilScreen() }
19          }
20        }
21      }
22    }
23  }
24 }
```

Código-fonte 7 – Configuração do NavController
Fonte: Elaborado pelo autor (2023)

Na **linha 10,** estamos criando uma instancia do "**NavController**" através da função "**rememberNavController**".

Na **linha 11,** utilizamos a função "**NavHost**", que é responsável por gerenciar as rotas para as telas que devem ser exibidas. O "**NavHost**" utiliza o "**navController**", que possui a "**backstack**" e o "**startDestination**" que é utilizado para indicar qual será a tela que deverá ser exibida quando o aplicativo for aberto pela primeira vez, que neste caso é a tela de login.

Nas **linhas 15 a 18**, estamos indicando quais são os destinos navegáveis pelo "**NavHost**". Essa função recebe dois parâmetros:

- **Route**: que utiliza um identificador único para cada destino que será acessado. Esse identificador é criado pelo desenvolvedor.
- **Função lambda**: onde informamos qual será a tela que deverá ser renderizada.

Execute a aplicação no emulador. Se tudo estiver correto, a aplicação deverá iniciar na tela de login, conforme a figura “Teste da tela Login”:

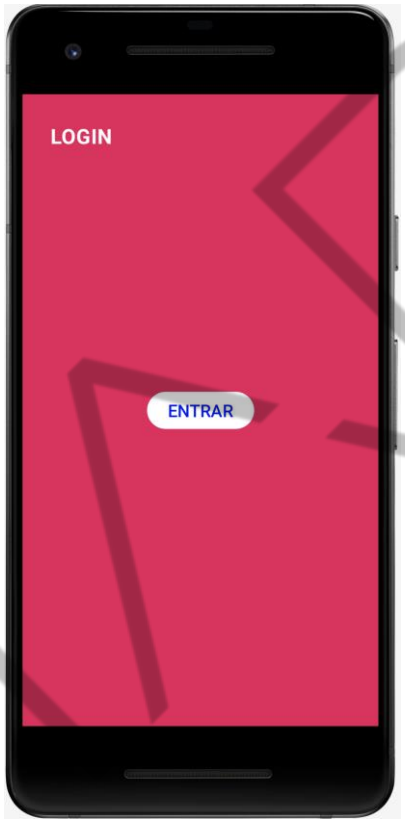

Figura 12 – Teste da tela Login
Fonte: Elaborado pelo autor (2023)

Ao abrirmos a aplicação pela primeira vez, a tela que será apresentada ao usuário é a “**LoginScreen**”.

Para acessar a tela do Menu ao pressionar o botão **'ENTRAR'** na tela de Login, é necessário ter acesso ao **'NavController'**. Portanto, faremos uma alteração na função **'LoginScreen'** para que possamos enviar o **'NavController'** e implementar a navegação corretamente. Veja abaixo a modificação na função **'LoginScreen'**:

```
@Composable
fun LoginScreen(navController: NavController) {
  Box(modifier = Modifier
    .fillMaxSize()
    .background(Color(0xFFED145B))
    .padding(32.dp)
  ){
    Text(
      text = "LOGIN",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White
    )
    Button(
      onClick = { navController.navigate("menu") },
      colors = ButtonDefaults.buttonColors(Color.White),
      modifier = Modifier.align(Alignment.Center)
    ) {
      Text(text = "ENTRAR", fontSize = 20.sp, color = Color.Blue)
    }
  }
}
```

Código-fonte 8 – Utilização do NavController
Fonte: Elaborado pelo autor (2023)

No código acima foi adicionado um novo parâmetro à função "**LoginScreen**", chamado "**navController**", que é do tipo "**NavController**". Isso significa que ao chamar a função "**LoginScreen**", é necessário fornecer um objeto "**NavController**". Dessa forma, a "**LoginScreen**" terá acesso à *backstack* e poderá realizar a navegação corretamente.

No clique do botão foi adicionado uma instrução que utiliza o método "**navigate**" do "**NavController**", para informar o identificador da tela que deverá ser acessada.

Antes de executarmos a aplicação, há um pequeno ajuste que devemos fazer no arquivo "MainActivity.kt". Na chamada da função "LoginScreen" devemos passar o "NavController". O código da "MainActivity" deverá se parecer com a listagem abaixo:

```
class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      NavegandoEntreTelasTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          val navController = rememberNavController()
          NavHost(
            navController = navController,
            startDestination = "login"
          ) {
            composable(route = "login") { LoginScreen(navController) }
            composable(route = "menu") { MenuScreen() }
```

```
                composable(route = "pedidos") { PedidosScreen() }
                composable(route = "perfil") { PerfilScreen() }
            }
        }
    }
}
}
```

Código-fonte 9 – Passagem do NavController para a função LoginScreen
Fonte: Elaborado pelo autor (2023)

Execute o aplicativo no emulador e clique no botão “**ENTRAR**”. Se tudo estiver correto, a aplicação exibirá a tela de menu.

Para que a funcionalidade de navegação seja possível em todas as telas da nossa aplicação, vamos adicionar o mesmo parâmetro que adicionamos à função “LoginScreen” em todas as outras funções de tela da nossa aplicação e implementar o clique de todos os botões para que a navegação ocorra corretamente.

As alterações necessárias deverão se parecer com as listagens de código abaixo:

Arquivo “**MenuScreen.kt**”

```
@Composable
fun MenuScreen(navController: NavController) {
  Box(modifier = Modifier
    .fillMaxSize()
    .background(Color(0xFF2C4EC7))
    .padding(32.dp)
  ){
    Text(
      text = "MENU",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White
    )
    Column(
      horizontalAlignment = Alignment.CenterHorizontally,
      modifier = Modifier
      .fillMaxWidth()
      .align(Alignment.Center)
    ) {
      Button(
        onClick = { navController.navigate("perfil") },
        colors = ButtonDefaults.buttonColors(Color.White),
        modifier = Modifier.size(width = 200.dp, height = 48.dp)
      ) {
        Text(text = "Perfil", fontSize = 20.sp, color = Color.Blue)
      }
      Spacer(modifier = Modifier.height(16.dp))
      Button(
        onClick = { navController.navigate("pedidos") },
        colors = ButtonDefaults.buttonColors(Color.White),
        modifier = Modifier.size(width = 200.dp, height = 48.dp)
```

```
        ) {
          Text(text = "Pedidos", fontSize = 20.sp, color = Color.Blue)
        }
        Spacer(modifier = Modifier.height(16.dp))
        Button(
          onClick = { navController.navigate("login") },
          colors = ButtonDefaults.buttonColors(Color.White),
          modifier = Modifier.size(width = 200.dp, height = 48.dp)
        ) {
          Text(text = "Sair", fontSize = 20.sp, color = Color.Blue)
        }
      }
    }
}
```

Código-fonte 10 – Adição do parâmetro NavController em MenuScreen
Fonte: Elaborado pelo autor (2023)

Arquivo “**PerfilScreen.kt**”

```
@Composable
fun PerfilScreen(navController: NavController) {
  Box(modifier = Modifier
    .fillMaxSize()
    .background(Color(0xFF329F6B))
    .padding(32.dp)
  ){
    Text(
      text = "PERFIL",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White
    )
    Button(
      onClick = { navController.navigate("menu") },
      colors = ButtonDefaults.buttonColors(Color.White),
      modifier = Modifier.align(Alignment.Center)
    ) {
      Text(text = "Voltar", fontSize = 20.sp, color = Color.Blue)
    }
  }
}
```

Código-fonte 11 – Adição do parâmetro NavController em PerfilScreen
Fonte: Elaborado pelo autor (2023)

Arquivo “**PedidosScreen.kt**”

```
@Composable
fun PedidosScreen(navController: NavController) {
  Box(modifier = Modifier
    .fillMaxSize()
    .background(Color(0xFFAFA9A9))
    .padding(32.dp)
  ){
    Text(
      text = "PEDIDOS",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White
    )
    Button(
      onClick = { navController.navigate("menu") },
      colors = ButtonDefaults.buttonColors(Color.White),
      modifier = Modifier.align(Alignment.Center)
    ) {
      Text(text = "Voltar", fontSize = 20.sp, color = Color.Blue)
    }
  }
}
```

Código-fonte 12 – Adição do parâmetro NavController em PedidosScreen
Fonte: Elaborado pelo autor (2023)

Arquivo “**MainActivity.kt**”

```
class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      NavegandoEntreTelasTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          val navController = rememberNavController()
          NavHost(
            navController = navController,
            startDestination = "login"
          ) {
            composable(route = "login") { LoginScreen(navController) }
            composable(route = "menu") { MenuScreen(navController) }
            composable(route = "pedidos") { PedidosScreen(navController) }
            composable(route = "perfil") { PerfilScreen(navController) }
          }
        }
      }
    }
  }
}
```

Código-fonte 13 – Correção das rotas em MainActivity.kt
Fonte: Elaborado pelo autor (2023)

Ao executar a aplicação no emulador você poderá navegar por todas as telas da aplicação. Experimente!

1.2 Passagem de parâmetros entre telas

Ao trabalhar com diversas telas é bastante comum precisarmos transportar dados de uma tela para outra. Quando utilizamos a biblioteca “**Navigation**”, isso pode ser feito de maneira bastante simples.

1.2.1 Parâmetros obrigatórios

Vamos enviar uma String para a tela “**PerfilScreen**” quando o botão “**Perfil**” da tela “**Menu**” for pressionado. Assim, vamos acrescentar um parâmetro do tipo “**String**” na função “**PerfilScreen**”. O código da função “**PerfilScreen**” deverá se parecer com o da listagem abaixo:

```
@Composable
fun PerfilScreen(navController: NavController, nome: String) {
  Box(modifier = Modifier
    .fillMaxSize()
    .background(Color(0xFF329F6B))
    .padding(32.dp)
  ){
    Text(
      text = "PERFIL - $nome",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White
    )
    Button(
      onClick = { navController.navigate("menu") },
      colors = ButtonDefaults.buttonColors(Color.White),
      modifier = Modifier.align(Alignment.Center)
    ) {
      Text(text = "Voltar", fontSize = 20.sp, color = Color.Blue)
    }
  }
}
```

Código-fonte 14 – Acréscimo do parâmetro nome
Fonte: Elaborado pelo autor (2023)

Na função “**onCreate**” da “**MainActivity**”, vamos acrescentar o parâmetro que deverá ser fornecido para a rota. O código deverá se parecer com a listagem abaixo:

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContent {
    NavegandoEntreTelasTheme {
      Surface(
        modifier = Modifier.fillMaxSize(),
        color = MaterialTheme.colorScheme.background
      ) {
        val navController = rememberNavController()
        NavHost(
          navController = navController,
          startDestination = "login"
        ) {
          composable(route = "login") {
            LoginScreen(navController) }
          composable(route = "menu") {
            MenuScreen(navController) }
          composable(route = "pedidos") {
            PedidosScreen(navController) }
          composable(route = "perfil/{nome}") {
            PerfilScreen(navController)
          }
        }
      }
    }
  }
}
```

Código-fonte 15 – Acréscimo do parâmetro nome a rota
Fonte: Elaborado pelo autor (2023)

Na função “composable” recebemos o parâmetro “**NavBackStackEntry**”, que contém o valor do parâmetro que está sendo passado no path da rota. Vamos recuperar este valor e passar na chamada para a função “PerfilScreen”. O código deverá se parecer com a listagem abaixo:

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContent {
    NavegandoEntreTelasTheme {
      Surface(
        modifier = Modifier.fillMaxSize(),
        color = MaterialTheme.colorScheme.background
      ) {
        val navController = rememberNavController()
        NavHost(
          navController = navController,
          startDestination = "login"
        ) {
          composable(route = "login") {
             LoginScreen(navController) }
          composable(route = "menu") {
             MenuScreen(navController) }
          composable(route = "pedidos") {
             PedidosScreen(navController) }
          composable(route = "perfil/{nome}") {
            val nome: String? =
             it.arguments?.getString("nome", "")
            PerfilScreen(navController, nome!!)
          }
        }
      }
    }
  }
}
```

Código-fonte 16 – Recuperação do parâmetro nome na rota
Fonte: Elaborado pelo autor (2023)

Vamos ajustar o evento de clique da tela “**MenuScreen**”, para que passe o parâmetro “**nome**” para a tela “**PerfilScreen**”. O código da função “**MenuScreen**” deve se parecer com a listagem abaixo:

```
@Composable
fun MenuScreen(navController: NavController) {
  Box(modifier = Modifier
    .fillMaxSize()
    .background(Color(0xFF2C4EC7))
    .padding(32.dp)
  ){
    Text(
      text = "MENU",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White
    )
    Column(
      horizontalAlignment = Alignment.CenterHorizontally,
      modifier = Modifier
```

```
        .fillMaxWidth()
        .align(Alignment.Center)
    ) {
        Button(
            onClick = {
                navController.navigate("perfil/Pedro da Silva")
            },
            colors = ButtonDefaults.buttonColors(Color.White),
            modifier = Modifier.size(width = 200.dp, height = 48.dp)
        ) {
            Text(text = "Perfil", fontSize = 20.sp, color =
Color.Blue)
        }
```

Código-fonte 17 – Passagem do valor do parâmetro nome na rota
Fonte: Elaborado pelo autor (2023)

Estamos passando o valor do parâmetro “**nome**” de forma manual, mas poderíamos obter esse dado de qualquer outra forma.

Execute a aplicação e navegue até a tela “Perfil”, que deverá se parecer com a figura “Exibindo o parâmetro nome”:

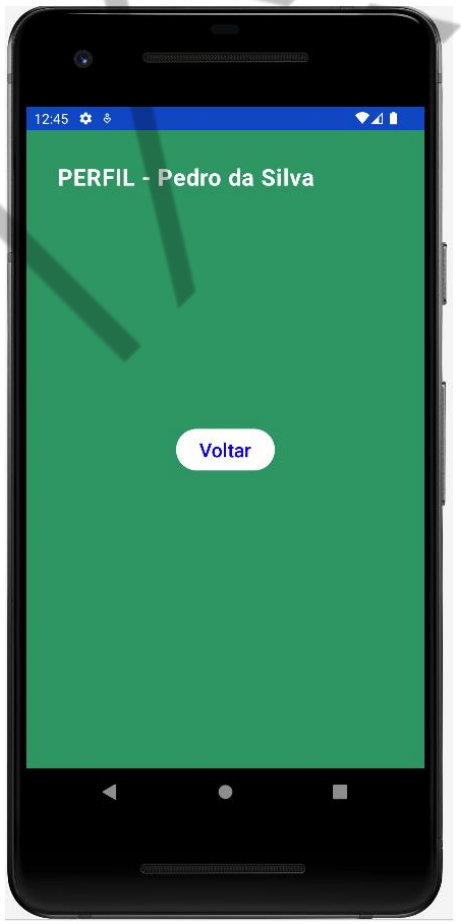

Figura 13 – Exibindo o parâmetro nome
Fonte: Elaborado pelo autor (2023)

1.2.2 Parâmetros opcionais

Nem sempre a passagem de parâmetros é obrigatória. Às vezes o parâmetro pode ser opcional. Vamos configurar a tela “**Pedidos**” de modo que a passagem de parâmetro seja opcional.

Vamos começar ajustando o código do “**NavHost**” para a tela “**Pedidos**” no arquivo “**MainActivity**”. Após o ajuste, o código deverá se parecer com a listagem abaixo:

```
composable(
  route = "pedidos?cliente={cliente}",
  arguments = listOf(navArgument(name = "cliente") {
    defaultValue = "Sem cliente"
  })
) {
  PedidosScreen(navController, it.arguments?.getString("cliente"))
}
```

Código-fonte 18 – Configurando parâmetro opcional na rota
Fonte: Elaborado pelo autor (2023)

A diferença entre a abordagem anterior e essa é a forma como o parâmetro é passado, que lembra bastante o uso de “**queryString**”. Outra mudança importante é o parâmetro “**arguments**”, que agora devemos colocar um valor padrão, para caso o valor não for fornecido. No exemplo anterior, utilizamos o valor de “**defaultValue**” para “**Sem cliente**”, ou seja, se ao chamarmos a tela “**Pedidos**” não fornecermos o valor, será utilizado “**Sem cliente**”.

Vamos adicionar um parâmetro “**String**” na função “**PedidosScreen**” no arquivo “**PedidosScreen.kt**”, além de utilizarmos na função o argumento recebido. O código da função “**PedidosScreen**” deverá se parecer com a listagem de código abaixo:

```
@Composable
fun PedidosScreen(navController: NavController, cliente: String?) {
  Box(modifier = Modifier
    .fillMaxSize()
    .background(Color(0xFFAFA9A9))
    .padding(32.dp)
  ){
    Text(
      text = "PEDIDOS - $cliente",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White
    )
    Button(
      onClick = { navController.navigate("menu") },
```

```
        colors = ButtonDefaults.buttonColors(Color.White),
        modifier = Modifier.align(Alignment.Center)
      ) {
        Text(text = "Voltar", fontSize = 20.sp, color = Color.Blue)
      }
    }
}
```

Código-fonte 19 – Adição do parâmetro cliente na função PedidosScreen
Fonte: Elaborado pelo autor (2023)

Execute a aplicação no emulador e navegue até a tela “Pedidos”. O resultado esperado deverá se parecer com a figura “Testando parâmetro vazio”:

Figura 14 – Testando parâmetro vazio
Fonte: Elaborado pelo autor (2023)

Quando executamos a aplicação, não passamos nenhum argumento para a tela “**Pedidos**”, então, foi utilizado o valor padrão “**Sem cliente**”. Vamos fazer uma pequena alteração no botão que abre a tela “Pedido” no arquivo “**MenuScreen.kt**” para inserirmos o valor do parâmetro. Seu código deverá se parecer com a listagem abaixo:

```
Button(
  onClick = { navController.navigate("pedidos?cliente=FIAP") },
  colors = ButtonDefaults.buttonColors(Color.White),
  modifier = Modifier.size(width = 200.dp, height = 48.dp)
) {
  Text(text = "Pedidos", fontSize = 20.sp, color = Color.Blue)
}
```

Código-fonte 20 – Fornecendo valor a um parâmetro opcional
Fonte: Elaborado pelo autor (2023)

Execute a aplicação novamente. O resultado deverá se parecer com a figura "Inserindo valor ao parâmetro opcional":

Figura 15 – Inserindo valor ao parâmetor opcional
Fonte: Elaborado pelo autor (2023)

1.2.3 Passando múltiplos parâmetros

É possível passar múltiplos valores entre as telas e definir o tipo do dado que será passado. Para isso, utilizamos o parâmetro "**arguments**" da função *composable*.

Vamos alterar o código da "**MainActivity**", para que possamos enviar os argumentos nome como um valor do tipo "**String**" e a idade como um valor do tipo "**Int**" para a tela de perfil. Seu código deverá se parecer com a listagem abaixo:

```
class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      NavegandoEntreTelasTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          val navController = rememberNavController()
          NavHost(
            navController = navController,
            startDestination = "login",
          ) {
            composable(route = "login") {
              LoginScreen(navController)
            }
            composable(route = "menu") {
              MenuScreen(navController)
            }
            composable(route = "pedidos") {
              PedidosScreen(navController)
            }
            composable(
              route = "perfil/{nome}/{idade}",
              arguments = listOf(
                navArgument("nome") {
                  type = NavType.StringType
                },
                navArgument("idade") {
                  type = NavType.IntType
                }
              )
            ) {
              val nome: String? =
                it.arguments?.getString("nome", "")
              val idade: Int? =
                it.arguments?.getInt("idade", 0)
              PerfilScreen(navController, nome!!, idade!!)
            }
          }
        }
      }
    }
  }
}
```

Código-fonte 21 – Passagem de parâmetros múltiplos
Fonte: Elaborado pelo autor (2023)

Através do parâmetro “**arguments**” da função *composable* é possível enviarmos uma lista de argumentos. O “**path**” do argumento “**route**” deverá contemplar todos os argumentos que se deseja passar.

Na função “**PerfilScreen**” vamos adicionar o parâmetro “idade” na lista de argumentos e adicioná-lo ao composable “**Text**”. Seu código deverá se parecer com a listagem abaixo:

```
@Composable
fun PerfilScreen(
  navController: NavController,
  nome: String,
  idade: Int
) {
  Box(modifier = Modifier
    .fillMaxSize()
    .background(Color(0xFF329F6B))
    .padding(32.dp)
  ){
    Text(
      text = "PERFIL - $nome tem $idade anos.",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold,
      color = Color.White
    )
    Button(
      onClick = { navController.navigate("menu") },
      colors = ButtonDefaults.buttonColors(Color.White),
      modifier = Modifier.align(Alignment.Center)
    ) {
      Text(
        text = "Voltar",
        fontSize = 20.sp,
        color = Color.Blue
      )
    }
  }
}
```

Código-fonte 22 – Adição do parâmetro idade a função PerfilScreen
Fonte: Elaborado pelo autor (2023)

Na função “**MenuScreen**” vamos alterar o parâmetro “**onClick**” do botão “Perfil” para acrescentar a idade na rota destino. O código da função “**MenuScreen.kt**” deverá se parecer com a listagem abaixo:

```
@Composable
fun MenuScreen(navController: NavController) {
  Box(
    modifier = Modifier
      .fillMaxSize()
      .background(Color(0xFF2C4EC7))
      .padding(32.dp)
  ) {
    Text(
      text = "MENU",
```

```
        fontSize = 24.sp,
        fontWeight = FontWeight.Bold,
        color = Color.White
      )
      Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier
          .fillMaxWidth()
          .align(Alignment.Center)
      ) {
        Button(
          onClick = {
            navController.navigate("perfil/Pedro/27")
          },
          colors = ButtonDefaults.buttonColors(Color.White),
          modifier = Modifier.size(width = 200.dp, height = 48.dp)
        ) {
```

Código-fonte 23 – Fornecendo parâmetros múltiplos à rota
Fonte: Elaborado pelo autor (2023)

Execute a aplicação e navegue até a tela de Perfil. A IU Perfil deverá se parecer com a figura “Testando passagem de parâmetros múltiplos”:

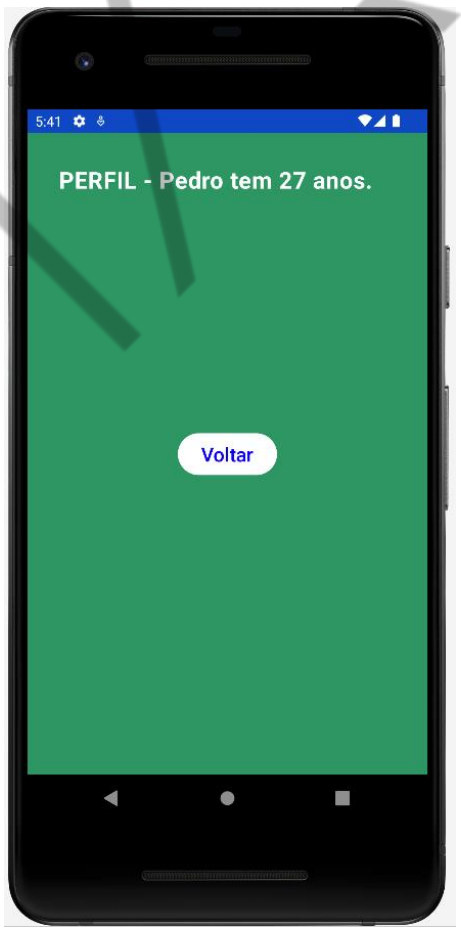

Figura 16 – Testando passagem de parâmetros múltiplos
Fonte: Elaborado pelo autor (2023)

1.3 Animação entre transição de telas

A troca entre as telas em uma aplicação Android ocorre de forma muito simples. Assim, podemos incluir animação entre a transição para tornar a experiência do usuário mais agradável.

1.3.1 Configuração da animação

Para a utilização de animação de transição entre telas Android é necessário adicionarmos uma biblioteca externa chamada "**Accompanist Navigation Animation**". Adicione na sessão "**dependencies**" do arquivo "**build.gradle (Module :app)**" a seguinte linha:

```
implementation "com.google.accompanist:accompanist-navigation-
animation:0.30.1"
```

Código-fonte 24 – Adicionando dependência para animação
Fonte: Elaborado pelo autor (2023)

Após a alteração, clique em "**Sync Now**" para sincronizar as dependências da nova biblioteca.

1.3.2 Implementando a animação

Para implementarmos a animação durante a transição de telas em uma aplicação Android, vamos fazer algumas poucas alterações em relação ao que já temos. Faça as seguintes alterações no método "**onCreate**" da classe "**MainActivity**". Seu código deverá se parecer com a listagem abaixo:

```
class MainActivity : ComponentActivity() {
  @OptIn(ExperimentalAnimationApi::class)
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      NavegandoEntreTelasTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          val navController = rememberAnimatedNavController()
```

```
            AnimatedNavHost(
              navController = navController,
              startDestination = "login",
            ) {
              composable(route = "login") {
                LoginScreen(navController)
              }
              composable(route = "menu") {
                MenuScreen(navController)
              }
              composable(route = "pedidos") {
                PedidosScreen(navController)
              }
              composable(
                route = "perfil/{nome}/{idade}",
                arguments = listOf(
                  navArgument("nome") {
                    type = NavType.StringType
                  },
                  navArgument("idade") {
                    type = NavType.IntType
                  }
                )
              ) {
                val nome: String? =
                  it.arguments?.getString(
                    "nome",
                    ""
                  )
                val idade: Int? =
                  it.arguments?.getInt(
                    "idade",
                    0
                  )
                PerfilScreen(
                  navController,
                  nome!!,
                  idade!!
                )
              }
            }
          }
        }
      }
    }
  }
}
```

Código-fonte 25 – Aplicando a biblioteca de animação
Fonte: Elaborado pelo autor (2023)

As classes utilizadas para a implementação de animação de transição entre as telas precisam ser importadas para o nosso projeto, portanto, certifique-se de que os imports estejam corretos. A lista de imports do arquivo “MainActivity” deve se parecer com a listagem abaixo:

```
//import androidx.navigation.compose.NavHost
//import androidx.navigation.compose.composable
//import androidx.navigation.compose.rememberNavController
```

```
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.ExperimentalAnimationApi
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.ui.Modifier
import androidx.navigation.NavType
import androidx.navigation.navArgument
import br.com.fiap.navegandoentretelas.sreens.LoginScreen
import br.com.fiap.navegandoentretelas.sreens.MenuScreen
import br.com.fiap.navegandoentretelas.sreens.PedidosScreen
import br.com.fiap.navegandoentretelas.sreens.PerfilScreen
import br.com.fiap.navegandoentretelas.ui.theme.NavegandoEntreTelasTheme
import com.google.accompanist.navigation.animation.AnimatedNavHost
import com.google.accompanist.navigation.animation.composable
import
com.google.accompanist.navigation.animation.rememberAnimatedNavController
```

Código-fonte 26 – Ajuste das importações na MainAcitivity
Fonte: Elaborado pelo autor (2023)

Atenção: todos os “**imports**” comentados no início da listagem devem ser substituídos pelos “**imports**” em negrito.

Ao executar, a aplicação não mudou, pois ainda não implementamos as animações que devem ocorrer na troca das telas.

Podemos aplicar o efeito de transição de telas em dois momentos:

- **exitTransition**: a tela atual sai da visão do usuário.
- **startTransition**: a próxima tela entra na visão do usuário.

O efeito que queremos na transição de telas em nossa aplicação será a seguinte:

1 – A tela atual desliza para a direita em um ciclo de 1s;

2 – A tela de destino desliza para a esquerda em um ciclo de 3s.

Após os ajustes, o seu código deverá se parecer com a listagem abaixo:

```
… trecho de Código omitido
setContent {
  NavegandoEntreTelasTheme {
    Surface(
      modifier = Modifier.fillMaxSize(),
      color = MaterialTheme.colorScheme.background
    ) {
      val navController = rememberAnimatedNavController()
      AnimatedNavHost(
        navController = navController,
        startDestination = "login",
        exitTransition = {
          slideOutOfContainer(towards =
          AnimatedContentScope.SlideDirection.End,
            animationSpec = tween(1000)
          )
        },
        enterTransition = {
          slideIntoContainer(towards =
            AnimatedContentScope.SlideDirection.Start,
            animationSpec = tween(3000)
          )
        }
      ) {
… trecho de código omitido
```

Código-fonte 27 – Inclusão dos efeitos de transição
Fonte: Elaborado pelo autor (2023)

No código acima, em "**exitTransition**", escolhemos a transição "**slideOutOfContainer**", que significa "**Deslizar para fora do container**". As configurações para essa animação utilizam os seguintes parâmetros:

- **towards:** este parâmetro indica a direção que a tela deverá seguir enquanto desliza. Para que a tela deslize para a direita, utilizamos "**AnimatedContentScope.SlideDirection.End"**.
- **animationSpec**: Indicamos que a transição deverá durar **1 segundo** (1000 milissegundos).

Em "**enterTransition**", escolhemos o efeito "**slideIntoContainer**", que significa "**Deslizar para dentro do container**". As configurações utilizadas foram as seguintes:

- **towards**: definimos que a animação deverá ocorrer da direita para a esquerda através de "**AnimatedContentScope.SlideDirection.Start**".
- **animationSpec**: Indicamos que a transição deverá durar **3 segundos** (3000 milissegundos).

Execute a aplicação, navegue entre as telas e observe o resultado.

1.3.3 Combinação de efeitos

É possível somar efeitos tanto para "**exitTransition**" quanto para "**startTransition**". Para isso, basta adicionar o sinal de mais (+) entre os efeitos de transição escolhidos.

Para adicionar um efeito de "**desaparecimento gradual**" durante a saída da tela, vamos acrescentar o efeito "**fadeOut**". Seu código deverá se parecer com a listagem de código abaixo:

```
… trecho de código omitido
val navController = rememberAnimatedNavController()
AnimatedNavHost(
  navController = navController,
  startDestination = "login",
  exitTransition = {
    //fadeOut(animationSpec = tween(1000))
    slideOutOfContainer(towards =
    AnimatedContentScope.SlideDirection.End,
      animationSpec = tween(1000)
    ) + fadeOut(animationSpec = tween(1000))
  },
  enterTransition = {
    //fadeIn(animationSpec = tween(2000))
    slideIntoContainer(towards =
      AnimatedContentScope.SlideDirection.Start,
      animationSpec = tween(3000)
  }
) { … trecho de código omitido
```

Código-fonte 28 – Combinando efeitos de transição
Fonte: Elaborado pelo autor (2023)

No código acima, acrescentamos à saída da tela o efeito "**fadeOut**" com duração de **1 segundo** (1000 milissegundos).

Execute a aplicação no emulador, navegue entre as telas e observe os efeitos de transição.

CONCLUSÃO

O uso do **Navigation**, a passagem de dados entre telas e a animação de transição no *Jetpack Compose* trazem uma abordagem completa e robusta para a construção de fluxos de navegação em aplicativos Android.

O **Navigation** oferece uma estrutura clara e declarativa para a definição dos destinos de navegação, permitindo criar uma arquitetura de navegação escalável e fácil de manter. Com a passagem de dados entre telas, é possível transmitir informações relevantes e personalizadas, enriquecendo a experiência do usuário.

Além disso, as animações de transição possibilitam a criação de efeitos visuais envolventes e polidos durante a troca de telas, adicionando um toque especial ao aplicativo. Através das animações, é possível criar transições suaves, deslizamentos elegantes e outros efeitos visuais que tornam a experiência do usuário mais agradável e profissional.

REFERÊNCIAS

ACCOMPANIST. **Como navegar com o Compose**. 2023. Disponível em: <https://google.github.io/accompanist/navigation-animation/> Acesso em: 3 jul. 2023.

DEVELOPERS. **Como navegar com o Compose**. 2023. Disponível em: <https://developer.android.com/jetpack/compose/navigation?hl=pt-br>. Acesso em: 3 jul. 2023.