PROGRAMAÇÃO II

AULA 6

Prof. Elton Masaharu Sato

CONVERSA INICIAL

Nesta etapa, iremos aprender sobre vários detalhes da produção de aplicativos, desde como realizar os testes de projeto, como criar aplicativos com design, como publicar um aplicativo e prepará-los para o futuro.

Os tópicos a serem abordados são:

- Testes de Unidade;
- Testes de Widget;
- Documentação do Flutter;
- Material Design;
- Publicando um APK.

TEMA 1 – *FLUTTER UNIT TESTING*

Flutter possui uma diversa quantidade de ferramentas para o desenvolvedor atingir os seus objetivos, desde a instalação até a publicação. Nesta seção, veremos como podemos testar o nosso aplicativo utilizando algumas das ferramentas que foram apresentadas em conteúdos anteriores.

Testar um aplicativo é uma das fases mais importantes durante o ciclo de desenvolvimento de uma aplicação, pois é nela que o desenvolvedor recebe o *feedback* do aplicativo e pode realizar ajustes, correções e melhorias. Fazer os testes de um aplicativo vai além de verificar o aplicativo por erros; também é possível analisar uma quantidade enorme de outros fatores, como:

1.1 OTIMIZAÇÃO

Uma das melhorias que um desenvolvedor pode fazer durante a fase de testes é otimizar o código e suas funções. Como foi visto em conteúdos anteriores, o uso de códigos como o *Async* e *Future* permitem que o aplicativo rode mais rápido, o que pode ser vital para um aplicativo. Ex.: uma tela de *login* que espera a confirmação do servidor antes de realizar qualquer outra tarefa pode parecer que está travado para o usuário, mesmo que o aplicativo esteja funcionando perfeitamente bem.

1.2 INTERFACE

Alterar a interface de um aplicativo pode ser uma das melhorias possíveis. Ao mexer no aplicativo múltiplas vezes, é possível identificar onde estão os pontos redundantes do aplicativo, em que o seu funcionamento poderia ser mais rápido se etapas desnecessárias forem puladas. Esta etapa requer também muito cuidado, pois o desenvolvedor deve sempre lembrar que como ele conhece profundamente o aplicativo e suas funções, é possível que o aplicativo não seja intuitivo para um usuário leigo. Ex.: separar as configurações corretas de aplicativo dentro de seus menus e telas. Se uma configuração como alterar o tema do aplicativo de modo claro e modo escuro não estiver em um local apropriado e intuitivo, o usuário pode acreditar que a configuração não foi implementada.

1.3 POLIMENTO

Essa melhoria costuma fazer a diferença entre um aplicativo funcional e um aplicativo de sucesso. Quando falamos de polimento, estaremos nos referindo à implementação de efeitos visuais, sonoros e hápticos (vibração do dispositivo) que possam melhorar a experiência do usuário. Em especial, são os efeitos que não possuem efeito prático nas funcionalidades do aplicativo e que servem somente como algum tipo de recompensa ou apoio para que o usuário tenha algum *feedback* do aplicativo. Ex.: em um aplicativo de jogo de cartas (Paciência, por exemplo), o efeito visual das cartas cascateando não agrega para o aplicativo em termos de funcionalidade, mas pode ser a diferença entre um usuário preferir o seu aplicativo de Paciência acima da concorrência.

1.4 ERROS

Finalmente, o motivo primário pelo qual um desenvolvedor realiza testes é procurar por *bugs* e outros erros de código que impeçam o aplicativo de funcionar como deveria. Isso se aplica a

qualquer tipo de funcionalidade, sistema, ou implementação, incluindo erros de otimização, interface e polimento. Ex.: uma tela de *login* não envia as informações certas ao servidor, algum efeito na tela não aparece quando deveria etc.

1.5 SEGURANÇA

De forma mais avançada, é bastante comum também realizar testes de segurança do aplicativo, procurando por falhas e brechas que possam comprometer os dados do usuário. Esse tema ficou bastante reforçado devido à aprovação da LGPD, a Lei Geral de Proteção de Dados, que responsabiliza a parte reveladora dos dados privados. Ex.: se a sua base de dados estiver aberta a ataques como o famoso SQL injection, a empresa do aplicativo pode ser processada pelo vazamento dos dados coletados dos usuários para o cadastro.

Para realizar esses testes, o Flutter disponibiliza algumas ferramentas, além do editor de código que estiver usando, que pode apresentar algumas funcionalidades a mais.

Neste tópico, veremos o *Unit Testing* (teste de unidade), que é uma forma rápida de verificar se uma parte do código roda como esperado. Vejamos o seguinte exemplo:

Faremos a seguinte alteração no código de exemplo do Flutter:

```
@override

 Widget build(BuildContext context) {

  return Scaffold(
   appBar: AppBar(
    title: Text(widget.title),
   ),
   body: Center(
    child: Column(
     mainAxisAlignment: MainAxisAlignment.center,
     children: <Widget>[
      Text(
       pushingButtonText.pushingText(_counter),
      ),
      Text(
       '$_counter',
       style: Theme.of(context).textTheme.headline4,
      ),
     ],
    ),
   ),
   floatingActionButton: FloatingActionButton(
    onPressed: _incrementCounter,
    tooltip: 'Increment',
    child: const Icon(Icons.add),
   ), // This trailing comma makes auto-formatting nicer for build
methods.
  );
 }

}

class pushingButtonText {
 static String pushingText(int counter){
  if(counter < 10){
   return 'Você pressionou poucas vezes:';
  } else if(counter < 50){
   return 'Você pressionou várias vezes:';
  } else{
   return 'Você pressionou muitas vezes:';
  }
 }
}
```

E então faremos o seguinte:

1. Adicionaremos a dependência de test ou flutter_test
2. Criaremos um arquivo de testes (sempre terminar o nome do arquivo como _test.dart)
3. Criaremos a classe de testes
4. Escreveremos os testes
5. Combinaremos os testes em grupos
6. Rodaremos os testes

E o código será o seguinte:

```
import 'package:testing/main.dart';

//import 'package:test/test.dart';
import 'package:flutter_test/flutter_test.dart';

void main(){
 group('pushingText',(){

 test('Texto com poucos cliques', (){

  var resultado = pushingButtonText.pushingText(6);
  expect(resultado, 'Você pressionou poucas vezes:');
 });

 test('Texto com 40 cliques', (){

  var resultado = pushingButtonText.pushingText(40);
  expect(resultado, 'Você pressionou várias vezes:');
 });

 test('Texto com 120 cliques', (){

    var resultado = pushingButtonText.pushingText(120);
    expect(resultado, 'Você pressionou muitas vezes:');
  });

 });

}
```

TEMA 2 – *FLUTTER WIDGET TESTING*

O Flutter possui um segundo método de testes, o *Widget Testing*. Este método funciona de forma muito similar ao *Unit Testing*, porém é desenvolvido para testar *widgets* inteiros, utilizando um *tester* (testador) que age como se fosse um usuário testando um aplicativo.

Algumas das funções disponíveis para o nosso testador são as seguintes:

- **enterText()**: essa função permite que o nosso testador simule uma entrada de texto em um campo de texto na tela e é bastante útil para automatizar testes de *login* e senha de usuários;
- **tap()**: essa função permite que o nosso testador simule um toque na tela, o qual ativa *widgets* como botões, permitindo que sejam analisados os efeitos de toques na tela;
- **drag()**: essa função permite que o nosso testador simule um movimento de arrastar na tela. O movimento começa do meio do *widget*, e arrasta-se de acordo com um parâmetro passado para a função.

E há também uma classe importante, o *finder* (procurador), que tem como objetivo procurar na tela atual do seu aplicativo, um *widget* que tenha os parâmetros passados a ele. Para complementar o trabalho do *finder*, temos o *matcher* (pareador), que, por sua vez, tem como objetivo verificar se o(s) *widget(s)* encontrados paream com a quantidade de *widgets* que se espera encontrar.

- **findsNothing**: verificador do *matcher* para saber se nenhum *widget* é igual ao procurado;
- **findsOneWidget**: verificador do *matcher* para saber se exatamente um widget é igual ao procurado;
- **findsWidgets**: verificador do *matcher* para saber se um ou mais *widgets* são iguais ao procurado;
- **findsNWidgets**: verificador do *matcher* para saber se exatamente N *widgets* são iguais ao procurado. Note que N é um parâmetro a ser passado para esta função.

Veremos como cada uma dessas peças do *widget tester* se encaixa no exemplo ao final do tópico. Para se criar um *widget Tester*, recomenda-se seguir os seguintes passos:

1. Adicione a dependência do flutter_test;
2. Crie um *widget* para ser testado;
3. Crie um *widget* to tipo *testWidgets*;
4. Construa um *widget* usando o *WidgetTester*;
5. Procure por um *widget* usando o *Finder*;

6. Verifique se o *widget* está funcionando com o *Matcher*;

2.1 ADICIONE A DEPENDÊNCIA DO FLUTTER_TEST

Antes de escrever qualquer teste, inclua a dependência a seguir dentro do arquivo pubspec.yaml:

```
dev_dependencies:
  flutter_test:
    sdk: flutter
```

2.2 CRIE UM *WIDGET* PARA SER TESTADO

Para o nosso exemplo, iremos utilizar o aplicativo padrão de contador quando se cria um novo projeto.

2.3 CRIE UM WIDGET DO TIPO *TESTWIDGETS*

Um novo projeto do Flutter deverá vir com um testador pronto. Mas caso seja de seu interesse criar um novo, as linhas de código necessárias para um novo testador são as seguintes:

```
void main() {
  testWidgets('Título do meu Testador', (WidgetTester tester) async
{
    // Código de teste aqui
  });
}
```

2.4 CONSTRUA UM *WIDGET* USANDO O *WIDGETTESTER*

Este é o momento em que escolhemos qual *widget* queremos testar. Utilizando a função *pumpWidget* descrita a seguir, escolhemos qual *widget* iniciar para realizarmos os testes.

```
void main() {
  testWidgets('Título do meu Testador', (WidgetTester tester) async
{
    await tester.pumpWidget(const MeuWidget());
  });
}
```

A função *pumpWidget* tem então como objetivo iniciar o *widget* para testes. E ele se manterá estático até que uma das duas seguintes funções sejam executadas:

- **pump()**: avança o *widget* em um frame, ou uma quantidade de tempo designada por parâmetro;
- **pumpAndSettle()**: repetidamente faz as chamadas de *pump()* até que não haja mais alterações planejadas para a tela. Essa função é útil para esperar até que animações na tela se completem.

2.5 PROCURE POR UM *WIDGET* USANDO O *FINDER*

Utilizando a função *finder* então, agora procuraremos pelas partes que queremos testar. Neste exemplo a seguir, procuramos se existe um *widget* com o texto 'Login'.

2.6 VERIFIQUE SE O *WIDGET* ESTÁ FUNCIONANDO COM O *MATCHER*

E finalmente, após abrirmos o *widget* e procurarmos pelo que queremos testar, devemos verificar se o *widget* existe ou se está na quantidade esperada de *widgets*.

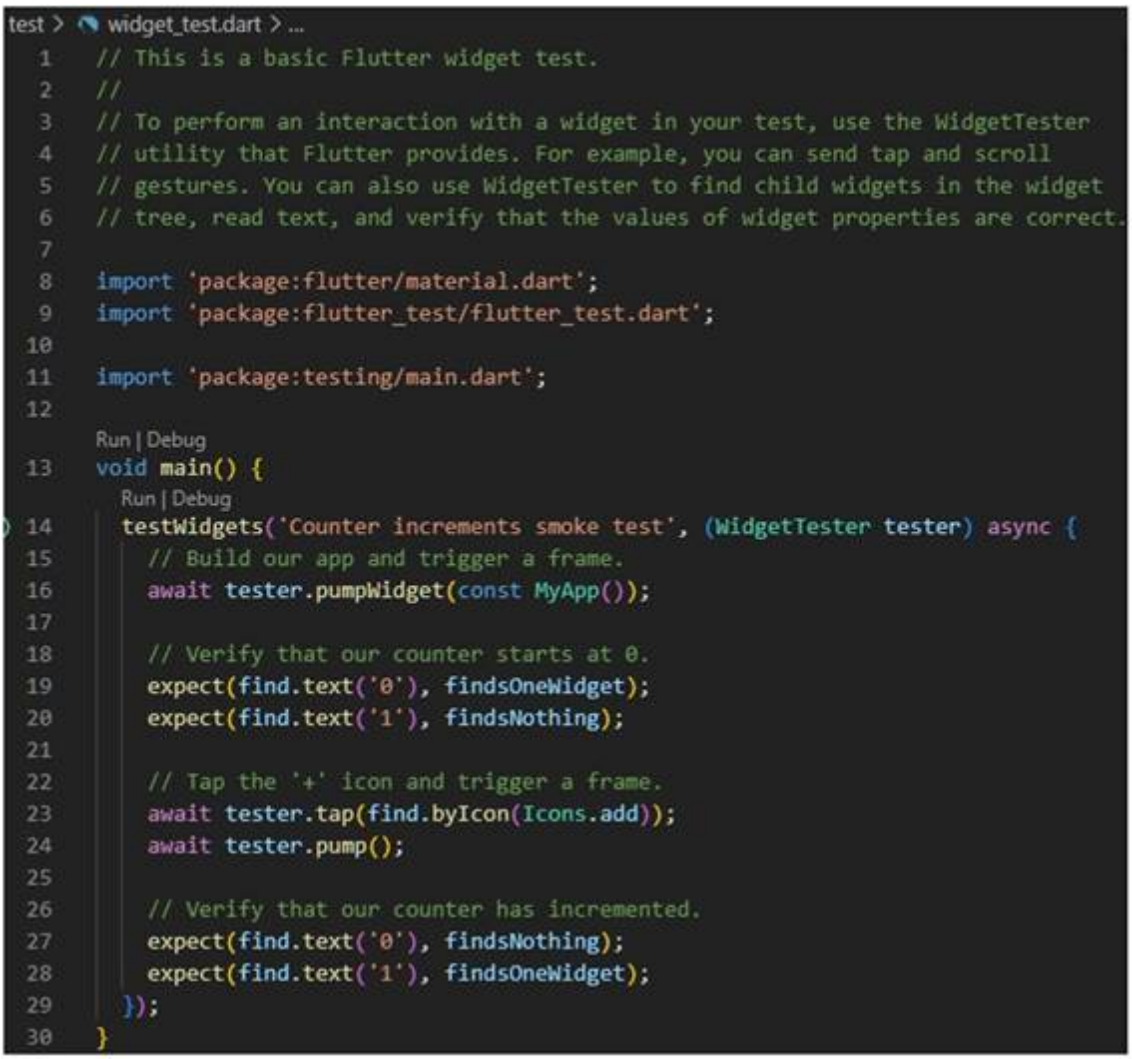
```
void main() {
  testWidgets('Título do meu Testador', (WidgetTester tester) async
{
    await tester.pumpWidget(const MeuWidget());

    final messageFinder = find.text('Login');
    expect(messageFinder, findsOneWidget);
  });
}
```

Este último então verifica se foi encontrado um *widget* com o texto "Login" em nosso *widget* "MeuWidget".

2.7 EXEMPLO

Veremos o código que vem pronto com cada novo projeto de Flutter para verificar a sua execução:

Figura 1 – Teste de *widget* do Flutter



TEMA 3 – FLUTTER DOCS

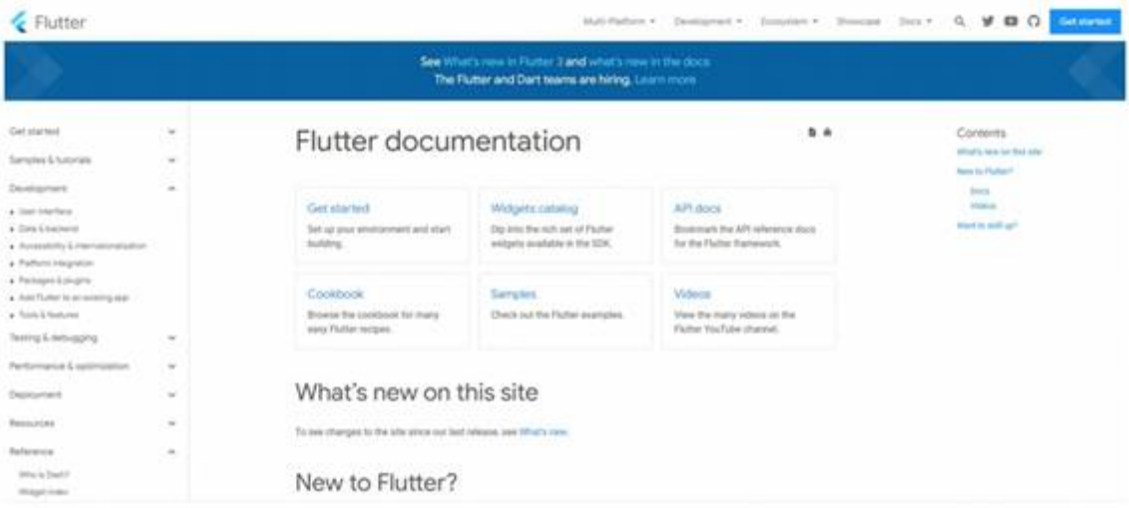
Por mais completo que qualquer curso de Flutter seja, ele sempre ficará defasado quando uma nova funcionalidade do Flutter estiver disponível, portanto, devemos fazer o possível para preparar você para o futuro da tecnologia. Neste tópico, iremos abordar uma das formas mais interativas e informativas sobre as alterações e funcionalidades da ferramenta.

Flutter Docs é um dos repositórios mais completos para se procurar por informações sobre o Flutter. No *site*, encontraremos diversas formas de aprender sobre o Flutter, e é altamente

recomendado que você faça uma visita ao *site* e dê uma explorada.

Caso você tenha problemas com inglês, recomenda-se que instale um *plugin* de tradução como o Google Translate para o seu navegador, permitindo assim que você tenha uma versão traduzida do *site*.

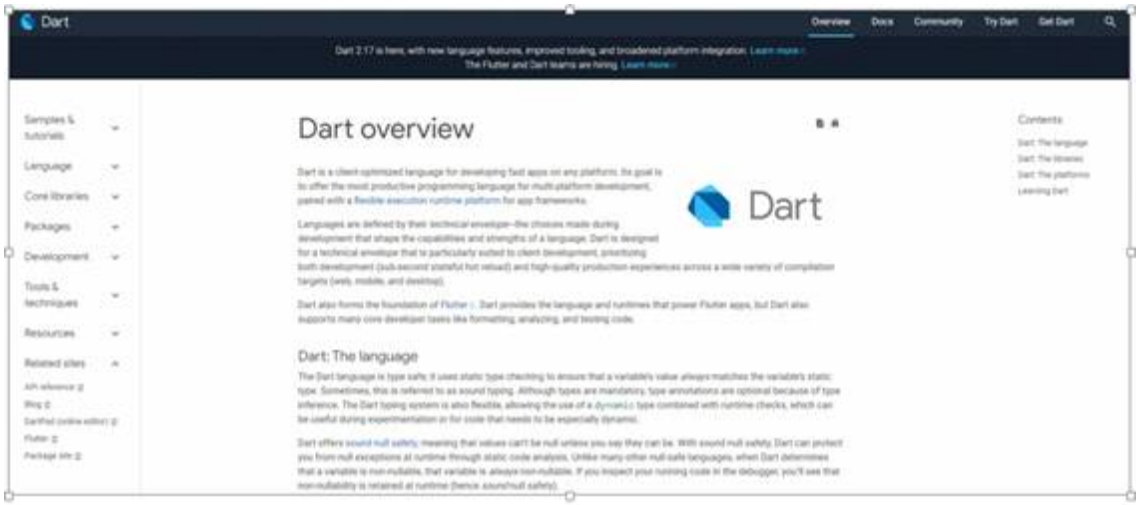
Figura 2 – Página principal do Flutter Docs



Fonte: Flutter, S.d.

A partir dele, podemos acessar o dart.dev através do *link* Dart Language Overview em *Get Started*.

Figura 3 – Página Principal do Dart.dev



Fonte: Dart, S.d.a

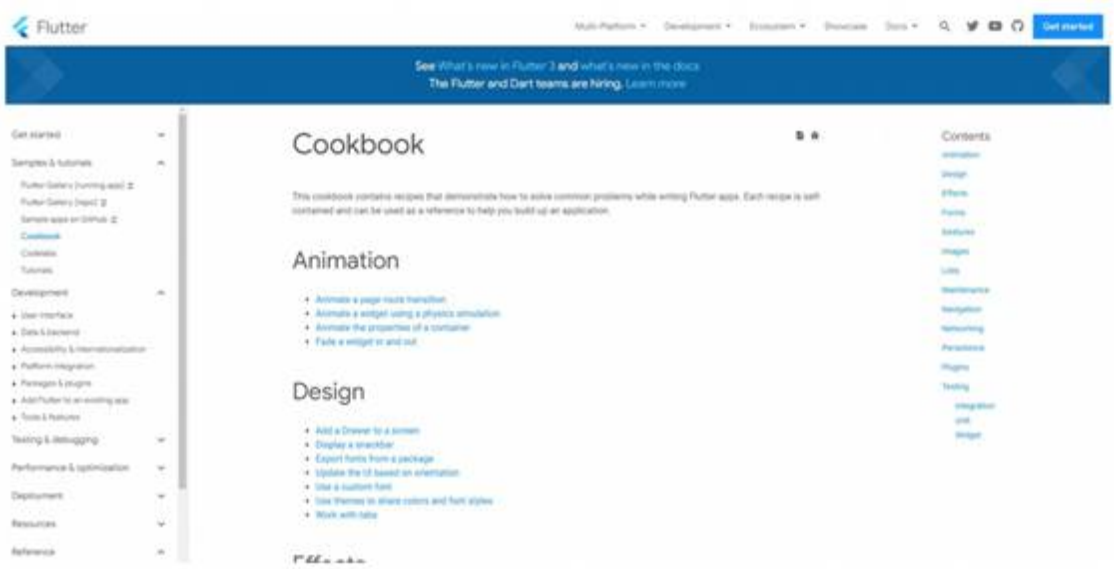
Em *Dart cheatsheet Codelab*, é possível relembrar e aprender conceitos de *dart* com exercícios dentro do próprio *site*.

Figura 4 – Cheatsheet Codelab

Fonte: Dart, S.d.

Voltando ao *site* do Flutter, podemos acessar um repositório muito útil chamado Cookbook, o qual seria um livro de receitas de códigos de Flutter.

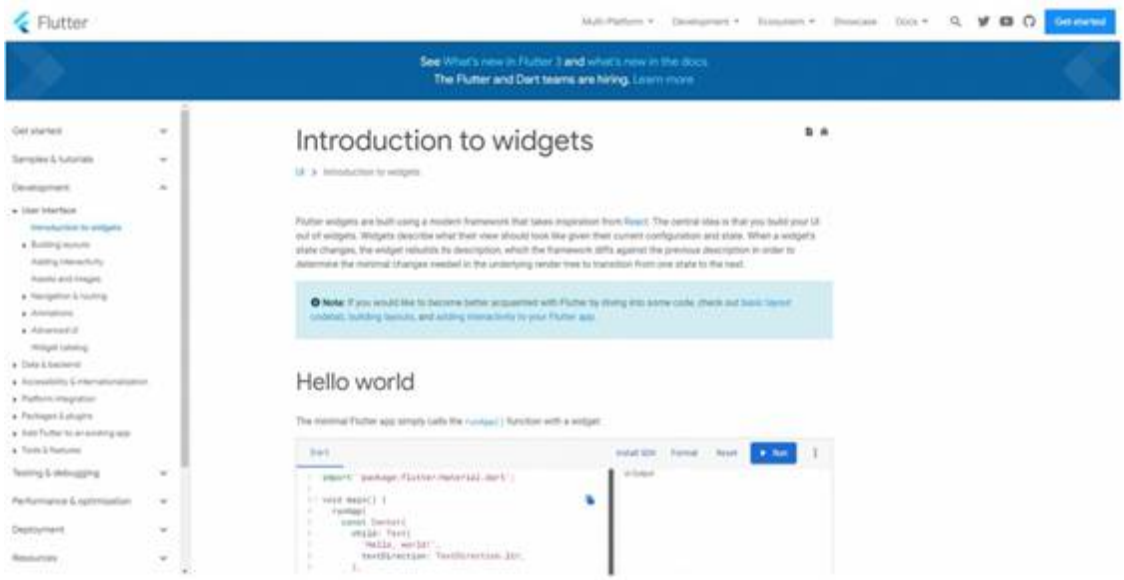
Figura 5 – *Cookbook*



Fonte: Flutter, S.d.

Na parte de *desenvolvimento*, é possível encontrar guias e tutoriais do *framework*.

Figura 6 – Introdução a *widgets*



Fonte: Flutter, S.d.

Em *Testing* e *Debugging*, podemos encontrar material referente à *debugging* e testes de aplicativo.

Figura 7 – *Testing* e *Debugging*

Fonte: Flutter, S.d.

Há também uma seção mais avançada sobre otimização e *performance*, para aqueles que vão além da simples funcionalidade e aparência.

Figura 8 – *Performance*

Fonte: Flutter, S.d.

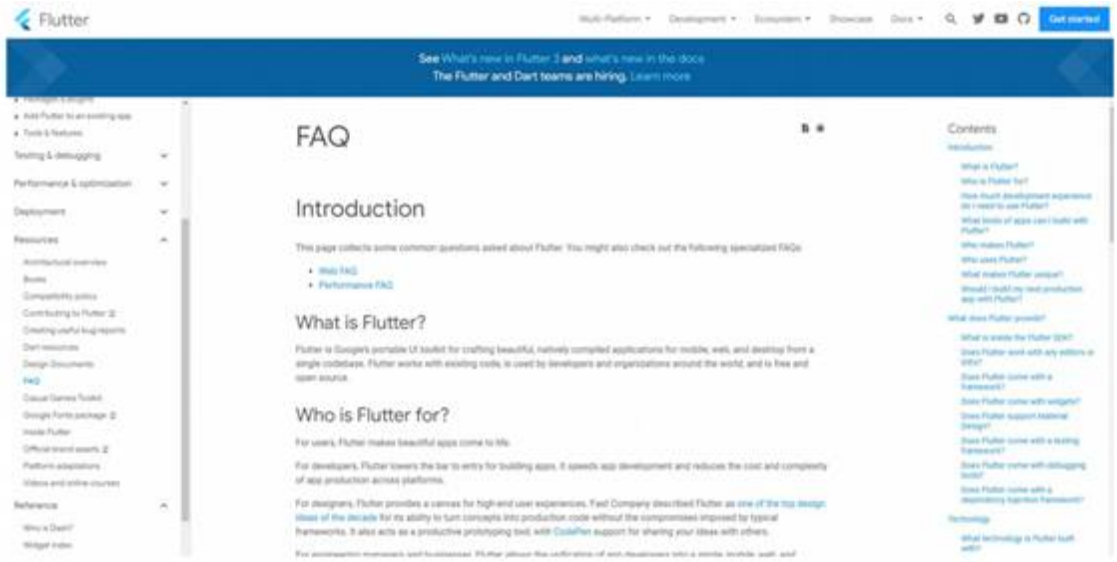
No *site* encontra-se uma pequena seção sobre publicação.

Figura 9 – Construção e publicação

Fonte: Flutter, S.d.

Ao final, encontramos também uma variedade de outros tópicos, como livros, política de compatibilidade, um FAQ, entre outros.

Figura 10 – FAQ



Fonte: Flutter, S.d.

TEMA 4 – MATERIAL *DESIGN*

Nesta seção, abordaremos o tema de *design*. Apesar de o foco do estudo não estar na parte criativa do *front-end*, é importante termos as noções de *design* necessárias para podermos utilizar em nossos aplicativos.

Um curso de *design* completo levaria anos para ser concluído, portanto usaremos uma ferramenta oficial da Google, o *site* Material Design, para aprendermos sobre como podemos utilizar os seus componentes da forma mais esteticamente agradável possível.

Veremos como utilizar as informações do *site* para melhorarmos os nossos aplicativos e entendermos um pouco mais do desenvolvimento de interfaces do *front-end*.

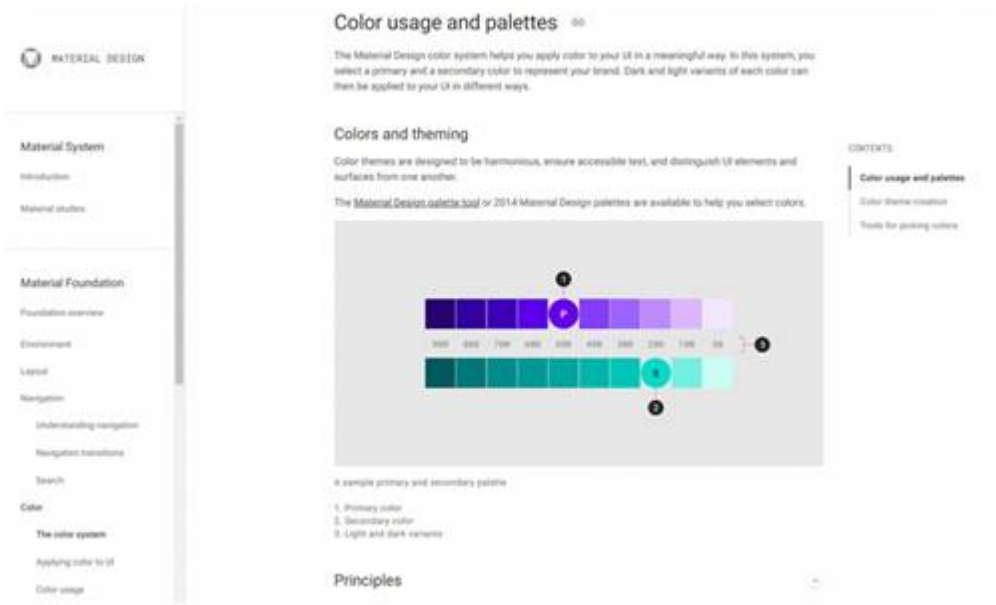
Assim como no tópico anterior, recomenda-se que se você não sabe inglês, que utilize um *plugin* de tradução em seu navegador, como o Google Translate, que traduz a página inteira de uma vez só.

Figura 11 – *Site* do Material *Design*

Fonte: Material *Design*, S.d.

Na parte de *Color*, poderemos aprender um pouco sobre a teoria das cores, e como utilizá-los da melhor forma possível. É interessante notar que o Flutter já vem com combinações de cores curadas por especialistas, mas o desenvolvedor pode a qualquer momento alterar o seu esquema para o seu desejado.

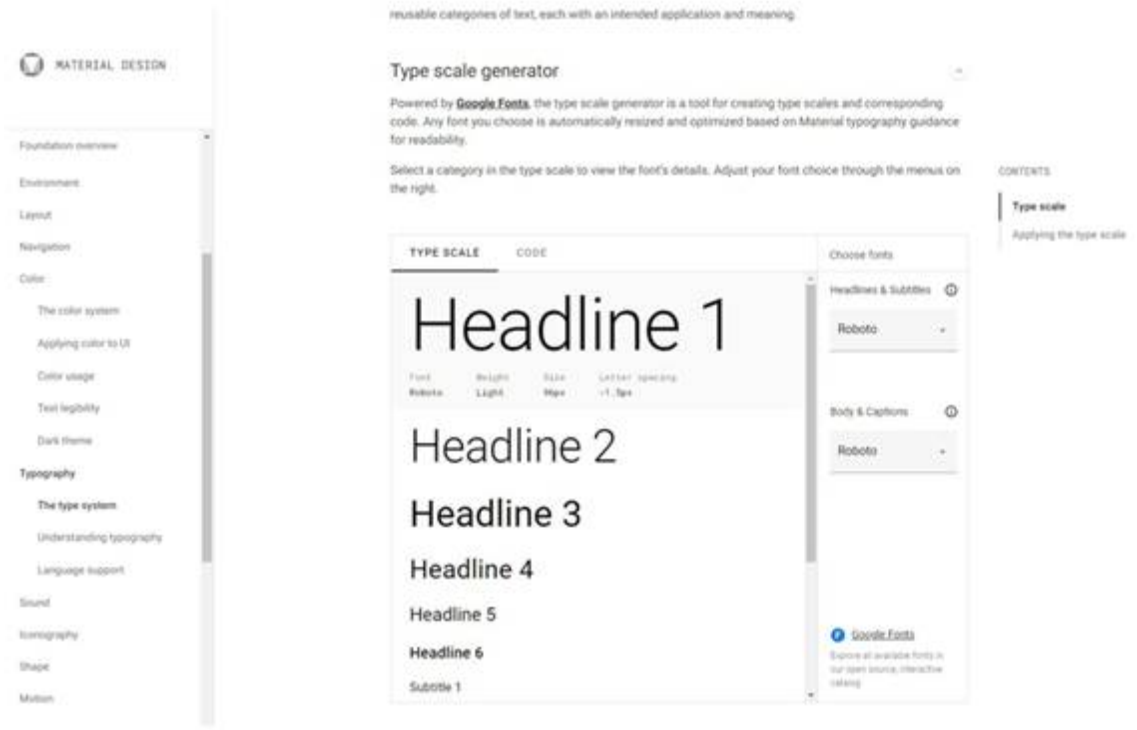
Figura 12 – Cores



Fonte: Material Design, S.d.

Há também uma seção de tipografia, no qual apresenta um simulador de tipografia e também um exemplo de escala de fontes.

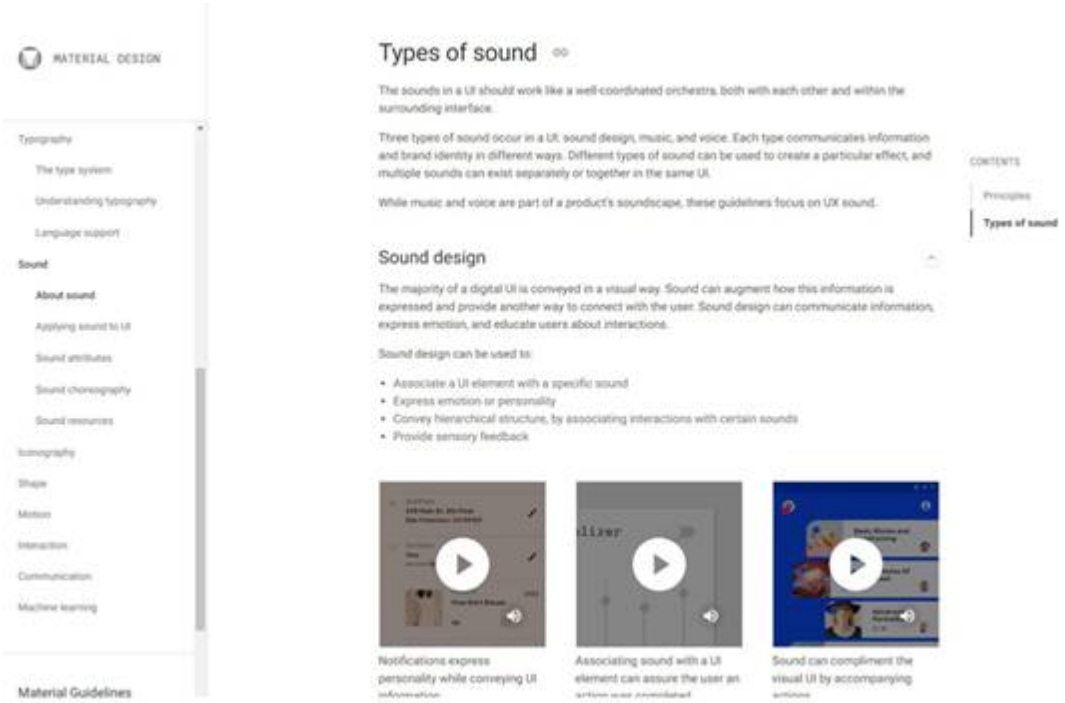
Figura 13 – Tipografia



Fonte: Material Design, S.d.

Existe uma seção sobre sonografia e sobre a importância de utilizar sons na interface, e também música e voz, dependendo da situação.

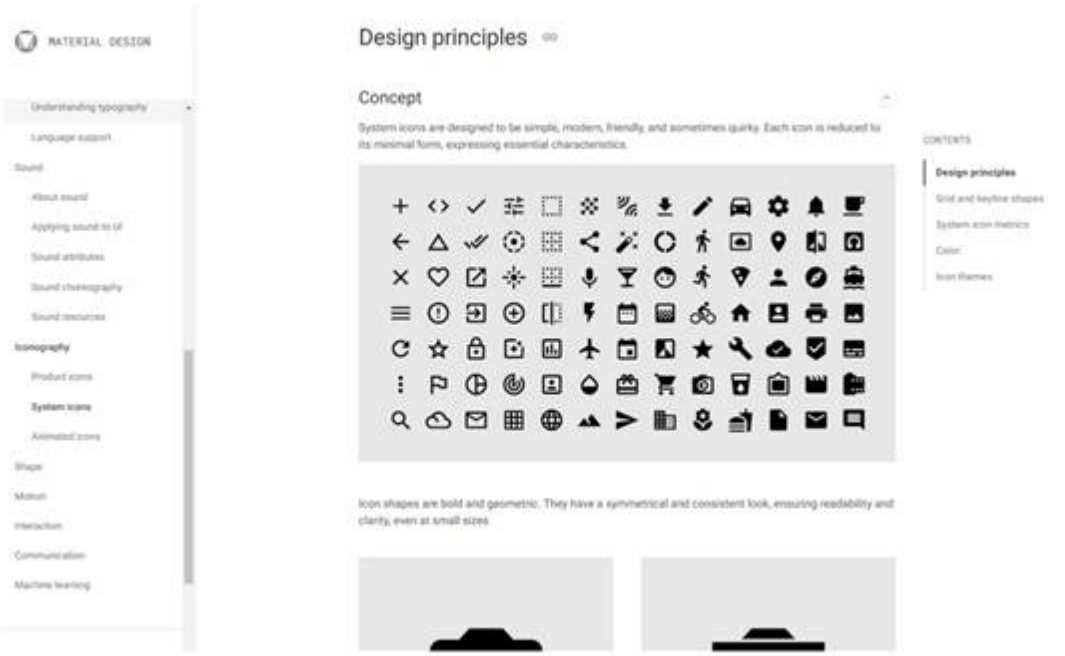
Figura 14 – Sons



Fonte: Materail Design, S.d.

Na parte de iconografia, podemos aprender sobre como criar ícones e animação de ícones.

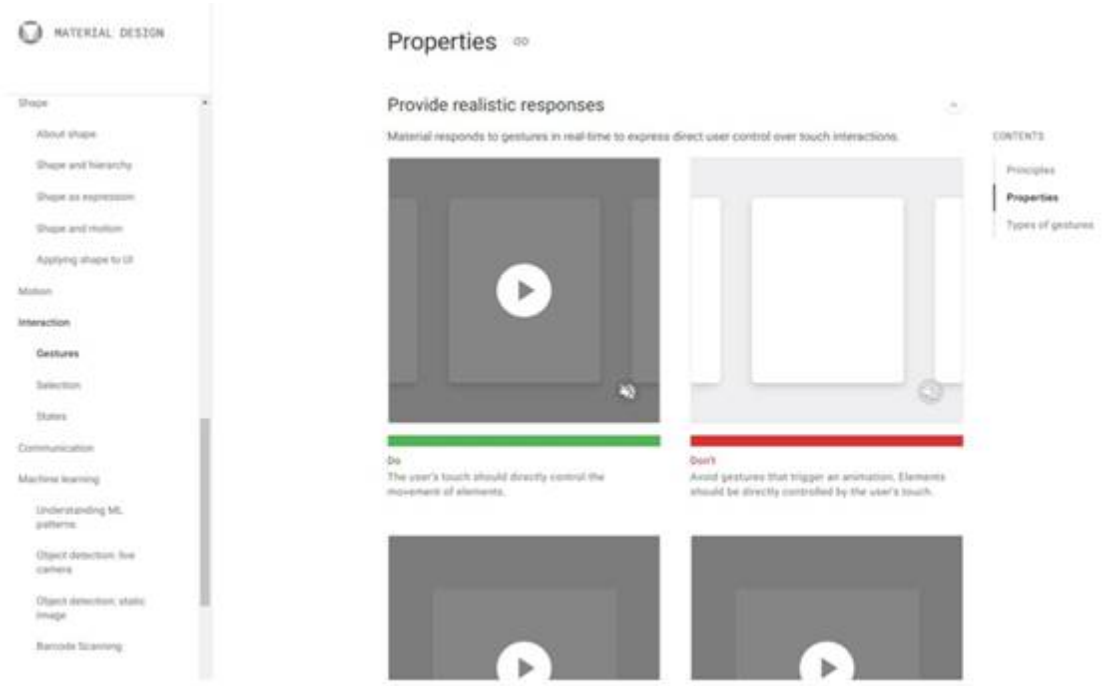
Figura 15 – Ícones



Fonte: Materail Design, S.d.

Na seção de gestos, há sugestões de como usar os gestos para controlar o aplicativo.

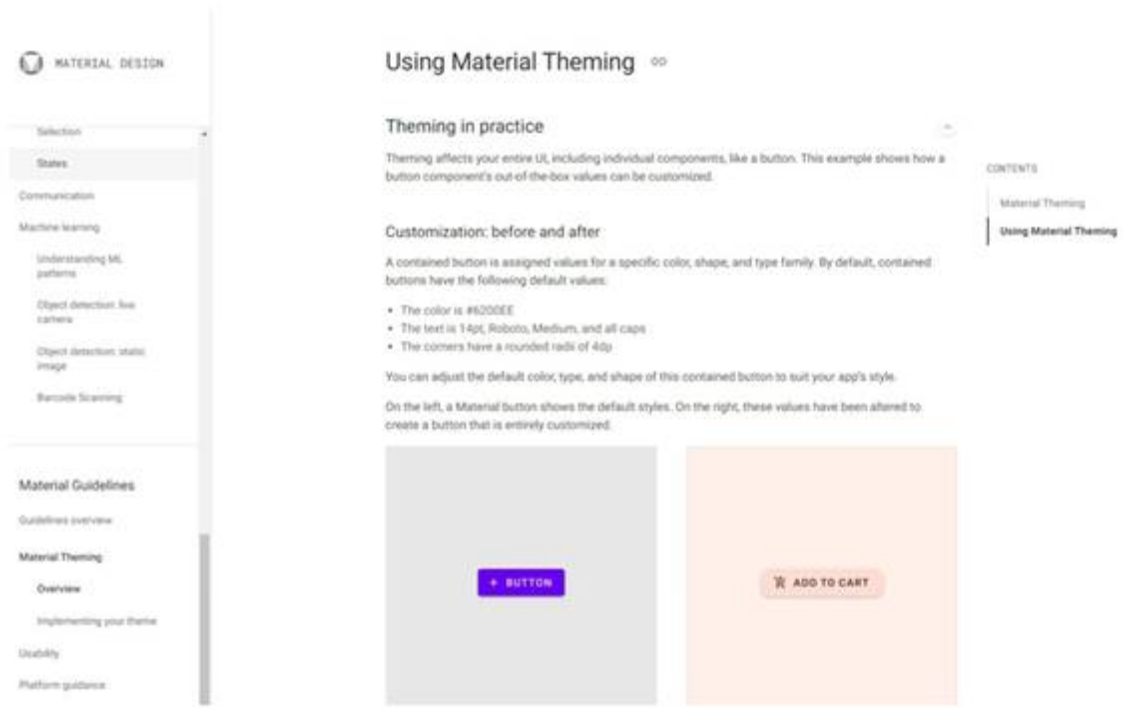
Figura 16 – Gestos



Fonte: Materal Design, S.d.

Finalmente, há uma parte sobre aprendizado de máquina e um guia do Material Design.

Figura 17 – Guia



Fonte: Material Design, S.d.

TEMA 5 – GERANDO E PUBLICANDO UM APK

Talvez a parte mais importante do desenvolvimento de um aplicativo é o seu lançamento e publicação, pois antes disso o projeto se mantém indisponível para os seus clientes.

Existem várias etapas para a geração e publicação de um APK, mas aqui vamos cobrir os seguintes tópicos:

1. Adicionar um ícone ao aplicativo;
2. Alterar o nome do pacote;
3. Construir o aplicativo para lançamento;
4. Publicar o aplicativo.

5.1 ADICIONAR UM ÍCONE AO APLICATIVO

Todo aplicativo Flutter vem com um ícone padrão do Flutter, assim como demonstrado a seguir no *site* indicado no *Saiba mais*.

Saiba mais

FLUTTER. Disponível em: <https://flutter.dev/>. Acesso em: 25 jul. 2022.

Para se alterarem os ícones utilizados, deve-se ir para a pasta do projeto e entrar em:

- nomeDoProjeto\aplicaçãoFlutter\android\app\src\main\res

Dentro desta pasta, encontra-se as 5 pastas de *mipmap*, cada um com um tamanho do mesmo ícone a ser utilizado. A forma mais fácil de alterar os ícones é criar um novo arquivo de imagem png, e substituir as imagens de ícones anteriores, removendo-as.

Lembre-se de manter as dimensões da imagem de cada ícone:

- hdpi: 72x72;
- mdpi: 48x48;
- xhdpi: 96x96;
- xxhdpi: 144x144;

- xxxhdpi: 192x192.

5.2 ALTERAR O NOME DO PACOTE

O nome do pacote do seu aplicativo, ou *package name*, é um nome que deve ser único em toda a loja de aplicativos. Assim como um nome de *site*, o nome do pacote do seu aplicativo não pode ser igual a nenhum outro aplicativo publicado.

A convenção é nomear os pacotes como se fossem um *site*. Quando se cria um novo aplicativo, o padrão utilizado é com.example.nomeDoAplicativo, as desenvolvedoras normalmente utilizam o nome do *site* da empresa como base, seguido pelo nome do aplicativo.

Por exemplo, se a Google lançar um aplicativo chamado *Teste*, o nome do pacote seria o nome do *site* (google.com) em ordem invertida: "com.google.Teste".

Se você possuir uma empresa com o *site* "nomeDaEmpresa.net", e lançar um aplicativo chamado *Exemplar*, por convenção, o nome do pacote será "net.nomeDaEmpresa.Exemplar".

Para alterar o nome do pacote, abriremos o arquivo build.gradle. Isso é possível ser feito no VSCode, ou diretamente no arquivo com um editor de texto. Procure por "nomeDoAplicativo/android/build.gradle", mas cuidado para não confundir com o arquivo presente em "nomeDoAplicativo/android/app/build.gradle".

Dentro do arquivo, procure pelo seguinte texto:

```
defaultConfig {
    // TODO: Specify your own unique Application ID
(https://developer.android.com/studio/build/application-id.html).
    applicationId "com.example.flutter_application_1"
    minSdkVersion flutter.minSdkVersion
    targetSdkVersion flutter.targetSdkVersion
    versionCode flutterVersionCode.toInteger()
    versionName flutterVersionName
}
```

E substitua o *applicationId* com o nome do pacote desejado.

5.3 CONSTRUIR O APLICATIVO PARA LANÇAMENTO

Para construir o aplicativo, é só rodar o comando em um novo terminal. Dentro do VSCode, é só clicar em Terminal → Novo Terminal, e então digitar a seguinte linha de código:

- flutter build apk

Esse comando irá gerar um arquivo apk dentro da pasta "nomeDoAplicativo/build/app/outputs/flutter-apk".

5.4 PUBLICAR O APLICATIVO

Para se publicar um aplicativo na Google Play Store, é necessário criar uma conta de desenvolvedor da Google, que possui um custo de 25 dólares.

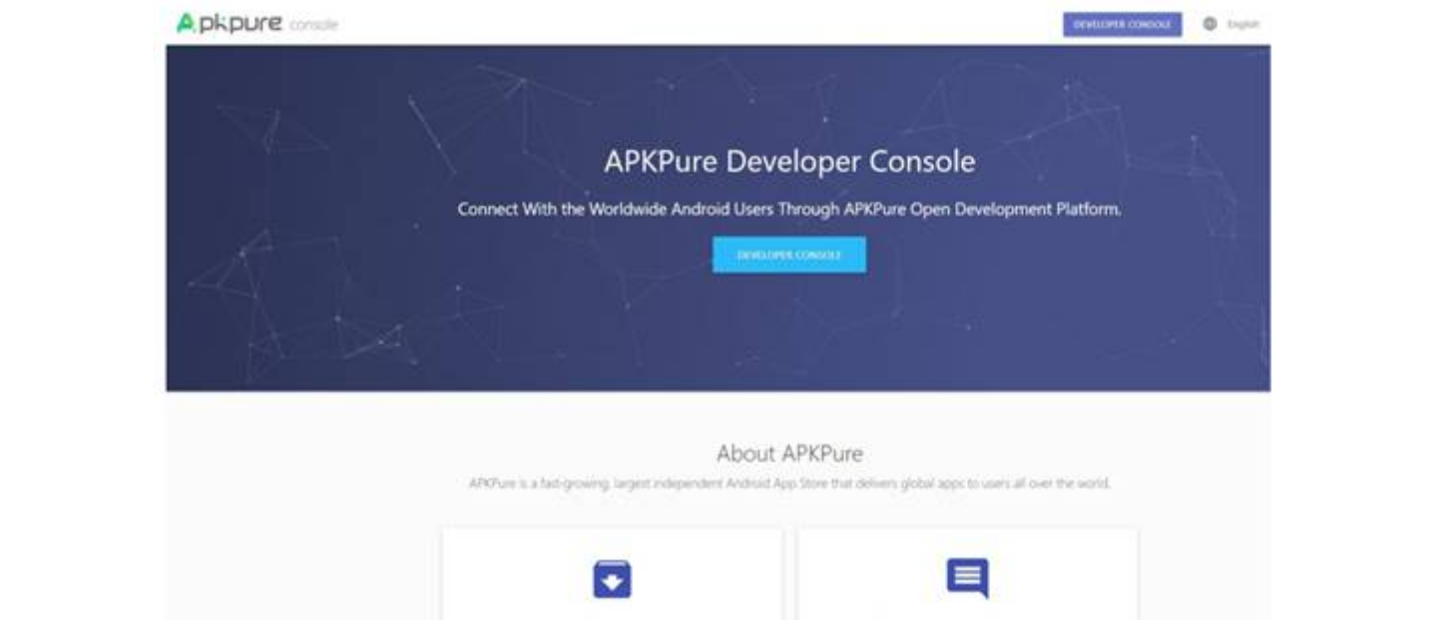
Portanto, para caráter educativo, iremos publicar o nosso aplicativo em uma loja alternativa chamada APKPure.

Entraremos no *site* de desenvolvedor da APKPure.

Saiba mais

APKURE. Disponível em: <https://developer.apkpure.com/>. Acesso em: 25 jul. 2022.

Figura 18 – *Site* de desenvolvedor da APKPure



Fonte: APKure, S.d.

Após criar uma conta gratuita no *site*, o *site* abrirá a opção de requisitar autoria de um aplicativo no *site*, ou adicionar uma nova aplicação. Ao adicionar uma nova aplicação, devemos preencher algumas informações como o nome do pacote, o nome do aplicativo, entre outros.

Uma parte importante do processo de publicação é que o desenvolvedor deverá anexar um *link* para uma política de privacidade do aplicativo.

Saiba mais

É possível criar um de forma gratuita acessando o *link* a seguir, lembrando que a geração ser gratuita ou não pode depender do tipo de aplicativo a usado.

APP PRIVACY POLICY. App Privacy Policy Generator Generator. Disponível em: <link>. Acesso em: 25 jul. 2022.

Após todos os dados serem completados, deve-se então aguardar até 2 dias úteis até que o sistema decida se o aplicativo será publicado ou não no *site*, sendo possível ver a situação do aplicativo pelo próprio *site*.

FINALIZANDO

Nesta etapa, aprendemos parte da pós-produção de um aplicativo, e também como utilizar as ferramentas de *debug* disponíveis do Flutter.

Nós nos preparamos para o futuro, sabendo dos melhores *sites* para aprender sobre a criação de aplicativos e nos mantermos atualizados com as futuras atualizações.

Vimos também o processo de publicação de um aplicativo e como fazê-lo.

REFERÊNCIAS

APKURE. Disponível em: <https://developer.apkpure.com/>. Acesso em: 25 jul. 2022.

DART. Disponível em: <https://dart.dev>. Acesso em: 25 jul. 2022a.

DART. Dart cheatsheet Codelab. **Dart**, S.d.b. Disponível em: <https://dart.dev/codelabs/dart-cheatsheet>. Acesso em: 25 jul. 2022.

FIREBASE. FlutterFire. Disponível em: <https://firebase.flutter.dev/>. Acesso em: 25 jul. 2022.

FLUTTER. Flutter Documentation. **Flutter**, S.d. Disponível em: <https://docs.flutter.dev/>. Acesso em: 25 jul. 2022.

MATERIAL DESIGN. Disponível em: <https://material.io/design/introduction>. Acesso em: 25 jul. 2022.