

Aqui está o **Passo 2** detalhado para configurar as dependências necessárias no `build.gradle` e organizar as bases de dados e bibliotecas do projeto.

Passo 2: Configuração das Dependências e Organização


2.1 Abrindo o `build.gradle`

O arquivo `build.gradle` é usado para gerenciar as bibliotecas que o projeto vai utilizar. Aqui, vamos adicionar bibliotecas essenciais como Retrofit (para chamadas de rede), Glide (para carregamento de imagens), e Room (para persistência de dados local).

1. **Localize o arquivo `build.gradle` (Module: app) :**
 - No painel esquerdo do Android Studio, vá até **Gradle Scripts** e clique no arquivo **`build.gradle` (Module: app)**.
2. **Adicionar as Dependências Necessárias**

No arquivo `build.gradle` (Module: app) , localize a seção `dependencies {}` e adicione as seguintes dependências:

gradle

 Copiar código

```
dependencies { // Retrofit para fazer chamadas HTTP implementation
'com.squareup.retrofit2:retrofit:2.9.0' implementation
'com.squareup.retrofit2:converter-gson:2.9.0' // Glide para carregar imagens de
produtos, banners, etc. implementation 'com.github.bumptech.glide:glide:4.12.0'
annotationProcessor 'com.github.bumptech.glide:compiler:4.12.0' // Room para
persistência local de dados implementation "androidx.room:room-runtime:2.4.3" kapt
"androidx.room:room-compiler:2.4.3" // Coroutines para operações assíncronas
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.5.2' implementation
'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.5.2' // ConstraintLayout para
criar layouts responsivos implementation
'androidx.constraintlayout:constraintlayout:2.1.0' // RecyclerView para exibir listas
de produtos implementation 'androidx.recyclerview:recyclerview:1.2.1' // Dependências
do AndroidX necessárias implementation 'androidx.core:core-ktx:1.7.0' implementation
'androidx.appcompat:appcompat:1.4.0' implementation
'com.google.android.material:material:1.5.0' implementation
'androidx.lifecycle:lifecycle-runtime-ktx:2.4.0' }
```

2.2 Sincronizando o Projeto com as Dependências

1. Depois de adicionar as dependências, clique no botão **"Sync Now"** que aparecerá no topo da janela do Android Studio. Isso garantirá que o projeto seja sincronizado com todas as bibliotecas e dependências adicionadas.
2. Quando a sincronização for concluída, o Android Studio baixará automaticamente as bibliotecas e as integrará ao seu projeto.

2.3 Configurando Retrofit para Chamadas de API

Agora que o Retrofit foi adicionado como uma dependência, vamos configurá-lo para fazer chamadas de API (caso o aplicativo precise se conectar a um backend para obter informações sobre

os produtos).

1. Criando o Client do Retrofit:

- Crie uma classe Kotlin chamada `RetrofitClient.kt` dentro do pacote `com.example.acaiStore.network` para gerenciar as chamadas de API:

kotlin

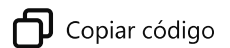


```
package com.example.acaiStore.network import retrofit2.Retrofit import
retrofit2.converter.gson.GsonConverterFactory object RetrofitClient { private
const val BASE_URL = "https://api.acaiStore.com" // Troque pela URL real da API
val instance: Retrofit by lazy { Retrofit.Builder() .baseUrl(BASE_URL)
.addConverterFactory(GsonConverterFactory.create()) .build() } }
```

2. Criando a Interface da API:

- Crie uma interface chamada `AcaiApiService.kt` que definirá os endpoints da API:

kotlin

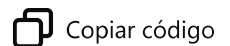


```
package com.example.acaiStore.network import retrofit2.Call import
retrofit2.http.GET // Supondo que a API retorne uma lista de produtos interface
AcaiApiService { @GET("/products") fun getProducts(): Call<List<AcaiProduct>> }
```

3. Usando Retrofit no MainActivity.kt :

- Agora, no arquivo `MainActivity.kt`, podemos usar Retrofit para fazer a requisição dos produtos:

kotlin



```
package com.example.acaiStore import android.os.Bundle import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity import
com.example.acaiStore.network.AcaiApiService import
com.example.acaiStore.network.RetrofitClient import retrofit2.Call import
retrofit2.Callback import retrofit2.Response class MainActivity :
AppCompatActivity() { override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState) setContentView(R.layout.activity_main) //
Inicializando Retrofit val apiService =
RetrofitClient.instance.create(AcaiApiService::class.java) // Fazendo a chamada à
API apiService.getProducts().enqueue(object : Callback<List<AcaiProduct>> {
override fun onResponse( call: Call<List<AcaiProduct>>, response:
Response<List<AcaiProduct>> ) { if (response.isSuccessful) { val products =
response.body() ?: emptyList() // Atualizar a UI com os produtos recebidos } else
{ Toast.makeText(this@MainActivity, "Erro ao buscar produtos",
Toast.LENGTH_SHORT).show() } } override fun onFailure(call:
Call<List<AcaiProduct>>, t: Throwable) { Toast.makeText(this@MainActivity, "Falha
na comunicação com a API", Toast.LENGTH_SHORT).show() } }) }
```

2.4 Configurando Glide para Carregamento de Imagens

Com Glide configurado no `build.gradle`, você pode usá-lo para carregar imagens de produtos em seu layout.

1. Usando Glide em um `ImageView`:

- No `RecyclerView.Adapter`, onde você exibe a lista de produtos, use Glide para carregar a imagem do produto diretamente de uma URL:

kotlin

 Copiar código

```
import com.bumptech.glide.Glide class AcaiAdapter(private val products:
List<AcaiProduct>) : RecyclerView.Adapter<AcaiAdapter.AcaiViewHolder>() { override
fun onCreateViewHolder(parent: ViewGroup, viewType: Int): AcaiViewHolder { val
view = LayoutInflater.from(parent.context) .inflate(R.layout.item_acai, parent,
false) return AcaiViewHolder(view) } override fun onBindViewHolder(holder:
AcaiViewHolder, position: Int) { val product = products[position]
holder.bind(product) } override fun getItemCount(): Int = products.size class
AcaiViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) { private val
productImageView: ImageView = itemView.findViewById(R.id.productImage) fun
bind(product: AcaiProduct) { // Usando Glide para carregar a imagem do produto
Glide.with(itemView.context) .load(product.imageUrl) .into(productImageView) } } }
```

2.5 Configurando Room para Persistência de Dados Local

Se você deseja que o aplicativo armazene dados localmente (como histórico de pedidos ou carrinho), Room é uma excelente biblioteca para isso.

1. Criando a Entidade `CartItem`:

- Crie uma classe Kotlin chamada `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 price:
Double, val quantity: Int )
```

2. Criando o DAO (Data Access Object):

- Crie uma interface `CartDao.kt` que gerenciará as interações com o banco de dados:

kotlin

 Copiar código

```
package com.example.acaiStore.data import androidx.room.Dao import
androidx.room.Insert import androidx.room.Query @Dao interface CartDao { @Insert
fun insertItem(item: CartItem) @Query("SELECT * FROM cart_items") fun
getAllItems(): List<CartItem> }
```

3. Configurando o Banco de Dados Room:

- Crie uma classe `AppDatabase.kt` para definir o banco de dados:

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 } } }
```

Agora você tem as dependências configuradas, e as bibliotecas Retrofit, Glide, e Room prontas para serem usadas no desenvolvimento do seu aplicativo de loja de açaí!