

No **Passo 10**, vamos discutir como fazer a **manutenção contínua do aplicativo e o lançamento de atualizações**. Após o aplicativo ser publicado, a manutenção contínua é crucial para garantir que o aplicativo funcione corretamente em diferentes versões do Android, que novos recursos sejam implementados e que eventuais bugs sejam corrigidos com base no feedback dos usuários.

Passo 10: Manutenção Contínua e Lançamento de Atualizações

10.1 Monitorando a Performance do Aplicativo

Depois de publicar o aplicativo, o primeiro passo é monitorar continuamente sua performance. Isso inclui observar o comportamento do aplicativo, coletar dados de uso e identificar erros que os usuários possam enfrentar.

1. Usando Firebase Crashlytics:

- **Crashlytics** é uma ferramenta do **Firebase** que oferece relatórios detalhados sobre travamentos do aplicativo. Ela ajuda a identificar os pontos onde o aplicativo pode estar falhando e dá informações sobre o que causou o erro.
- Para integrar o **Crashlytics** ao seu aplicativo:

1. Conectar o Firebase ao Projeto:

- No Android Studio, vá até **Tools > Firebase**.
- Selecione **Crashlytics** e clique em **Connect your app to Firebase**.
- Siga as instruções para associar seu aplicativo a um projeto no Firebase.

2. Adicionar as Dependências de Firebase e Crashlytics:

- Após conectar o aplicativo ao Firebase, adicione as dependências de **Crashlytics** no arquivo `build.gradle (Module: app)`:

gradle

 Copiar código

```
dependencies { // Firebase Crashlytics implementation 'com.google.firebase:firebase-crashlytics:18.2.1' implementation  
'com.google.firebase:firebase-analytics:21.2.0' }
```


3. Configurar Crashlytics:

- O Firebase Crashlytics estará ativo automaticamente após a configuração.
- Agora você pode monitorar erros e falhas no painel do Firebase.

2. Monitoramento de Desempenho com Firebase Performance Monitoring:

- O **Firebase Performance Monitoring** é outra ferramenta importante para acompanhar a velocidade do aplicativo e identificar gargalos de desempenho.
- Para ativá-lo, siga o mesmo processo de configuração que o **Crashlytics** e adicione a dependência:

gradle

 Copiar código

```
dependencies { // Firebase Performance Monitoring implementation 'com.google.firebase:firebase-perf:20.1.0' }
```

Isso permitirá monitorar o desempenho em áreas como o tempo de carregamento de atividades, latência de chamadas de rede e uso de recursos.

10.2 Respondendo ao Feedback dos Usuários

1. Coletando e Respondendo ao Feedback dos Usuários:

- Monitore o feedback que os usuários deixam na Play Store. Responda a avaliações e dúvidas dos usuários para mostrar que o aplicativo está sendo mantido ativamente.
- Use as sugestões e críticas dos usuários para melhorar o aplicativo. Por exemplo, se muitos usuários pedirem por um novo recurso ou relataram um bug específico, priorize essas melhorias nas próximas atualizações.

2. Enquetes e Pesquisas:

- Você também pode incluir enquetes ou formulários diretamente no aplicativo para coletar feedback contínuo.
- Exemplo de como você pode mostrar uma pesquisa de satisfação dentro do aplicativo:

kotlin

 Copiar código

```
AlertDialog.Builder(this) .setTitle("Como está sua experiência?") .setMessage("Sua opinião nos ajuda a melhorar. Avalie sua experiência com nosso app!") .setPositiveButton("Avaliar") { dialog, which -> // Ação de redirecionar para a Play Store ou abrir formulário } .setNegativeButton("Fechar", null) .show()
```

10.3 Lançamento de Atualizações

Quando você identificar que melhorias ou correções de bugs precisam ser implementadas, você deve planejar uma nova versão do aplicativo. Vamos ver como gerenciar isso.

1. Gerenciamento de Versões:

- Cada vez que você lançar uma atualização, é necessário incrementar o número de versão no arquivo `build.gradle` (Module: app) :

gradle



```
defaultConfig { applicationId "com.example.acaiStore" minSdkVersion 21 targetSdkVersion 31 versionCode 2 // Aumente o número a cada nova versão versionName "1.1" // Aumente a versão de forma mais visível ao usuário }
```

- `versionCode` : Um número que aumenta em cada atualização e que o Google Play usa para diferenciar versões.
- `versionName` : A versão que o usuário vê na Play Store. Isso pode ser algo como "1.1" ou "2.0" para indicar mudanças significativas ou menores.

2. Gerando o APK ou AAB para a Atualização:

- Para gerar um novo APK ou AAB assinado, siga o processo que já cobrimos no **Passo 9**. Certifique-se de que a nova versão foi corretamente assinada com o mesmo keystore que a versão anterior.

3. Publicando a Atualização:

- No **Google Play Console**, faça upload do novo APK ou AAB, preencha a seção de **Notas da Versão**, detalhando o que foi alterado ou corrigido na nova versão.
- Envie a atualização para revisão. Se for um bug crítico, você pode marcar como uma **atualização de emergência**.

10.4 Adição de Novos Recursos

À medida que o aplicativo cresce, você pode querer adicionar novos recursos para melhorar a experiência dos usuários. Vamos ver como planejar e implementar novos recursos sem comprometer a estabilidade do aplicativo.

1. Planejamento de Novos Recursos:

- Baseie-se no feedback dos usuários e nas tendências do mercado para adicionar novos recursos. Defina claramente o que você deseja adicionar na nova versão e quais serão as mudanças em relação à versão anterior.
- Exemplo de novos recursos para o seu aplicativo de loja de açaí:

- **Cupons de Desconto:** Permitir que os usuários insiram cupons para obter descontos.
- **Histórico de Pedidos:** Permitir que os usuários vejam seus pedidos anteriores.

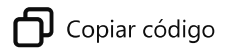
2. Implementando Novos Recursos de Forma Gradual:

- Sempre implemente novos recursos de forma gradual e bem testada para evitar a quebra do aplicativo. Use testes A/B para lançar novos recursos para uma pequena porcentagem de usuários antes de liberar para todos.

Exemplo de um novo recurso de cupom de desconto:

- **Layout XML** para inserir o cupom:

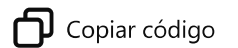
xml



```
<EditText android:id="@+id/couponCode" android:layout_width="match_parent" android:layout_height="wrap_content"
android:hint="Insira seu cupom"/>
```

- **Kotlin** para aplicar o cupom:

kotlin



```
val couponCode = findViewById<EditText>(R.id.couponCode) val applyCouponButton = findViewById<Button>(R.id.applyCouponButton)
applyCouponButton.setOnClickListener { val code = couponCode.text.toString() if (code == "DESCONTO10") { val discount = 0.10
// 10% de desconto val newTotalPrice = CartManager.getTotalPrice() * (1 - discount) totalPriceTextView.text = "Total com
desconto: R$ $newTotalPrice" } else { Toast.makeText(this, "Cupom inválido", Toast.LENGTH_SHORT).show() } }
```

10.5 Garantindo a Compatibilidade com Novas Versões do Android

1. Testes em Diferentes Versões do Android:

- Sempre que uma nova versão do Android for lançada, teste o aplicativo em diferentes versões para garantir a compatibilidade. Use emuladores ou dispositivos físicos para testar o funcionamento do aplicativo em versões mais antigas e recentes do Android.

2. Atualização do SDK e Bibliotecas:

- Verifique se as bibliotecas e o SDK Android usados no projeto estão atualizados para garantir compatibilidade e segurança.

- Atualize o `targetSdkVersion` no arquivo `build.gradle` sempre que uma nova versão estável do SDK Android for lançada.

10.6 Testes Automatizados

Para garantir que o aplicativo continua funcionando conforme esperado após cada atualização, é importante automatizar os testes.

1. Teste Unitário:

- Escreva testes unitários para garantir que funções críticas, como cálculos no carrinho e integração com APIs, continuam funcionando conforme esperado.
- Exemplo de um teste unitário simples:

kotlin

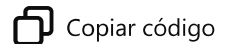


```
@Test fun testTotalPriceCalculation() { val product1 = AcaiProduct(1, "Açaí", "Delicioso", 12.99, "url") val product2 = AcaiProduct(2, "Açaí com morango", "Com morango", 15.99, "url") CartManager.addToCart(product1, 2) // 2 unidades do produto 1 CartManager.addToCart(product2, 1) // 1 unidade do produto 2 assertEquals(41.97, CartManager.getTotalPrice(), 0.001) }
```

2. Teste de Interface (UI):

- Use **Espresso** para testar automaticamente a navegação e a interface do usuário.
- Exemplo de um teste de navegação com Espresso:

kotlin



```
@Test fun testNavigateToCart() { onView(withId(R.id.fab_cart)).perform(click()) onView(withId(R.id.cartRecyclerView)).check(matches(isDisplayed())) }
```

10.7 Ciclo de Vida de Manutenção

Manter o aplicativo atualizado e funcionando bem ao longo do tempo é um processo contínuo. Estabeleça um ciclo de vida de manutenção, definindo prazos para revisar e lançar novas atualizações.

- **Ciclo de Atualizações Menores:** A cada 2-3 meses, revise o aplicativo para resolver bugs menores ou melhorar o desempenho.
- **Atualizações Maiores:** A cada 6-12 meses, planeje e implemente novos recursos ou grandes melhorias.

Resumo

No **Passo 10**, focamos na manutenção contínua do aplicativo, monitoramento de erros e desempenho, lançamento de atualizações, e adição de novos recursos. Este é um processo que mantém o aplicativo funcionando bem e relevante para os usuários. Automatize testes, monitore feedback e continue melhorando o aplicativo com base nas necessidades dos usuários e novas tendências tecnológicas. Se precisar de mais ajuda com algum desses aspectos, estou à disposição!