Agora, no **Passo 6**, vamos implementar a **persistência de dados usando o Room**, que permitirá ao aplicativo salvar e carregar os itens do carrinho de compras em um banco de dados local. Assim, mesmo que o aplicativo seja fechado, os dados do carrinho serão preservados. Vamos detalhar cada etapa da implementação.

Passo 6: Persistência de Dados com Room

6.1 Configurando o Room no Gradle

Antes de usarmos o Room, precisamos garantir que as dependências necessárias estão configuradas no arquivo `build.gradle`.

1. **Abra o arquivo** `build.gradle (Module: app)` e certifique-se de que as seguintes dependências estão adicionadas:

gradle

Copiar código

```
dependencies { // Room para persistência de dados local implementation "androidx.room:room-runtime:2.4.3" kapt
"androidx.room:room-compiler:2.4.3" // Kotlin Coroutines para operações assíncronas implementation
'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.5.2' implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.5.2' }
```

2. **Sincronize o projeto**:
   - Depois de adicionar as dependências, clique em "**Sync Now**" para sincronizar o projeto.

6.2 Criando a Entidade `CartItem`

O `CartItem` será a representação de cada item do carrinho que será armazenado no banco de dados.

1. **Crie a classe** `CartItem.kt`:
   - No pacote `com.example.acaiStore`, crie um novo arquivo Kotlin chamado `CartItem.kt`.
2. **Código da Entidade** `CartItem.kt`:

kotlin

Copiar código

```
package com.example.acaiStore.data import androidx.room.Entity import androidx.room.PrimaryKey @Entity(tableName = "cart_items")
data class CartItem( @PrimaryKey(autoGenerate = true) val id: Int = 0, val name: String, val description: String, val price:
```

```
Double, val imageUrl: String, val quantity: Int )
```

Aqui, estamos criando uma entidade `CartItem`, que contém os dados de um item no carrinho, como o nome, descrição, preço, imagem e quantidade.

6.3 Criando o DAO (Data Access Object)

O DAO é a interface responsável pelas operações de acesso ao banco de dados, como inserir, excluir e consultar itens.

1. **Crie a interface** `CartDao.kt`:
   - No pacote `com.example.acaiStore.data`, crie um novo arquivo Kotlin chamado `CartDao.kt`.
2. **Código da Interface** `CartDao.kt`:

kotlin

Copiar código

```
package com.example.acaiStore.data import androidx.room.Dao import androidx.room.Insert import androidx.room.Query @Dao interface CartDao { // Inserir um item no carrinho @Insert suspend fun insertItem(item: CartItem) // Deletar todos os itens do carrinho (ao finalizar a compra, por exemplo) @Query("DELETE FROM cart_items") suspend fun clearCart() // Buscar todos os itens do carrinho @Query("SELECT * FROM cart_items") suspend fun getAllItems(): List<CartItem> }
```

Aqui, usamos três operações principais:

- `insertItem`: Insere um item no banco de dados.
- `clearCart`: Remove todos os itens do banco de dados.
- `getAllItems`: Retorna todos os itens do carrinho armazenados no banco de dados.

6.4 Configurando o Banco de Dados Room

Agora precisamos criar uma classe que estenda o `RoomDatabase` para configurar o banco de dados.

1. **Crie a classe** `AppDatabase.kt`:
   - No pacote `com.example.acaiStore.data`, crie um novo arquivo Kotlin chamado `AppDatabase.kt`.

2. **Código da Classe** `AppDatabase.kt`:

kotlin

Copiar código

```
package com.example.acaiStore.data import android.content.Context import androidx.room.Database import androidx.room.Room import androidx.room.RoomDatabase @Database(entities = [CartItem::class], version = 1) abstract class AppDatabase : RoomDatabase() { abstract fun cartDao(): CartDao companion object { @Volatile private var INSTANCE: AppDatabase? = null fun getDatabase(context: Context): AppDatabase { return INSTANCE ?: synchronized(this) { val instance = Room.databaseBuilder( context.applicationContext, AppDatabase::class.java, "acai_store_db" ).build() INSTANCE = instance instance } } } }
```

A classe `AppDatabase` é o ponto de acesso ao banco de dados. O método `getDatabase` retorna uma instância do banco de dados usando o padrão Singleton para garantir que só uma instância do banco de dados será criada.

6.5 Atualizando o `CartManager` para Usar Room

Agora, vamos atualizar o `CartManager` para salvar e carregar os itens do carrinho usando o Room.

1. **Abra ou crie a classe** `CartManager.kt` e atualize-a para usar o banco de dados Room:

kotlin

Copiar código

```
package com.example.acaiStore import android.content.Context import com.example.acaiStore.data.AppDatabase import com.example.acaiStore.data.CartItem import kotlinx.coroutines.CoroutineScope import kotlinx.coroutines.Dispatchers import kotlinx.coroutines.launch import kotlinx.coroutines.withContext object CartManager { private var cartItems = mutableListOf<CartItem>() // Função para adicionar um item ao carrinho e salvar no banco de dados fun addToCart(context: Context, product: AcaiProduct, quantity: Int = 1) { val newItem = CartItem( name = product.name, description = product.description, price = product.price, imageUrl = product.imageUrl, quantity = quantity ) cartItems.add(newItem) // Inserir o item no banco de dados CoroutineScope(Dispatchers.IO).launch { val cartDao = AppDatabase.getDatabase(context).cartDao() cartDao.insertItem(newItem) } } // Função para carregar os itens do banco de dados fun loadCartItems(context: Context, onCartLoaded: (List<CartItem>) -> Unit) { CoroutineScope(Dispatchers.IO).launch { val cartDao = AppDatabase.getDatabase(context).cartDao() val items = cartDao.getAllItems() withContext(Dispatchers.Main) { cartItems = items.toMutableList() onCartLoaded(items) } } } // Função para limpar o carrinho fun clearCart(context: Context) { cartItems.clear() // Limpar o banco de dados CoroutineScope(Dispatchers.IO).launch { val cartDao = AppDatabase.getDatabase(context).cartDao() cartDao.clearCart() } } //
```

```
Função para obter os itens do carrinho fun getCartItems(): List<CartItem> = cartItems // Função para calcular o preço total do carrinho fun getTotalPrice(): Double = cartItems.sumOf { it.price * it.quantity } }
```

Aqui, atualizamos o `CartManager` para:

- Usar corrotinas para inserir e carregar os itens do banco de dados de forma assíncrona.
- Adicionar um produto ao carrinho e salvar no banco de dados.
- Carregar os itens do banco de dados ao abrir o carrinho.

6.6 Atualizando o `CartActivity` para Carregar Itens do Banco de Dados

Agora, vamos atualizar a `CartActivity` para carregar os itens do carrinho a partir do banco de dados quando o usuário abrir a tela.

1. **Abra ou crie a classe** `CartActivity.kt` e atualize-a para carregar os itens do banco de dados:

kotlin

Copiar código

```
package com.example.acaiStore import android.os.Bundle import android.widget.Button import android.widget.TextView import androidx.appcompat.app.AppCompatActivity import androidx.recyclerview.widget.LinearLayoutManager import androidx.recyclerview.widget.RecyclerView import com.example.acaiStore.data.CartItem class CartActivity : AppCompatActivity() { private lateinit var recyclerView: RecyclerView private lateinit var totalPriceTextView: TextView private lateinit var checkoutButton: Button override fun onCreate(savedInstanceState: Bundle?) { super.onCreate(savedInstanceState) setContentView(R.layout.activity_cart) recyclerView = findViewById(R.id.cartRecyclerView) totalPriceTextView = findViewById(R.id.totalPrice) checkoutButton = findViewById(R.id.checkoutButton) // Carregar itens do carrinho do banco de dados CartManager.loadCartItems(this) { cartItems -> setupRecyclerView(cartItems) updateTotalPrice(cartItems) } // Finalizar compra e limpar o carrinho checkoutButton.setOnClickListener { CartManager.clearCart(this) finish() // Fecha a tela após limpar o carrinho } } // Configura o RecyclerView com os itens do carrinho private fun setupRecyclerView(cartItems: List<CartItem>) { recyclerView.layoutManager = LinearLayoutManager(this) recyclerView.adapter = CartAdapter(cartItems) } // Atualiza o valor total dos itens no carrinho private fun updateTotalPrice(cartItems: List<CartItem>) { val totalPrice = cartItems.sumOf { it.price * it.quantity } totalPriceTextView.text = "Total: R$ $totalPrice" } }
```

Aqui, usamos a função `CartManager.loadCartItems()` para carregar os itens do banco de dados e exibi-los na `CartActivity`. Quando o usuário finaliza a compra, o carrinho é limpo tanto na memória quanto no banco de dados.

6.7 Criando o Adapter para o Carrinho (`CartAdapter`)

Agora, vamos criar um `RecyclerView.Adapter` para exibir os itens do carrinho.

1. **Crie a classe** `CartAdapter.kt`:

kotlin

Copiar código

```kotlin
package com.example.acaiStore import android.view.LayoutInflater import android.view.View import android.view.ViewGroup import android.widget.ImageView import android.widget.TextView import androidx.recyclerview.widget.RecyclerView import com.bumptech.glide.Glide import com.example.acaiStore.data.CartItem class CartAdapter( private val cartItems: List<CartItem> ) : RecyclerView.Adapter<CartAdapter.CartViewHolder>() { override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CartViewHolder { val view = LayoutInflater.from(parent.context) .inflate(R.layout.item_acai, parent, false) return CartViewHolder(view) } override fun onBindViewHolder(holder: CartViewHolder, position: Int) { val cartItem = cartItems[position] holder.bind(cartItem) } override fun getItemCount(): Int = cartItems.size class CartViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) { private val productImage: ImageView = itemView.findViewById(R.id.productImage) private val productName: TextView = itemView.findViewById(R.id.productName) private val productDescription: TextView = itemView.findViewById(R.id.productDescription) private val productPrice: TextView = itemView.findViewById(R.id.productPrice) fun bind(cartItem: CartItem) { productName.text = cartItem.name productDescription.text = cartItem.description productPrice.text = "R$ ${cartItem.price} x ${cartItem.quantity}" // Carregar a imagem do produto usando Glide Glide.with(itemView.context) .load(cartItem.imageUrl) .into(productImage) } } }
```

6.8 Testando a Persistência de Dados

Agora que o Room foi integrado, podemos testar o aplicativo para garantir que os itens adicionados ao carrinho sejam salvos e recuperados corretamente.

Resumo

Com isso, o aplicativo agora tem persistência de dados com Room, permitindo que os itens do carrinho sejam armazenados localmente no dispositivo e carregados quando necessário. A funcionalidade de adicionar, remover e salvar o estado do carrinho está completa.