

Agora vamos ao **Passo 4**, que é focado em implementar as funcionalidades do aplicativo de barbearia, como **Login, Agendamento de Serviço, Notificações Push, Avaliações de Serviço e Histórico de Agendamentos**.

4. Implementação de Funcionalidades


4.1 Implementação de Login e Cadastro

4.1.1 Tela de Login

No arquivo `LoginActivity.kt`, implementamos a lógica de autenticação usando **Firebase Authentication** ou outra API. Aqui, usaremos um exemplo com **Firebase Authentication**.

Dependências no `build.gradle`:


groovy

 Copiar código

```
// Firebase Authentication implementation 'com.google.firebase:firebase-auth-  
ktx:21.0.1'
```

Código Kotlin para o Login (`LoginActivity.kt`):

kotlin

 Copiar código

```
import android.content.Intent import android.os.Bundle import android.widget.Button  
import android.widget.EditText import android.widget.Toast import  
androidx.appcompat.app.AppCompatActivity import com.google.firebase.auth.FirebaseAuth  
class LoginActivity : AppCompatActivity() { private lateinit var auth: FirebaseAuth  
override fun onCreate(savedInstanceState: Bundle?) {  
super.onCreate(savedInstanceState) setContentView(R.layout.activity_login) auth =  
FirebaseAuth.getInstance() val emailEditText = findViewById<EditText>  
(R.id.emailEditText) val passwordEditText = findViewById<EditText>  
(R.id.passwordEditText) val loginButton = findViewById<Button>(R.id.loginButton)  
loginButton.setOnClickListener { val email = emailEditText.text.toString() val  
password = passwordEditText.text.toString() if (email.isNotEmpty() &&  
password.isNotEmpty()) { loginUser(email, password) } else { Toast.makeText(this, "Por  
favor, preencha todos os campos", Toast.LENGTH_SHORT).show() } } } private fun  
loginUser(email: String, password: String) { auth.signInWithEmailAndPassword(email,  
password) .addOnCompleteListener(this) { task -> if (task.isSuccessful) { // Login  
bem-sucedido, navegue para a tela principal val intent = Intent(this,  
MainActivity::class.java) startActivity(intent) finish() } else { Toast.makeText(this,  
"Erro no login", Toast.LENGTH_SHORT).show() } } } }
```

4.1.2 Tela de Cadastro

Criamos uma tela de cadastro para novos usuários.

Código Kotlin para o Cadastro (`RegisterActivity.kt`):

kotlin

 Copiar código

```
import android.content.Intent import android.os.Bundle import android.widget.Button
import android.widget.EditText import android.widget.Toast import
androidx.appcompat.app.AppCompatActivity import com.google.firebase.auth.FirebaseAuth
class RegisterActivity : AppCompatActivity() { private lateinit var auth: FirebaseAuth
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState) setContentView(R.layout.activity_register) auth =
FirebaseAuth.getInstance() val emailEditText = findViewById<EditText>
(R.id.emailEditText) val passwordEditText = findViewById<EditText>
(R.id.passwordEditText) val registerButton = findViewById<Button>(R.id.registerButton)
registerButton.setOnClickListener { val email = emailEditText.text.toString() val
password = passwordEditText.text.toString() if (email.isNotEmpty() &&
password.isNotEmpty()) { registerUser(email, password) } else { Toast.makeText(this,
"Por favor, preencha todos os campos", Toast.LENGTH_SHORT).show() } } } private fun
registerUser(email: String, password: String) {
auth.createUserWithEmailAndPassword(email, password) .addOnCompleteListener(this) {
task -> if (task.isSuccessful) { // Cadastro bem-sucedido, navegue para a tela
principal val intent = Intent(this, MainActivity::class.java) startActivity(intent)
finish() } else { Toast.makeText(this, "Erro no cadastro", Toast.LENGTH_SHORT).show()
} } } }
```


4.2 Implementação de Agendamento de Serviço

Agora vamos implementar a lógica para que o cliente possa agendar um serviço. Isso envolve selecionar o serviço e a data/hora, e depois salvar esse agendamento no banco de dados.

4.2.1 Agendamento de Serviço (ScheduleActivity.kt)

No arquivo ScheduleActivity.kt , vamos capturar a escolha do cliente e salvar o agendamento.

kotlin

 Copiar código

```
import android.os.Bundle import android.widget.Button import android.widget.DatePicker
import android.widget.Spinner import android.widget.Toast import
androidx.appcompat.app.AppCompatActivity import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers import kotlinx.coroutines.launch class
ScheduleActivity : AppCompatActivity() { private lateinit var appDatabase: AppDatabase
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState) setContentView(R.layout.activity_schedule)
appDatabase = AppDatabase.getDatabase(this) val serviceSpinner = findViewById<Spinner>
(R.id.serviceSpinner) val datePicker = findViewById<DatePicker>(R.id.datePicker) val
scheduleButton = findViewById<Button>(R.id.scheduleButton)
scheduleButton.setOnClickListener { val serviceId =
serviceSpinner.selectedItemPosition + 1 // Exemplo para obter o ID do serviço val data
= "${datePicker.dayOfMonth}/${datePicker.month + 1}/${datePicker.year}" val
agendamento = AgendamentoEntity( id = 0, // ID será gerado automaticamente clienteId =
1, // ID do cliente logado (será obtido do sistema de sessão) barbeiroId = 1, // ID de
um barbeiro exemplo servicoId = serviceId, data = data, horario = "14:00" // Horário
fixo por exemplo ) CoroutineScope(Dispatchers.IO).launch {
appDatabase.agendamentoDao().inserirAgendamento(agendamento) runOnUiThread {
```

```
Toast.makeText(this@ScheduleActivity, "Agendamento realizado com sucesso!",  
Toast.LENGTH_SHORT).show() } } } }
```

Explicação:

- O usuário seleciona um serviço e a data.
- Quando o botão "Agendar" é clicado, o agendamento é salvo no banco de dados usando Room.

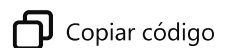
4.3 Implementação de Notificações Push

Usamos o **Firebase Cloud Messaging (FCM)** para enviar notificações push de lembrete para os clientes sobre seus agendamentos.

4.3.1 Configuração do FCM

Adicione as dependências no `build.gradle` :

```
groovy
```

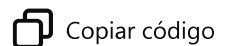


```
implementation 'com.google.firebase:firebase-messaging:22.0.0'
```

4.3.2 Receber Notificações (`MyFirebaseMessagingService.kt`)

Crie um serviço para lidar com as notificações:

```
kotlin
```



```
import com.google.firebase.messaging.FirebaseMessagingService import  
com.google.firebase.messaging.RemoteMessage class MyFirebaseMessagingService :  
FirebaseMessagingService() { override fun onMessageReceived(remoteMessage:  
RemoteMessage) { super.onMessageReceived(remoteMessage) // Lidar com a notificação  
recebida e exibir uma notificação local val notificationTitle =  
remoteMessage.notification?.title val notificationBody =  
remoteMessage.notification?.body // Exemplo de exibir notificação usando  
NotificationManager (pode ser personalizado) } override fun onNewToken(token: String)  
{ super.onNewToken(token) // Enviar o token para o servidor para fins de notificação }  
}
```

4.4 Implementação de Avaliações de Serviço

Permite que os clientes avaliem o serviço após a conclusão do agendamento.

4.4.1 Lógica de Avaliação (`RatingActivity.kt`)

Crie uma tela onde os clientes podem avaliar o barbeiro e o serviço.

```
kotlin
```



```
import android.os.Bundle import android.widget.Button import android.widget.RatingBar  
import android.widget.Toast import androidx.appcompat.app.AppCompatActivity import
```

```
kotlinx.coroutines.CoroutineScope import kotlinx.coroutines.Dispatchers import
kotlinx.coroutines.launch class RatingActivity : AppCompatActivity() { private
lateinit var appDatabase: AppDatabase override fun onCreate(savedInstanceState:
Bundle?) { super.onCreate(savedInstanceState) setContentView(R.layout.activity_rating)
appDatabase = AppDatabase.getDatabase(this) val ratingBar = findViewById<RatingBar>
(R.id.ratingBar) val submitButton = findViewById<Button>(R.id.submitButton)
submitButton.setOnClickListener { val rating = ratingBar.rating val barbeiroId = 1 //
ID do barbeiro avaliado CoroutineScope(Dispatchers.IO).launch {
appDatabase.barbeiroDao().atualizarAvaliacao(barbeiroId, rating) runOnUiThread {
Toast.makeText(this@RatingActivity, "Avaliação enviada!", Toast.LENGTH_SHORT).show() }
} } } }
```

Explicação:


- O cliente avalia o barbeiro e o serviço, e essa avaliação é atualizada no banco de dados.

4.5 Implementação de Histórico de Agendamentos

Mostra ao cliente uma lista de seus agendamentos anteriores usando um RecyclerView .

4.5.1 Histórico de Agendamentos (HistoryActivity.kt)

kotlin

 Copiar código

```
import android.os.Bundle import androidx.appcompat.app.AppCompatActivity import
androidx.recyclerview.widget.LinearLayoutManager import
androidx.recyclerview.widget.RecyclerView import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers import kotlinx.coroutines.launch class
HistoryActivity : AppCompatActivity() { private lateinit var appDatabase: AppDatabase
private lateinit var historyAdapter: HistoryAdapter override fun
onCreate(savedInstanceState: Bundle?) { super.onCreate(savedInstanceState)
setContentView(R.layout.activity_history) appDatabase = AppDatabase.getDatabase(this)
val recyclerView = findViewById<RecyclerView>(R.id.historyRecyclerView)
recyclerView.layoutManager = LinearLayoutManager(this) historyAdapter =
HistoryAdapter(listOf()) recyclerView.adapter = historyAdapter carregarAgendamentos()
} private fun carregarAgendamentos() { CoroutineScope(Dispatchers.IO).launch { val
agendamentos = appDatabase.agendamentoDao().buscarAgendamentosPorClienteId(1) //
Exemplo com clienteId=1 runOnUiThread {
historyAdapter.atualizarAgendamentos(agendamentos) } } } }
```

Explicação:

- **RecyclerView:** Mostra os agendamentos passados.
- **Adapter:** Atualiza a lista dinamicamente após a consulta ao banco de dados.

Conclusão

Com esse passo, você implementou as principais funcionalidades do aplicativo: login, cadastro, agendamento de serviço, notificações push, avaliações de serviço e histórico de agendamentos. Essas funcionalidades são essenciais para o funcionamento completo do aplicativo de barbearia.