Implementação de um Aplicativo em Java com o Android Studio v2025.2

Autor: Marcos Virgilio da Costa
Versão 14.20251118

Índice

Introdução

Esse documento apresenta uma sequência de passos para o desenvolvimento de um aplicativo Android com a linguagem Java, utilizando a IDE Android Studio.

A implementação do Aplicativo Android exemplificada no decorrer do documento contempla as seguintes funcionalidades:

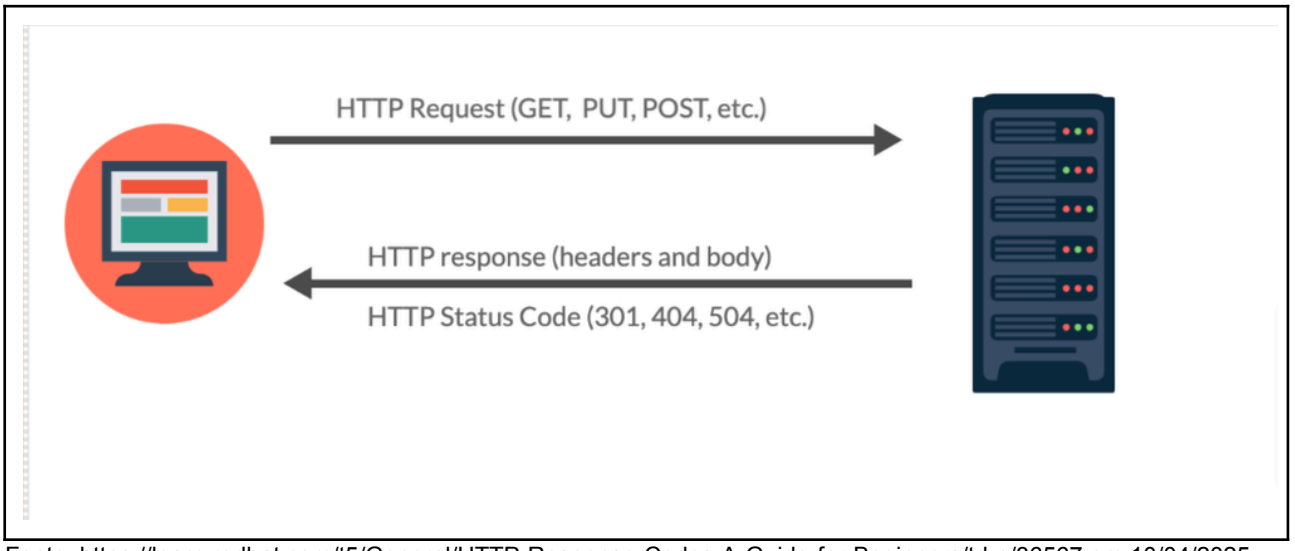
- Barra de Botões Inferior para Navegação entre as principais telas do aplicativo;
- Tela (Fragment) para o Cadastro de Usuário, utilizando um formulário;
- Requisição (JSON Array Request) para um Web Service que retorna um array de objetos (Perfis de Acesso) no formato JSON, para popular a lista de opções de um componente Spinner da tela de cadastro;
- Requisição (JSON Object Request) com os valores informados nos campos do formulário, no formato JSON, para um Web Service de cadastro;
- Tela (Fragment) para a Consulta de Usuários, utilizando o componente *RecyclerViewAdapter* para a listagem dos usuários já cadastrados;
- Requisição (JSON Array Request) para um Web Service que retorna um array de objetos (usuários cadastrados) no formato JSON;

Esse documento não contempla os passos para a implementação do lado servidor, ou seja, dos Web Services, uma vez que a IDE Android Studio é destinada especificamente para a implementação de aplicativos Android.

Entretanto, no final do documento são disponibilizados os códigos fontes dos Web Services consumidos pelo aplicativo, na linguagem PHP, apenas para a validação e testes do aplicativo, uma vez que os mesmos não possuem nenhuma implementação de segurança como JWT (Json Web Token), OAuth 2.0, etc.

O Modelo de comunicação Request x Response

O modelo de comunicação Request x Response (Requisição-Resposta) é um paradigma fundamental na computação distribuída e na arquitetura de sistemas, especialmente na comunicação cliente-servidor. Ele descreve um padrão de interação onde um cliente envia uma requisição para um servidor, e o servidor, após processar essa requisição, envia uma resposta de volta ao cliente.



Fonte: https://learn.redhat.com/t5/General/HTTP-Response-Codes-A-Guide-for-Beginners/td-p/36567 em 10/04/2025.

Provavelmente os exemplos mais comuns desse modelo de comunicação são a navegação na web (browser enviando uma requisição HTTP e o servidor web respondendo com a página HTML), as chamadas de API (aplicações solicitando dados ou serviços de outras aplicações por uma rede) e os sistemas de gerenciamento de banco de dados (SGBD).

Características Principais:

- Iniciação pelo Cliente: A comunicação é sempre iniciada pelo cliente, que envia uma solicitação específica de informação ou ação para o servidor.
- Natureza Síncrona (na forma mais básica): Em sua forma mais simples, o cliente aguarda (bloqueia) a resposta do servidor antes de prosseguir com outras operações. No entanto, variações assíncronas também existem.
- Direcionalidade: A comunicação é predominantemente bidirecional, com uma requisição fluindo do cliente para o servidor e uma resposta fluindo do servidor de volta para o cliente.

Principais Métodos de Request

- **GET:** Utilizado para **recuperar** dados de um recurso específico. É um método que não modifica o servidor, mas é inseguro porque as informações são geralmente passadas de forma aberta na URL.

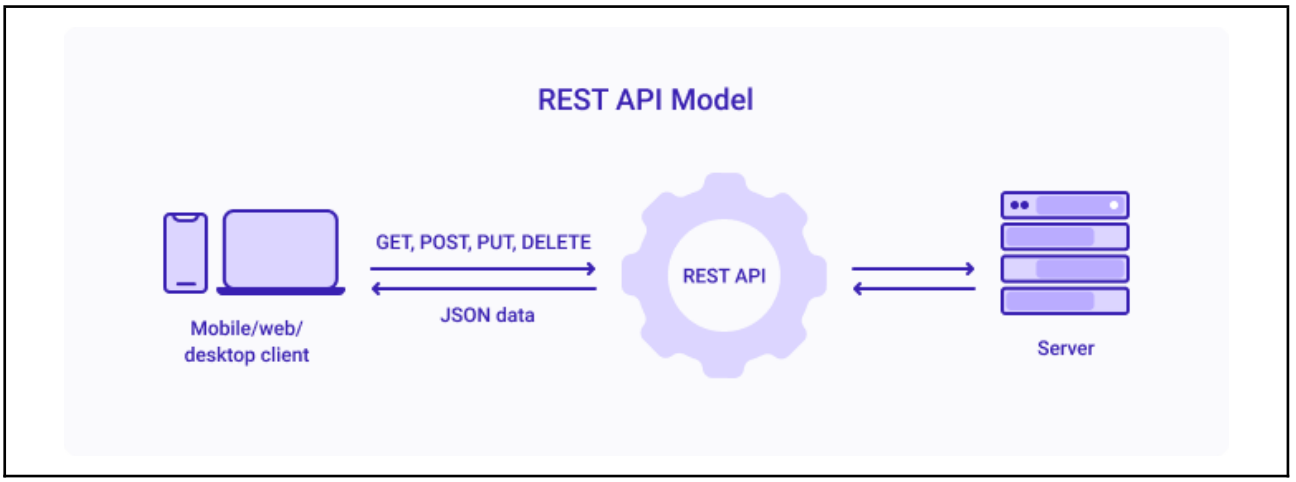
- **POST:** Utilizado para **enviar dados** para o servidor para criar um novo recurso ou submeter dados a serem processados. É um pouco mais seguro porque os dados são geralmente enviados no corpo da requisição.
- **PUT:** Utilizado para **atualizar** um recurso existente ou criar um novo recurso no URI fornecido. Neste método os dados que representam o recurso completo são enviados no corpo da requisição.
- **DELETE:** Utilizado para **remover** um recurso específico no servidor. Ao tentar deletar um recurso inexistente resulta na resposta "não encontrado".
- **PATCH:** Utilizado para aplicar **modificações parciais** a um recurso. É semelhante ao PUT, mas em vez de substituir todo o recurso, aplica apenas as alterações especificadas no corpo da requisição.
- **HEAD:** Similar ao GET, mas **não retorna o corpo da resposta**. É útil para obter metadados sobre um recurso (como tipo de conteúdo, tamanho, última modificação) sem transferir o conteúdo em si.
- **OPTIONS:** Utilizado para **descrever as opções de comunicação** para o recurso de destino. Permite que o cliente determine os métodos HTTP suportados pelo servidor para um determinado recurso.
- **CONNECT:** Estabelece um **túnel** para o servidor identificado pelo recurso de destino. É comumente usado para estabelecer conexões seguras (como HTTPS) através de um proxy.
- **TRACE:** Realiza um **teste de loop-back** da mensagem para o servidor. O servidor retorna a mensagem recebida, permitindo que o cliente veja quaisquer alterações ou adições que proxies ou outros intermediários possam ter feito.

Web Services

Permitem a comunicação e integração entre sistemas diferentes usando tecnologias web e protocolos padrões da internet, como HTTP. Eles funcionam como um "tradutor" universal, independentemente da linguagem de programação ou plataforma de hardware subjacente. Sua principal finalidade é permitir a troca de informações entre sistemas, independentes de plataforma ou linguagem computacional.

No esquema abaixo é apresentada a arquitetura do sistema proposto nesta apostila, que utiliza Web services do lado servidor e um aplicativo Android desenvolvido com a linguagem Java do lado cliente.

Os Web Services estão sendo representados na figura abaixo como “REST API”, mas a implementação proposta não precisa utilizar Web Services com arquitetura RESTFul especificamente, sendo possível utilizar APIs simples.



Fonte: Adaptado de https://hevodata.com/learn/api-vs-rest-api/ em 20/07/2023.

JavaScript Object Notation (JSON)

É um padrão aberto e independente de troca de dados simples entre sistemas.

Os objetos JSON são definidos entre chaves {} e podem conter múltiplas tags do tipo **nome:valor,** como nos exemplos abaixo:

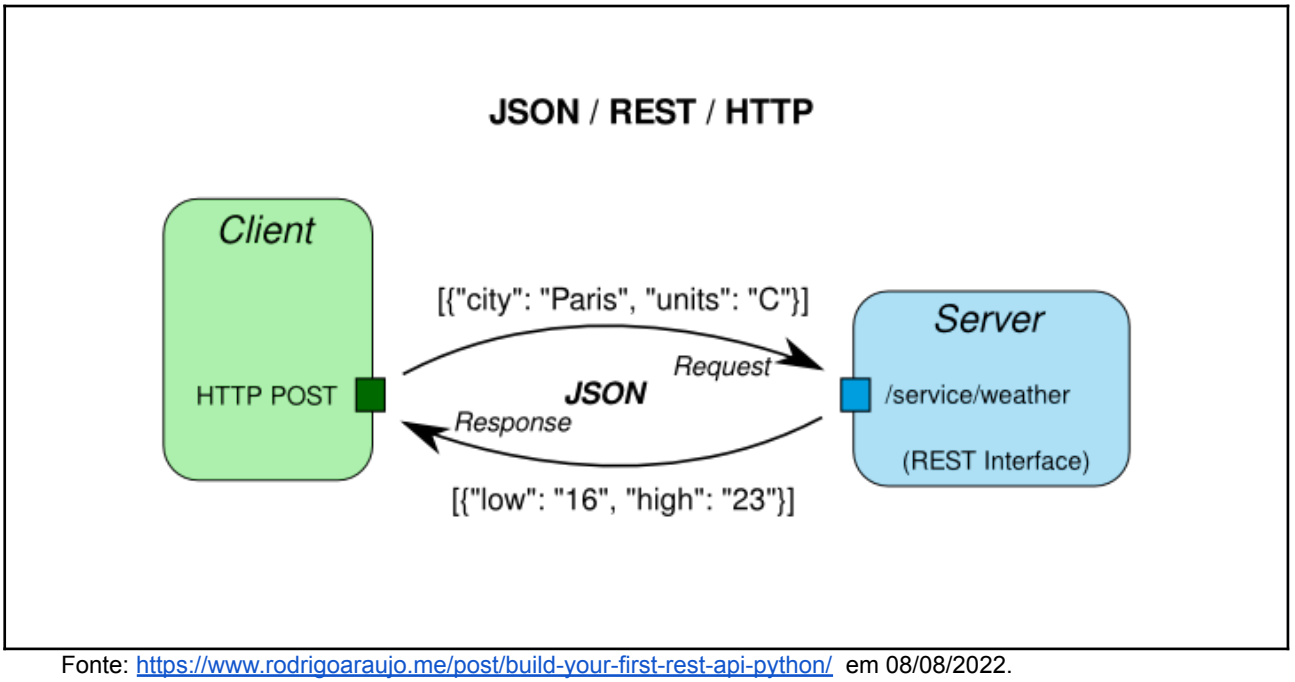
```
{ "nome" : "Jose Carlos" , "sobrenome" : "Macoratti" }

{"id":1,      "descricao":"Notebook      14",      "serial":"PRD-5381",
"marca":"Acer"}
```

As trocas de dados entre sistemas geralmente consiste em uma troca de grandes quantidades de dados, e o formato JSON permite a criação de ***arrays*** de objetos JSON, que ficam entre os caracteres de [ ], como no exemplo abaixo:

```
[ { "nome":"Jose Carlos" , "sobrenome":"Macoratti" },
{ "nome":"Ana Clara " , "sobrenome":"Lima" },
{ "nome":"Pedro Luiz" , "sobrenome":"Ramos" } ]
```

Justamente pela sua simplicidade e compactação, os objetos JSON são usados como um padrão para troca de dados entre Web Services, como mostrado no esquema abaixo:



Fonte: https://www.rodrigoaraujo.me/post/build-your-first-rest-api-python/ em 08/08/2022.

Criando um novo projeto no Android Studio

Passos para criar um novo projeto simples no Android Studio.

1. Na tela inicial do Android Studio, acione o botão ***New Project***;
2. Ao abrir a próxima tela, selecione o modelo ***Empty Views Activity*** e acione o botão ***Next*** para continuar***;***
3. Na próxima tela, preencha o campo ***Name*** com o nome desejado para o aplicativo (o nome vai aparecer na gaveta de aplicativos do celular, junto com o ícone do mesmo).
4. No campo ***Package Name***, informe o nome completo do pacote do aplicativo. Esta informação tem como objetivo gerar um identificador único para o aplicativo na Play Store do Google, que não permitirá a publicação se já houver um igual.
   Sugestão: `br.com.instituicao.mobile.nomedoaplicativo;`
   Exemplo: `br.com.xapsoftware.mobile.seg;`
5. No campo ***Save Location***, informe a pasta de armazenamento dos arquivos do projeto no sistema operacional;
6. Selecione a linguagem ***Java*** no campo ***Language***;
7. Selecionar a versão mínima do Android que será compatível com o Aplicativo. Essa decisão depende de cada projeto, porque restringe a instalação do aplicativo em aparelhos que tenham versões mais antigas do Android;
8. Acione o botão **Finish**;

Ativando o Sistema de Controle de Versão Git

1. Acione a opção de menu ***VCS>Enable Version Control Integration***;
2. Selecione a opção ***Git*** na caixa de seleção;
3. Acione o botão ***OK*** para confirmar o sistema escolhido;

Compartilhando os fontes do projeto no GitHub

1. Executar os comandos abaixo no Terminal do Android Studio, para configurar o usuário padrão do Github no Android Studio. Substituir `username` pelo seu nome de login no GitHub mantendo as aspas duplas e `email@email.com` o pelo seu email de login do GitHub, também mantendo as aspas duplas :

```
git config --global user.name "username"

git config --global user.email "email@email.com"
```

2. Acione a opção de menu ***Git>GitHub>Share Project*** on ***GitHub***;

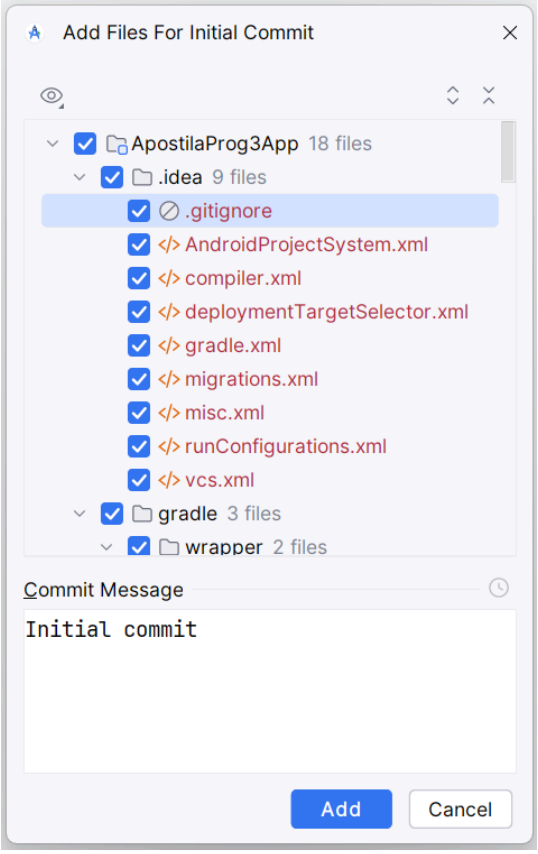
3. Na pequena tela que abriu, informe no campo ***Repository Name*** o nome que deseja para o repositório do projeto no ***GitHub***;

4. Mantenha o campo ***Remote*** com o valor que já veio preenchido;

5. Informe uma breve descrição do projeto, se desejar, no campo ***Description***;

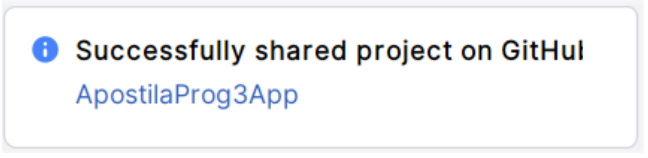
6. Acione o botão ***Share*** para confirmar;

7. Será aberta uma nova tela com todos os arquivos do projeto selecionados, como na figura abaixo:



8. Acione o botão ***Add*** para confirmar a inclusão dos arquivos no repositório do ***GitHub***;

9. Após o envio dos arquivos para o repositório, uma mensagem deve ser apresentada no canto inferior direito, confirmando a ação;

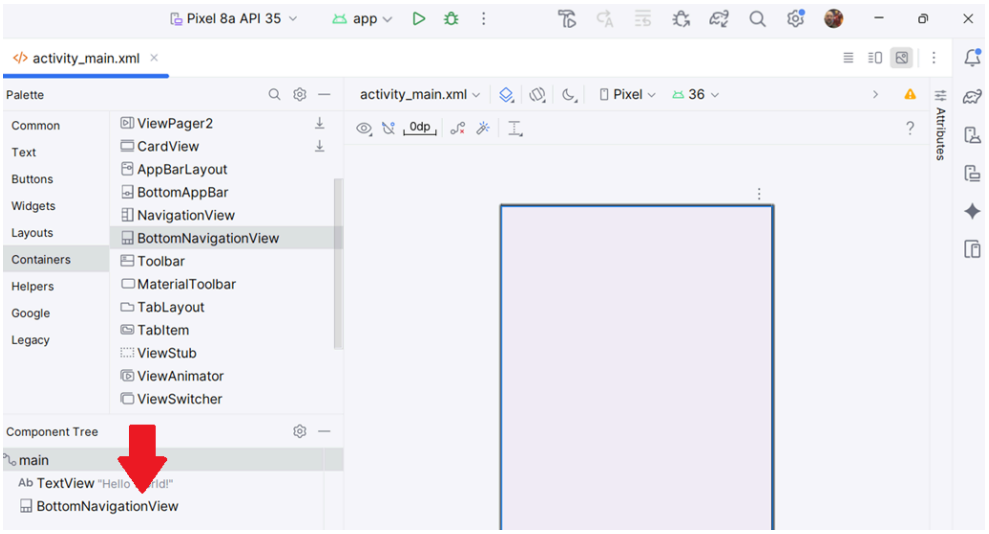


Configurando uma Barra de Navegação Inferior na tela principal

Adicionando um componente BottomNavigationView ao Layout do ActivityMain

1. Abra o arquivo `activity_main.xml` na pasta `res>layout`;

2. Arraste o componente `ButtonNavigationView` para o final da `Component Tree`, como demonstrado com a seta vermelha na figura abaixo;



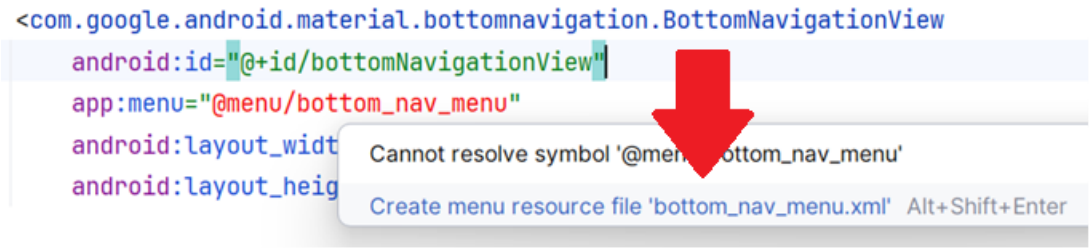
3. Depois de arrastar o componente, altere o modo de visualização para ***Code***, para que possamos editar o código XML do `activity_main.xml`;
4. Incluir uma tag **`<fragment/>`** antes da tag **`<com.google.android.material.bottomnavigation.BottomNavigationView/>`** que será usada para apresentar cada ***layout/ Fragment*** correspondente a navegação acionada;

```
<fragment
    android:id="@+id/nav_host_fragment_activity_main"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    app:defaultNavHost="true"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toTopOf="@+id/bottomNavigationView"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:navGraph="@navigation/mobile_navigation" />
```

5. Agora altere a tag **android:layout_height** do **BottomNavigationView**, mudando o valor de **"match_parent"** para **"wrap_content"**
6. Ainda no **BottomNavigationView**, inclua as tags listadas abaixo **em vermelho**;

```
<com.google.android.material.bottomnavigation.BottomNavigationView
   android:id="@+id/bottomNavigationView"
   app:labelVisibilityMode="labeled"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintEnd_toEndOf="parent"
   app:menu="@menu/bottom_nav_menu"
         android:layout_width="match_parent"
         android:layout_height="wrap_content" />
```

7. Mantenha o cursor do mouse parado sobre o texto **"@menu/bottom _nav_menu"** até que apareça a opção **Create menu resource file …** como demonstrado na figura abaixo;



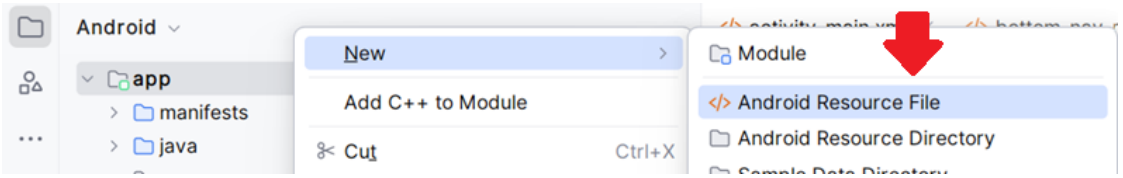
8. Clique sobre a opção **Create menu resource file …**;
9. Na tela ***New Resource File*** não altere nada e apenas acione o botão **OK**;
10. Observe que foi gerado o arquivo `botton_nav_menu.xml` na pasta `res>menu`;
11. Ao final das alterações o arquivo `activity_main.xml` deve ficar com os dois componentes, como listado no exemplo abaixo:

```
<fragment
   android:id="@+id/nav_host_fragment_activity_main"
   android:name="androidx.navigation.fragment.NavHostFragment"
   android:layout_width="match_parent"
   android:layout_height="0dp"
   app:defaultNavHost="true"
   app:layout_constraintTop_toTopOf="parent"
   app:layout_constraintBottom_toTopOf="@+id/bottomNavigationView"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintEnd_toEndOf="parent"
   app:navGraph="@navigation/mobile_navigation" />
```

```
<com.google.android.material.bottomnavigation.BottomNavigationView
   android:id="@+id/bottomNavigationView"
   app:labelVisibilityMode="labeled"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintEnd_toEndOf="parent"
   app:menu="@menu/bottom_nav_menu"
   android:layout_width="match_parent"
   android:layout_height="wrap_content" />
```

Adicionando um recurso de navegação ao projeto

1. Acione o menu de contexto do mouse sobre a pasta **app** e acione o menu **new>Android Resource File**;

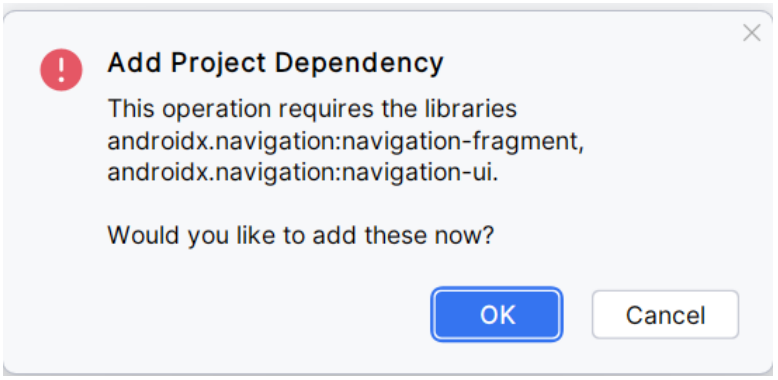


2. Ao abrir a tela ***New Resource File***, informe no campo ***File name:*** o seguinte texto `mobile_navigation`***;***

3. No campo ***Resource type:*** selecione a opção **Navigation*;***

4. Acione o botão **OK**;

5. Quando aparecer a tela perguntando se as bibliotecas de navegação devem ser adicionadas ao projeto (figura abaixo), acione o botão **OK**;



6. Observe que foi gerado o arquivo `mobile_navigation.xml` na pasta `res>nevigation` do projeto;

Adicionando uma tela de cadastro

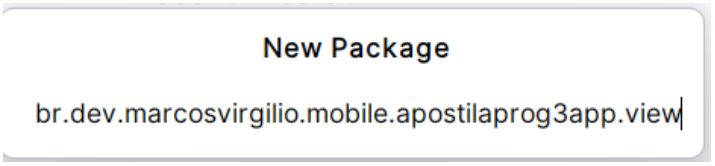
Incluindo um Objeto Fragment (Tela) ao projeto

1. Este projeto segue a arquitetura **MVC**, que divide as classes entre ***Model*** (lógica de negócios e dados), a ***View*** (interface do usuário) e o ***Controller*** (que atua como intermediário entre as outras duas);

2. Selecione a pasta do pacote principal do projeto em **java>br.com…**;

3. Acione o menu de contexto com o mouse e acione a opção de menu ***New>Package***;

4. Como a tela é uma interface com o usuário, na tela ***New Package***, informe o texto ***view*** após o nome do pacote existente e acione a tecla ***Enter*** do teclado para confirmar;



5. Selecione o novo package criado ***view*** e acione a opção de menu ***New>Fragment>Fragment(Blank)***;

6. No campo ***Fragment Name***, informe o nome desejado para a ***Classe***. Nesse exemplo vamos construir uma tela de cadastro de usuários, então o nome sugerido para a ***Classe Fragment*** será **CadUsuarioFragment**;

7. O campo ***Layout Name*** será ajustado automaticamente com base no nome do ***Fragment*** informado;

8. Verifique se a linguagem selecionada é **Java** e acione o botão ***Finish***;

Definindo o layout do formulário de cadastro

1. Abra o arquivo “fragment_cad_usuario.xml” localizado na pasta “res>layout>”;
2. Adicione o código abaixo **entre o início da tag `<FrameLayout>` … e o final da tag `</FrameLayout>`;**

```
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >
        <TextView
```

```
        android:id="@+id/textViewNm"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Nome" />
    <EditText
        android:id="@+id/etNome"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName" />
    <TextView
        android:id="@+id/textViewMail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="e-mail" />
    <EditText
        android:id="@+id/etMail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textEmailAddress" />
    <TextView
        android:id="@+id/textViewSenha"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="e-mail" />
    <EditText
        android:id="@+id/etSenha"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPassword" />
    <TextView
          android:id="@+id/textViewSexo"
     android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="Sexo" />
<RadioGroup
          android:id="@+id/rgSexo"
          android:layout_width="match_parent"
          android:layout_height="match_parent" >
          <TableLayout
          android:layout_width="match_parent"
          android:layout_height="match_parent">
          <TableRow
               android:layout_width="match_parent"
```

```
                android:layout_height="match_parent" >
                <RadioButton
                      android:id="@+id/radioButtonMas"
                      android:layout_width="match_parent"
                      android:layout_height="wrap_content"
                      android:text="Masculino" />
                <RadioButton
                      android:id="@+id/radioButtonFem"
                      android:layout_width="match_parent"
                      android:layout_height="wrap_content"
                      android:text="Feminino" />
                <RadioButton
                      android:id="@+id/radioButtonOutro"
                      android:layout_width="match_parent"
                      android:layout_height="wrap_content"
                      android:text="Outro" />
          </TableRow>
          </TableLayout>
</RadioGroup>
<TextView
          android:id="@+id/textViewPerfil"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="Perfil" />
<Spinner
          android:id="@+id/spPerfis"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:entries="@array/perfis" />
   <TextView
        android:id="@+id/textViewNasc"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Data Nascimento" />
   <CalendarView
        android:id="@+id/cvDataNasc"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
   <CheckBox
        android:id="@+id/cbAceite"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Aceito os termos ...." />
   <Button
        android:id="@+id/btSalvar"
        android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:text="Salvar" />
        <Space
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize" />
    </LinearLayout>
</ScrollView>
```

3. Não é recomendado o uso de strings (textos) fixos diretamente definidos nos componentes de ***layouts*** (XML). O correto é utilizar os ***String Resources***, incluindo tags **`<string></string>`** no arquivo "strings.xml" para cada ID de atributo de componente do layout, como por exemplo o botão salvar abaixo:

    - TAG **`<string>`** no arquivo "strings.xml":
      `<string name="botao_salvar">Salvar</string>`

    - Uso do **`name="botao_salvar"`** no componente ***button*** do layout (xml)
      `android:text="@string/botao_salvar"`

4. Após substituir os **`android:text`** por **`@string/`** salve os arquivos alterados.

Explicação das TAGs de layout utilizadas

- A tag "**`ScrollView`**" inclui uma barra de rolagem na tela, que engloba todas as demais tags dentro dela;
- A tag "**`LinearLayout`**" define um layout dentro da barra de rolagem, que alinha as demais tags uma embaixo da outra automaticamente, nesse caso o layout vai por os componentes na vertical **`android:orientation="vertical"`**;
- As tags "**`TextView`**" são componentes que apresentam um texto fixo na tela, similar ao que outras linguagens denominam "Label";
- As tags "**`EditText`**" adicionam componentes do tipo campos de inserção de dados, similares aos "Inputs" em outras linguagens;
- As tags "**`CalendarView`**" adicionam componentes do tipo Calendário para seleção de data pelo usuário;
- As tags "**`Spinner`**" adicionam componentes do tipo lista de seleção, também conhecidos como **`ComboBox`**;
- As tags "**`RadioGroup`**" adicionam a funcionalidade de seleção única entre os seus componentes "**`RadioButton`**";
- As tags "**`CheckBox`**" adicionam componentes do tipo caixa de marcação;

Spinner - Definindo uma lista fixa de opções de seleção

Caso seja necessário incluir um campo do tipo caixa de seleção na sua tela, o componente no Android é o Spinner, também conhecido como "ComboBox" em outras linguagens, siga os passos abaixo:

1. Incluir uma TAG **<string-array> </string-array> no arquivo strings.xml**, contendo as opções de seleção desejadas, como no exemplo abaixo:

```
<string-array name="perfis">
    <item>Administrativo</item>
    <item>Contabilidade</item>
    <item>Expedição</item>
    <item>Vendas</item>
</string-array>
```

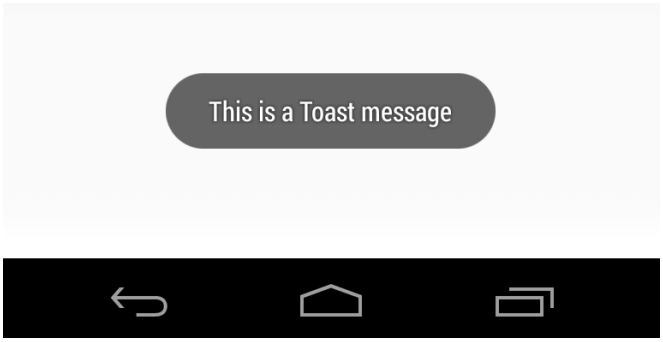
2. Vincular a tag **<string-array>** no componente **<Spinner>** da tela, definindo o valor de **android:entries** com o **name** definido em **<string-array name="perfis">**, como no exemplo abaixo:

```
<Spinner
    android:id="@+id/spPerfis"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/perfis" />
```

3. Salvar as alterações realizadas nos arquivos.

Principais Classes de Janelas de Mensagem

Classe Toast



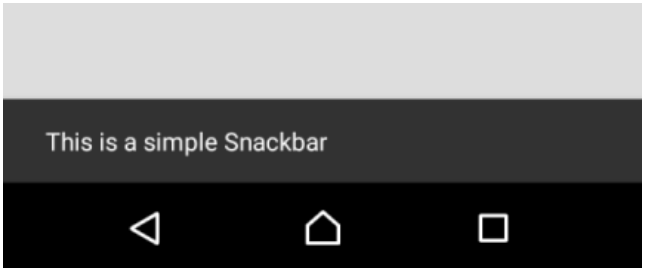
Fonte: https://developer.android.com/ em 12/04/2023

Código de exemplo:

```
Toast msg = Toast.makeText(
   view.getContext(),
   "Mensagem desejada!",
   Toast.LENGTH_LONG);
msg.show();
```

LENGTH_LONG, LENGTH_SHORT = Tempo de permanência da janela aberta.

Classe Snackbar



Fonte: https://developer.android.com/ em 12/04/2023

Código de exemplo:

```
Snackbar.make(
   view,
   "Mensagem desejada!",
   Snackbar.LENGTH_LONG).show();
```

LENGTH_LONG, LENGTH_SHORT = Tempo de permanência da janela aberta.

Inserindo um botão na barra de navegação inferior para abrir a tela de cadastro

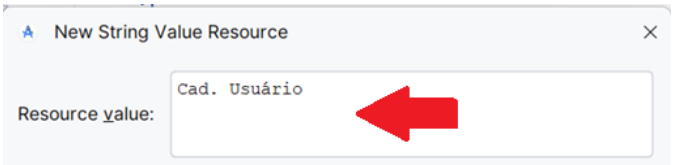
Para abrir uma nova tela, precisamos adicionar um novo botão na barra inferior do aplicativo.

1. Abra o arquivo **bottom_nav_menu.xml** localizado na pasta ***res>menu*** do projeto.
2. Inclua uma tag **<item/>** no arquivo para criar um botão para abrir a nova tela, conforme o exemplo abaixo.

```
<item
    android:id="@+id/navigation_cad_usuario"
    android:icon="@android:drawable/ic_menu_add"
    android:title="@string/title_cad_usuario" />
```

3. Observe que o texto `@string/title_cad_usuario` da tag `android:title` ficou em vermelho, indicando um erro. Isso é porque não criamos um índice no arquivo **strings.xml** que fica em **res>values**;
4. Pare com o cursor do mouse sobre o texto `@string/title_cad_usuario` até aparecer a janela de sugestões de correção de erros, como na figura abaixo:

5. Clique sobre a opção **`Create string value resource …`**;
6. Quando a tela ***New String Value Resource*** abrir, basta informar no campo ***Resource value:*** o texto que você quer que apareça no botão, que nesse exemplo é `Cad. usuario`, e depois acione o botão **OK**;



7. Observe que no arquivo ***strings.xml*** da pasta ***res>values,*** foi criada uma tag **`<string name="title_cad_usuario">Cad. Usuário</string>`**;
8. Futuramente, para alterar o texto do botão, basta alterar o arquivo strings.xml;

Adicionando uma navegação para abrir a tela de cadastro

Após inserir o botão na barra de botões inferior, agora precisamos programar uma navegação para que a tela possa ser chamada dos eventos do aplicativo;

1. Abra o arquivo **mobile_navigation.xml** localizado na pasta ***res>navigation***;
2. No modo de visualização **Design**, acione o botão  para adicionar a nova tela ao recurso de navegação;
3. Selecione a nova tela **fragment_cad_usuario** que foi adicionada ao projeto;
4. Altere o modo de visualização para **Code**, para alterar o código XML;
5. Observe que o arquivo já possui a tag **<fragment/>** que corresponde a tela que que foi adicionada;
6. Agora precisamos ajustar o **android:id** da nova navegação para que fique igual ao **android:id** definido para o botão que vai abrir essa tela, que nesse caso é **"@+id/navigation_cad_usuario"**;
7. Precisamos também ajustar o **android:label** com o string usado no botão, que no caso foi **"@string/title_cad_usuario"**;
8. A tag ficará como no exemplo abaixo:

```
<fragment
        android:id="@+id/navigation_cad_usuario"
        android:name="br.com.instituicao.view.CadUsuarioFragment"
        android:label="@string/title_cad_usuario"
        tools:layout="@layout/fragment_cad_usuario" />
```

9. Na tag **<navigation/>** altere o valor de **app:startDestination** com o mesmo **@id/navigation_cad_usuario**, porque senão o sistema não terá uma tela inicial válida e não irá executar;
10. Abaixo um exemplo da tag **<navigation/>** ajustada:

```
<navigation
xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      android:id="@+id/mobile_navigation"
      app:startDestination="@id/navigation_cad_usuario">
```

11. Não esqueça de salvar o arquivo XML após a alteração;

Registrando a nova navegação na Classe MainActivity

1. Na Classe MainActivity.java, incluir o código de programação listado abaixo para declarar os objetos de navegação do aplicativo:

```
// 1. Inicia o objeto BottomNavigationView
BottomNavigationView navView =
      findViewById(R.id.bottomNavigationView);
// 2. Instancia o objeto NavController
NavController navController = Navigation.findNavController(
      this, R.id.nav_host_fragment_activity_main);
// 3. Instancia do AppBar
AppBarConfiguration appBarConfiguration =
      new AppBarConfiguration.Builder(
      R.id.navigation_cad_usuario).build();
// 4. Conecte a barra de navegação ao NavController
NavigationUI.setupWithNavController(navView, navController);
```

2. Os comandos devem ser incluídos no final do método ***onCreate()*** da Classe e antes do "**}**" final do método;
3. Não esqueça de salvar as alterações realizadas no arquivo.

Alterando a tela de Cadastro para carregar os itens do Spinner a partir de um Web Service

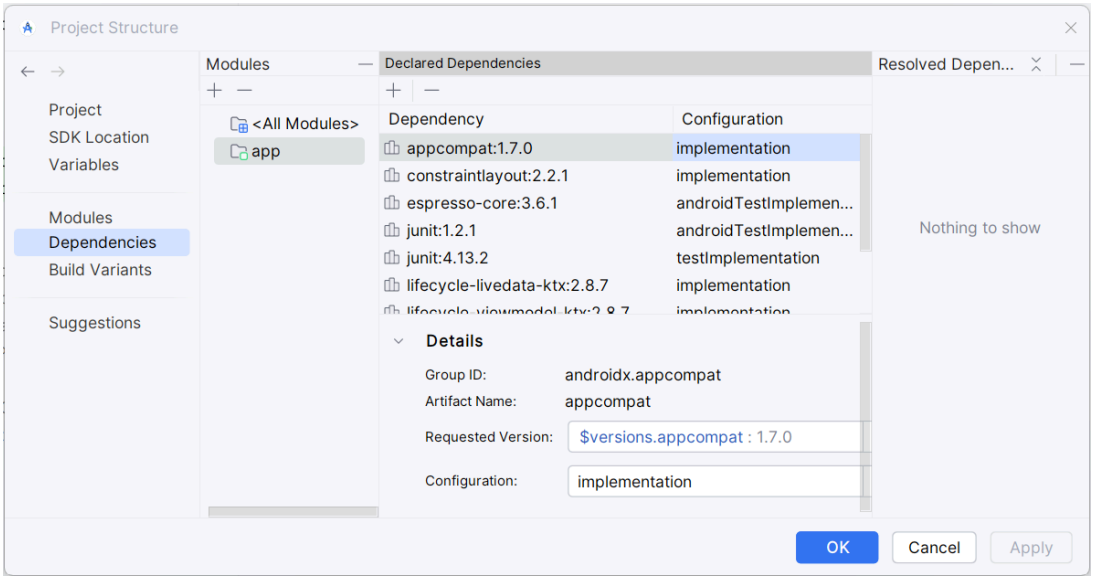
No layout da tela de cadastro de usuários, definido previamente, o componente Spinner foi configurado para apresentar itens fixos, definidos em um array no arquivo de recurso **strings.xml**.

Os próximos passos vão sobrepor os itens fixos por itens que retornam no formato *json* do *web service* **conperfis.php**, utilizando a biblioteca ***Volley***.

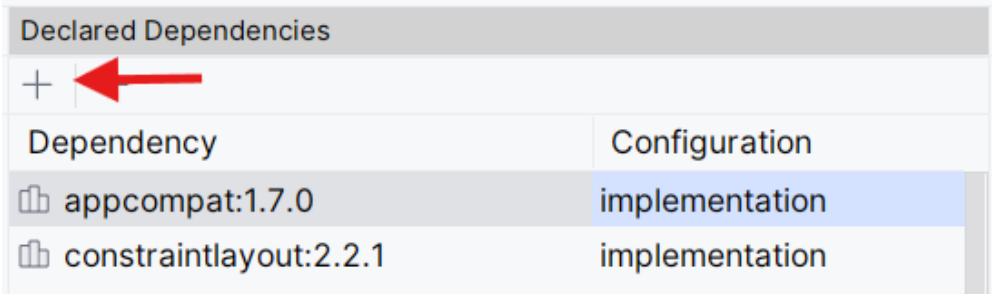
Adicionando a biblioteca Volley ao projeto

1. Abrir o menu ***Project Structure***;

2. Na janela aberta, selecione a opção ***Dependencies*** na primeira coluna da esquerda e selecione ***app*** na segunda coluna (Modules), como na figura abaixo;

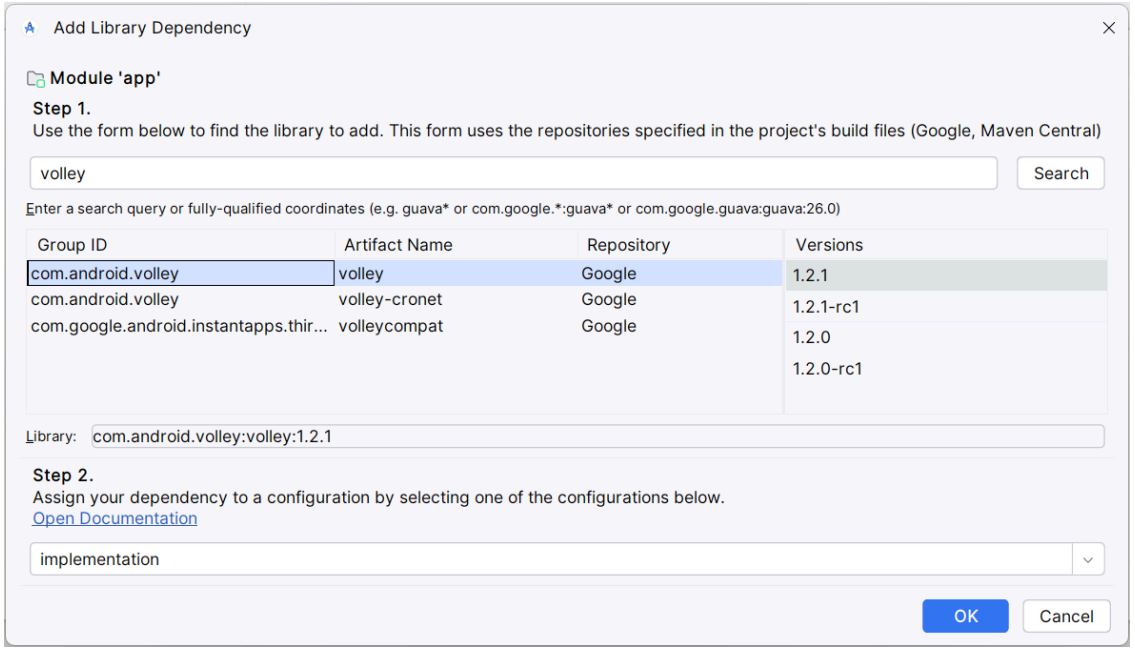


3. Acione o botão + abaixo do título *Declared Dependencies*, indicado com uma seta vermelha na figura abaixo;



4. Selecione a opção 1 ***Library Dependency***;

5. Na janela aberta, digite “volley” no campo de busca e acione o botão ***Search*** localizado do lado direito do campo de digitação, para pesquisar as versões disponíveis da biblioteca Volley, como na figura abaixo;



6. Após a consulta, selecione a versão mais recente da biblioteca Volley da lista. Evite as versões que contenham “rc-N”, porque são versões ***release candidate***, ou seja, ainda estão em testes.

7. Adicionar o botão ***OK*** para instalar a biblioteca selecionada no projeto;

Definindo o objeto da Classe View como atributo da Classe Fragment

Esse passo é super importante, porque o objeto da Classe View é um representação da tela, ou seja, será necessário acessá-lo em todos os métodos da Classe Fragment. Se deixarmos ele como objeto só do método OnCreateView(), ele só estará visível para este método e não para os demais da Classe, como por exemplo os que tratam os cliques dos botões.

1. Ajustar a Classe Fragment, declarando o objeto da classe View como atributo da classe e não apenas um retorno do método OnCreateView().

```
/*Objeto View como atributo de escopo da classe e não mais apenas do escopo do método */
private View view;
```

2. Inicializar o atributo `this.view` com o retorno do método `inflate`, no método onCreateView(), Caso não esteja sendo inicializado em algum outro método da classe ou ainda se não esteja sendo inicializado como um objeto com outro nome;

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup
container,Bundle savedInstanceState) {
   // Ajustando o atributo para receber o retorno no método
   this.view = inflater.inflate(R.layout.fragment_cad_usuario,
        container, false);
   // Ajustando o return para o novo atributo
   return this.view;
}
```

Definindo o Spinner do Layout XML como objeto Java

Para que possamos interagir com os elementos das telas, precisamos programar um vínculo entre a tag XML e um objeto Java, por intermédio do ID da tag XML do componente da tela.

1. Declare o objeto que corresponde ao campo da tela como atributo privado da Classe Fragment. Observe que a TAG do Layout XML tem o mesmo nome da Classe Java do objeto, como no exemplo abaixo:

```
private Spinner spPerfis;
```

2. Vincular o componente do layout XML com o objeto Java no método onCreateView() do Fragment;

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup
container,Bundle savedInstanceState) {
   this.view = inflater.inflate(R.layout.fragment_cad_usuario,
        container, false);
   //spinner
   this.spPerfis = (Spinner) view.findViewById(R.id.spPerfis);
   // return default
   return this.view;
}
```

Implementando a chamada do Web Service

IP para o localhost do computador pelo emulador do Android Studio = 10.0.2.2

1. Incluir os atributos do Volley na classe Fragment;

```
private RequestQueue requestQueue;
```

2. Inicializar os objetos do Volley no método **onCreateView()**;

```
//instanciando a fila de requests - caso o objeto seja o view
this.requestQueue = Volley.newRequestQueue(view.getContext());
//inicializando a fila de requests do SO
this.requestQueue.start();
```

3. Incluir um novo pacote ao projeto (model) abaixo do pacote raiz, uma vez que essa nova classe não tem a finalidade de Interface de usuário (View) e de modelo de negócio da aplicação;
4. Criar uma nova Classe de negócio Java para representar cada Perfil que retornar do *web service*, como no exemplo abaixo;

```
public class Perfil {
    private int idPerfil;
    private String nmPerfil;
    private String dePerfil;
    //método toString() para o adapter do spinner
    @Override
    public String toString() {return this.nmPerfil;}
    public int getIdPerfil() {return idPerfil;}
    public void setIdPerfil(int idPerfil) {
   this.idPerfil = idPerfil;}
    public String getNmPerfil() {return nmPerfil;}
    public void setNmPerfil(String nmPerfil) {
   this.nmPerfil = nmPerfil;}
    public String getDePerfil() {return dePerfil;}
    public void setDePerfil(String dePerfil) {
   this.dePerfil = dePerfil;}
}
```

5. Implementar um método privado para consultar os dados do *web service*;

```
private void consultaPerfis() throws JSONException {
    //requisição para o Rest Server
    JsonArrayRequest jsonArrayReq = null;
    try {
   jsonArrayReq = new JsonArrayRequest(
        Request.Method.POST,
        "http://10.0.2.2:8080/seg/conperfis.php",
        new JSONArray("[{}]"),
        response -> {
           try {
              //se a consulta não veio vazia
              if (response != null) {
                //array list para receber a resposta
                ArrayList<Perfil> lista = new ArrayList<>();
                //preenchendo ArrayList com JSONArray recebido
                for (int pos=0; pos < response.length(); pos++){
                  JSONObject jo = response.getJSONObject(pos);
                  Perfil perfil = new Perfil();
                perfil.setIdPerfil(jo.getInt("idPerfil"));
                perfil.setNmPerfil(jo.getString("nmPerfil"));
                  lista.add(pos, perfil);
                }
                //Criando um adapter para o spinner
                ArrayAdapter<Perfil> adapter = new ArrayAdapter<>(
                      requireContext(),
                     android.R.layout.simple_spinner_item,
                      lista);
              //Definindo o adapter do spinner
                spPerfis.setAdapter(adapter);
              } else {
                Snackbar mensagem = Snackbar.make(view,
                  "A consulta não retornou nenhum registro!",
                  Snackbar.LENGTH_LONG);
                mensagem.show();
              }
           } catch (JSONException e) {
              Snackbar mensagem = Snackbar.make(view,
```

```
                    "Ops!Problema com o arquivo JSON: " + e,
                Snackbar.LENGTH_LONG);
                mensagem.show();
            }
        },
        error -> {
            //mostrar mensagem que veio do servidor
            Snackbar mensagem = Snackbar.make(view,
                "Ops! Houve um problema ao realizar a consulta: " +
                error.toString(), Snackbar.LENGTH_LONG);
            mensagem.show();
         }
      );
    } catch (JSONException e) {
    throw new RuntimeException(e);
    }
    //colocando nova request para fila de execução
    requestQueue.add(jsonArrayReq);
}
```

6. Implementar a chamada do método `consultaPerfis()` no final do método **onCreateView()**, antes do ***return*** do método, como no exemplo abaixo:

```
try {
  this.consultaPerfis();
} catch (JSONException e) {
  throw new RuntimeException(e);
}
```

7. Adicionar a permissão de acesso a internet no app, no arquivo manifest;

```
<uses-permission android:name="android.permission.INTERNET" />
```

8. Adicionar a permissão para o tráfego de texto via internet dentro da tag application, no arquivo manifest;

```
android:usesCleartextTraffic="true"
```

Salvando os dados do formulário de cadastro utilizando web service

Implementando a Classe Usuario

1. Incluir um novo pacote ao projeto (model) abaixo do pacote raiz, uma vez que essa nova classe não tem a finalidade de Interface de usuário (View) e de modelo de negócio da aplicação;
2. Adicionar uma nova classe Java denominada Usuario ao projeto, dentro do novo pacote.
3. Essa nova Classe deve conter os atributos que correspondam aos dados que serão informados na tela pelo usuário, com um atributo adicional que corresponda ao seu identificador único (ID). Exemplo do código Java da Classe Usuario abaixo:

```
public class Usuario {
    //atributos
    private int id;
    private String nome;
    private String email;
    private String senha;
    private boolean aceiteTermos;
    private String dataNascimento;
    private int sexo;
    private int perfil;
    //construtor
    public Usuario() {
        this.id = 0;
        this.nome = "";
        this.email = "";
        this.senha = "";
        this.aceiteTermos = false;
        this.dataNascimento = "1900-01-01";
        this.sexo = 0;
        this.perfil = 0;
    }
    //métodos GET
    public boolean isAceiteTermos() {return aceiteTermos;}
    public String getDataNascimento() {return dataNascimento;}
    public int getSexo() {return sexo;}
    public int getPerfil() {return perfil;}
    public int getId() {return id;}
    public String getNome() {return nome;}
    public String getEmail() {return email;}
    public String getSenha() {return senha;}
```

```
//métodos SET
public void setDataNascimento(String dataNascimento) {
    //Verificando se o String recebido é uma data válida
    SimpleDateFormat formato =
            new SimpleDateFormat("yyyy-MM-dd");
    try {
        Date data = (Date) formato.parse(dataNascimento);
        //se chegar até aqui não deu erro no parser
       this.dataNascimento = dataNascimento;
    } catch (ParseException e) {
        throw new IllegalArgumentException("Data inválida!");
    }
}
public void setId(int id) {
    //Verificando se o ID recebido é válido
    if (id > 0) {
        this.id = id;
    } else {
        throw new IllegalArgumentException(
                    "O ID deve ser maior que zero");
    }
}
public void setEmail(String email) {
    //Verificando se o email recebido é válido
    if (email.contains("@") && email.contains(".")) {
        this.email = email;
    } else {
        throw new IllegalArgumentException(
                    "O email deve conter um @");
    }
}
public void setSenha(String senha) {
    /* Verificando se a senha recebida é válida
    Exluir letras (a-zA-Z), números (0-9) e espaços (\s).
    Sobram os caracteres especiais !@#$%^&*()_+={}[]:;"'<>,.?/|
    */
    String specialCharacters = "[^a-zA-Z0-9\\s]";
    if (senha.length() > 6 && senha.matches(".*" +
     specialCharacters + ".*")) {
        this.senha = senha;
    } else {
        throw new IllegalArgumentException(
          "A senha deve caractere especial e 6 caracteres");
    }
}
```

```
    public void setAceiteTermos(boolean aceiteTermos) {
        //Não precisa verificação sempre será true ou false
        this.aceiteTermos = aceiteTermos;
    }
    public void setSexo(int sexo) {
        //Verificando se o sexo recebido é válido
        if (sexo >= 0 || sexo <= 3) {
            this.sexo = sexo;
        } else {
            throw new IllegalArgumentException("Sexo Inválido");
        }
    }
    public void setPerfil(int perfil) {
        //Verificando se o perfil recebido é válido
        if (perfil >= 0 && perfil <=2) {
            this.perfil = perfil;
        } else {
            throw new IllegalArgumentException("Perfil Inválido!");
        }
    }
    public void setNome(String nome) {
        /*Verificando se o nome recebido é válido
        p{L} = Qualquer caracter Unicode (ã, ç, é, etc.)
        p{N} = Qualquer numero Unicode
        s = espaço em branco
        ^ e $ garantem que o string restrinja os caracteres*/
        String regex = "^[\\p{L}\\p{N}\\s]+$";
        if (nome.length() > 3 && nome.matches(regex)) {
            this.nome = nome;
        } else {
            throw new IllegalArgumentException(
                    "O nome não pode ter caracteres especiais!");
        }
    }
}
```

Adicionando os Construtores e Métodos para conversão do objeto da Classe para JSON e Vice-versa

Como o aplicativo vai utilizar o formato JSON como padrão para o conteúdo das mensagens trocadas com os Web Services, precisamos implementar métodos e construtores na classe que façam a conversão do estado do objeto Java para um objeto JSON e vice-versa.

Abaixo é apresentado um exemplo de arquivo JSON que representa o objeto Java **Usuario** do projeto.

```
{
  "idUsuario": 1,
  "nmUsuario": "João da Silva",
  "deEmail": "joao.silva@example.com",
  "deSenha": "senha123",
  "cdSexo": 1,
  "idPerfil": 1,
  "dtNascimento": "1990-05-08",
  "opTermo": 1
}
```

1. Para que os objetos possam ser convertidos em objetos JSON, precisamos definir alguns métodos e construtores, como no exemplo abaixo:

```
//CONSTRUTOR - Inicializa os atributos para gerar Objeto Json
public Usuario () {
    this.setId(0);
    this.setNome("");
    this.setSenha("");
    this.setEmail("");
    this.setAceiteTermos(false);
    this.setDataNascimento("1970-01-01");
    this.setSexo(0);
    this.setPerfil(0);
}

/*
CONSTRUTOR - inicializa atributos a partir de um objeto JSon
recebido como parâmetro
*/
public Usuario (JSONObject jp) {
    try {
        this.setId(jp.getInt("idUsuario"));
        this.setNome(jp.getString("nmUsuario"));
        this.setSenha(jp.getString("deSenha"));
        this.setEmail(jp.getString("deEmail"));
```

```
            this.setAceiteTermos(jp.getBoolean("opTermo"));
            this.setDataNascimento(jp.getString("dtNascimento"));
            this.setSexo(jp.getInt("cdSexo"));
            this.setPerfil(jp.getInt("idPerfil"));
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    //Metodo que retorna o estado do objeto Java no formato objeto JSon
    public JSONObject toJsonObject() {
        JSONObject json = new JSONObject();
        try {
            json.put("idUsuario", this.id);
            json.put("nmUsuario", this.nome);
            json.put("deSenha", this.senha);
            json.put("deEmail", this.email);
            json.put("opTermo", this.aceiteTermos);
            json.put("dtNascimento", this.dataNascimento);
            json.put("cdSexo", this.sexo);
            json.put("idPerfil", this.perfil);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        return json;
    }
```

Programando o click do botão salvar na tela de cadastro

Vinculando os campos dos Layouts XML com objetos Java

Para que possamos interagir com os componentes dos layouts das telas, precisamos programar um vínculo (**Binding**) entre as tags XML do Layout e um objeto Java, usando para isso os IDs das tags XML dos componentes da tela.

1. O primeiro passo é declarar os objetos que correspondem aos campos da tela como atributos privados da classe **Fragment**.

   Observe que os nomes das TAGs do Layout XML tem os mesmos nomes das Classes Java dos componentes (`EditText,CheckBox, etc.`);

   É sugerido também que os nomes dos objetos Java tenham o mesmo nome dos IDs das TAGs dos componentes visuais do Layout XML;

2. Abaixo o código da declaração dos objetos Java para os componentes do Layout XML:

```
private EditText etNome;
private EditText etMail;
private EditText etSenha;
private CheckBox cbAceite;
private RadioGroup rgSexo;
private CalendarView cvDataNasc;
private Button btSalvar;
```

3. Vincular o componente do layout XML com os objeto Java no método ***onCreateView()*** do **Fragment**;

```
//Binding dos Objetos com os componentes XML
this.etNome = (EditText) view.findViewById(R.id.etNome);
this.etMail = (EditText) view.findViewById(R.id.etMail);
this.etSenha = (EditText) view.findViewById(R.id.etSenha);
//radio group
this.rgSexo = (RadioGroup) view.findViewById(R.id.rgSexo);
//checkBox
this.cbAceite = (CheckBox) view.findViewById(R.id.cbAceite);
//calendarView
this.cvDataNasc = (CalendarView) view.findViewById(R.id.cvDataNasc);
this.btSalvar = (Button) view.findViewById(R.id.btSalvar);
```

Implementando a Classe Fragment como Listener de cliques de componentes

1. Adicionar a implementação da interface **OnClickListener** na Classe **Fragment**;

```
implements View.OnClickListener{
```

2. Implementar o método ***onClick()*** da interface `View.OnClickListener`;

```
@Override
public void onClick(View view) {
    if (view.getId() == R.id.btSalvar) {
        //instanciando objeto de negócio
        Usuario u = new Usuario();
        //populando objeto com dados da tela
        u.setNome(this.etNome.getText().toString());
        u.setEmail(this.etMail.getText().toString());
        u.setSenha(this.etSenha.getText().toString());
        //indice radio group button selecionado
        int radioButtonID = rgSexo.getCheckedRadioButtonId();
        View radioButton = rgSexo.findViewById(radioButtonID);
        u.setSexo(rgSexo.indexOfChild(radioButton));
        //status do checkBox
        u.setAceiteTermos(this.cbAceite.isChecked());
        //objeto do item selecionado do Spinner
        int posPerfil = spPerfis.getSelectedItemPosition();
        Perfil perfil = (Perfil)
             spPerfis.getItemAtPosition(posPerfil);
        u.setPerfil(perfil.getIdPerfil());
        //Pegando a Data do CalendarView
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        String dtSelecao = sdf.format(new
Date(cvDataNasc.getDate()));
        u.setDataNascimento(dtSelecao);
        //mensagem de sucesso
        Context context =  view.getContext();
        CharSequence text = "salvo com sucesso!";
        int duration = Toast.LENGTH_SHORT;
        Toast toast = Toast.makeText(context, text, duration);
        toast.show();
    }
}
```

Definindo a Classe listener para os componentes Java

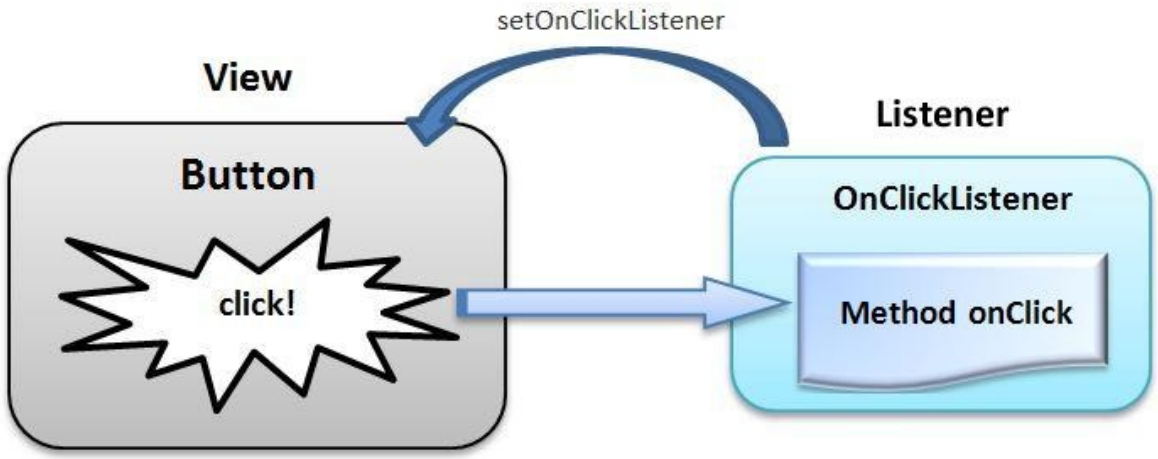
Após Implementar o método ***onClick()*** da interface `View.OnClickListener`, a classe Fragment passa a ter a "habilidade" de tratar os eventos de cliques em componentes da tela. Entretanto, para que os componentes passem a ter seus eventos de cliques tratados pela classe, cada um precisa ser configurado para isso explicitamente, como explicado nos passos abaixo.

1. Definir a própria Classe **`this`** como listener dos componentes que terão o evento onClick() ativados. Geralmente são os botões, mas todos podem ter o click ativado;

Exemplo do código para definição do Listener para os componentes Java:

```
//definindo o listener do botão
this.btSalvar.setOnClickListener(this);
```

Após o comando acima, o botão `btSalvar` passou a ter seus eventos de cliques tratados pela classe Fragment, mas especificamente pelo método OnCLick(), que agora é acionado cada vez que o botão é clicado, como ilustra a figura abaixo.



Fonte: https://www.includehelp.com/android/independent-view-1-setonclicklistener.aspx em 18/11/2025.

Chamada do Web Service de cadastro

IP para acesso ao localhost do emulador do Android Studio = 10.0.2.2

1. Adicionar acesso a internet ao app no manifest (CASO NÃO TENHA AINDA)

```
<uses-permission android:name="android.permission.INTERNET" />
```

2. E adicionar a permissão para o tráfego de texto via internet dentro da tag application (CASO NÃO TENHA AINDA)

```
android:usesCleartextTraffic="true"
```

3. Implementar um método privado para realizar o request para cadastro dos dados do usuário, passando o objeto da classe Usuario como parâmetro;

```
private void cadastrarUsuario(Usuario u) throws JSONException {
   //requisição para o Rest Server
   JsonObjectRequest jsonObjectReq = new JsonObjectRequest(
     Request.Method.POST,
     "http://10.0.2.2:8080/seg/cadusuario.php",
     u.toJsonObject(),
     response -> {
       try {
             //se a consulta não veio vazia
             if (response != null) {
                   Context context = requireContext();
               if (response.getBoolean("success")) {
                   //limpar campos da tela
                   this.etNome.setText("");
                     this.etMail.setText("");
                         this.etSenha.setText("");
                   //primeiro item dos spinners
                   this.spPerfis.setSelection(0);
                }
                //mostrando a mensagem que veio do JSON
               Toast toast = Toast.makeText(
                   view.getContext(),
                   response.getString("message"),
                         Toast.LENGTH_SHORT);
                toast.show();
             } else {
             //mostrar mensagem do response == null
             Snackbar mensagem = Snackbar.make(
                   view,
                   "A consulta não retornou nada!",
```

```
                    Snackbar.LENGTH_LONG);
                mensagem.show();
            }
        } catch (Exception e) {
        //mostrar mensagem da exception
             Snackbar mensagem = Snackbar.make(
            view,
            "Ops!Problema com o arquivo JSON: " + e,
            Snackbar.LENGTH_LONG);
        mensagem.show();
        }
    },
    error -> {
      //mostrar mensagem que veio do servidor
            Snackbar mensagem = Snackbar.make(
            view,
                  "Ops! Houve um problema: " + error.toString(),
            Snackbar.LENGTH_LONG);
            mensagem.show();
    }
 );
 //colocando nova request para fila de execução
 requestQueue.add(jsonObjectReq);
}
```

4. Implementar a chamada do método privado `cadastrarUsuario(Usuario u)` no método ***onClick*();**

   Substituir o código abaixo, que mostra uma mensagem de sucesso:

```
//mensagem de sucesso
Context context = view.getContext();
CharSequence text = "salvo com sucesso!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText
       (context, text, duration);
toast.show();
```

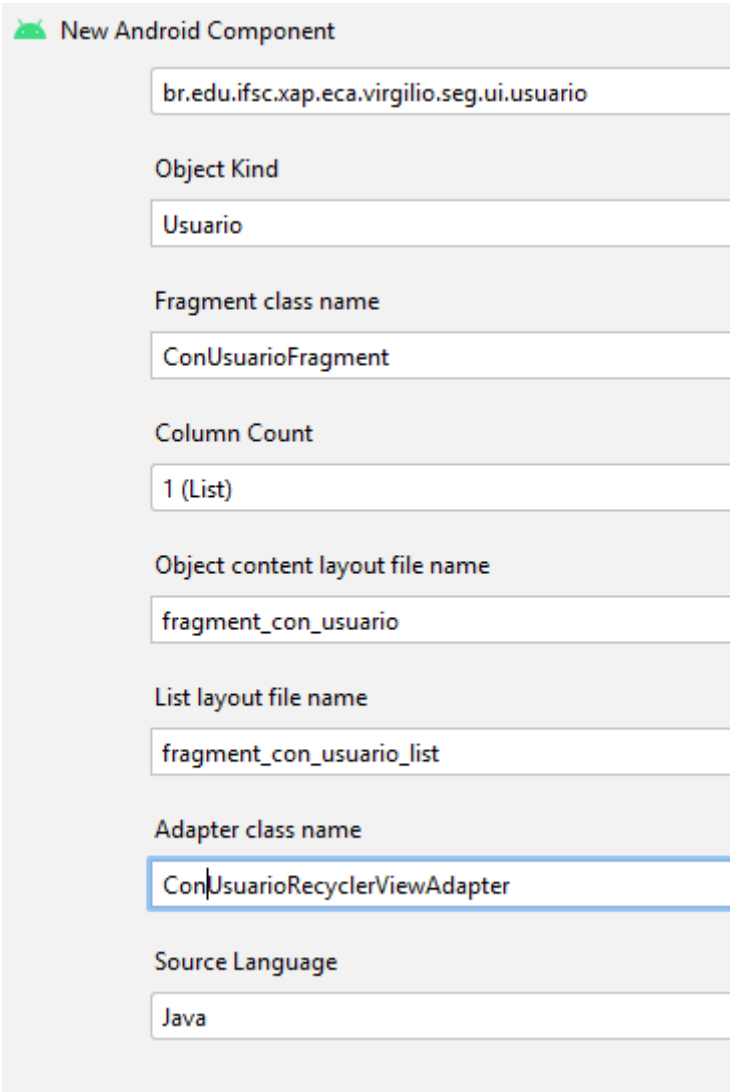
   Pelo código abaixo, com a chamada do método `cadastrarUsuario()`:

```
//chamada do web service de cadastro
try {
   cadastrarUsuario(u);
} catch (JSONException e) {
   throw new RuntimeException(e);
}
```

Adicionando uma tela de consulta ao aplicativo

Incluindo um componente Fragment(List) ao projeto

1. Selecione o pacote **view** do projeto e acione o botão de contexto do mouse;
2. Acione a opção de menu ***New>Fragment>Fragment(List);***
3. Preencha os campos da tela de acordo com a finalidade do Fragment para o aplicativo, mantendo os padrões de projeto do Android;
4. No exemplo abaixo é apresentada uma sugestão de preenchimento, dentro dos padrões, no contexto de a tela servir para a consulta de usuários, uma vez que o segmento inicial da apostila foi o cadastro de usuários;



Adicionando um botão na barra inferior para abertura do Fragment (List)

Para abrir uma nova tela, precisamos adicionar um novo botão na barra inferior do aplicativo.

1. Abra o arquivo “bottom_nav_menu.xml” localizado na pasta “res>menu” do projeto.
2. Observe que o arquivo já possui três tags do tipo <item/>, que correspondem aos botões já existentes da barra inferior do aplicativo;
3. Inclua uma nova tag <item/> no arquivo para o botão que abrirá a nova tela, conforme o exemplo abaixo.
   ATENÇÃO - a posição da tag <item/> em relação as demais tags no arquivo vai definir a posição do botão na barra;

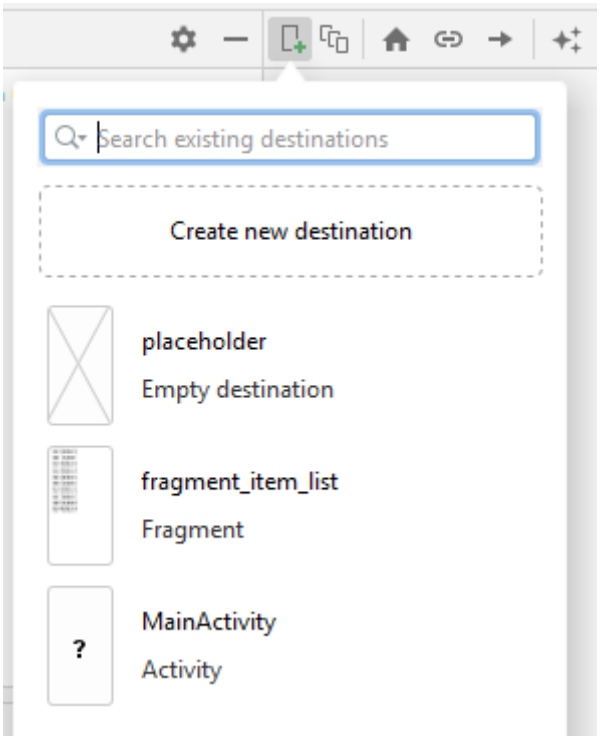
```
<item
    android:id="@+id/navigation_con_usuario"
    android:icon="@android:drawable/ic_menu_sort_by_size"
    android:title="@string/title_con_usuario" />
```

4. Não esqueça de salvar o arquivo após a alteração;

Adicionando uma navegação para abrir o Fragment (List)

Após inserir o botão na barra de botões inferior, agora precisamos programar uma navegação para que a tela possa ser chamada dos eventos do aplicativo;

1. Abra o arquivo “mobile_navigation.xml” localizado na pasta “res>navigation” do projeto.
2. No modo de visualização **Design**, acione o botão para adicionar uma nova tela as navegações já existentes  ;
3. Procure e selecione a nova tela Fragment (List) que foi adicionada ao projeto, que deverá estar entre as opções apresentadas na tela demonstrativa abaixo.



4. Depois de adicionar a tela no grupo de navegações do projeto, altere o modo de visualização para **Code**, para que possamos configurar a navegação pelo código XML;
5. Observe que o arquivo já possui tags do tipo <fragment/> ou <activity/>, que correspondem às telas que já estão com navegações disponíveis no aplicativo;
6. Agora precisamos configurar o **`android:id`** da nova navegação para que fique igual ao **`android:id`** definido para o botão que vai abrir essa tela, que no exemplo desta apostila é **`"@+id/navigation_con_usuario";`**

```
<fragment
    android:id="@+id/navigation_con_usuario"
android:name="br.com.instituicao.view.ConUsuarioFragment"
    android:label="@string/title_con_usuario"
    tools:layout="@layout/fragment_con_usuario_list" />
```

7. Precisamos ajustar também o **`android:label`** criando um **res > values > strings.xml** com o título desejado para a tela;
8. Não esqueça de salvar o arquivo após a alteração;

Registrando a nova navegação na Classe MainActivity

1. Observe que no método onCreate() da Classe, no momento de instanciar o objeto **`"appBarConfiguration",`** são passados como parâmetros os IDs das navegações;
2. Precisamos registrar a nossa nova navegação junto com as demais, já existentes, para que a mesma funcione corretamente, como no texto em destaque no exemplo abaixo:

```
AppBarConfiguration appBarConfiguration = new
      AppBarConfiguration.Builder(
        R.id.navigation_con_usuario,
           R.id.navigation_cad_usuario).build();
```

3. Salvar as alterações realizadas nos arquivos.

Refatorando a Classe …RecyclerViewAdapter

Para definir os itens que aparecem na lista de um Fragment (List), precisamos substituir os objetos da classe "`PlaceHolder`" que foi gerada automaticamente, pela nossa classe de negócio, que corresponde aos objetos/ dados que queremos apresentar.

1. Apague a classe **PlaceHolder**, acionando o menu de contexto do mouse sobre ela e acionando a opção Delete;
2. Na classe "`...RecyclerViewAdapter`", substitua as ocorrências da classe **PlaceHolder**, que foi apagada do projeto nos passos anteriores, pela classe **Item**. Cuidado para não esquecer de substituir todas as ocorrências, inclusive na parte de comentários no início do arquivo;
3. No método "`onBindViewHolder`", ajuste o código para usar os métodos definidos na Classe **Usuario**;
4. **ATENÇÃO! Quando o método não retornar String, é preciso realizar a conversão utilizando o método valueOf() da Classe String, como no exemplo abaixo:**

   ```
   String.valueOf(mValues.get(position).getId());
   ```

5. No código abaixo, estão em destaque amarelo, os pontos de ajuste da Classe e das chamadas dos métodos que precisam ser ajustados:

```
    /**
    * {@link RecyclerView.Adapter} that can display a {@link Usuario}.
    * TODO: Replace the implementation with code for your data type.
    */
    public class ConUsuarioRecyclerViewAdapter extends
RecyclerView.Adapter<ListaUsuarioRecyclerViewAdapter.ViewHolder> {

        private final List<Usuario> mValues;

        public ListRecyclerViewAdapter(List<Usuario> items) {
            mValues = items;
        }

        @Override
```

```
    public ViewHolder onCreateViewHolder(ViewGroup parent,
    int viewType) {
    return new ViewHolder(FragmentUsuarioItemBinding.inflate
    (LayoutInflater.from(parent.getContext()),
    parent, false));

    }

    @Override
    public void onBindViewHolder(final ViewHolder holder, int
    position) {
       holder.mItem = mValues.get(position);
       holder.mIdView.setText(
          String.valueOf(mValues.get(position).getId()));
       holder.mContentView.setText(
          mValues.get(position).getNome());
    }

    @Override
    public int getItemCount() {
        return mValues.size();
    }

    public class ViewHolder extends RecyclerView.ViewHolder {
        public final TextView mIdView;
        public final TextView mContentView;
        public Usuario mItem;

        public ViewHolder(FragmentUsuarioItemBinding binding) {
            super(binding.getRoot());
            mIdView = binding.itemNumber;
            mContentView = binding.content;
        }

        @Override
        public String toString() {
            return super.toString()+
              "'" + mContentView.getText() + "'";
        }
    }
}
```

6. Não esqueça de salvar as alterações em todas as classes modificadas;:

Chamada para o Web Service de Consulta de Usuários

IP para acesso ao localhost do emulador do Android Studio = 10.0.2.2

1. Incluir a implementação das interfaces necessárias na classe Fragment ou Activity

```
public class ConSensorFragment extends Fragment
      implements Response.ErrorListener, Response.Listener {
```

2. Implementar os métodos das interfaces `ErrorListener` e `Listener`;

```
@Override
public void onErrorResponse(VolleyError error) { }
@Override
public void onResponse(Object response) { }
```

3. Incluir os seguintes atributos na Classe Fragment de consulta;

```
//Lista que vai armazenar os objetos que retornam do Web Service
private ArrayList<Usuario> usuarios;
//Fila de requests da biblioteca Volley
private RequestQueue requestQueue;
//Objeto da biblioteca Volley que faz o request para o Web Service
private JsonArrayRequest jsonArrayReq;
//Objeto view que representa a tela utilizado em diversos metodos
private View view;
```

4. Substituir o objeto local `View view` pelo atributo da Classe **`this.view`** no método onCreateView(), como no exemplo abaixo:

```
@Override
public View onCreateView(LayoutInflater inflater,
ViewGroup container,Bundle savedInstanceState) {
   this.view = inflater.inflate(R.layout.fragment_con_usuario_list,
     container, false);
   …
   return this.view;
}
```

5. Inicializar os objetos da biblioteca Volley no método *onCreateView*();

```
//instanciando a fila de requests - caso o objeto seja o view
this.requestQueue = Volley.newRequestQueue(view.getContext());
//inicializando a fila de requests do SO
this.requestQueue.start();
```

6. Implementar a chamada da consulta do WebService REST Server no método que corresponde ao evento desejado;

```
//requisição para o Rest Server tipo POST
jsonArrayReq = new JsonArrayRequest(
        Request.Method.POST,
        "http://10.0.2.2:8080/seg/conusuario.php",
        new JSONArray("[{}]"),
        this, this);
//Inclui a request na fila
requestQueue.add(jsonArrayReq);
return this.view;
```

7. Implementar o método *onResponse*();

```
@Override
public void onResponse(Object response) {
   try {
       //array Json para receber a resposta do webservice
       JSONArray jsonArray = null;
       jsonArray = new JSONArray(response.toString());
       //se a consulta não veio vazia passar para array list
       if (jsonArray != null) {
           //objeto java
           Usuario usuario = null;
           //array list para receber a resposta
           this.usuarios = new ArrayList<Usuario>();
           //preenchendo ArrayList com JSONArray recebido
           for (int pos=0,pos<jsonArray.length();pos++) {
               JSONObject jo = jsonArray.getJSONObject(pos);
               usuario = new Usuario(jo);
               this.usuarios.add(usuario);
           }
           /*
           O código abaixo já estava no método onCreateView().
           Mas foi movido para cá, porque só pode ser
           executado se o ArrayList não estiver vazio.
           */
           if (view instanceof RecyclerView) {
               Context context = view.getContext();
               RecyclerView recyclerView =
                   (RecyclerView) view;
               if (mColumnCount <= 1) {
                   recyclerView.setLayoutManager(
                   new LinearLayoutManager(context));
```

```
                } else {
                    recyclerView.setLayoutManager(
                   new GridLayoutManager(context,
                    mColumnCount));
                }
                recyclerView.setAdapter(
              new SensorRecyclerViewAdapter(this.usuarios));
            }
        }else {
            Snackbar mensagem = Snackbar.make(view,
                    "A consulta não retornou nenhum registro!",
                    Snackbar.LENGTH_LONG);
            mensagem.show();
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

8. Implementar o método onErrorResponse()

```
@Override
public void onErrorResponse(VolleyError error) {
    //mostrar mensagem que veio do servidor
    Snackbar mensagem = Snackbar.make(view,
            "Ops! Houve um problema ao realizar a consulta: " +
          error.toString(), Snackbar.LENGTH_LONG);
    mensagem.show();
}
```

9. Adicionar a permissão de acesso a internet no aplicativo, alterando o arquivo manifest do projeto (CASO NÃO TENHA AINDA);

```
<uses-permission android:name="android.permission.INTERNET" />
```

10. Ainda no arquivo manifest, adicionar a permissão de transmissão de texto dentro da tag `<application/>` (CASO NÃO TENHA AINDA)

```
android:usesCleartextTraffic="true"
```

Adicionando click nos itens da consulta

Programando o evento onClick() na Classe ViewHolder

Agora que os itens do Fragment (List) foram definidos, precisamos identificar qual item foi clicado pelo usuário, para que possamos implementar o comportamento específico do aplicativo.

Para isso usaremos a **Inner Class** (Classe declarada dentro de outra Classe) **ViewHolder**, que foi declarada dentro da classe ...**RecyclerViewAdapter**.

1. Implementar a interface `View.OnClickListener` na Classe `ViewHolder`;

```
public class ViewHolder extends RecyclerView.ViewHolder
      implements View.OnClickListener {
```

2. Incluir o método `onClick()` da interface `View.OnClickListener` na Classe `ViewHolder`, como exemplificado no código abaixo;

```
@Override
public void onClick(View view) {
    //CLICK - pegar posicao que foi clicada
    int adapterposition = this.getLayoutPosition();
    //mostrar posição clicada
    Context context =  view.getContext();
    CharSequence text = "Posicao = " + adapterposition;
    int duration = Toast.LENGTH_SHORT;
    Toast toast = Toast.makeText(context, text, duration);
    toast.show();
}
```

3. Adicionar a ***annotation*** `@NonNull` antes do parâmetro `FragmentListBinding binding` no construtor da Classe **ViewHolder;**

```
public ViewHolder(@NonNull FragmentListBinding binding) {
```

4. Definir o listener para o componente binding, no final do construtor da Classe **ViewHolder;**

```
binding.getRoot().setOnClickListener(this);
```

Acionando uma tela de edição ao aplicativo

Para apresentar os dados do item clicado (selecionado) em uma tela de edição, são sugeridos os seguintes passos:

1. Construção da tela em si, ou seja, incluir um novo componente Fragment no projeto e configurar todo o layout da tela com os componentes necessários;
2. Incluir uma navegação específica para carregar a nova tela de edição, similar ao que foi feito para incluir uma navegação para os botões da Barra de Navegação Inferior, mas sem a inclusão dos botões;
3. Alterar o método `onClick()` da Classe `ViewHolder` para salvar o ID do item que foi clicado (selecionado) na consulta, para que seus dados possam ser recuperados e apresentados na nova tela de edição;
   Para passar o ID do item para a tela de edição, podem ser usadas diversas estratégias de passagem de dados entre Fragments, como ***Safe Args*** e até mesmo o uso de um objeto ***Singleton***;
4. Acionar a navegação para abertura da tela de edição, como no código de exemplo abaixo:

```
Navigation.findNavController(view).navigate(R.id.navigation_edt_usuario)
;
```

5. Construir um **Web Service** para realizar uma consulta no banco de dados do sistema, recebendo um JSon contendo o ID e retornando um JSon com os todos os dados "alteráveis" correspondentes;
6. Implementar no método `onCreate()` ou `onCreateView()` da tela de edição, a chamada do Web Service de consulta por ID e o carregamento dos dados para alteração nos componentes da tela;
7. Construir um **Web Service** para realizar o ***UPDATE*** dos dados no banco de dados, chamado no evento ***onClick()*** do botão salvar da tela de edição;

Web Services em linguagem PHP

Os fontes dos Web Services para a realização dos testes do aplicativo implementado nesta apostila, podem ser obtidos no seguinte repositório: **https://github.com/marcosvirgilio/ApostilaProg3Api**

Arquivos JSON para teste dos webservices

Para testar os Web Services, podem ser utilizados aplicativos para a geração de ***Requests***, como o Postman ou Insomnia.

O único web service que precisa receber um arquivo json como parâmetro é o cadusuario.php, que pode ser testado com o arquivo JSON listado abaixo:

```
{ "nmUsuario": "João da Silva",
  "deEmail": "joao.silva@example.com",
  "deSenha": "senha123",
  "cdSexo": 1,
  "idPerfil": 1,
  "dtNascimento": "1990-05-08",
  "opTermo": 1}
```

Script SQL para a criação do Banco de Dados em um SGBD MySQL

```
CREATE TABLE Perfil (
    idPerfil INT AUTO_INCREMENT PRIMARY KEY,
    nmPerfil VARCHAR(250) NOT NULL,
    dePerfil VARCHAR(250) NULL
);

CREATE TABLE Usuario (
    idUsuario INT AUTO_INCREMENT PRIMARY KEY,
    nmUsuario VARCHAR(100) NOT NULL,
    deEmail VARCHAR(100) NOT NULL UNIQUE,
    deSenha VARCHAR(255) NOT NULL,
    cdSexo TINYINT NOT NULL,
    idPerfil INT NOT NULL,
    dtNascimento DATE NOT NULL,
    opTermo TINYINT(1) NOT NULL DEFAULT 0,
    dtCadastro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_usuario_perfil
        FOREIGN KEY (idPerfil)
        REFERENCES Perfil(idPerfil)
);
```