Fundamentos de aplicativos

Apps Android podem ser criados usando o Kotlin, a linguagem de programação Java e as linguagens C++. As Ferramentas do SDK do Android compilam seu código, além de todos os arquivos de dados e recursos, em um APK ou em um Android App Bundle.

Um *pacote Android*, que é um arquivo com sufixo .apk, contém o conteúdo de um app Android necessário no momento da execução. Esse é o arquivo que dispositivos Android usam para instalar o app.

Um Android App Bundle, que é um arquivo com sufixo .aab, tem o conteúdo de um projeto de app Android, incluindo alguns metadados extras que não são necessários no momento da execução. Um AAB é um formato de publicação e não pode ser instalado em dispositivos Android. Isso **adia a geração e a assinatura do APK para uma fase posterior.**

Ao distribuir seu app pelo Google Play, por exemplo, os servidores geram APKs otimizados que contêm apenas os recursos e códigos exigidos pelo dispositivo específico que solicita a instalação do app.

Cada app Android reside no próprio sandbox de segurança, protegido pelos seguintes recursos de segurança do Android:

- O sistema operacional Android é um sistema Linux multiusuário em que cada app é um usuário diferente.
- Por padrão, o sistema atribui a cada app um ID de usuário exclusivo do Linux, que é usado apenas pelo sistema e é desconhecido para o app. O sistema define permissões para todos os arquivos em um app de modo que somente o ID do usuário atribuído a esse app possa acessá-los.
- Cada processo tem a própria máquina virtual (VM), portanto, o código de um app é executado isoladamente de outros apps.
- Por padrão, cada aplicativo é executado no próprio processo do Linux. O sistema Android inicia o processo quando qualquer componente do app precisa ser executado e, em seguida, encerra o processo quando ele não é mais necessário ou quando o sistema precisa recuperar memória para outros apps.

O sistema Android implementa o *princípio de privilégio mínimo* (link em inglês). Ou seja, cada app, por padrão, tem acesso apenas aos componentes necessários para a execução do trabalho e nada mais. Isso cria um ambiente muito seguro em que um app não pode acessar partes do sistema para as quais não tem permissão.

No entanto, existem maneiras de um app compartilhar dados com outros apps e de um app acessar os serviços do sistema:

- É possível fazer com que dois apps compartilhem o mesmo ID de usuário do Linux. Nesse caso, eles podem acessar os arquivos um do outro. Para economizar recursos do sistema, os apps com o mesmo ID de usuário também podem ser configurados para serem executados no mesmo processo do Linux e compartilharem a mesma VM. Os apps também precisam ser assinados com o mesmo certificado.
- Um app pode solicitar permissão para acessar dados do dispositivo, como a localização, a câmera e a conexão Bluetooth. O usuário precisa conceder essas permissões de forma explícita. Para mais informações sobre permissões, consulte Permissões no Android.

O restante deste documento apresenta os conceitos a seguir:

- Componentes fundamentais de biblioteca que definem o aplicativo.
- O arquivo de manifesto em que você declara os componentes e os recursos de dispositivo necessários para o app.
- Recursos separados do código do app e que permitem otimizar o comportamento dele para uma variedade de configurações de dispositivos.

Componentes do app

Os componentes são os elementos básicos de um app Android. Cada componente é um ponto de entrada pelo qual o sistema ou um usuário pode entrar no seu app. Alguns componentes dependem de outros.

Há quatro tipos de componentes de app:

- Atividades
- Serviços
- Broadcast receivers
- Provedores de conteúdo

Cada tipo tem uma finalidade distinta e tem um ciclo de vida distinto que define como um componente é criado e destruído. As seções a seguir descrevem os quatro tipos de componentes do aplicativo.

Atividades

Uma *atividade* é o ponto de entrada para interagir com o usuário. Ele representa uma única tela com uma interface do usuário. Por exemplo, um app de e-mails pode ter uma atividade que mostra uma lista de novos e-mails, outra atividade para escrever um e-mail e outra para ler e-mails. Embora as atividades funcionem juntas para formar uma experiência do usuário coesa no app de e-mails, cada uma é independente das outras.

Um app diferente pode iniciar qualquer uma dessas atividades se o app de e-mails permitir. Por exemplo, um app de câmera pode iniciar a atividade no app de e-mails para escrever um novo e-mail e permitir que o usuário compartilhe uma imagem.

Uma atividade facilita as principais interações a seguir entre sistema e aplicativo:

- Monitorar o que é importante para o usuário no momento (o que está na tela) para que o sistema continue executando o processo que hospeda a atividade.
- Saber quais processos usados anteriormente contêm atividades interrompidas a que o usuário pode retornar e priorizar esses processos de forma mais alta para mantê-los disponíveis.
- Ajudar o app a processar o encerramento do processo para que o usuário possa retornar às atividades com o estado anterior restaurado.
- Oferecer uma maneira para que os apps implementem fluxos de usuários entre si e para que o sistema coordene esses fluxos. O principal exemplo é o compartilhamento.

Implemente uma atividade como uma subclasse da classe Activity. Para mais informações sobre a classe Activity, consulte Introdução a atividades.

Serviços

Um *serviço* é um ponto de entrada de uso geral para manter um app em execução em segundo plano por todos os tipos de motivos. Ele é um componente executado em segundo plano para realizar operações de longa duração ou trabalhar para processos remotos. Serviços não apresentam uma interface do usuário.

Por exemplo, um serviço pode tocar música em segundo plano enquanto o usuário está em um app diferente ou buscar dados na rede sem bloquear a interação do usuário com uma atividade. Outro componente, como uma atividade, pode iniciar o serviço e permitir que ele seja executado ou vinculado a ele para interagir com ele.

Há dois tipos de serviços que informam ao sistema como gerenciar um app: serviços iniciados e serviços vinculados.

Os **serviços iniciados** orientam o sistema a mantê-los em execução até que o trabalho seja concluído. Isso pode ser a sincronização de alguns dados em segundo plano ou a reprodução de músicas mesmo depois que o usuário sai do app. Sincronizar dados em segundo plano ou tocar músicas representam diferentes tipos de serviços iniciados, que o sistema processa de forma diferente:

- A reprodução de música é algo de que o usuário está diretamente ciente, e o app comunica isso ao sistema indicando que quer estar em primeiro plano, com uma notificação para informar ao usuário que ela está em execução. Nesse caso, o sistema prioriza manter o processo desse serviço em execução, porque o usuário terá uma experiência ruim se ele for encerrado.
- Um serviço normal de segundo plano não é algo de que o usuário está diretamente ciente, então o sistema tem mais liberdade para gerenciar o próprio processo. Ele pode ser eliminado, reiniciando o serviço algum tempo depois, se precisar de RAM para coisas que são de interesse mais imediato do usuário.

Os **serviços vinculados** são executados porque algum outro app (ou o sistema) disse que quer usar o serviço. Um serviço vinculado fornece uma API para outro processo, e o sistema sabe que há uma dependência entre esses processos. Portanto, se o processo A estiver vinculado a um serviço no processo B, o sistema vai saber que precisa manter o processo B e o serviço dele em execução para A. Além disso, se o processo A é importante para o usuário, ele sabe tratar o processo B como algo que também importa para o usuário.

Devido à flexibilidade deles, os serviços são elementos básicos úteis para todos os tipos de conceitos de sistema de nível superior. Plano de fundo interativo, listeners de notificação, protetores de tela, métodos de entrada, serviços de acessibilidade e muitos outros recursos principais do sistema são criados como serviços que os aplicativos implementam e aos quais o sistema se vincula quando são executados.

Um serviço é implementado como uma subclasse de Service. Para mais informações sobre a classe Service, consulte a Visão geral dos serviços.

Observação:caso seu app seja destinado ao Android 5.0 (API de nível 21) ou versões mais recentes, use a classe **JobScheduler** para programar ações. O JobScheduler tem a vantagem de economizar bateria ao programar jobs de maneira ideal para reduzir o consumo de energia e trabalhar com a API Soneca. Para mais informações sobre como usar essa classe, consulte a documentação de referência **JobScheduler**.

Broadcast receivers

Um *broadcast receiver* é um componente que permite que o sistema envie eventos ao app fora de um fluxo normal de usuários para que ele possa responder a anúncios de transmissão do sistema. Como os broadcast receivers são outra entrada bem definida no app, o sistema pode entregar transmissões até mesmo para apps que não estejam em execução no momento.

Assim, por exemplo, um app pode agendar um alarme para postar uma notificação e informar o usuário sobre um evento futuro. Como o alarme é entregue a um BroadcastReceiver no app, não é necessário que o app permaneça em execução até que o alarme dispare.

Muitas transmissões são originadas do sistema, como uma transmissão que anuncia que a tela está desligada, a bateria está fraca ou que uma imagem foi capturada. Os apps também podem iniciar transmissões, como para permitir que outros apps saibam que alguns dados foram transferidos por download para o dispositivo e estão disponíveis para uso.

Embora os broadcast receivers não mostrem uma interface do usuário, eles podem criar uma notificação na barra de status para alertar o usuário quando um evento de transmissão ocorre. No entanto, o mais comum é que um broadcast receiver seja apenas um *gateway* para outros componentes e faça uma quantidade mínima de trabalho.

Por exemplo, um broadcast receiver pode programar um JobService para realizar um trabalho com base em um evento usando JobScheduler. Os broadcast receivers geralmente envolvem apps que interagem entre si. Por isso, é importante estar ciente das implicações de segurança ao configurá-los.

Um broadcast receiver é implementado como uma subclasse de BroadcastReceiver, e cada transmissão é entregue como um objeto Intent. Para mais informações, consulte a classe BroadcastReceiver.

Provedores de conteúdo

Um *provedor de conteúdo* gerencia um conjunto compartilhado de dados do app que você pode armazenar no sistema de arquivos, em um banco de dados SQLite, na Web ou em qualquer outro local de armazenamento permanente que seu app possa acessar. Por meio do provedor de conteúdo, outros apps podem consultar ou modificar os dados, se o provedor de conteúdo permitir.

Por exemplo, o sistema Android oferece um provedor de conteúdo que gerencia os dados de contato do usuário. Qualquer app com as permissões adequadas pode consultar o provedor de conteúdo, por exemplo, usando ContactsContract.Data, para ler e gravar informações sobre uma pessoa específica.

É tentador pensar em um provedor de conteúdo como uma abstração em um banco de dados, porque há muita API e suporte integrados a eles para esse caso comum. No entanto, eles têm um propósito principal diferente do ponto de vista do design de sistema.

Para o sistema, um provedor de conteúdo é um ponto de entrada em um app para publicar itens de dados nomeados, identificados por um esquema de URI. Assim, um app pode decidir como mapear os dados que contém para um namespace de URI, distribuindo esses URIs para outras entidades que, por sua vez, podem usá-los para acessar os dados. Existem algumas coisas específicas que isso permite que o sistema faça ao gerenciar um aplicativo:

- A atribuição de um URI não exige que o app permaneça em execução. Portanto, os URIs podem persistir após o fechamento dos apps

proprietários. O sistema só precisa garantir que o app proprietário ainda esteja em execução ao recuperar os dados do app no URI correspondente.

- Os URIs também fornecem um importante modelo de segurança de controle preciso. Por exemplo, um app pode colocar o URI de uma imagem que está na área de transferência, mas deixar o provedor de conteúdo bloqueado para que outros apps não possam acessá-lo livremente. Quando um segundo app tenta acessar esse URI na área de transferência, o sistema pode permitir que ele acesse os dados usando uma *concessão de permissão do URI* temporária para acessar os dados apenas por trás desse URI, e nada mais no segundo app.

Os provedores de conteúdo também são úteis para ler e gravar dados particulares no seu app e não compartilhados.

Um provedor de conteúdo é implementado como uma subclasse de ContentProvider e precisa implementar um conjunto padrão de APIs que permitem que outros apps realizem transações. Para ver mais informações, consulte o guia para desenvolvedores sobre Provedores de conteúdo.

Um aspecto exclusivo do design do sistema Android é que qualquer app pode iniciar um componente de outro app. Por exemplo, se você quiser que o usuário capture uma foto com a câmera do dispositivo, provavelmente há outro app que faz isso, e seu app pode usá-lo em vez de desenvolver uma atividade para capturar uma foto por conta própria. Não é necessário incorporar nem mesmo vincular ao código do app de câmera. Em vez disso, é possível iniciar a atividade no app de câmera que captura uma foto. Quando concluída, a foto até retorna ao aplicativo em questão para ser usada. Para o usuário, parece que a câmera é realmente parte do seu app.

Quando o sistema inicia um componente, ele inicia o processo desse app, se ele ainda não está em execução, e instancia as classes necessárias para o componente. Por exemplo, se o app iniciar a atividade no app de câmera que captura uma foto, essa atividade vai ser executada no processo pertencente ao app da câmera, não no processo do app. Portanto, ao contrário dos apps da maioria dos outros sistemas, os apps Android não têm um único ponto de entrada: não há função main().

Como o sistema executa cada app em um processo separado com permissões de arquivo que restringem o acesso a outros apps, o app não pode ativar diretamente um componente de outro. No entanto, o sistema Android pode. Para ativar um componente em outro app, envie uma mensagem ao sistema que especifica sua *intent* para iniciar um componente específico. Em seguida, o sistema ativa o componente.

Ativar componentes (INTENTS)

Uma mensagem assíncrona chamada *intent* ativa três dos quatro tipos de componentes: atividades, serviços e broadcast receivers. Os intents vinculam componentes individuais uns aos outros no ambiente de execução. Você pode pensar

neles como mensageiros que solicitam uma ação de outros componentes, independente do componente que pertence ao seu app ou a outro.

Uma intent é criada com um objeto Intent, que define uma mensagem para ativar um componente específico (uma intent *explícita*) ou um tipo específico de componente (uma intent *implícita*).

Para atividades e serviços, um intent define a ação a ser executada, como *visualizar* ou *enviar* algo, e pode especificar o URI dos dados a serem usados, entre outras coisas que o componente que está sendo iniciado pode precisar saber.

Por exemplo, uma intent pode transmitir uma solicitação de uma atividade para mostrar uma imagem ou abrir uma página da Web. Em alguns casos, você pode iniciar uma atividade para receber um resultado. Nesse caso, a atividade também retorna o resultado em uma Intent. Você também pode emitir uma intent para permitir que o usuário escolha um contato pessoal e o retorne a você. A intent de retorno inclui um URI que aponta para o contato escolhido.

Para broadcast receivers, a intent define o aviso de transmissão. Por exemplo, uma transmissão para indicar que a bateria do dispositivo está fraca inclui apenas uma string de ação conhecida que indica que *a bateria está fraca*.

Ao contrário de atividades, serviços e broadcast receivers, os provedores de conteúdo são ativados quando direcionados por uma solicitação de um ContentResolver. O resolvedor de conteúdo processa todas as transações diretas com o provedor de conteúdo, e o componente que executa transações com o provedor chama métodos no objeto ContentResolver. Isso deixa uma camada de abstração por motivos de segurança entre o provedor de conteúdo e o componente que solicita informações.

Há dois métodos para ativar cada tipo de componente:

- Você pode iniciar uma atividade ou oferecer algo novo para fazer transmitindo uma Intent para startActivity() ou, quando quiser que a atividade retorne um resultado, startActivityForResult().
- No Android 5.0 (API de nível 21) e versões mais recentes, você pode usar a classe JobScheduler para programar ações. Para versões anteriores do Android, você pode iniciar um serviço ou fornecer novas instruções a um serviço em andamento transmitindo uma Intent para startService(). Você pode se vincular ao serviço transmitindo um Intent para bindService().
- Você pode iniciar uma transmissão transmitindo um Intent para métodos como sendBroadcast() ou sendOrderedBroadcast().
- Você pode executar uma consulta a um provedor de conteúdo chamando query() em um ContentResolver.

Para saber mais sobre intents, consulte o documento Intents e filtros de intents. Os documentos a seguir fornecem mais informações sobre a ativação de componentes

específicos: Introdução a atividades, Visão geral dos serviços, BroadcastReceiver e Provedores de conteúdo.

O arquivo de manifesto

Antes que o sistema Android possa iniciar um componente de app, ele precisa saber que o componente existe lendo o *arquivo de manifesto* do app, AndroidManifest.xml. O app declara todos os componentes nesse arquivo, que está na raiz do diretório do projeto do app.

O manifesto faz outras coisas além de declarar os componentes do app, como:

- Identifica todas as permissões do usuário que o app exige, como acesso à Internet ou acesso de leitura aos contatos do usuário.
- Declara o nível mínimo da API exigido pelo app com base nas APIs que ele usa.
- Declara os recursos de hardware e software usados ou exigidos pelo app, como câmera, serviços Bluetooth ou tela multitoque.
- Declara as bibliotecas de API a que o app precisa ser vinculado (além das APIs do framework do Android), como a Biblioteca Google Maps.

Declarar componentes

A tarefa principal do manifesto é informar ao sistema os componentes do aplicativo. Por exemplo, um arquivo de manifesto pode declarar uma atividade desta forma:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    <activity android:name="com.example.project.ExampleActivity"
            android:label="@string/example_label" ... >
    </activity>
    ...
  </application>
</manifest>
```

No elemento <application>, o atributo android:icon aponta para recursos de um ícone que identifica o app.

No elemento <activity>, o atributo android:name especifica o nome totalmente qualificado da classe da subclasse Activity, e o atributo android:label especifica uma string a ser usada como o rótulo da atividade visível ao usuário.

É necessário declarar todos os componentes do aplicativo que usam os elementos a seguir:

- Elementos <activity> para atividades

- Elementos <service> para serviços
- Elementos <receiver> para broadcast receivers
- Elementos <provider> para provedores de conteúdo

Atividades, serviços e provedores de conteúdo incluídos na origem, mas não declarados no manifesto, não ficam visíveis para o sistema e, consequentemente, não podem ser executados. No entanto, os broadcast receivers podem ser declarados no manifesto ou criados dinamicamente no código como objetos BroadcastReceiver e registrados no sistema chamando registerReceiver().

Para saber mais sobre a estrutura do arquivo de manifesto do seu app, consulte Visão geral do manifesto do app.

Declarar funcionalidades do componente

Conforme discutido na seção Ativar componentes, você pode usar um Intent para iniciar atividades, serviços e broadcast receivers. Para isso, nomeie explicitamente o componente de destino usando o nome da classe do componente na intent. Também é possível usar uma intent implícita, que descreve o tipo de ação a ser realizada e, opcionalmente, os dados em que ela será realizada. A intent implícita permite que o sistema encontre um componente no dispositivo que possa realizar a ação e iniciá-la. Se houver vários componentes que possam executar a ação descrita pela intent, o usuário selecionará qual deles usar.

Cuidado:se você usar uma intent para iniciar uma **Service**, use uma intent explícita para verificar se o app é seguro. O uso de uma intent implícita para iniciar um serviço representa um risco de segurança, porque não é possível determinar qual serviço responde à intent, e o usuário não pode ver qual serviço é iniciado. No Android 5.0 (nível 21 da API) e versões mais recentes, o sistema gera uma exceção se você chama **bindService()** com uma intent implícita. Não declare filtros de intent para seus serviços.

O sistema identifica os componentes que podem responder a uma intent comparando a intent recebida aos *filtros de intent* fornecidos no arquivo de manifesto de outros apps no dispositivo.

Ao declarar uma atividade no manifesto do app, você pode incluir filtros de intent que declaram os recursos da atividade para que ela possa responder a intents de outros apps. Para fazer isso, adicione um elemento <intent-filter> como filho do elemento de declaração do componente.

Por exemplo, se você criar um app de e-mails com uma atividade de escrever um novo e-mail, poderá declarar um filtro de intent para responder a intents "enviar" e enviar um novo e-mail, conforme mostrado neste exemplo:

```
<manifest ... >
    ...
```

```
  <application ... >
    <activity android:name="com.example.project.ComposeEmailActivity">
      <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <data android:type="*/*" />
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Se outro app criar uma intent com a ação ACTION_SEND e a transmitir para startActivity(), o sistema poderá iniciar a atividade para que o usuário possa redigir um e-mail e enviá-lo.

Para saber mais sobre a criação de filtros de intent, consulte o documento Intents e filtros de intent.

Declarar requisitos do app

Há vários dispositivos com tecnologia Android e nem todos oferecem os mesmos recursos e funcionalidades. Para evitar que o app seja instalado em dispositivos que não têm os recursos necessários, é importante definir claramente um perfil para os tipos de dispositivos com suporte do app. Para isso, declare os requisitos de dispositivo e software no arquivo de manifesto.

A maioria dessas declarações é apenas informativa. O sistema não as lê, mas serviços externos, como o Google Play, as leem para oferecer filtragem aos usuários quando eles pesquisam apps no dispositivo.

Por exemplo, suponha que seu app exija uma câmera e use APIs introduzidas no Android 8.0 (nível 26 da API). É necessário declarar esses requisitos. Os valores para minSdkVersion e targetSdkVersion são definidos no arquivo build.gradle do módulo do app:

```
android {
 ...
 defaultConfig {
  ...
  minSdkVersion 26
  targetSdkVersion 29
 }
}
```

Observação:não defina **minSdkVersion** e **targetSdkVersion** diretamente no arquivo de manifesto, já que eles são substituídos pelo Gradle durante o processo de build. Para mais informações, consulte Especificar requisitos de nível da API.

Declare o recurso de câmera no arquivo de manifesto do app:

```
<manifest ... >
  <uses-feature android:name="android.hardware.camera.any"
                android:required="true" />
  ...
</manifest>
```

Com as declarações mostradas nesses exemplos, dispositivos que *não* têm câmera ou têm uma versão do Android *anterior* à 8.0 não podem instalar seu app do Google Play. No entanto, você também pode declarar que o app usa a câmera, mas não a *exige*. Para fazer isso, defina o atributo required como false, confira se o dispositivo tem uma câmera durante a execução e desative todos os recursos relacionados, conforme necessário.

Confira mais informações sobre como gerenciar a compatibilidade do seu app com diferentes dispositivos na Visão geral da compatibilidade do dispositivo.

Recursos do app

Um app Android é composto de mais do que apenas código. Ela exige recursos separados do código-fonte, como imagens, arquivos de áudio e tudo relacionado à apresentação visual do app. Por exemplo, você pode definir animações, menus, estilos, cores e o layout das interfaces do usuário da atividade com arquivos XML.

O uso de recursos do app facilita a atualização de várias características do app sem modificar o código. Fornecer conjuntos de recursos alternativos permite otimizar o app para diversas configurações de dispositivo, como diferentes idiomas e tamanhos de tela.

Para cada recurso incluído no projeto Android, as ferramentas de build do SDK definem um ID de número inteiro exclusivo, que você pode usar para referenciar o recurso do código do app ou de outros recursos definidos no XML. Por exemplo, se o app tiver um arquivo de imagem chamado logo.png (salvo no diretório res/drawable/), as ferramentas do SDK vão gerar um ID de recurso chamado R.drawable.logo. Esse ID é mapeado para um número inteiro específico do app, que pode ser usado para referenciar a imagem e inseri-la na interface do usuário.

Um dos aspectos mais importantes de fornecer recursos separados do código-fonte é a capacidade de fornecer recursos alternativos para diferentes configurações do dispositivo.

Por exemplo, ao definir strings de interface em XML, você pode traduzir as strings para outros idiomas e salvá-las em arquivos separados. Em seguida, o Android aplica as strings de idioma adequadas à IU com base em um *qualificador* de idioma anexado ao nome do diretório de recursos, como res/values-fr/ para valores de string em francês e a configuração de idioma do usuário.

O Android oferece suporte a muitos qualificadores para recursos alternativos. O qualificador é uma string curta incluída no nome dos diretórios de recursos para definir a configuração do dispositivo em que esses recursos são usados.

Por exemplo, você pode criar layouts diferentes para suas atividades dependendo da orientação e do tamanho da tela do dispositivo. Quando a tela do dispositivo está na orientação retrato (altura), você pode querer um layout com botões organizados verticalmente, mas quando a tela está na orientação paisagem (larga), convém alinhar os botões horizontalmente. Para mudar o layout dependendo da orientação, você pode definir dois layouts e aplicar o qualificador adequado ao nome do diretório de cada layout. Em seguida, o sistema aplica automaticamente o layout adequado conforme a orientação atual do dispositivo.

Para saber mais sobre os diferentes tipos de recursos que você pode incluir no aplicativo e como criar recursos alternativos para diferentes configurações de dispositivos, leia Visão geral dos recursos de app. Para saber mais sobre as práticas recomendadas e como projetar apps robustos com qualidade de produção, consulte o Guia para a arquitetura do app.