

Aqui está o **Passo 6 detalhado**, onde vamos implementar funcionalidades avançadas para o aplicativo **Tech Burguer**, incluindo integração com uma API de pagamento e autenticação de usuários com o Firebase. Esse passo é opcional, mas acrescenta recursos importantes, como pagamento e login, que são comuns em aplicativos de pedidos.

6. Adicionar Funcionalidades Avançadas (Opcional)

Passo 6.1: Integrar API de Pagamento (Stripe)

Usaremos a API **Stripe** para processar pagamentos no aplicativo. Para simplificar, a integração com o Stripe será feita utilizando a biblioteca **Stripe SDK para Android**.

Passo 6.1.1: Configurar Stripe no Projeto

1. Adicionar Dependências do Stripe:

- Abra o arquivo **build.gradle (Module: app)**.
- Adicione a dependência do Stripe SDK.

gradle

 Copiar código

```
dependencies { implementation 'com.stripe:stripe-android:20.10.0' }
```

2. Sincronizar o Projeto: Clique em "Sync Now" para sincronizar as dependências.

Passo 6.1.2: Criar uma Conta no Stripe e Obter as Credenciais

1. Acesse Stripe e crie uma conta.
2. Vá para o painel do Stripe e obtenha suas **chaves API** (public key e secret key).
 - **Public Key** será usada no aplicativo.
 - **Secret Key** será usada no backend (não no app diretamente).

Passo 6.1.3: Configurar a Activity de Pagamento

Agora, vamos configurar uma nova Activity para processar o pagamento com o Stripe.

1. Crie uma nova Activity chamada **PaymentActivity**.
2. Adicione o layout para a página de pagamento em **res/layout/activity_payment.xml**:

xml

 Copiar código

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:layout_width="match_parent"
android:layout_height="match_parent" android:orientation="vertical" android:padding="16dp"> <!-- Input para o valor total -->
<TextView android:id="@+id/paymentAmountTextView" android:layout_width="wrap_content" android:layout_height="wrap_content"
android:text="Valor total: R$ 0,00" android:textSize="18sp" android:layout_gravity="center_horizontal" /> <!-- Input para os
dados do cartão --> <com.stripe.android.view.CardInputWidget android:id="@+id/cardInputWidget"
android:layout_width="match_parent" android:layout_height="wrap_content" android:layout_marginTop="16dp" /> <!-- Botão para pagar
--> <Button android:id="@+id/payButton" android:layout_width="wrap_content" android:layout_height="wrap_content"
android:text="Pagar" android:layout_gravity="center_horizontal" android:layout_marginTop="16dp" /> </LinearLayout>
```

Passo 6.1.4: Implementar a Lógica de Pagamento

1. Abra o arquivo **PaymentActivity.kt** e adicione a lógica para processar o pagamento.

kotlin

 Copiar código

```
package com.example.techburguer import android.os.Bundle import android.widget.Button import android.widget.TextView import
android.widget.Toast import androidx.appcompat.app.AppCompatActivity import com.stripe.android.PaymentConfiguration import
com.stripe.android.Stripe import com.stripe.android.model.PaymentIntentParams import com.stripe.android.view.CardInputWidget
class PaymentActivity : AppCompatActivity() { private lateinit var stripe: Stripe override fun onCreate(savedInstanceState:
Bundle?) { super.onCreate(savedInstanceState) setContentView(R.layout.activity_payment) // Configurar Stripe com a chave pública
PaymentConfiguration.init(applicationContext, "your_public_key_from_stripe_dashboard") // Referências aos componentes da
interface val amountTextView = findViewById<TextView>(R.id.paymentAmountTextView) val cardInputWidget =
findViewById<CardInputWidget>(R.id.cardInputWidget) val payButton = findViewById<Button>(R.id.payButton) // Obter o valor do
pedido (exemplo de R$ 50,00) val orderTotal = 50.00 amountTextView.text = "Valor total: R$ %.2f".format(orderTotal) //
Inicializar Stripe stripe = Stripe(applicationContext, PaymentConfiguration.getInstance(applicationContext).publishableKey) //
Lidar com o clique no botão "Pagar" payButton.setOnClickListener { val cardDetails = cardInputWidget.paymentMethodCreateParams if
```

```
(cardDetails != null) { // Criar PaymentIntent no backend (exemplo de código de servidor) val paymentIntentParams = PaymentIntentParams.createWithPaymentMethodCreateParams( paymentMethodCreateParams = cardDetails, clientSecret = "payment_intent_client_secret" // Este valor vem do backend ) stripe.confirmPayment(this, paymentIntentParams) } else { Toast.makeText(this, "Cartão inválido!", Toast.LENGTH_SHORT).show() } } }
```

Explicação:

- **PaymentConfiguration:** Inicializa o Stripe com a chave pública.
- **CardInputWidget:** Permite ao usuário inserir os dados do cartão.
- **stripe.confirmPayment:** Processa o pagamento no Stripe utilizando o client secret que seria gerado pelo backend.

Nota: A implementação completa do Stripe requer um backend para criar o PaymentIntent e fornecer o client_secret. No código acima, você precisa substituir "payment_intent_client_secret" por um valor real obtido do backend.


Passo 6.2: Autenticação de Usuários com Firebase

A autenticação com o **Firebase Authentication** permitirá que os usuários façam login no aplicativo, garantindo uma experiência mais personalizada. Vamos configurar a autenticação com **email e senha**.

Passo 6.2.1: Adicionar o Firebase ao Projeto

1. No Firebase Console, crie um novo projeto ou adicione o aplicativo a um projeto existente.
2. Siga as instruções para conectar o aplicativo Android ao Firebase.
3. No **build.gradle (Project: app)**, adicione o plugin do Firebase:

gradle


 Copiar código

```
dependencies { implementation platform('com.google.firebase:firebase-bom:31.1.0') implementation 'com.google.firebase:firebase-auth-ktx' }
```

Passo 6.2.2: Criar a Interface de Login

1. Crie uma nova Activity chamada **LoginActivity**.
2. No arquivo **res/layout/activity_login.xml**, crie o layout para login com email e senha:

xml

 Copiar código

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:layout_width="match_parent"
android:layout_height="match_parent" android:orientation="vertical" android:padding="16dp"> <EditText
android:id="@+id/emailEditText" android:layout_width="match_parent" android:layout_height="wrap_content" android:hint="Email" />
<EditText android:id="@+id/passwordEditText" android:layout_width="match_parent" android:layout_height="wrap_content"
android:hint="Senha" android:inputType="textPassword" /> <Button android:id="@+id/loginButton"
android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Entrar"
android:layout_gravity="center_horizontal" /> <Button android:id="@+id/registerButton" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:text="Registrar" android:layout_gravity="center_horizontal"
android:layout_marginTop="8dp" /> </LinearLayout>
```

Passo 6.2.3: Implementar a Lógica de Autenticação no Firebase

1. Abra o arquivo **LoginActivity.kt**.
2. Adicione o seguinte código para implementar o login e o registro:

kotlin

 Copiar código

```
package com.example.techburguer 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)
// Inicializar FirebaseAuth auth = FirebaseAuth.getInstance() // Referências aos componentes da interface val emailEditText =
findViewById<EditText>(R.id.emailEditText) val passwordEditText = findViewById<EditText>(R.id.passwordEditText) val loginButton =
findViewById<Button>(R.id.loginButton) val registerButton = findViewById<Button>(R.id.registerButton) // Lidar com o login
loginButton.setOnClickListener { val email = emailEditText.text.toString() val password = passwordEditText.text.toString() if
(email.isNotEmpty() && password.isNotEmpty()) { auth.signInWithEmailAndPassword(email, password) .addOnCompleteListener { task ->
if (task.isSuccessful) { Toast.makeText(this, "Login bem-sucedido!", Toast.LENGTH_SHORT).show() val intent = Intent(this,
```

```
MainActivity::class.java) startActivity(intent) } else { Toast.makeText(this, "Falha no login: ${task.exception?.message}",
Toast.LENGTH_SHORT).show() } } } else { Toast.makeText(this, "Preencha todos os campos!", Toast.LENGTH_SHORT).show() } } // Lidar
com o registro de novos usuários registerButton.setOnClickListener { val email = emailEditText.text.toString() val password =
passwordEditText.text.toString() if (email.isNotEmpty() && password.isNotEmpty()) { auth.createUserWithEmailAndPassword(email,
password) .addOnCompleteListener { task -> if (task.isSuccessful) { Toast.makeText(this, "Registro bem-sucedido!",
Toast.LENGTH_SHORT).show() } else { Toast.makeText(this, "Falha no registro: ${task.exception?.message}",
Toast.LENGTH_SHORT).show() } } } } else { Toast.makeText(this, "Preencha todos os campos!", Toast.LENGTH_SHORT).show() } } } }
```

Explicação:

- **auth.signInWithEmailAndPassword:** Faz o login de um usuário existente.
- **auth.createUserWithEmailAndPassword:** Registra um novo usuário.
- **Intent:** Após o login bem-sucedido, o usuário é redirecionado para o `MainActivity`.

Passo 6.3: Testar as Funcionalidades Avançadas

1. Execute o aplicativo no emulador ou dispositivo real.
2. Teste a funcionalidade de login e registro.
3. Teste o fluxo de pagamento simulado para garantir que o Stripe funcione corretamente.

Com isso, o **Passo 6** está completo. Agora, o aplicativo **Tech Burguer** tem funcionalidades avançadas, incluindo integração com o Stripe para pagamentos e autenticação de usuários com o Firebase.