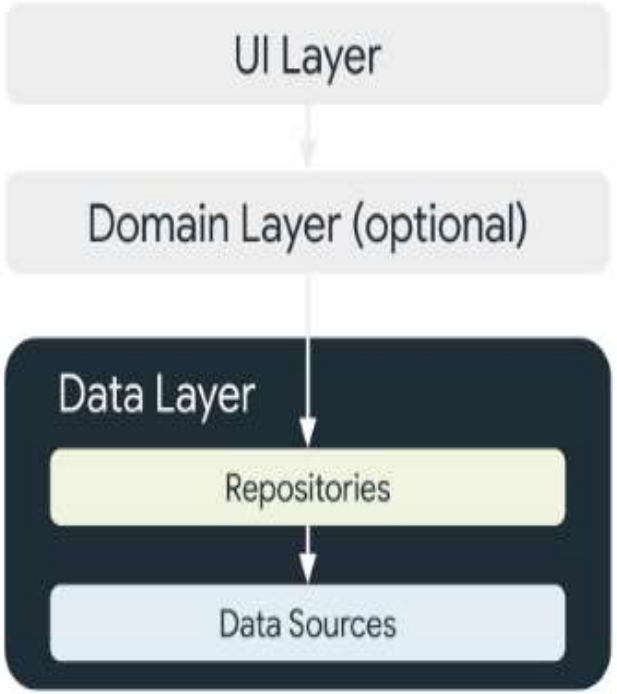
Desenvolvimento Mobile

Professor Maurício Buess

mbuess@up.edu.br

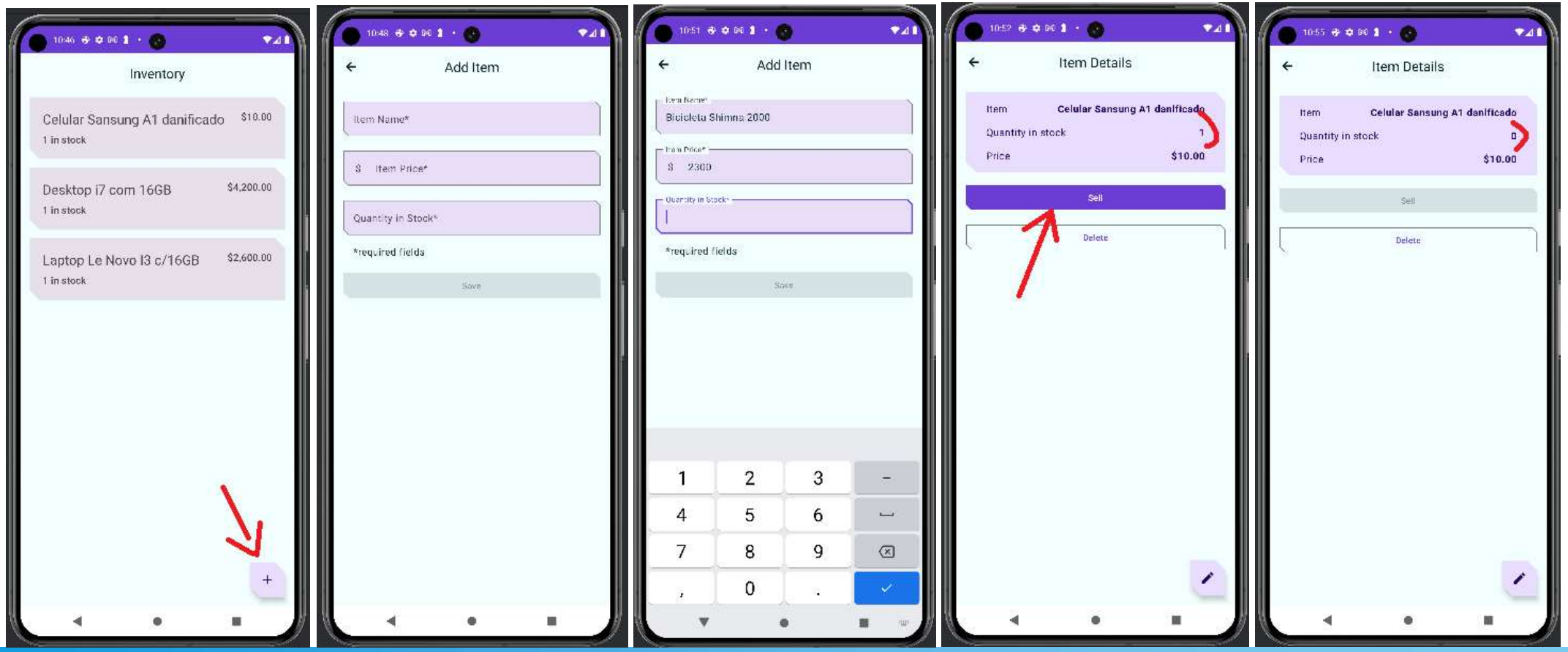
- **Persistência de dados com o Room**

- **Room**
  - Biblioteca de persistência (gravação) de dados que faz parte do JetPack Compose;
  - Camada de abstração sobre o SQLite;
- Camada de abstração → funções que escondem a complexidade (implementação) fornecendo uma interface com um conjunto de funções – no caso, para manipulação do SQLite;
- Room passa a ser considerado como uma fonte de dados



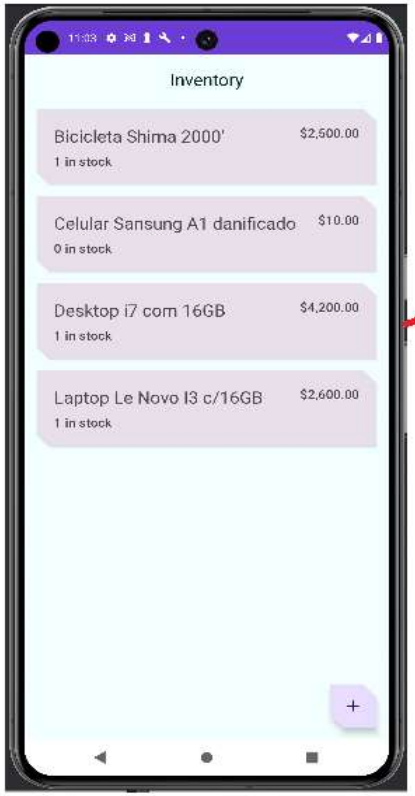
Desenvolvimento Mobile

- App de registro de inventário:



Desenvolvimento Mobile

- Descompacte o arquivo inventory_starter.zip
- A versão inicial (starter) não está concluída pois não grava os dados;

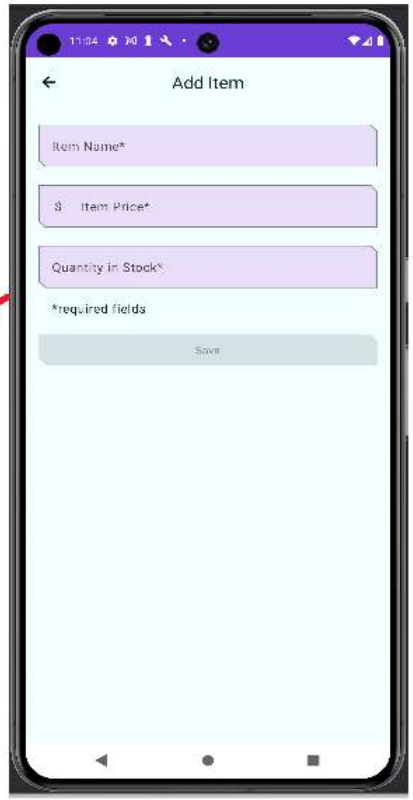


ui/home/HomeScreen.kt

Tela inicial do app, que contém os elementos combináveis para mostrar a lista de inventário.

ui/item/ItemEntryScreen.kt

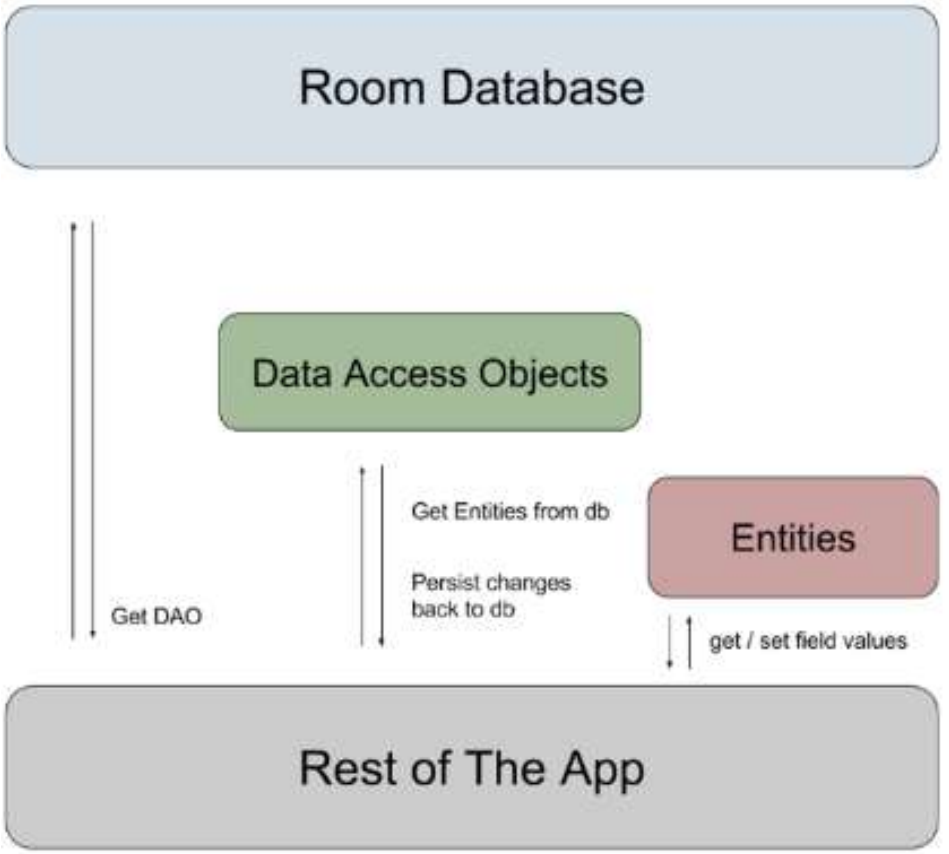
Essa tela é semelhante à ItemEditScreen.kt. Ambas têm campos de texto para os detalhes do item.



Desenvolvimento Mobile

- **Room: 3 componentes importantes**

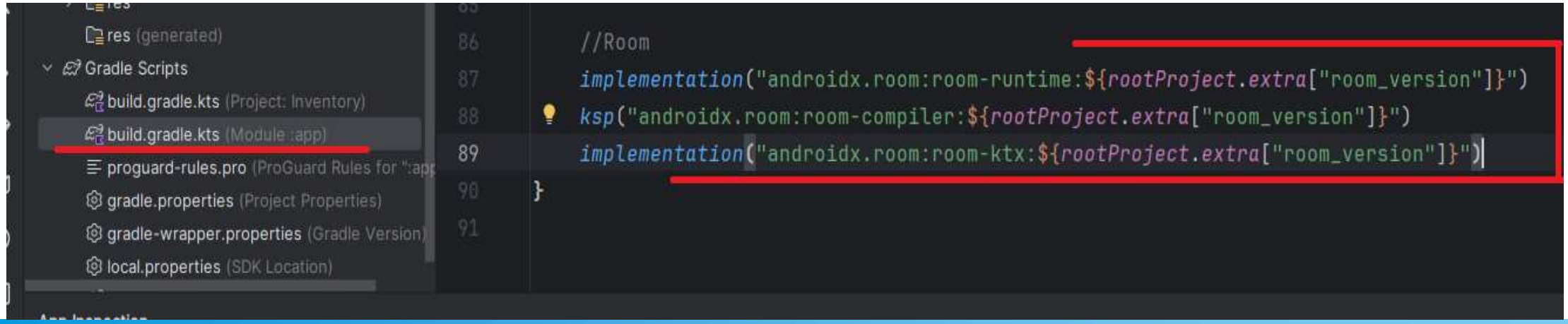
- Room entities → representam as tabelas do B.D.;
- DAOs → (Date Acess Object) oferecem métodos para manipular dados no B.D.;
- Database Class → a classe B.D. do Room oferece ao App instâncias do DAOs associados ao B.D.



Desenvolvimento Mobile

- **App passo a passa**

- Descompacte o arquivo inventory_starter.zip;
- Abra o projeto com a IDE do Android Studio;
- Abra o arquivo build.gradle.kts (app);
- Localize o bloco de dependência e insira o código abaixo:



Desenvolvimento Mobile

- Abra o pacote data no pacote base com.example.inventory
- No pacote data, abra a classe Item do Kotlin, que representa uma entidade de banco de dados no app.

```
class Item(
    val id: Int,
    val name: String,
    val price: Double,
    val quantity: Int
)
```

Classes de dados são usadas para armazenar dados no Kotlin e são definidas com a palavra-chave **data**. Os objetos de classe de dados do Kotlin têm alguns benefícios extras. Por exemplo, o compilador gera utilitários automaticamente para comparar, mostrar e copiar, como toString(), copy() e equals().

- Transforme a classe Item em uma classe de dado (faça isso somente inserindo a palavra **data** antecedendo class.

```
data class Item(
    val id: Int,
    val name: String,
    val price: Double,
    val quantity: Int
)
```

Desenvolvimento Mobile

- Insira as anotações (anotations) e faça as importações requisitadas, deixando o código como na figura abaixo:

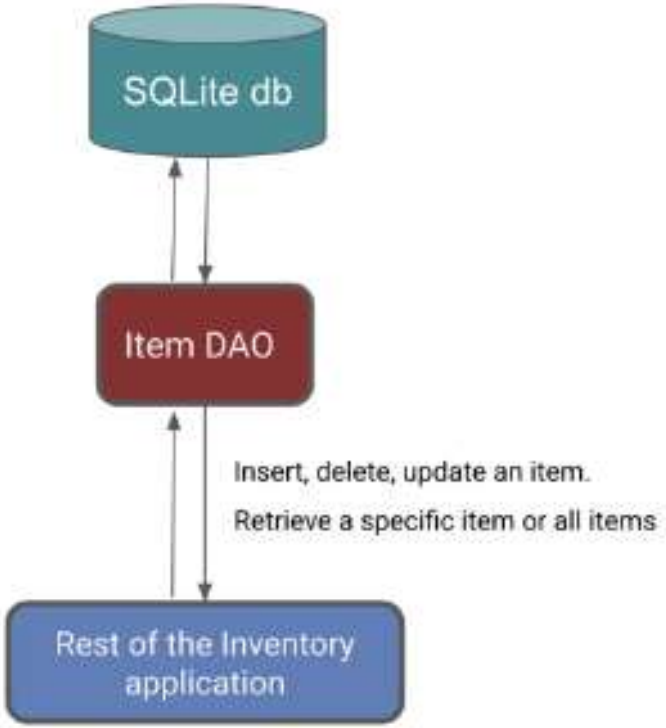
```
17  package com.example.inventory.data
18
19  import androidx.room.Entity
20  import androidx.room.PrimaryKey
21
22
23  /**
24   * Entity data class represents a single row in the database.
25   */
26  @Entity(tableName = "items")
27  data class Item(
28      @PrimaryKey(autoGenerate = true)
29      val id: Int = 0,
30      val name: String,
31      val price: Double,
32      val quantity: Int
33  )
34
```

- **DAO da entidade Item**

- DAO é um padrão usado para separar a camada de persistência do restante do aplicativo, fornecendo uma interface abstrata, conforme o princípio de responsabilidade única.
- DAO oculta e separa o restante do aplicativo de todas as complexidades envolvidas na execução das operações do banco de dados na camada de persistência. Isso permite mudar a camada de dados de maneira independente do código que usa os dados.

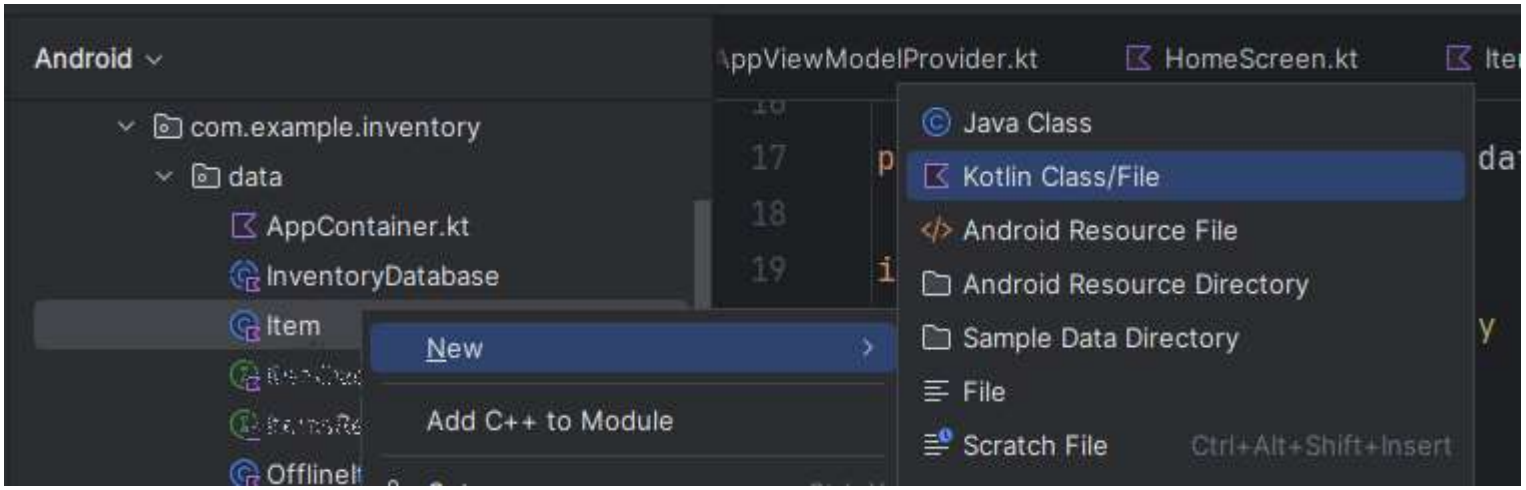
Desenvolvimento Mobile

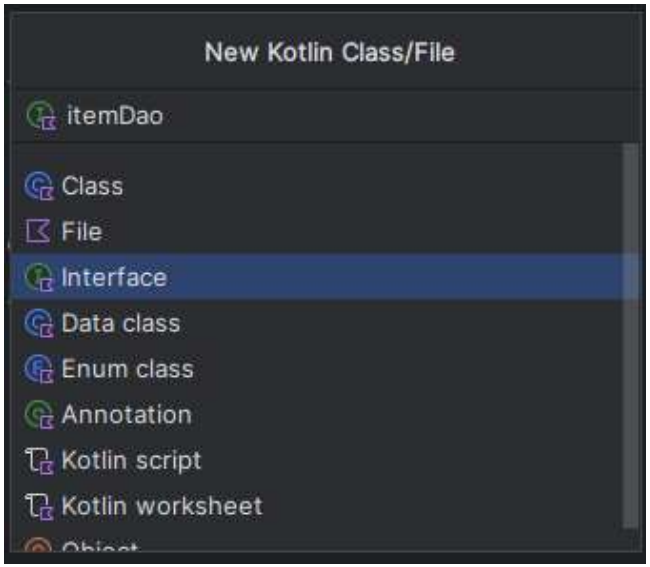
- Para o app de inventário, é necessário fazer o seguinte:
  - Inserir ou adicionar um novo item.
  - Atualizar um item existente para mudar o nome, o preço e a quantidade.
  - Buscar um item específico com base na chave primária id.
  - Buscar todos os itens para poder mostrá-los.
  - Excluir uma entrada do banco de dados.



Desenvolvimento Mobile

- No pacote data, crie a interface Kotlin ItemDao.kt.






```
import androidx.room.Dao

@Dao
interface ItemDao {
}
```

- A seguir o corpo da interface será preenchido:

Desenvolvimento Mobile

```kotlin
import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import androidx.room.Update
import kotlinx.coroutines.flow.Flow
@Dao
interface ItemDao {
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insert(item: Item)

    @Update
    suspend fun update(item: Item)

    @Delete
    suspend fun delete(item: Item)

    @Query("SELECT * FROM items WHERE id = :id")
    fun getItem(id : Int): Flow<Item>

    @Query("SELECT * FROM items ORDER BY name ASC")
    fun getAllItems(): Flow<List<Item>>
}
```

- **Criando instância do B.D.**

- A classe Database fornece ao app instâncias dos DAOs definidos. O app pode usar os DAOs para extrair dados do banco de dados, como instâncias dos objetos da entidade de dados associados. Ele também pode usar as entidades de dados definidas para atualizar linhas das tabelas correspondentes ou criar novas linhas para inserção.
- Será usada a classe RoomDatabase abstrata e será adicionada a anotação @Database a ela. Essa classe tem um método que retorna a instância atual do RoomDatabase caso o banco de dados não exista.

- No pacote data, crie uma classe do Kotlin com o nome InventoryDatabase.kt.
- No arquivo InventoryDatabase.kt, transforme InventoryDatabase em uma classe abstract que estenda o RoomDatabase.
- Adicione a anotação @Database à classe. Desconsidere o erro de parâmetros ausentes que vai ser corrigido na próxima etapa.

```
import androidx.room.Database
import androidx.room.RoomDatabase

@Database
abstract class InventoryDatabase : RoomDatabase() {}
```

- Ajuste a anotação @Database para:

```
@Database(entities = [Item::class], version = 1, exportSchema = false)
```

Desenvolvimento Mobile

- No corpo da classe, declare uma função abstrata que retorne o ItemDao para que o banco de dados saiba sobre o DAO e reproduza o código abaixo:

```
12      companion object {
13          @Volatile
14          private var Instance: InventoryDatabase? = null
15
16          fun getDataBase(context: Context): InventoryDatabase {
17              return Instance ?: synchronized( lock: this) {
18                  Room.databaseBuilder(context, InventoryDatabase::class.java,  name: "item_database")
19                      .fallbackToDestructiveMigration()
20                      .build() InventoryDatabase
21                      .also { Instance = it }
22              }
23          }
24      }
25  }
```

Desenvolvimento Mobile

- **Implementando o repositório**
  - Abra o arquivo ItemsRepository.kt no pacote data.
  - Adicione as funções abaixo à interface, que são mapeadas para a implementação do DAO.

```
1   > / Copyright (C) 2023 The Android Open Source Project .../
16
17    package com.example.inventory.data
18
19    import kotlinx.coroutines.flow.Flow
20
21    /**
22     * Repository that provides insert, update, delete, and
23     * retrieve of [Item] from a given data source.
24     */
25    interface ItemsRepository {
26        fun getItemStream(id : Int): Flow<Item?>
27        fun getAllItemsStream(): Flow<List<Item>>
28
29        suspend fun insertItem(item: Item)
30        suspend fun updateItem(item: Item)
31        suspend fun deleteItem(item: Item)
32    }
```

Desenvolvimento Mobile

- Abra o arquivo OfflineItemsRepository.kt no pacote data e reproduza o código abaixo:

```
1   / Copyright (C) 2023 The Android Open Source Project .../
16
17  package com.example.inventory.data
18
19  import kotlinx.coroutines.flow.Flow
20
21  class OfflineItemsRepository(private val itemDao: ItemDao) : ItemsRepository {
22
23      override fun getAllItemsStream(): Flow<List<Item>> = itemDao.getAllItems()
24      override fun getItemStream(id: Int): Flow<Item?> = itemDao.getItem(id)
25
26      override suspend fun insertItem(item: Item) = itemDao.insert(item)
27
28      override suspend fun updateItem(item: Item) = itemDao.update(item)
29
30      override suspend fun deleteItem(item: Item) = itemDao.delete(item)
31  }
```

Desenvolvimento Mobile

- **Implementar a classe AppContainer**
- Iinstanciar o B.D. e transmiter a instância DAO para a classe OfflineItemsRepository.
- Abra o arquivo AppContainer.kt no pacote data e reproduza o código.

```
24  interface AppContainer {
25      val itemsRepository: ItemsRepository
26  }
27
28  /**
29   * [AppContainer] implementation that provides instance of [OfflineItemsRepository]
30   */
31  class AppDataContainer(private val context: Context) : AppContainer {
32      /**
33       * Implementation for [ItemsRepository]
34       */
35      override val itemsRepository: ItemsRepository by lazy {
36          OfflineItemsRepository(InventoryDatabase.getDataBase(context).itemDao())
37      }
38  }
```

Desenvolvimento Mobile

- Até o momento, foi criado um banco de dados com as classes de interface. Mas, para salvar os dados transitórios do app e também acessar o banco de dados, precisa-se atualizar os ViewModels.
- ViewModels interagem com o banco de dados pelo DAO e fornecem dados à interface.
- Todas as operações do banco de dados precisam ser executadas na linha de execução de interface principal e isso será feito com coroutine e o viewModelScope.
- Coroutine → é um container para operações assíncronas que garante uma forma concisa e eficiente de acesso, permitindo que o código aguarde o resultado de uma operação sem bloquear a execução. São usadas para buscar dados da rede ou executar operações de E/S, enquanto mantém a responsividade da interface do usuário.
- Coroutine → cooperation + routine (rotina colaborativa)

Desenvolvimento Mobile

- Transmissão do repositório para o arquivo ItemEntryViewModel.kt – verifique e reproduza o código:

```
16
17  package com.example.inventory.ui.item
18
19  import ...
26
27  class ItemEntryViewModel(private val itemsRepository: ItemsRepository) : ViewModel() {
28      var itemUiState by mutableStateOf(ItemUiState())
29          private set
30
31      fun updateUiState(itemDetails: ItemDetails) {
32          itemUiState =
33              ItemUiState(itemDetails = itemDetails, isEntryValid = validateInput(itemDetails))
34      }
35
36      private fun validateInput(uiState: ItemDetails = itemUiState.itemDetails): Boolean {
37          return with(uiState) { this: ItemDetails
38              name.isNotBlank() && price.isNotBlank() && quantity.isNotBlank()
39          }
40      }
```

Desenvolvimento Mobile

```
41
42      suspend fun saveItem() {
43          if (validateInput()) {
44              itemsRepository.insertItem(itemUiState.itemDetails.toItem())
45          }
46      }
47  }
48
49  data class ItemUiState(
50      val itemDetails: ItemDetails = ItemDetails(),
51      val isEntryValid: Boolean = false
52  )
53
54  data class ItemDetails(
55      val id: Int = 0,
56      val name: String = "",
57      val price: String = "",
58      val quantity: String = "",
59  )
60
```

Desenvolvimento Mobile

```
60
61  fun ItemDetails.toItem(): Item = Item(
62      id = id,
63      name = name,
64      price = price.toDoubleOrNull() ?: 0.0,
65      quantity = quantity.toIntOrNull() ?: 0
66  )
67
68  fun Item.formatedPrice(): String {
69      return NumberFormat.getCurrencyInstance().format(price)
70  }
71
72  fun Item.toItemUiState(isEntryValid: Boolean = false): ItemUiState = ItemUiState(
73      itemDetails = this.toItemDetails(),
74      isEntryValid = isEntryValid
75  )
76
77  fun Item.toItemDetails(): ItemDetails = ItemDetails(
78      id = id,
79      name = name,
80      price = price.toString(),
81      quantity = quantity.toString()
82  )
```

Desenvolvimento Mobile

- **Atualizando o ViewModel da itemEntry**
- Transmitir o repositório para o arquivo ItemEntryViewModel.kt. e também salvar no B.D. os detalhes do item inseridos na tela Add Item.
- Reproduza o código a seguir:

```
26
27  class ItemEntryViewModel(private val itemsRepository: ItemsRepository) : ViewModel() {
28      var itemUiState by mutableStateOf(ItemUiState())
29          private set
30
31      fun updateUiState(itemDetails: ItemDetails) {
32          itemUiState =
33              ItemUiState(itemDetails = itemDetails, isEntryValid = validateInput(itemDetails))
34      }
35
36      private fun validateInput(uiState: ItemDetails = itemUiState.itemDetails): Boolean {
37          return with(uiState) { this: ItemDetails
38              name.isNotBlank() && price.isNotBlank() && quantity.isNotBlank()
39          }
40      }
```

Desenvolvimento Mobile

```
40      }
41
42      suspend fun saveItem() {
43          if (validateInput()) {
44              itemsRepository.insertItem(itemUiState.itemDetails.toItem())
45          }
46      }
47  }
48
49  data class ItemUiState(
50      val itemDetails: ItemDetails = ItemDetails(),
51      val isEntryValid: Boolean = false
52  )
53
54  data class ItemDetails(
55      val id: Int = 0,
56      val name: String = "",
57      val price: String = "",
58      val quantity: String = "",
59  )
60
```

Desenvolvimento Mobile

```
60
61  fun ItemDetails.toItem(): Item = Item(
62      id = id,
63      name = name,
64      price = price.toDoubleOrNull() ?: 0.0,
65      quantity = quantity.toIntOrNull() ?: 0
66  )
67
68  fun Item.formatedPrice(): String {
69      return NumberFormat.getCurrencyInstance().format(price)
70  }
71
72  fun Item.toItemUiState(isEntryValid: Boolean = false): ItemUiState = ItemUiState(
73      itemDetails = this.toItemDetails(),
74      isEntryValid = isEntryValid
75  )
76
77  fun Item.toItemDetails(): ItemDetails = ItemDetails(
78      id = id,
79      name = name,
80      price = price.toString(),
81      quantity = quantity.toString()
82  )
```

Desenvolvimento Mobile

- Atualize o initializer do modelo de visualização de entrada do item no ui/AppViewModelProvider.kt e transmita a instância do repositório como um parâmetro (verifique e reproduza o código).

```
31  /**
32   * Provides Factory to create instance of ViewModel for the entire Inventory app
33   */
34  object AppViewModelProvider {
35      val Factory = viewModelFactory { this: InitializerViewModelFactoryBuilder
36          // Initializer for ItemEditViewModel
37          initializer { this: CreationExtras
38              ItemEditViewModel(
39                  this.createSavedStateHandle(),
40                  inventoryApplication().container.itemsRepository
41              )
42          }
43          // Initializer for ItemEntryViewModel
44          initializer { this: CreationExtras
45              ItemEntryViewModel(inventoryApplication().container.itemsRepository)
46          }
47
```

Desenvolvimento Mobile

```
47
48              // Initializer for ItemDetailsViewModel
49              initializer { this: CreationExtras
50                  ItemDetailsViewModel(
51                      this.createSavedStateHandle(),
52                      inventoryApplication().container.itemsRepository
53                  )
54              }
55
56              // Initializer for HomeViewModel
57              initializer { this: CreationExtras
58                  HomeViewModel(inventoryApplication().container.itemsRepository)
59              }
60          }
61      }
62
63      /**
64       * Extension function to queries for [Application] object and returns an instance of
65       * [InventoryApplication].
66       */
67      fun CreationExtras.inventoryApplication(): InventoryApplication =
68          (this[AndroidViewModelFactory.APPLICATION_KEY] as InventoryApplication)
69
```

Desenvolvimento Mobile

- **Tutorial do elemento ItemEntryBody()**
- No arquivo ui/item/ItemEntryScreen.kt, o elemento ItemEntryBody() é implementado , observe esse elemento ItemEntryBody() na chamada de função ItemEntryScreen() - compare e reproduza o código

```
17  package com.example.inventory.ui.item
18
19  > import ...
52
53  object ItemEntryDestination : NavigationDestination {
54      override val route = "item_entry"
55      override val titleRes = "Add Item"
56  }
57
58  @OptIn(ExperimentalMaterial3Api::class)
59  @Composable
60  fun ItemEntryScreen(
61      navigateBack: () -> Unit,
62      onNavigateUp: () -> Unit,
63      canNavigateBack: Boolean = true,
64      viewModel: ItemEntryViewModel = viewModel(factory = AppViewModelProvider.Factory)
65  ) {
66      val coroutineScope = rememberCoroutineScope()
67
```

Desenvolvimento Mobile

```
67
68      Scaffold(
69          topBar = {
70              InventoryTopAppBar(
71                  title = stringResource(ItemEntryDestination.titleRes),
72                  canNavigateBack = canNavigateBack,
73                  navigateUp = onNavigateUp
74              )
75          }
76      ) { innerPadding ->
77          ItemEntryBody(
78              itemUiState = viewModel.itemUiState,
79              onItemValueChange = viewModel::updateUiState,
80              onSaveClick = {
81                  coroutineScope.launch { this: CoroutineScope
82                      viewModel.saveItem()
83                      navigateBack()
84                  }
85              },
```

Desenvolvimento Mobile

```
85              },
86              modifier = Modifier
87                  .padding(
88                      start = innerPadding.calculateStartPadding(LocalLayoutDirection.current),
89                      end = innerPadding.calculateEndPadding(LocalLayoutDirection.current),
90                      top = innerPadding.calculateTopPadding()
91                  )
92                  .verticalScroll(rememberScrollState())
93                  .fillMaxWidth()
94          )
95      }
96  }
97
98  @Composable
99  fun ItemEntryBody(
100     itemUiState: ItemUiState,
101     onItemValueChange: (ItemDetails) -> Unit,
102     onSaveClick: () -> Unit,
103     modifier: Modifier = Modifier
104 ) {
```

Desenvolvimento Mobile

```
105     Column(
106         verticalArrangement = Arrangement.spacedBy(dimensionResource(id = 20dp)),
107         modifier = modifier.padding(dimensionResource(id = 16dp))
108         ) { this: ColumnScope
109         ItemInputForm(
110             itemDetails = itemUiState.itemDetails,
111             onValueChange = onItemValueChange,
112             modifier = Modifier.fillMaxWidth()
113         )
114
115         Button(
116
117             onClick = onSaveClick,
118             enabled = itemUiState.isEntryValid,
119             shape = MaterialTheme.shapes.small,
120             modifier = Modifier.fillMaxWidth()
121         ) { this: RowScope
122             Text(text = stringResource("Save"))
123         }
124     }
125 }
```

- **Mais informações / Detalhes & Créditos**

- O material aqui apresentado foi retirado do seguinte endereço web:

https://developer.android.com/codelabs/basic-android-kotlin-compose-persisting-data-room#0