

Agora vamos detalhar o **Passo 3**, que se concentra no desenvolvimento do backend utilizando **Kotlin** para a lógica do aplicativo. Este passo abrange a criação de modelos de dados, gerenciamento de banco de dados com **Room**, comunicação com APIs externas usando **Retrofit**, e o gerenciamento de sessão com **SharedPreferences**.


3. Desenvolvimento do Backend (Kotlin)

3.1 Modelo de Dados (Data Classes)

Primeiro, definimos as classes de dados para representar as entidades principais do aplicativo: **Cliente**, **Barbeiro**, **Serviço** e **Agendamento**.

Cliente.kt :


kotlin

 Copiar código

```
data class Cliente( val id: Int, val nome: String, val email: String, val telefone: String )
```

Barbeiro.kt :


kotlin

 Copiar código

```
data class Barbeiro( val id: Int, val nome: String, val especialidade: String, val avaliacao: Float )
```

Servico.kt :

kotlin

 Copiar código

```
data class Servico( val id: Int, val nome: String, val preco: Double, val descricao: String )
```

Agendamento.kt :

kotlin

 Copiar código

```
data class Agendamento( val id: Int, val clienteId: Int, val barbeiroId: Int, val servicoId: Int, val data: String, val horario: String )
```

Essas classes representam as entidades fundamentais do sistema e serão utilizadas para armazenar e manipular dados no aplicativo.

3.2 Banco de Dados Local com Room

Agora, configuramos o banco de dados local usando o **Room**. O Room facilita o armazenamento local de dados no dispositivo Android e fornece uma abstração simples sobre o SQLite.

3.2.1 Criação de Entidades

Vamos transformar as classes de dados em entidades do Room. As entidades serão usadas para criar tabelas no banco de dados local.

ClienteEntity.kt :


kotlin

 Copiar código

```
import androidx.room.Entity import androidx.room.PrimaryKey @Entity(tableName =
"clientes") data class ClienteEntity( @PrimaryKey(autoGenerate = true) val id: Int,
val nome: String, val email: String, val telefone: String )
```

BarbeiroEntity.kt :


kotlin

 Copiar código

```
import androidx.room.Entity import androidx.room.PrimaryKey @Entity(tableName =
"barbeiros") data class BarbeiroEntity( @PrimaryKey(autoGenerate = true) val id: Int,
val nome: String, val especialidade: String, val avaliacao: Float )
```

AgendamentoEntity.kt :

kotlin

 Copiar código


```
import androidx.room.Entity import androidx.room.PrimaryKey @Entity(tableName =
"agendamentos") data class AgendamentoEntity( @PrimaryKey(autoGenerate = true) val id:
Int, val clienteId: Int, val barbeiroId: Int, val servicoId: Int, val data: String,
val horario: String )
```

3.2.2 DAO (Data Access Object)

Agora, vamos criar as interfaces DAO, que definem os métodos para acessar e modificar os dados nas tabelas.

ClienteDao.kt :


kotlin

 Copiar código

```
import androidx.room.Dao import androidx.room.Insert import androidx.room.Query @Dao
interface ClienteDao { @Insert suspend fun inserirCliente(cliente: ClienteEntity)
@Query("SELECT * FROM clientes WHERE id = :id") suspend fun buscarClientePorId(id:
Int): ClienteEntity }
```

BarbeiroDao.kt :

kotlin

 Copiar código

```
import androidx.room.Dao import androidx.room.Insert import androidx.room.Query @Dao
interface BarbeiroDao { @Insert suspend fun inserirBarbeiro(barbeiro: BarbeiroEntity)
@Query("SELECT * FROM barbeiros WHERE id = :id") suspend fun buscarBarbeiroPorId(id:
Int): BarbeiroEntity }
```

AgendamentoDao.kt :

kotlin

 Copiar código

```
import androidx.room.Dao import androidx.room.Insert import androidx.room.Query @Dao
interface AgendamentoDao { @Insert suspend fun inserirAgendamento(agendamento:
AgendamentoEntity) @Query("SELECT * FROM agendamentos WHERE clienteId = :clienteId")
suspend fun buscarAgendamentosPorClienteId(clienteId: Int): List<AgendamentoEntity> }
```

3.2.3 Configuração do Banco de Dados

Agora, criamos a classe `AppDatabase`, que é responsável por fornecer uma instância do banco de dados.

`AppDatabase.kt` :

kotlin

 Copiar código

```
import android.content.Context import androidx.room.Database import androidx.room.Room
import androidx.room.RoomDatabase @Database(entities = [ClienteEntity::class,
BarbeiroEntity::class, AgendamentoEntity::class], version = 1) abstract class
AppDatabase : RoomDatabase() { abstract fun clienteDao(): ClienteDao abstract fun
barbeiroDao(): BarbeiroDao abstract fun agendamentoDao(): AgendamentoDao 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,
"barbearia_db" ).build() INSTANCE = instance instance } } }
```

3.3 API de Comunicação com Retrofit

Se o aplicativo precisa se comunicar com um servidor, usamos o **Retrofit** para gerenciar as chamadas de API.

3.3.1 Criação do Cliente Retrofit

Primeiro, adicionamos a dependência do Retrofit no `build.gradle` :

kotlin

 Copiar código

```
implementation 'com.squareup.retrofit2:retrofit:2.9.0' implementation
'com.squareup.retrofit2:converter-gson:2.9.0'
```

Agora, criamos o cliente Retrofit:

`ApiClient.kt` :

kotlin

 Copiar código

```
import retrofit2.Retrofit import retrofit2.converter.gson.GsonConverterFactory object
ApiClient { private const val BASE_URL = "https://api.exemplo.com/" val retrofit:
Retrofit by lazy { Retrofit.Builder() .baseUrl(BASE_URL)
.addConverterFactory(GsonConverterFactory.create()) .build() } }
```

3.3.2 Definir os Endpoints da API

Agora, criamos uma interface para definir as rotas da API:

BarbeariaApi.kt :

kotlin

 Copiar código

```
import retrofit2.Call import retrofit2.http.GET import retrofit2.http.Path interface
BarbeariaApi { @GET("servicos") fun buscarServicos(): Call<List<Servico>>
@GET("clientes/{id}") fun buscarCliente(@Path("id") id: Int): Call<Cliente> }
```

3.4 Gerenciamento de Sessões com SharedPreferences


O **SharedPreferences** é usado para armazenar informações persistentes, como o estado de login do usuário.

3.4.1 Salvar Sessão do Usuário

Crie uma classe para gerenciar as preferências:

SessionManager.kt :

kotlin

 Copiar código

```
import android.content.Context import android.content.SharedPreferences class
SessionManager(context: Context) { private val sharedPreferences: SharedPreferences =
context.getSharedPreferences("user_session", Context.MODE_PRIVATE) fun
salvarSessaoUsuario(clienteId: Int) { val editor = sharedPreferences.edit()
editor.putInt("CLIENTE_ID", clienteId) editor.apply() } fun buscarSessaoUsuario(): Int
{ return sharedPreferences.getInt("CLIENTE_ID", -1) } fun limparSessao() { val editor
= sharedPreferences.edit() editor.clear() editor.apply() } }
```

3.5 Implementando Funcionalidades no MainActivity

Por exemplo, vamos adicionar a lógica de salvar a sessão de login em MainActivity.kt :

kotlin

 Copiar código

```
class MainActivity : AppCompatActivity() { private lateinit var sessionManager:
SessionManager override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState) setContentView(R.layout.activity_main)
sessionManager = SessionManager(this) val loginButton = findViewById<Button>
(R.id.loginButton) loginButton.setOnClickListener { val clienteId = 1 // Aqui você
buscaria o ID do cliente após o login sessionManager.salvarSessaoUsuario(clienteId) //
Navegar para a tela principal } } }
```

Conclusão

Neste passo, configuramos o backend do aplicativo, definimos os modelos de dados, configuramos o banco de dados local com Room, implementamos chamadas de API com Retrofit, e gerenciamos as sessões de login com SharedPreferences. Agora, seu aplicativo está pronto para começar a armazenar e processar dados de forma eficiente.