Desenvolvimento Mobile

Professor Maurício Buess

mbuess@up.edu.br

github.com/mauriciobuess

- Objetivos:
  - Compreender elemento ViewModel
  - Resolução atividade aula 11 versão 1 (protótipo)
  - Atividade

- **ViewModel**:
  - Componente fundamental na arquitetura do Android Jetpack, especialmente quando se trabalha com o Composable;
  - Ele ajuda a gerenciar o estado da UI e a lógica de negócios de forma que sobreviva às mudanças de configuração, como rotações de tela

- **ViewModel**:
  - Classe que armazena e gerencia dados relacionados à UI de forma a sobreviver a mudanças de configuração, como a rotação da tela.
  - O ViewModel faz parte do pacote androidx.lifecycle e é projetado para ser usado com componentes de arquitetura, como LiveData e LifecycleOwner

- **Onde usar o ViewModel:**

  - **Persistência de Dados**: Dados armazenados no ViewModel persistem durante as mudanças de configuração, evitando perda de dados e necessidade de reinicialização.
  - **Separação de Preocupações**: O ViewModel separa a lógica de negócios e os dados da UI, facilitando o teste e a manutenção.
  - **Gerenciamento de Estado**: Ele ajuda a gerenciar o estado da UI e lógica de forma eficiente.

- **Criando a ViewModel:**

•Verifique as dependências (Gradle):

  - androidx.lifecycle:lifecycle-viewmodel-compose:2.7.0
  - androidx.compose.runtime:runtime-livedata:1.6.0
  - Utilize a versão mais recente

•Crie uma classe ViewModel que gerencia o estado da sua classe:

```
class MainViewModel : ViewModel() {
    // Estado gerenciado pelo ViewModel
    private val _pessoa = mutableStateOf<Pessoa?>(null)
    val pessoa: Pessoa?
        get() = _pessoa.value

    // Função para atualizar o estado
    fun setPessoa(pessoa: Pessoa) {
        _pessoa.value = pessoa
    }
}
```

- **Usar o ViewModel no Jetpack Compose:**

```
import androidx.compose.runtime.Composable
import androidx.lifecycle.viewmodel.compose.viewModel


@Composable
fun MyScreen() {
    // Obter o ViewModel
    val viewModel: MainViewModel = viewModel()
    // Usar o estado do ViewModel
    val pessoa = viewModel.pessoa
    // Exibir dados ou modificar estado
}
```

- **Integrar com a Navegação**

```
@Composable
fun AppNavHost(navController: NavHostController, viewModel: MainViewModel) {

    NavHost(navController = navController, startDestination = "menu") {
        composable("menu") { MenuPrincipal(navController, viewModel)       }
        composable("dados") { TelaDados(navController, viewModel) }
        composable("compartilha") { TelaCompartilha(navController, viewModel)      }
    }
}
```

- **Persistir Dados no ViewModel -** Para persistir e gerenciar o estado, atualize o ViewModel em resposta a eventos da UI:

```
@Composable
fun TelaDados(navController: NavHostController, viewModel: MainViewModel) {
    var nome by remember { mutableStateOf(viewModel.pessoa?.nome ?: "") }
    var numeroTelefone by remember { mutableStateOf(viewModel.pessoa?.numeroTelefone ?: "") }
    var descricao by remember { mutableStateOf(viewModel.pessoa?.descricao ?: "") }
    Column(modifier = Modifier.fillMaxSize().padding(16.dp)
        ,horizontalAlignment = Alignment.CenterHorizontally
        ,verticalArrangement = Arrangement.Center
    ) {Text("Nome:")
```

- **Persistir Dados no ViewModel –** (cont.):

```
    OutlinedTextField(value = nome,onValueChange = { nome = it })
    // Similar para outros campos...
    Button(onClick = {val pessoa = Pessoa(nome, numeroTelefone, descricao)
        viewModel.setPessoa(pessoa) // Atualiza o ViewModel
        navController.popBackStack() // Voltar para a tela anterior
    }) {Text("Voltar")}
  }
}
```

- **Resumindo ViewModel**

  - **Instância Única**: O ViewModel deve ser uma instância única durante o ciclo de vida do Activity ou Fragment. Use viewModel() para obter a instância no Jetpack Compose.

  - **Persistência**: O estado dentro do ViewModel persiste durante mudanças de configuração.

  - **Separação de Preocupações**: Mantenha a lógica de negócios e o estado da UI separados da apresentação.

Solução atividade em sala da aula anterior

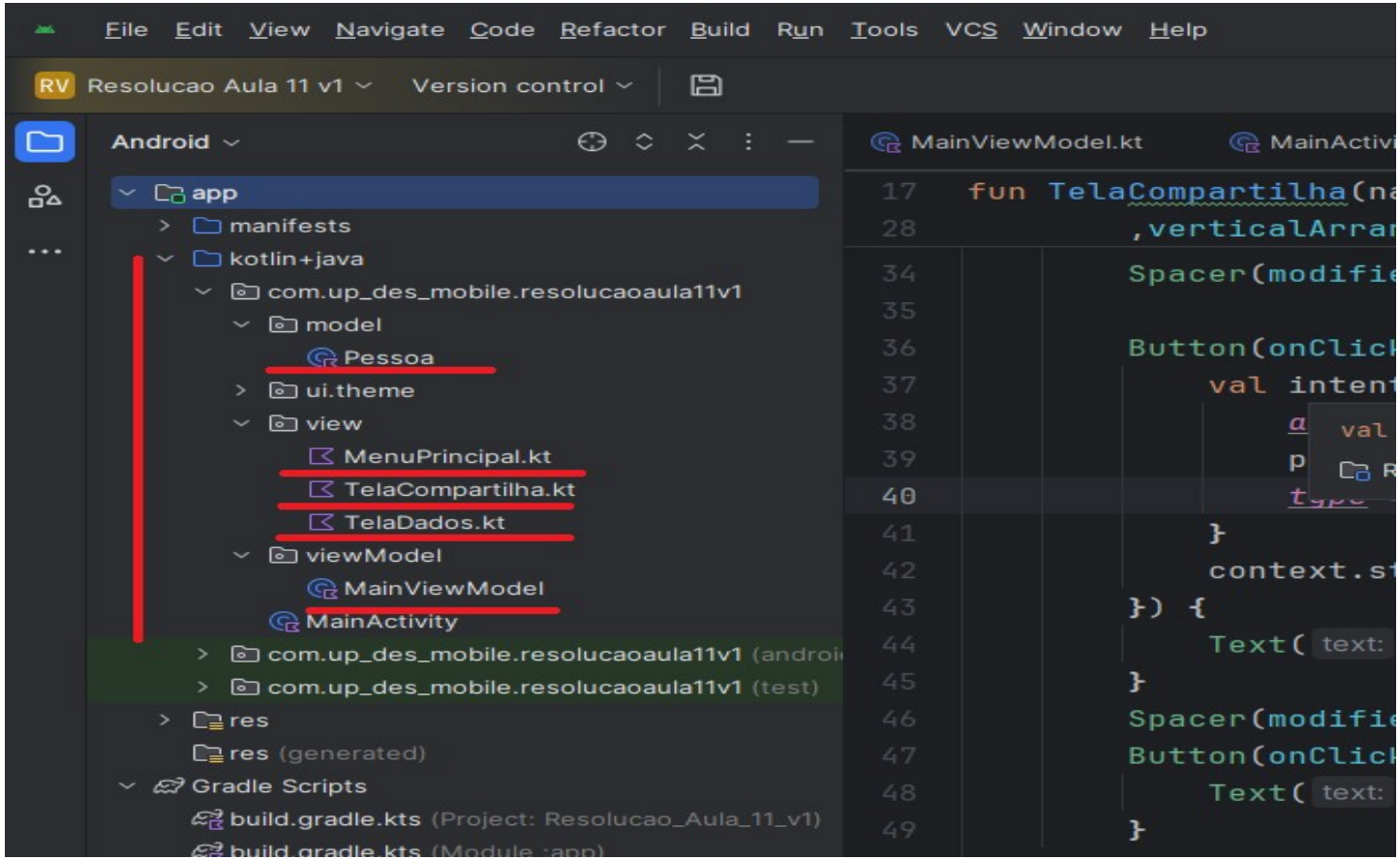
Desenvolver um aplicativo que permitirá ao usuário inserir informações pessoais e compartilhar esses dados com outros aplicativos. A primeira versão do aplicativo deve funcionar como um protótipo funcional básico. Após sucesso, incluir funcionalidades adicionais para tornar o aplicativo mais robusto e responsivo a diferentes situações.

Requisitos do Aplicativo

- Tela de Menu Principal → Opções: "Dados" e "Compartilhamento"
- Tela de Dados → Campos para inserir: Nome, Número de telefone e Descrição do biotipo
- Tela de Compartilhamento → Exibir os dados preenchidos na tela "Dados" e Campo para inserir um texto denominado "Intenção". Botão "Compartilhar" que permitirá compartilhar os dados com o aplicativo escolhido pelo usuário

Universidade POSITIVO

Organização proposta de pastas do projeto



- O primeiro passo seria visualizar e definir uma classe complexa que venha representar os dados pessoais exigidos no enunciado;

- Em seguida, pensar na estrutura de funcionamento do App
  - questão de menus,
  - quantidade e layout de telas
  - e persistência de dados

- Definir uma estratégia que garanta a preservação de dados em situações corriqueiras e específicas;

- Por fim, codificar.

- **Criando a classe de dados Pessoa:**

```
data class Pessoa(
    val nome: String,
    val numeroTelefone: String,
    val descricao: String
)
```

- A classe pessoa será instânciada uma vez, gerando o objeto pessoa;
- O objeto pessoa deverá ter seu conteúdo preservado e,
- O objeto pessoa deverá ser acessível em todos os métodos e funções:
  - Poderíamos usar o objeto pessoa como parâmetro de entrada das funções @Composable
  - Em muitas situações o Compose não aceita dados complexos em parâmetros

- **Criando uma ViewModel:**

```
class MainViewModel : ViewModel() {
    private val _pessoa = mutableStateOf<Pessoa?>(null)
    val pessoa: Pessoa?
        get() = _pessoa.value

    fun setPessoa(pessoa: Pessoa) {
        _pessoa.value = pessoa
    }
}
```

- Pode-se compreender que a utilização de uma ViewModel é a “criação de um container” que conterá e disponibilizará os objetos que nos interessam.

- **Ajuste da MainActivity:**

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            ResolucaoAula11V1Theme {
                Surface(modifier = Modifier.fillMaxSize(), color = MaterialTheme.colors.background) {
                    val viewModel: MainViewModel = viewModel()
                    val navController = rememberNavController()

                    AppNavHost(navController, viewModel)
                }
            }
        }
    }
}
```

- **val viewModel: MainViewModel = viewModel()**
  - Esta linha é usada para obter uma instância do ViewModel no Jetpack Compose.

- **val navController = rememberNavController()**
  - Esta linha é usada para criar um NavController no Jetpack Compose.

- **`AppNavHost(navController, viewModel)`**
  - Chamada da função que fará o “mapeamento” das telas navegáveis, passando como parâmetro navController e viewModel que são valores tratados no conceito de state.

- **MenuPrincipal.kt**

```
@Composable
fun AppNavHost(navController: NavHostController, viewModel: MainViewModel) {
    NavHost(navController = navController, startDestination = "menu") {
        composable("menu") {
            MenuPrincipal(navController, viewModel)
        }
        composable("dados") {
            TelaDados(navController, viewModel)
        }
        composable("compartilha") {
            TelaCompartilha(navController, viewModel)
        }
    }
}
```

- **MenuPrincipal.kt (cont.)**

```
@Composable
fun MenuPrincipal(navController: NavHostController
                  ,viewModel : MainViewModel) {
    val pessoa = viewModel.pessoa

    Column(modifier = Modifier.fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {pessoa?.let {Text(text = it.nome)
            Text(text = it.numeroTelefone)
            Text(text = it.descricao)
        } ?: Text(text = "Não há dados pessoais")
        Button(onClick = { navController.navigate("dados") }) {
            Text("Dados")
        }
        Button(
            onClick = {
                if (pessoa != null) {
                    navController.navigate("compartilha")
                }
            }
        ) {
            Text("Compartilhamento")
        }
    }
}
```

- **TelaDados.kt**

```
@Composable
fun TelaDados(navController: NavHostController
                    , viewModel: MainViewModel)

{
   var nome by remember{mutableStateOf(viewModel.pessoa?.nome ?: "") }
   var numeroTelefone by remember  { mutableStateOf(viewModel.pessoa?.numeroTelefone ?: "") }

   var descricao by remember  { mutableStateOf(viewModel.pessoa?.descricao ?: "") }

   Column(modifier = Modifier.fillMaxSize().padding(16.dp),horizontalAlignment = Alignment.CenterHorizontally,
      verticalArrangement = Arrangement.Center
   ) {Text("Nome:")
      OutlinedTextField(value = nome, onValueChange = { nome = it },
         keyboardOptions = KeyboardOptions.Default.copy(
            keyboardType = KeyboardType.Text,
            imeAction = ImeAction.Next
         )
      )
```

- **TelaDados.kt (cont.)**

```
        Text("Número de telefone:")
        OutlinedTextField( value = numeroTelefone, onValueChange = { numeroTelefone = it },
            keyboardOptions = KeyboardOptions.Default.copy(
                keyboardType = KeyboardType.Phone,
                imeAction = ImeAction.Next
            )
        )
        Text("Descrição do biotipo:")
        OutlinedTextField( value = descricao, onValueChange = { descricao = it },
            keyboardOptions = KeyboardOptions.Default.copy(
                keyboardType = KeyboardType.Text,
                imeAction = ImeAction.Done
            )
        )
        Button(onClick = { val pessoa = Pessoa(nome, numeroTelefone, descricao)
            viewModel.setPessoa(pessoa) // Atualiza a pessoa na ViewModel
            navController.popBackStack() // Voltar para a tela anterior
        }) { Text("Voltar") }
    }
}
```

- **TelaCompartilha.kt**

```
 @Composable
fun TelaCompartilha(navController: NavHostController, viewModel : MainViewModel) {
    var pessoa = viewModel.pessoa
    val context = LocalContext.current
    var intention by remember { mutableStateOf("") }

    val personalData = pessoa?.let {
        "Nome: ${it.nome} \nTelefone: ${it.numeroTelefone} \nDescrição: ${it.descricao}"
    } ?: "Nenhum dado pessoal disponível."

    Column(modifier = Modifier.fillMaxSize().padding(16.dp)
        ,horizontalAlignment = Alignment.CenterHorizontally
        ,verticalArrangement = Arrangement.Center) {
        Text("Dados Pessoais Preenchidos:")
        Text(personalData)
```

- **TelaCompartilha.kt (cont.)**

```
Text("Intenção:")
 TextField(value = intention, onValueChange = { intention = it })

 Button(onClick = { val intent = Intent().apply { action = Intent.ACTION_SEND
        putExtra(Intent.EXTRA_TEXT, "$personalData\nIntenção: $intention")
        type = "text/plain" }


    context.startActivity(Intent.createChooser(intent, "Compartilhe"))
 }) { Text("Compartilhar") }
 Button(onClick = { navController.popBackStack() }) {
    Text("Voltar")
 }
 }
}
```

Atividade em sala de aula (referente aula 11)

- **Versão 2: Protótipo Operacional**
- Passo 4: Habilitação Condicional dos Botões
  - Na tela de "Compartilhamento", desative o botão "Compartilhar" até que o campo "Intenção" contenha texto.
  - Desative o botão "Compartilhamento" na tela de menu principal até que os dados estejam preenchidos na tela "Dados".

Atividade em sala de aula (referente aula 11)

- Dica: Utilize condições baseadas no estado dos campos de texto para habilitar ou desabilitar os botões.

- Passo 5: Manutenção do Estado ao Rotacionar a Tela
  - Assegure-se de que os dados inseridos na tela "Dados" e "Intenção" na tela de "Compartilhamento" sejam mantidos quando o dispositivo for rotacionado.
  - Dica: Utilize rememberSaveable para manter o estado dos campos de texto.

- Versão 3: Funcionalidade Adicional
  - Passo 6: Verificação de Conexão com a Internet
    - Na tela de "Compartilhamento", habilite o botão "Compartilhar" apenas se houver uma conexão com a internet.
    - Caso a internet não esteja disponível, exiba uma mensagem para o usuário informando sobre a falta de conexão.