



Cursos Livres

FIC - Formação Inicial e Continuada

Desenvolvedor de Aplicações para Android

Autor(a): Helena Strada Franco de Souza

Revisão voluntária: Gustavo Donegá

Sumário

Conhecendo o Projeto.....	5
Ferramentas para o Desenvolvimento	6
SDK.....	6
AVD.....	7
Criando o Projeto	8
Etapas do Desenvolvimento.....	12
Primeira Etapa	17
Construindo a tela de Login	17
Organizando os estilos da nossa aplicação	18
colors.xml.....	19
dimen.xml	22
strings.xml.....	26
Realizando nossa ação de login.....	29
Arquivos.....	40
Resumo	40
Segunda Etapa	42
Criando a nossa tela principal	42
Redirecionando o usuário para uma nova tela.....	44
Mostrando a nossa primeira lista	46
Criando o nosso modelo de dados do Aluno.....	50
Personalizando nossa ListView.....	55
Criando o layout personalizado.....	56
Populando o adapter com os dados do aluno.....	58
Arquivos.....	63
Resumo	64
Terceira Etapa.....	65
Criando a activity de formulário	65
Alterando o ícone do botão	66
Criando uma ação para o floating button.....	69
Abrindo o calendário e a câmera.....	74
Solicitando permissão de acesso da câmera	76

Salvando os dados na lista.....	78
Retornando para a Home	79
Adicionando uma mensagem de sucesso para o usuário	82
Arquivos.....	82
Resumo	82
Quarta etapa.....	83
Selecionando um item da lista	83
Levando o ID do aluno para a próxima tela	84
Populando o formulário com os dados do aluno	87
Ajustando o método salvar	89
Arquivos.....	91
Resumo	91
Quinta Etapa	92
Criando o menu personalizado	92
Habilitando o menu personalizado	93
Abrindo um menu de decisão	96
Arquivos.....	99
Resumo	99
Sexta Etapa.....	100
Adicionando um novo item no menu.....	100
Trabalhando com mapas.....	101
Criando uma nova chave.....	103
Colocando a chave no projeto	104
Testando a primeira parte do mapa.....	107
Colocando o endereço do aluno no mapa.....	107
Transformando endereço em latitude e longitude	108
Adicionando um zoom no mapa	111
Sétima Etapa	113
Conhecendo o armazenamento interno da aplicação	113
Introdução ao SQLite	113
SQLite no android	113
Armazenamento no Android.....	113
Conhecendo um pouco mais sobre banco de dados.....	114

SQLite na Prática	115
Esquema e Contrato.....	115
Criando nosso banco de dados.....	115
Alterando nossa estrutura	116
Criando nosso helper	117
Definindo os métodos.....	118
Instanciando.....	120
Conteúdo Extra	122
Evolução das listas.....	122
Passo-a-passo	122
Layout personalizado	124
Adapter e ViewHolder	124
Adapter	125
ViewHolder.....	127
Selecionando um item da lista	128
Introdução WebServices	131
Vantagens.....	131
REST	131
Recursos.....	132
Representação.....	132
Mensagens	132
Padrão	132
Retrofit.....	133
Referências	139

Conhecendo o Projeto

O objetivo do projeto será realizar um cadastro de alunos. Sendo que o cadastro e a listagem só poderão ser realizados por usuários que tiverem feito o login com sucesso.

No projeto serão desenvolvidos: splashscreen (tela de abertura), listagem, localização da casa do aluno, trabalhar com datas no Android, diálogos de decisão, adapters, intent, activity, sharedpreferences, SQLite e alguns outros recursos do Android.

Essa apostila terá como foco o desenvolvimento da aplicação, com alguns detalhes sobre os recursos que estaremos utilizando e algumas dicas ou informações adicionais.

Todos os tópicos que veremos a seguir, terão apostilas individuais, caso tenha dúvida em um assunto específico, poderá realizar uma consulta individual.

Ferramentas para o Desenvolvimento

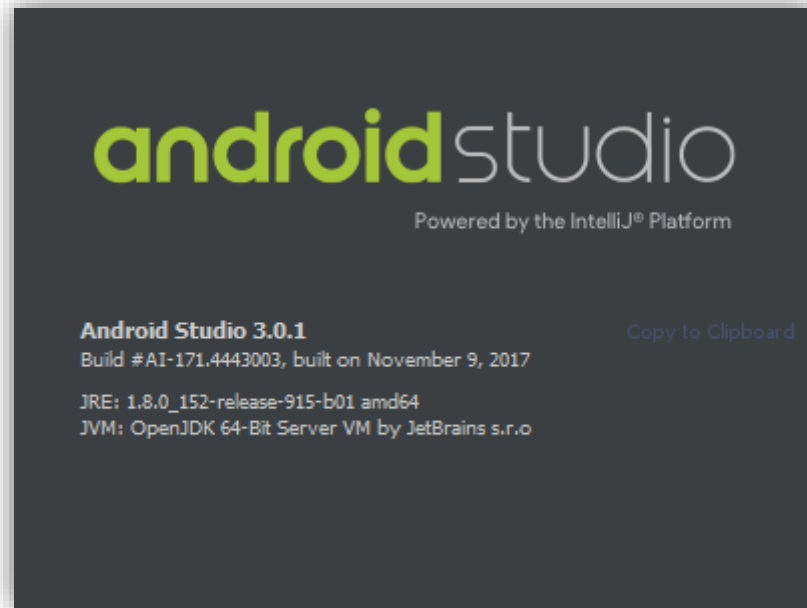


Figura 1 - Versão do Android Studio

Android Studio 3.0.1

Build #AI-171.4443003, built on November 9, 2017

JRE: 1.8.0_152-release-915-b01 amd64

JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o

Windows 10 10.0

minSdkVersion 16

targetSdkVersion 26

SDK

O Android SDK (Kit de Desenvolvimento de Software para Android) é um pacote com ferramentas para o desenvolvimento de aplicativos para android.

A SDK é utilizada pelo Android Studio (um programa que facilita a utilização das ferramentas).

Sendo assim, a SDK é composta pelo conjunto de APIs, ferramentas de desenvolvimento, emuladores de dispositivos, documentação e código fonte de aplicações exemplo.

AVD

O AVD Manager é uma interface gráfica que você criar e gerenciar dispositivos virtuais Android.



Utilize este ícone para abrir o gerenciamento.

Criando o Projeto

Vamos abrir o Android Visual Studio e criar um novo projeto chamado '*android-fic-escolas-dev*'.

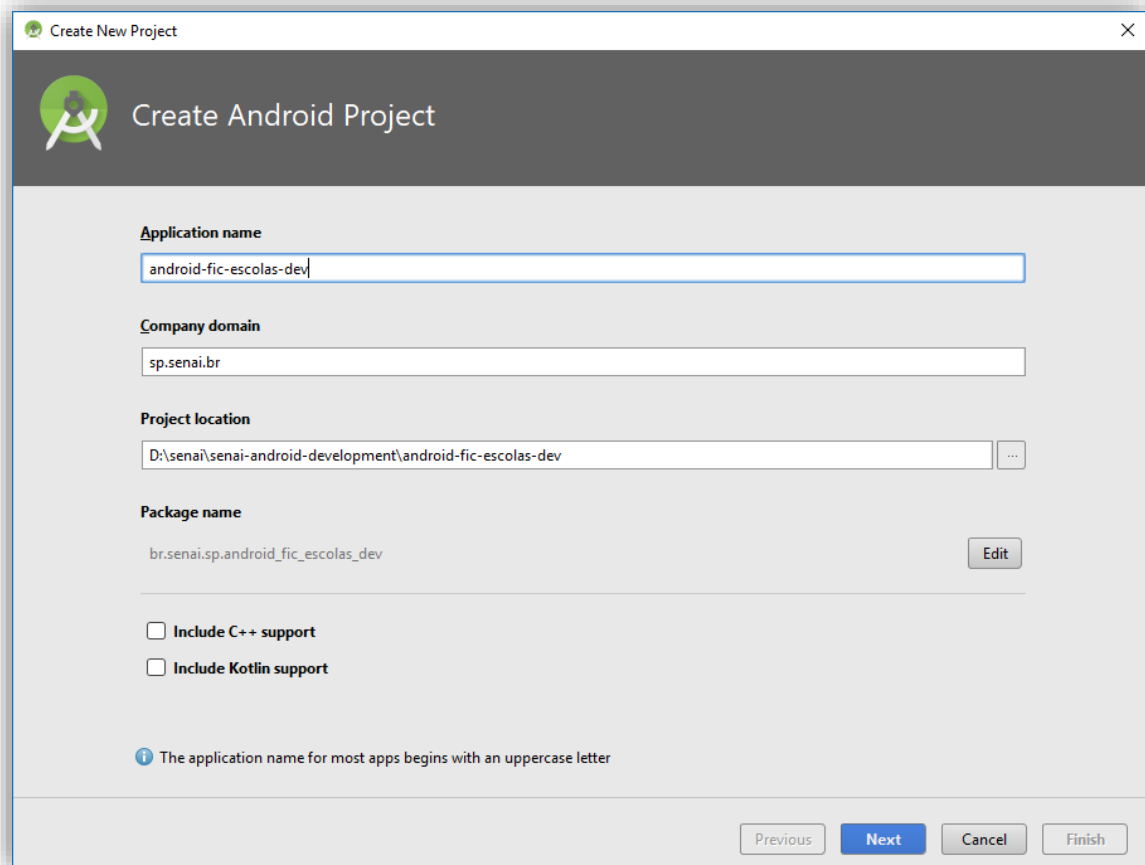


Figura 2 - Criando um novo projeto

Clicar em 'Next'.

Escolheremos a versão da API que será a versão mínima requerida para que o nosso aplicativo funcione. Vale lembrar que, ao começarmos a desenvolver uma nova aplicação ainda sem público, queremos que o máximo de usuários possam utilizá-lo.

Para termos essa média, o Android disponibiliza uma porcentagem de quantos dispositivos possuem a versão mínima para que consiga executar o nosso aplicativo.

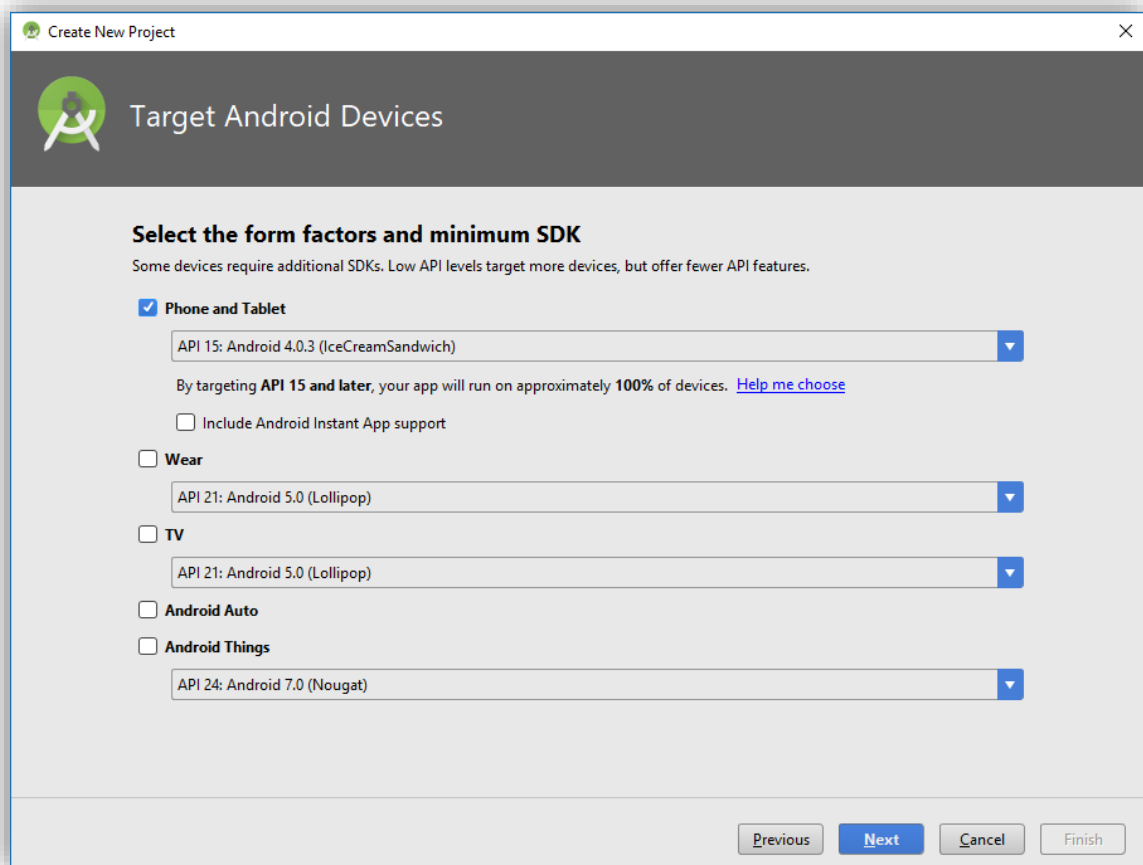


Figura 3 - Escolhendo o alvo da versão mínima para desenvolvimento

Clicar em 'Next'.

Nesta tela, iremos informar para qual layout inicial desejamos utilizar em nossa primeira tela. Lembrando que, nós podemos alterar essa informação posteriormente.

Vamos utilizar a opção que não tem nenhum recurso e/ou informação já pré-definida.

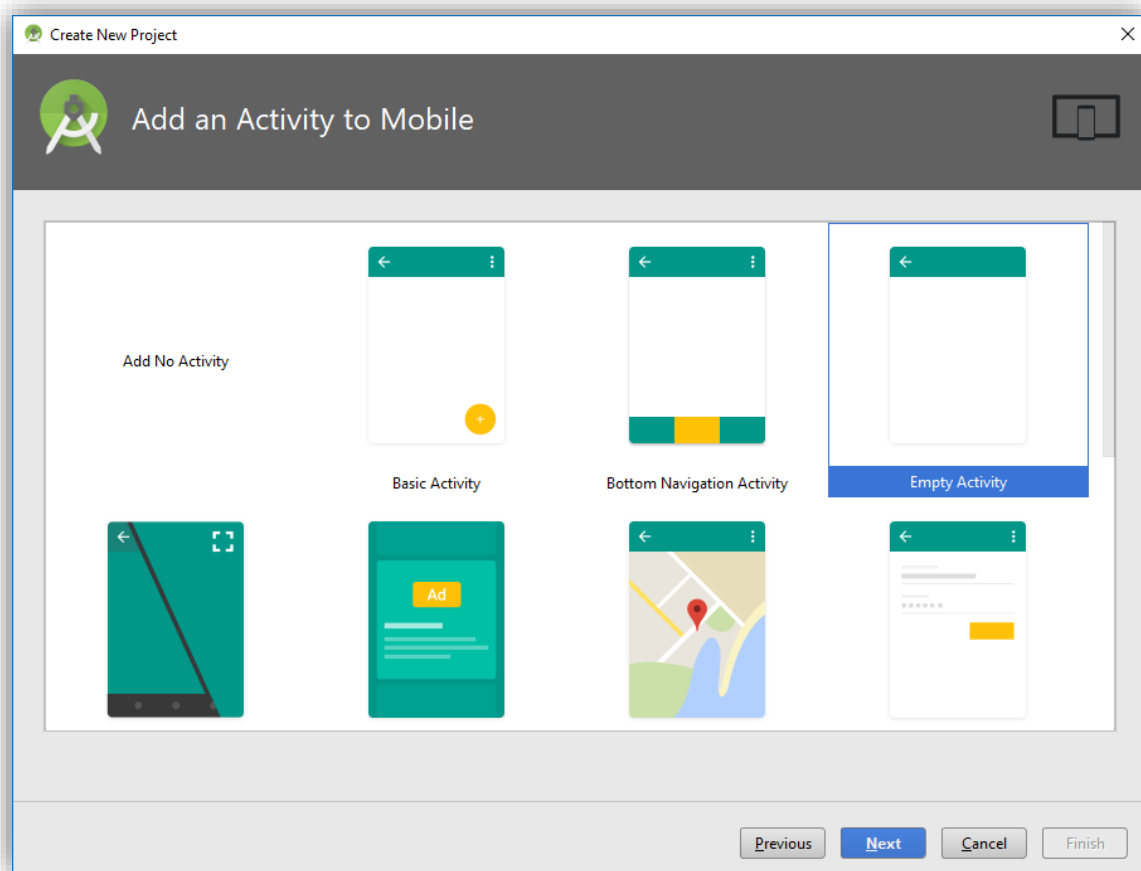


Figura 4 - Escolhendo o estilo da nossa activity principal

Clicar em 'Next'.

Na próxima tela, iremos definir o nome da nossa activity principal, a activity inicial que iremos trabalhar.

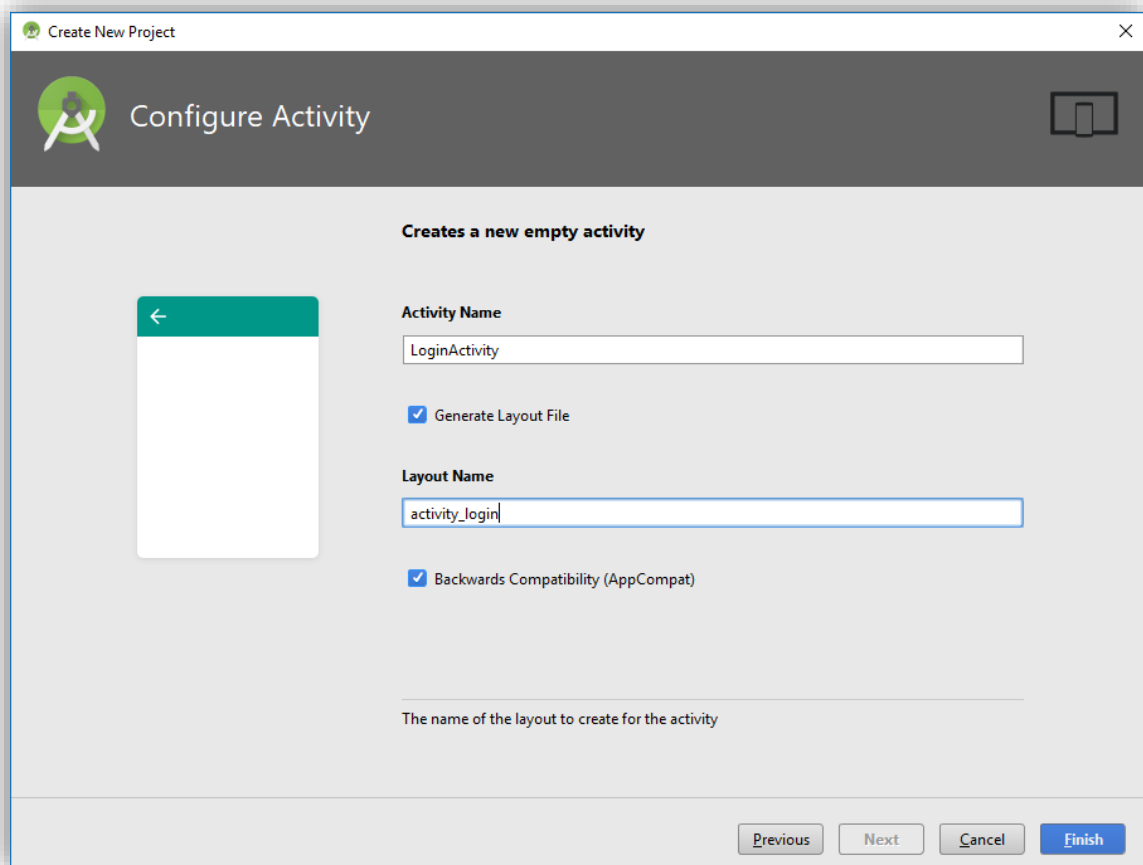
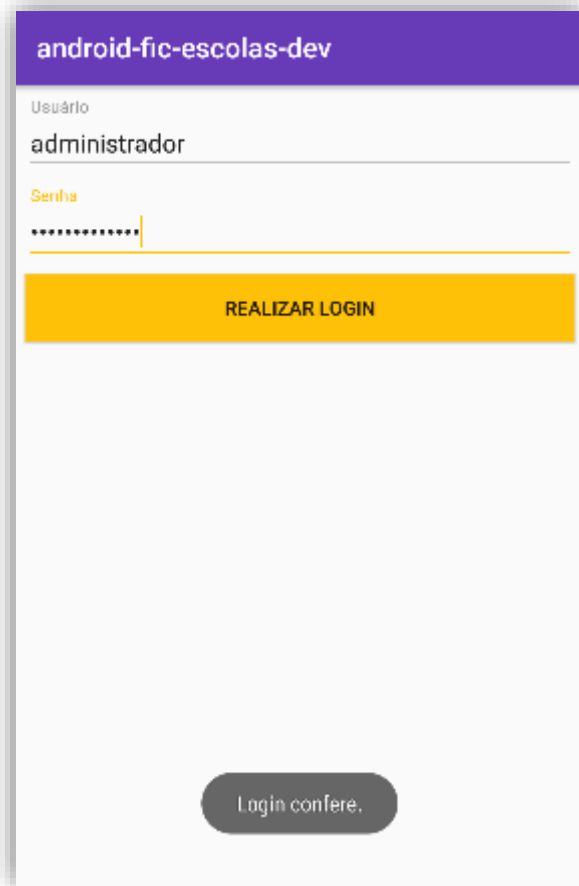


Figura 5 - Escolhendo o nome da nossa primeira activity

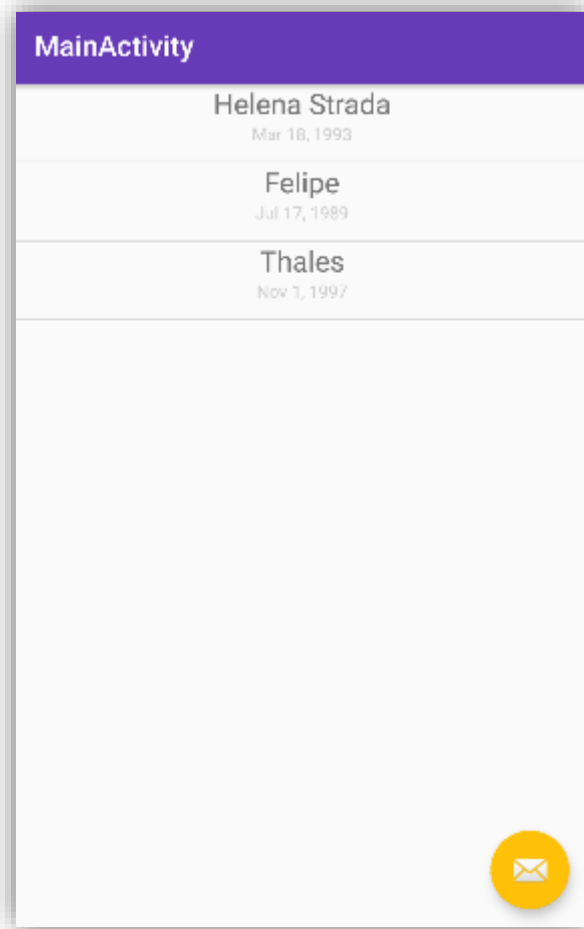
Clicar em 'Finish'.

Etapas do Desenvolvimento

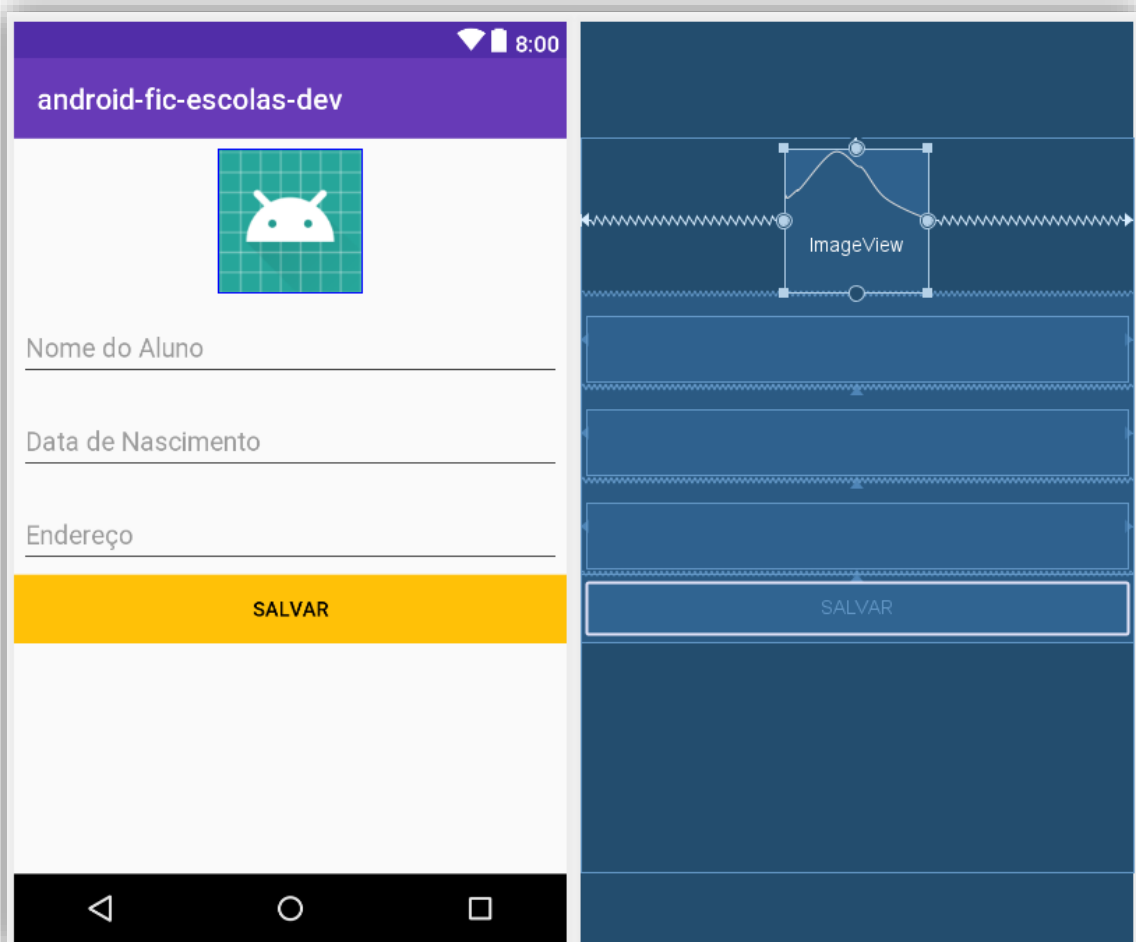
Vamos desenvolver o nosso aplicativo em etapas. A primeira etapa será desenvolver a nossa tela de login e suas funcionalidades. Além disso, nessa etapa, vamos aprender a alterar as cores dos nossos aplicativos, entre outras funções.



Na segunda etapa, iremos criar a home do nosso usuário logado para mostrar a listagem de alunos (personalizada) cadastrados no sistema.

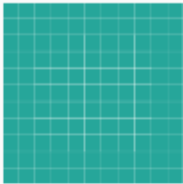


Na terceira etapa, faremos a criação de uma tela para realizar o cadastro de um novo aluno com os seguintes campos: imagem, nome, data de nascimento e endereço.



Na quarta etapa, iremos alterar o registro de um aluno.

Editar Aluno



Nome do Aluno

Helena Strada

Data de Nascimento

Mar 18, 1993

Endereço

Av. Paulista, 1578 - Bela Vista, São Paulo - SP

SALVAR

Na quinta etapa iremos aprender a como incluir um menu personalizado em nossa lista e a como remover um item de nossa lista.

MainActivity

Helena Strada

Mar 18, 1993

Felipe

Jul 17, 1989

Adriana Sales

Jan 1, 1997

Editar

Deletar

Na sexta etapa, iremos mostrar o endereço do aluno em um mapa



Na sétima etapa, iremos alterar o aplicativo para que os nossos dados sejam salvos no SQLite.

Na etapa final, faremos uma SplashScreen.

Primeira Etapa

Construindo a tela de Login

Vamos acessar o xml da activity de login que criamos e criar dois campos. Um campo para o usuário digitar o e-mail e outro campo para o usuário digitar a senha. Além disso, iremos adicionar um botão para o usuário realize o login.

activity_login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
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:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="br.senai.sp.android_fic_escolas_dev.LoginActivity">

    <android.support.design.widget.TextInputLayout
        android:id="@+id/tilLoginEmail"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_margin="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <android.support.design.widget.TextInputEditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Usuário" />

    </android.support.design.widget.TextInputLayout>

    <android.support.design.widget.TextInputLayout
        android:id="@+id/tilLoginSenha"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_margin="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/tilLoginEmail">

        <android.support.design.widget.TextInputEditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Senha" />

    </android.support.design.widget.TextInputLayout>

    <Button
        android:id="@+id/btLoginUsuario"
        android:layout_width="match_parent"
```

```
android:layout_height="wrap_content"
android:layout_margin="8dp"
android:text="Realizar Login"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/tilLoginSenha" />
```

```
</android.support.constraint.ConstraintLayout>
```

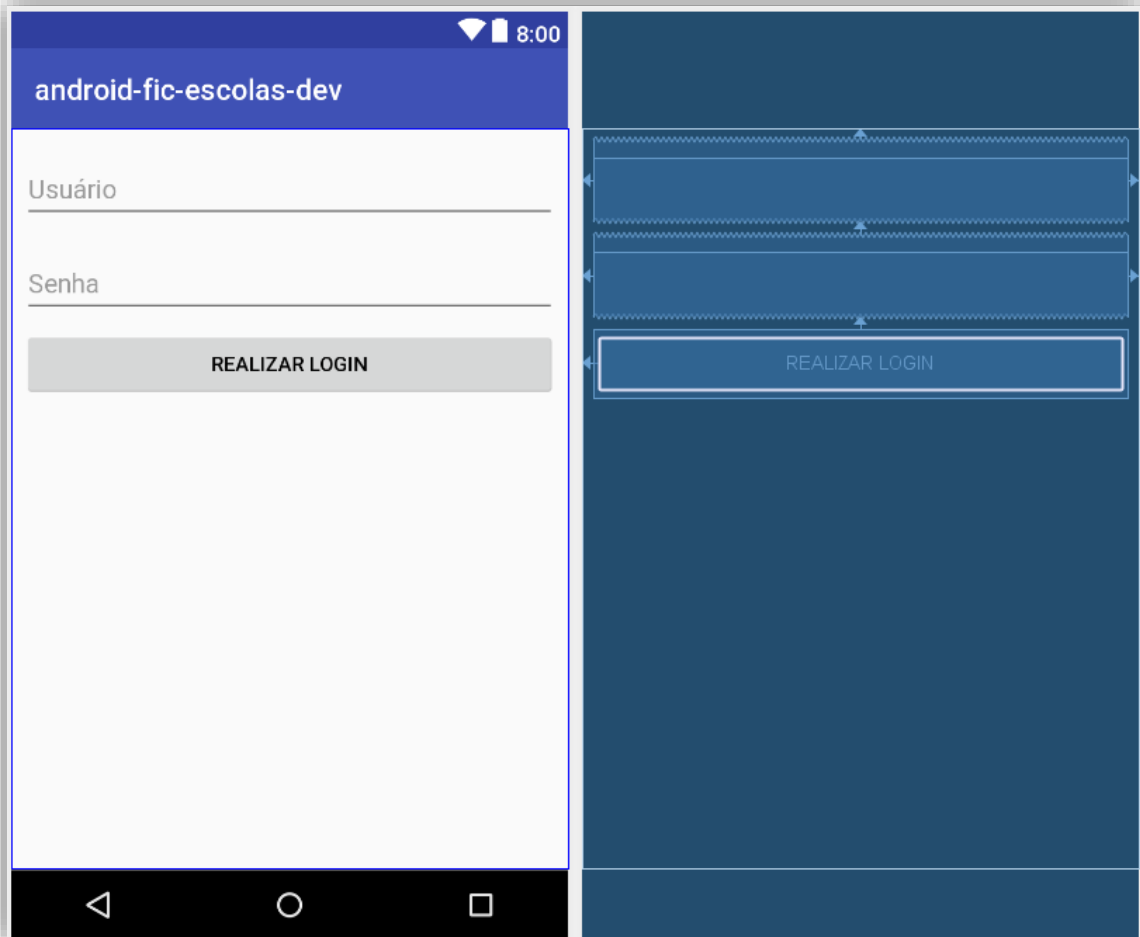


Figura 6 - Tela de Login

Ainda não iremos realizar nenhuma ação. Vamos, primeiro, definir os estilos da aplicação, para depois implementarmos o login.

Organizando os estilos da nossa aplicação

Para realizar e mudar os estilos da nossa aplicação, iremos trabalhar inicialmente com três arquivos. O *styles.xml*, *dimen.xml* e o *strings.xml*.

O styles ficará responsável pelas cores da aplicação. O dimen pelo layout (espaçamento e dimensões). O strings ficará responsável pelas strings que desejaremos escrever.

colors.xml

Vamos abrir o arquivo colors.xml localizado dentro da pasta 'res > values'. O arquivo está com as configurações iniciais que o próprio projeto cria para nós, bem como as outras configurações iniciais estarão pré-configuradas.

Vamos alterar este arquivo para deixarmos o app com as cores que desejarmos.

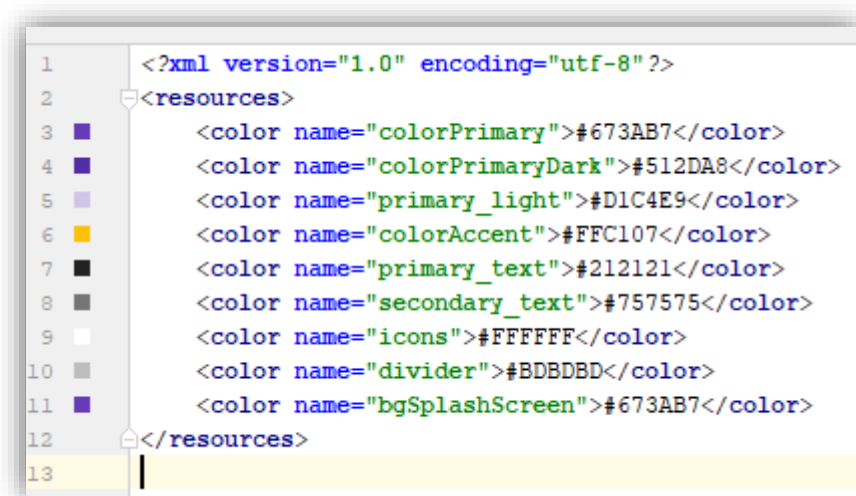


Figura 7 - Arquivo colors.xml

"Copiável":

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="colorPrimary">#673AB7</color>
  <color name="colorPrimaryDark">#512DA8</color>
  <color name="primary_light">#D1C4E9</color>
  <color name="colorAccent">#FFC107</color>
  <color name="primary_text">#212121</color>
  <color name="secondary_text">#757575</color>
  <color name="icons">#FFFFFF</color>
  <color name="divider">#BDBDBD</color>
  <color name="bgSplashScreen">#673AB7</color>
</resources>
```

Ao visualizarmos a nossa tela de login, podemos perceber que as cores foram atualizadas.

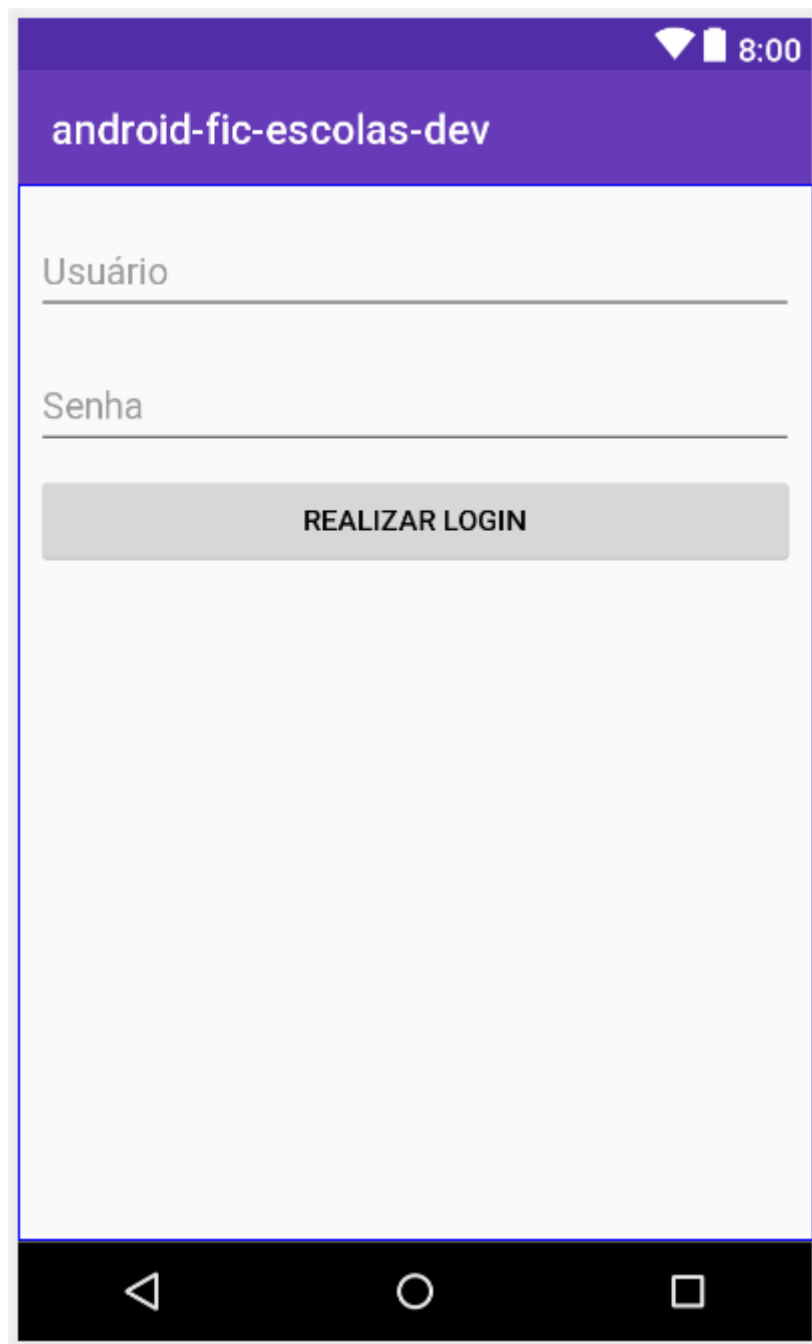


Figura 8 - Cor atualizada da tela de login

É importante destacar que o nome, você poderá definir bem como as cores que deseja utilizar. Neste caso, estamos apenas seguindo as boas práticas de desenvolvimento.

Outra dica: com o Material Palette, você consegue selecionar duas cores de preferência e ele disponibiliza o arquivo xml pra você apenas copiar e colocar na sua aplicação.

<https://www.materialpalette.com/>

Além disso, ele te mostra um preview de como ficará seu app com as cores selecionadas.

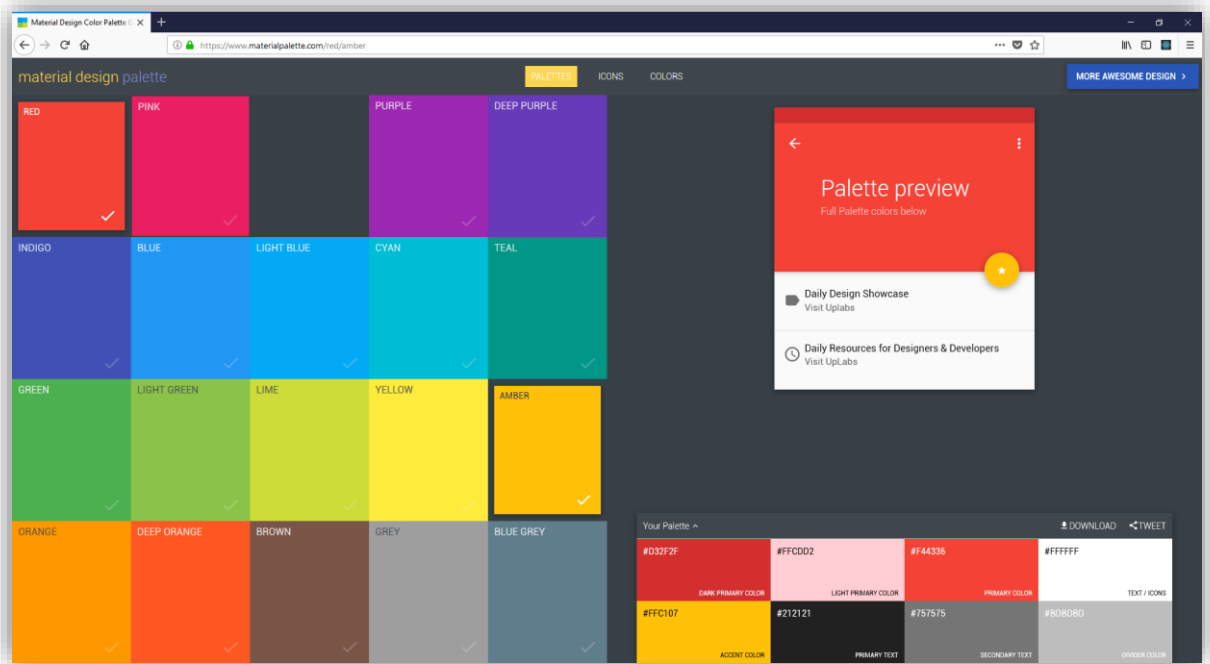


Figura 9 – Selecionando as cores no Material Palette

Após a seleção de cores que deseja, basta apenas realizar o download e copiar as cores para o colors.xml.

Vamos realizar uma nova alteração na tela de login para mostrarmos como o colors funciona. Voltando ao arquivo activity_login.xml, iremos deixar a cor de fundo do botão de realizar login, amarela.

No botão, iremos adicionar a linha que segue com a referência da cor que desejamos:

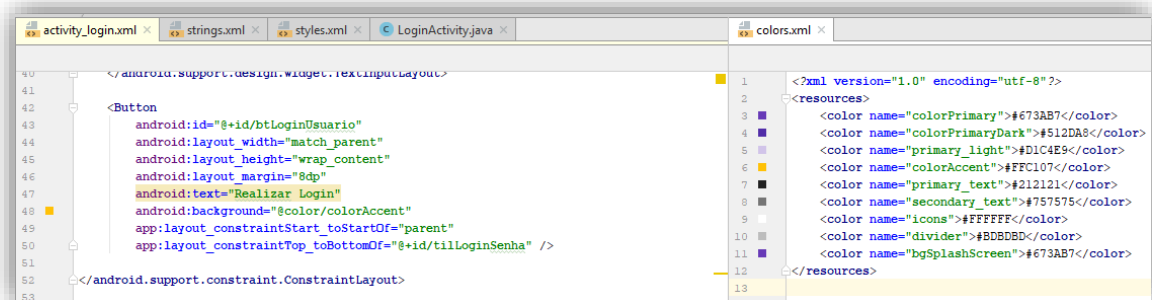


Figura 10 - Editando a cor do botão da tela de login

android:background="@color/colorAccent"

Após realizarmos essa alteração, podemos ver o resultado.

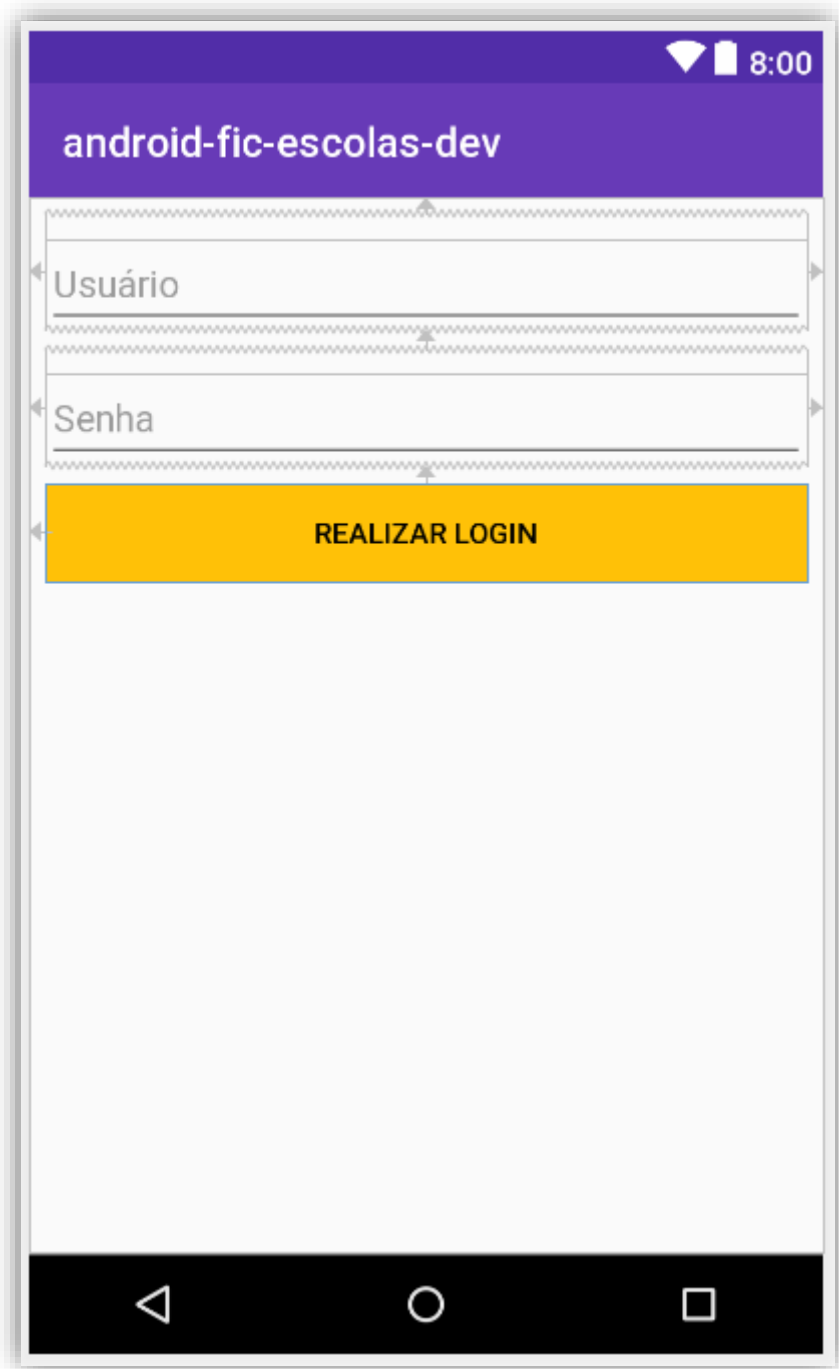


Figura 11 - Botão de login com a cor alterada

dimen.xml

Continuando na nossa tela de login, podemos reparar um detalhe que em uma única activity, não teríamos muito problema, porém, imagina um aplicativo real. Quantas telas teríamos?

Uma dica: tem referência com os espaçamentos e margens.

Se verificarmos que todas as nossas views possuem a mesma margem de 8dp, por quê não colocarmos ela como uma “variável”? Para isto, utilizamos o `dimen.xml`.

Por padrão, ele não é criado. Nós iremos criá-lo dentro da mesma pasta que o `colors.xml`.

Clique com o botão direito na pasta ‘res > values’.

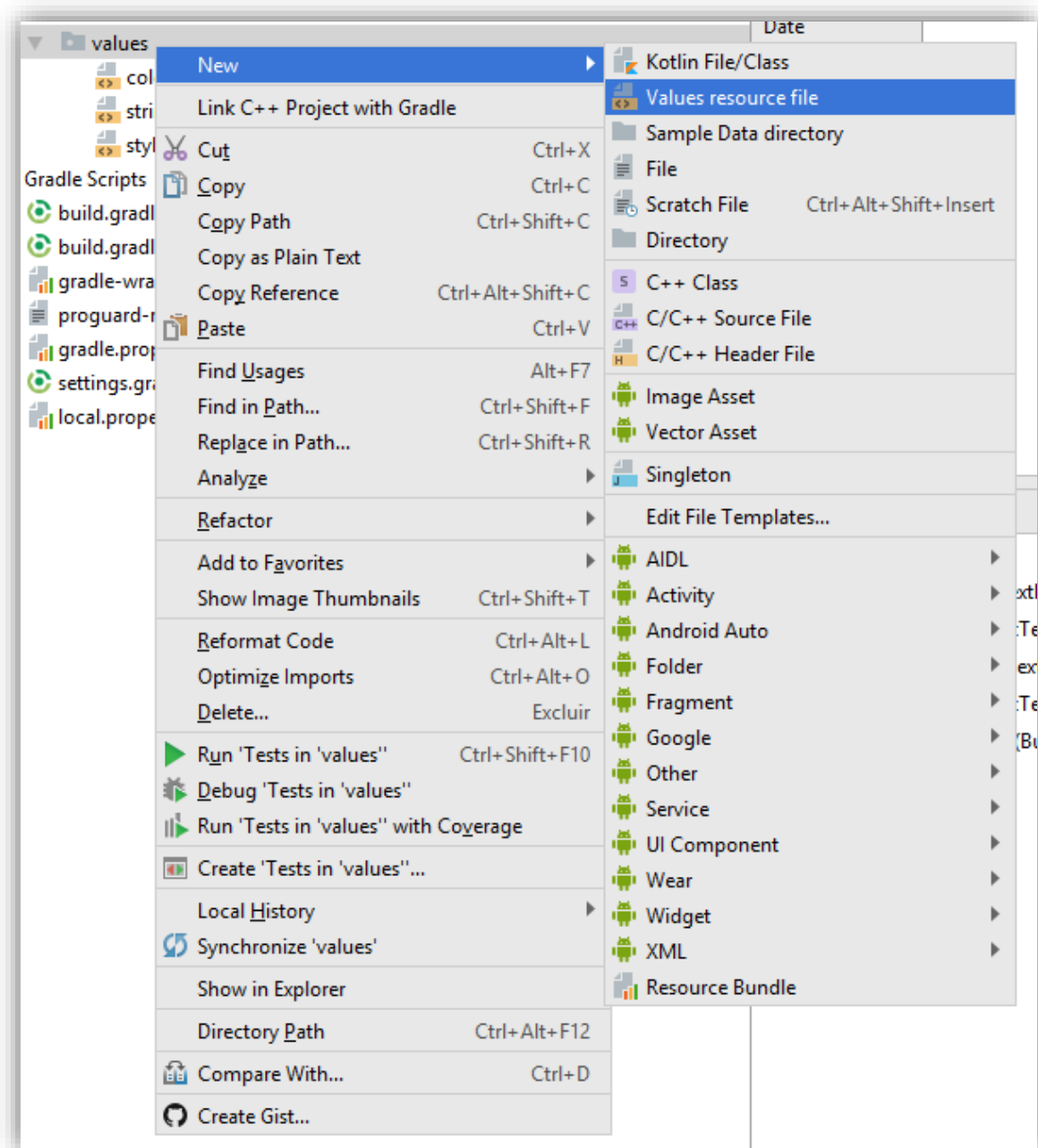


Figura 12 - Criando o arquivo `dimen.xml`

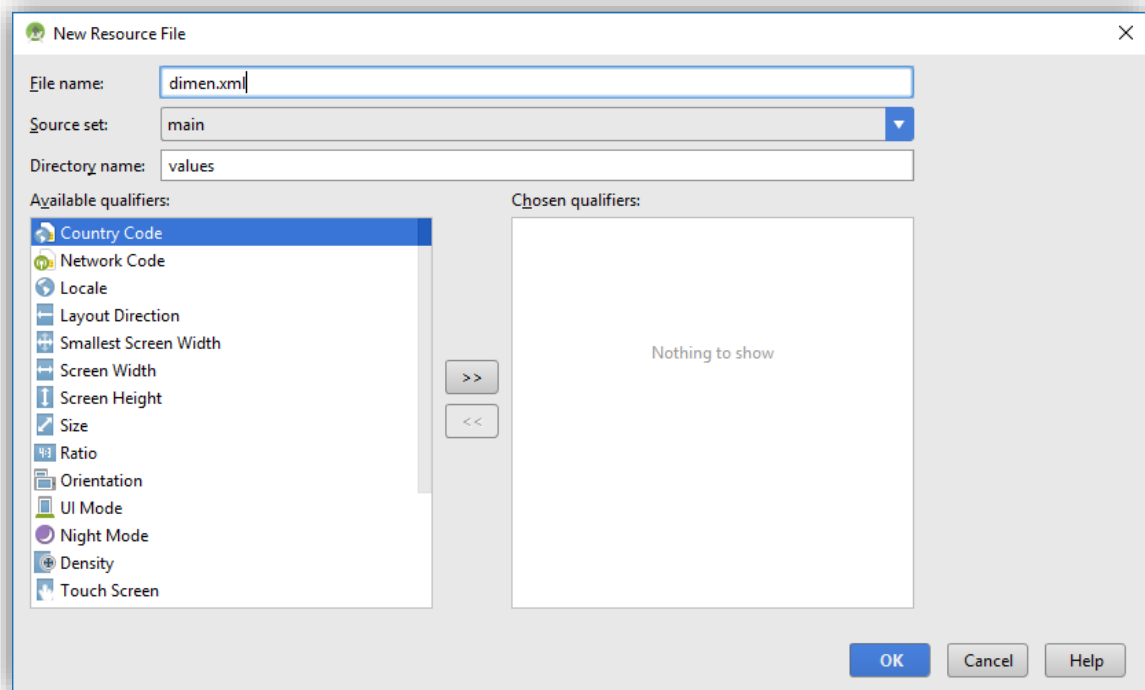


Figura 13 - Inserindo o nome do arquivo que deseja criar (dimen.xml)

Clicar em 'OK'.

Veremos que o arquivo foi criado com sucesso.

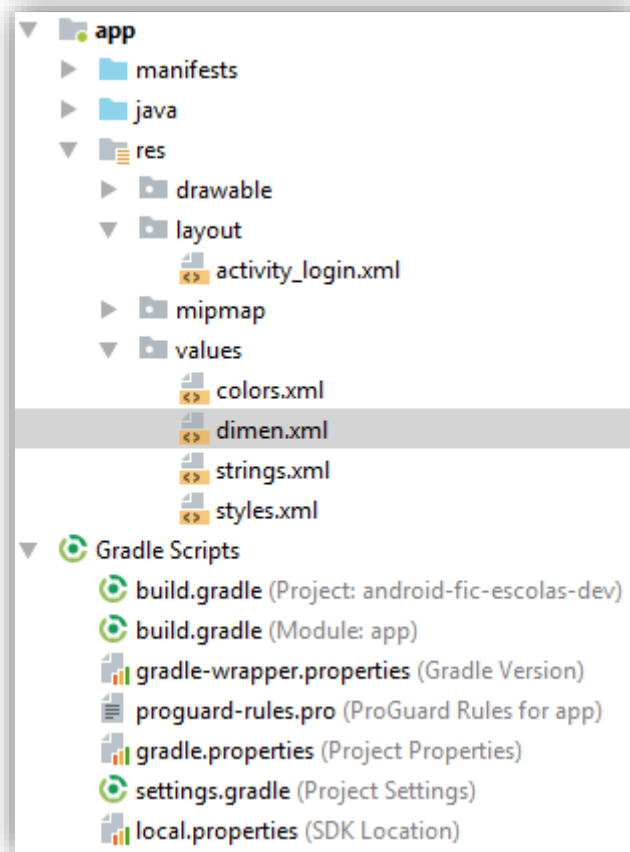


Figura 14 - Arquivo dimen.xml criado com sucesso

Criaremos uma nova propriedade que será a margem padrão de 8dp.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="margem_padrao_8dp">8dp</dimen>
</resources>
```

Iremos alterar novamente a nossa activity_login.xml.

Ao invés de escrevermos diretamente para cada item, iremos fazer a referência para o nosso arquivo de layout.

```
android:layout_margin="8dp"
```

```
android:layout_margin="@dimen/margem_padrao_8dp"
```

Depois da alteração, nosso layout ficará da seguinte maneira.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
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:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="br.senai.sp.android_fic_escolas_dev.LoginActivity">

<android.support.design.widget.TextInputLayout
    android:id="@+id/tilLoginEmail"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/margem_padrao_8dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">

    <android.support.design.widget.TextInputEditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Usuário" />

</android.support.design.widget.TextInputLayout>

<android.support.design.widget.TextInputLayout
    android:id="@+id/tilLoginSenha"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/margem_padrao_8dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tilLoginEmail">

    <android.support.design.widget.TextInputEditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Senha" />

</android.support.design.widget.TextInputLayout>

<Button
    android:id="@+id/btLoginUsuario"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/margem_padrao_8dp"
    android:text="Realizar Login"
    android:background="@color/colorAccent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tilLoginSenha" />

</android.support.constraint.ConstraintLayout>

```

A sugestão de trabalhar com os arquivos de “values” que são disponibilizados é: evitar que seja necessário fazer a refatoração muitas vezes. Uma vez que eu queira alterar, por exemplo, o tamanho dessa margem, basta apenas eu alterar em um único local.

strings.xml

E por enquanto, vamos alterar o nosso último arquivo. O strings.xml. Iremos seguir a mesma linha de raciocínio dos outros arquivos. No arquivo colors.xml, vimos que ele serve para alterar de forma única, as cores da nossa aplicação. No arquivos dimen.xml, alterar os layouts referentes a espaçamento, margens, dinamismo.

No arquivo strings.xml iremos alterar coisas referentes a títulos, frases, palavras que sejam fixas. Assim como os outros dois arquivos, eles ficam localizados dentro de 'res > values'.

Ao acessarmos a nossa activity_login.xml e vermos as dicas que o próprio android nos oferece, podemos verificar que ele "aconselha" a não deixar as propriedades escritas diretamente no texto.

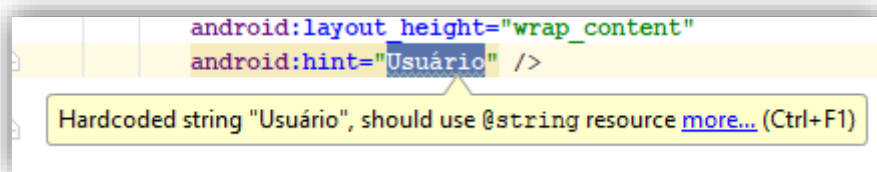


Figura 15 - Dica do Android Studio referente a strings.xml

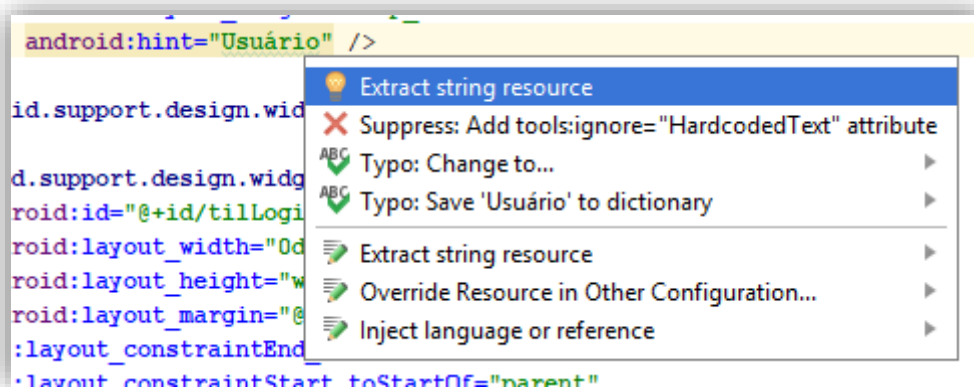


Figura 16 - Criando um novo recurso de string

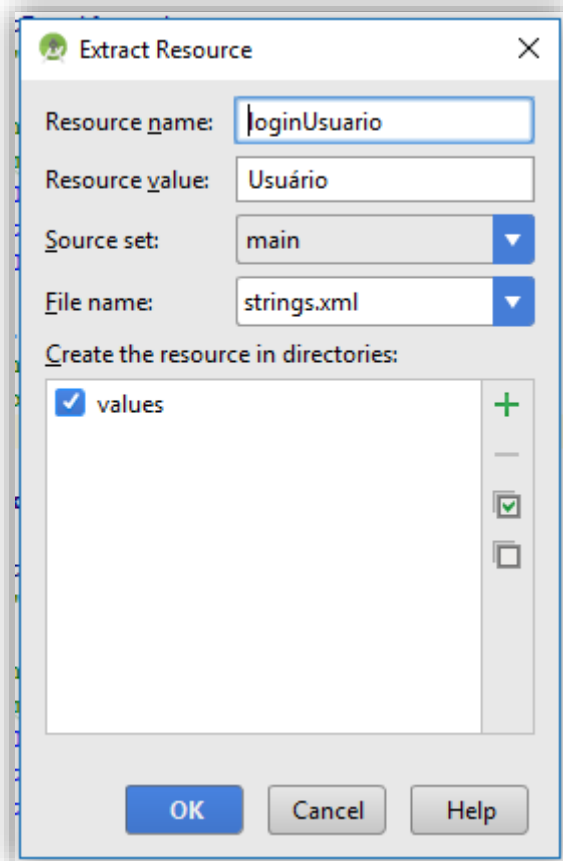


Figura 17 - Criando um novo recurso

Clique em 'OK'.

```
<android.support.design.widget.TextInputEditText  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/loginUsuario" />
```

Ao acessarmos o arquivo strings.xml, podemos verificar que o recurso foi criado.

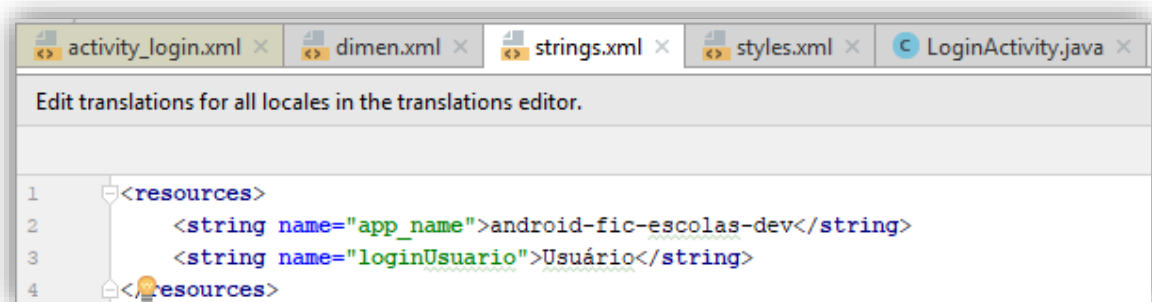


Figura 18 - String criada

Esta é apenas uma abordagem. Você também pode acessar diretamente o arquivo `strings.xml` e realizar a criação das strings que você deseja manualmente.

Realizando nossa ação de login

Até agora, nós apenas mexemos no layout da nossa tela de login. E aprendemos a como mexer em alguns arquivos de configuração. Essa abordagem inicial, é apenas para mostrar como manipulamos os arquivos na tela e como colocamos propriedades em suas configurações.

Uma vez que criamos a tela e seu respectivo layout, vamos começar a colocar as ações que desejamos. Quando queremos realizar um login, precisamos verificar se o e-mail e a senha coincidem com o cadastro.

Por enquanto, trabalharemos com listas para realizar o nosso “registro de cadastros”.

Vamos começar a desenvolver a estrutura de pasta do nosso projeto (classes Java) e da nossa aplicação para atender o nosso problema.

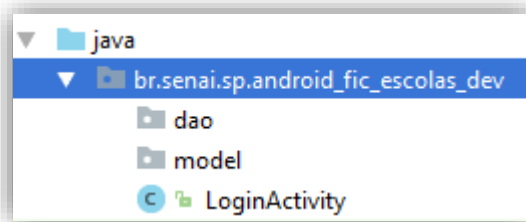


Figura 19 - Criando a estrutura da aplicação

Uma vez criada a nossa estrutura, precisamos criar o modelo que irá corresponder ao nosso usuário, e o nosso dao (data access object) que irá realizar a consulta na nossa lista criada para verificar se a conta que o usuário irá digitar, irá coincidir com os valores que inicialmente temos em nossa lista.

Para isto, iremos criar mais dois arquivos. Um arquivo chamado `Usuario.java` dentro da pasta `model`, e outro arquivo chamado `UsuarioDao.java` dentro da pasta `dao`.

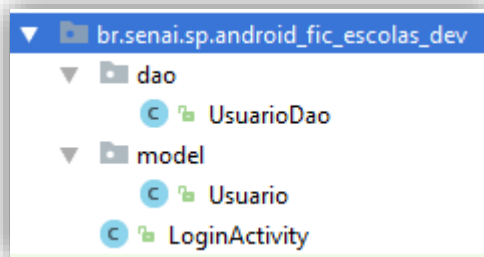


Figura 20 - Estrutura do Usuário criada

Por enquanto, apenas criamos a estrutura do que desejamos utilizar. Nós ainda não fizemos nenhuma referência na nossa LoginActivity, não colocamos nenhuma ação nele.

Vamos fazer os seguintes passos: buscarmos as referências das views na nossa LoginActivity e após isso, precisamos consultar para verificar se os dados que o usuário digitou, conferem com os dados que estão na base.

Primeiro passo:

Buscamos a referência das views de nossa tela. Vale lembrar que, nós colocamos um identificador para cada item da nossa view que desejamos ter referência.

```
public class LoginActivity extends AppCompatActivity {

    private TextInputLayout tilLoginEmail;
    private TextInputLayout tilLoginSenha;
    private Button btLoginUsuario;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        // fazendo as referências das nossas views
        tilLoginEmail = findViewById(R.id.tilLoginEmail);
        tilLoginSenha = findViewById(R.id.tilLoginSenha);
        btLoginUsuario = findViewById(R.id.btLoginUsuario);
    }
}
```

Segundo passo:

Buscamos as informações que o usuário irá digitar na tela e imprimir tanto para o usuário visualizar, quanto no log da nossa aplicação.

```

public class LoginActivity extends AppCompatActivity {

    // views
    private TextInputLayout tilLoginEmail;
    private TextInputLayout tilLoginSenha;
    private Button btLoginUsuario;

    // variáveis do usuário
    private String loginDigitado;
    private String senhaDigitada;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        // fazendo as referências das nossas views
        tilLoginEmail = findViewById(R.id.tilLoginEmail);
        tilLoginSenha = findViewById(R.id.tilLoginSenha);
        btLoginUsuario = findViewById(R.id.btLoginUsuario);

        btLoginUsuario.setOnClickListener(new verificarUsuarioDigitado());
    }

    private class verificarUsuarioDigitado implements View.OnClickListener {
        @Override
        public void onClick(View view) {

            loginDigitado = tilLoginSenha.getEditText().getText().toString();
            senhaDigitada = tilLoginSenha.getEditText().getText().toString();
            Toast.makeText(getApplicationContext(), text: loginDigitado + senhaDigitada, Toast.LENGTH_LONG).show();
            Log.d( tag: "Usuario: ", msg: loginDigitado + senhaDigitada);

        }
    }
}

```

Figura 21 - Buscando os dados do usuário digitado

"Copiável":

```

public class LoginActivity extends AppCompatActivity {

    // views
    private TextInputLayout tilLoginEmail;
    private TextInputLayout tilLoginSenha;
    private Button btLoginUsuario;

    // variáveis do usuário
    private String loginDigitado;
    private String senhaDigitada;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        // fazendo as referências das nossas views
        tilLoginEmail = findViewById(R.id.tilLoginEmail);
        tilLoginSenha = findViewById(R.id.tilLoginSenha);
        btLoginUsuario = findViewById(R.id.btLoginUsuario);

        btLoginUsuario.setOnClickListener(new
verificarUsuarioDigitado());
    }
}

```

```

    }

    private class verificarUsuarioDigitado implements
View.OnClickListener {
        @Override
        public void onClick(View view) {

            loginDigitado =
tilLoginSenha.getEditText().getText().toString();
            senhaDigitada =
tilLoginSenha.getEditText().getText().toString();
            Toast.makeText(getApplicationContext(), loginDigitado +
senhaDigitada, Toast.LENGTH_LONG).show();
            Log.d("Usuario: ", loginDigitado + senhaDigitada);

        }
    }
}

```

Uma vez conseguimos buscar os valores do usuário digitado, precisamos fazer qual ação? Verificar se o usuário confere com o usuário cadastrado em nossa lista. Porém, nós ainda não criamos o nosso modelo e nem o nosso dao. Ou melhor, criamos apenas o arquivo, mas não colocamos nenhuma implementação nele.

Neste momento, iremos realizar a implementação.

Vamos começar colocando as propriedades dentro do objeto Usuário. E depois, vamos realizar a sua implementação.

Usuario.java.


```

public class Usuario {

    private Long id;
    private String email;
    private String senha;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getSenha() {
        return senha;
    }

    public void setSenha(String senha) {
        this.senha = senha;
    }

    public Usuario(String email, String senha) {
        this.email = email;
        this.senha = senha;
    }

}

```

Figura 22 - Usuario.java

Copiável:

```

public class Usuario {

    private Long id;
    private String email;
    private String senha;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

```

```

    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getSenha() {
        return senha;
    }

    public void setSenha(String senha) {
        this.senha = senha;
    }

    public Usuario(String email, String senha) {
        this.email = email;
        this.senha = senha;
    }

    // gerar hashCode, equals e toString()
}

```

UsuarioDao.java.

```

public class UsuarioDao {

    public static UsuarioDao manager = new UsuarioDao();

    // Lista aonde serão armazenados os alunos
    private List<Usuario> lista;

    private long id = 1;

    private UsuarioDao() {
        lista = new ArrayList<>();
        lista.add(new Usuario( email: "a", senha: "a"));
    }

    public Usuario localizar(Usuario usuarioRecebido) {

        Usuario procurarUsuario = null;

        for (Usuario u : lista) {
            if ((u.getEmail().equals(usuarioRecebido.getEmail())) && (u.getSenha().equals(usuarioRecebido.getSenha()))) {
                procurarUsuario = u;
                return procurarUsuario;
            }
        }
        return procurarUsuario;
    }

}

```

Figura 23 - UsuarioDao.java

Copiável:


```

package br.senai.sp.android_fic_escolas_dev.dao;

import java.util.ArrayList;
import java.util.List;

import br.senai.sp.android_fic_escolas_dev.model.Usuario;

/**
 * Created by Helena Strada on 23/03/2018.
 */

public class UsuarioDao {

    public static UsuarioDao manager = new UsuarioDao();

    // Lista aonde serão armazenados os alunos
    private List<Usuario> lista;

    private long id = 1;

    private UsuarioDao() {
        lista = new ArrayList<>();
        lista.add(new Usuario("administrador", "administrador"));
    }

    public Usuario localizar(Usuario usuarioRecebido) {

        Usuario procurarUsuario = null;

        for (Usuario u : lista) {
            if ((u.getEmail().equals(usuarioRecebido.getEmail())) &&
(u.getSenha().equals(usuarioRecebido.getSenha()))) {
                procurarUsuario = u;
                return procurarUsuario;
            }
        }
        return procurarUsuario;
    }

}

```

Precisamos agora, ao invés de apenas apresentarmos os valores sendo digitados na tela, procurar esse valor na lista de usuários que criamos e verificar se o usuário digitado, confere com o valor armazenado em nossa atual lista.

Além disso, vamos aprender a manipular um outro elemento na tela. Uma vez que o campo senha não deve permanecer a vista para que outras pessoas possam visualizar o que está sendo digitado, iremos trocar o tipo de campo de entrada.

Na activity_login.xml, vamos colocar o tipo de entrada para "textPassword".

```

<android.support.design.widget.TextInputLayout
    android:id="@+id/tilLoginSenha"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/margem_padrao_8dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"

```

```
app:layout_constraintTop_toBottomOf="@+id/tilLoginEmail">  
  
<android.support.design.widget.TextInputEditText  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:hint="Senha"  
    android:inputType="textPassword" />  
  
</android.support.design.widget.TextInputLayout>
```

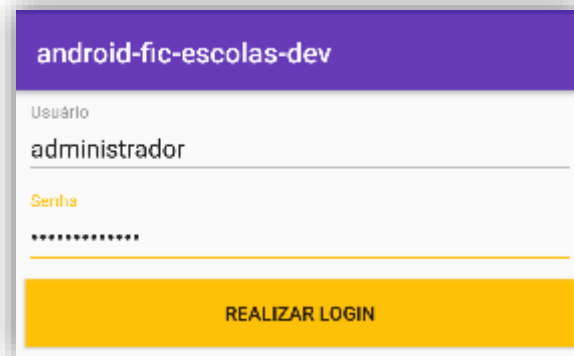


Figura 24 - `inputType = "textPassword"`

Basta apenas realizarmos na nossa LoginActivity, a validação para o usuário digitado.

```

public class LoginActivity extends AppCompatActivity {

    // views
    private TextInputLayout tilLoginEmail;
    private TextInputLayout tilLoginSenha;
    private Button btLoginUsuario;

    // variáveis do usuário
    private String loginDigitado;
    private String senhaDigitada;

    private UsuarioDao dao = UsuarioDao.manager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        // fazendo as referências das nossas views
        tilLoginEmail = findViewById(R.id.tilLoginEmail);
        tilLoginSenha = findViewById(R.id.tilLoginSenha);
        btLoginUsuario = findViewById(R.id.btLoginUsuario);

        btLoginUsuario.setOnClickListener(new verificarUsuarioDigitado());
    }

    private class verificarUsuarioDigitado implements View.OnClickListener {
        @Override
        public void onClick(View view) {

            loginDigitado = tilLoginSenha.getEditText().getText().toString();
            senhaDigitada = tilLoginSenha.getEditText().getText().toString();
            Usuario usuarioConferirNaLista = new Usuario(loginDigitado, senhaDigitada);

            if (dao.localizar(usuarioConferirNaLista) != null) {
                Toast.makeText(getApplicationContext(), text: "Login confere.", Toast.LENGTH_LONG).show();
            } else {
                Toast.makeText(getApplicationContext(), text: "Usuário ou senha incorretos.", Toast.LENGTH_SHORT).show();
            }
        }
    }
}

```

Figura 25 - LoginActivity validando email e senha digitados.

Copiável:

```

package br.senai.sp.android_fic_escolas_dev;

import android.support.design.widget.TextInputLayout;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import br.senai.sp.android_fic_escolas_dev.dao.UsuarioDao;
import br.senai.sp.android_fic_escolas_dev.model.Usuario;

public class LoginActivity extends AppCompatActivity {

    // views
    private TextInputLayout tilLoginEmail;
    private TextInputLayout tilLoginSenha;

```

```

private Button btLoginUsuario;

// variáveis do usuário
private String loginDigitado;
private String senhaDigitada;

private UsuarioDao dao = UsuarioDao.manager;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);

    // fazendo as referências das nossas views
    tilLoginEmail = findViewById(R.id.tilLoginEmail);
    tilLoginSenha = findViewById(R.id.tilLoginSenha);
    btLoginUsuario = findViewById(R.id.btLoginUsuario);

    btLoginUsuario.setOnClickListener(new
verificarUsuarioDigitado());
}

private class verificarUsuarioDigitado implements
View.OnClickListener {
    @Override
    public void onClick(View view) {

        loginDigitado =
tilLoginSenha.getText().getText().toString();
        senhaDigitada =
tilLoginSenha.getText().getText().toString();
        Usuario usuarioConferirNaLista = new
Usuario(loginDigitado, senhaDigitada);

        if (dao.localizar(usuarioConferirNaLista) != null) {
            Toast.makeText(getApplicationContext(), "Login
confere.", Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(getApplicationContext(), "Usuário ou
senha incorretos.", Toast.LENGTH_SHORT).show();
        }
    }
}
}

```

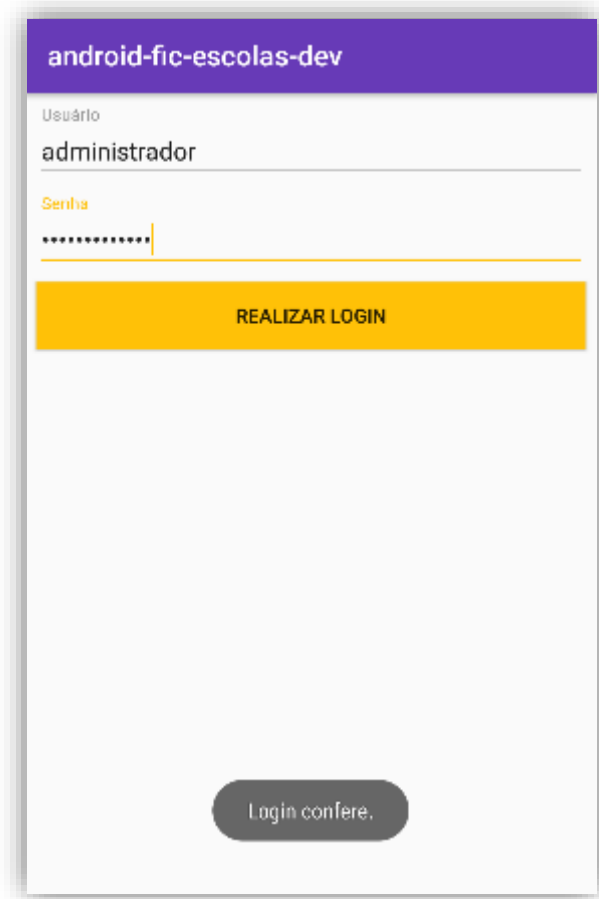


Figura 26 - Após validação com os dados da lista.

Uma vez que conseguimos manipular os valores que foram digitados, ou seja, verificar se o usuário e a senha foram digitados de maneira correta, precisamos realizar alguma ação após a validação. Em um sistema comum, após o usuário digitar o e-mail e a senha corretos, ele é redirecionado para a tela de “home” do usuário. Ou para a tela principal e/ou inicial do sistema/aplicativo.

No nosso caso, uma vez que o login foi feito com sucesso, iremos redirecioná-lo para a tela de listagem de alunos.

Nossa primeira etapa do projeto está pronta. Vamos partir para a segunda etapa que será criar a nossa home e realizar a listagem de alunos.

Arquivos

Projeto: android-fic-escolas-dev-primeira-etapa

Documentos: android.docx, android-estrutura.docx, android-resources.docx

Resumo

Nesta primeira etapa do projeto, nós apresentamos uma visão geral do Android, como funciona um pouco sua estrutura, quais arquivos você precisa manipular para personalizar seu aplicativo e a como manipular alguns elementos simples de tela (Button, EditText, TextInputLayout), um pouco mais sobre ConstraintLayout e a como fazer referência a essas views e buscar/tratar os dados que o usuário irá digitar para validação.

Nas próximas etapas, veremos como ir além de uma tela. Vamos criar novas telas, realizar listagens de registros, criar listas personalizadas e muito mais.

Segunda Etapa

Criando a nossa tela principal

Uma vez que o usuário realizar o login com sucesso, iremos redirecioná-lo para a tela principal da aplicação. Que será a nossa listagem de alunos cadastrados.

Uma dica: como nós iremos trabalhar com mais de uma activity, vamos separar as nossas activities (nossas "telas"), das demais informações de modelo, dao, e outras informações do nosso projeto. Para isto, vamos criar um novo pacote chamado *views*.

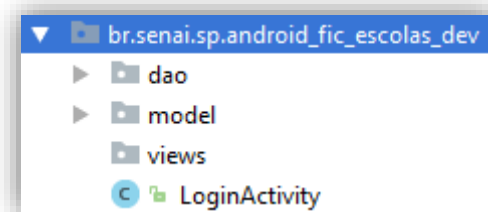


Figura 27 - Criar um pacote chamado views.

Uma vez criado o pacote, vamos criar a nossa activity principal, que conterà, como disse anteriormente, a listagem de alunos cadastrados.

Dentro do pacote *views* que acabamos de criar, vamos criar uma nova activity do tipo Basic Activity, chamada *MainActivity*.

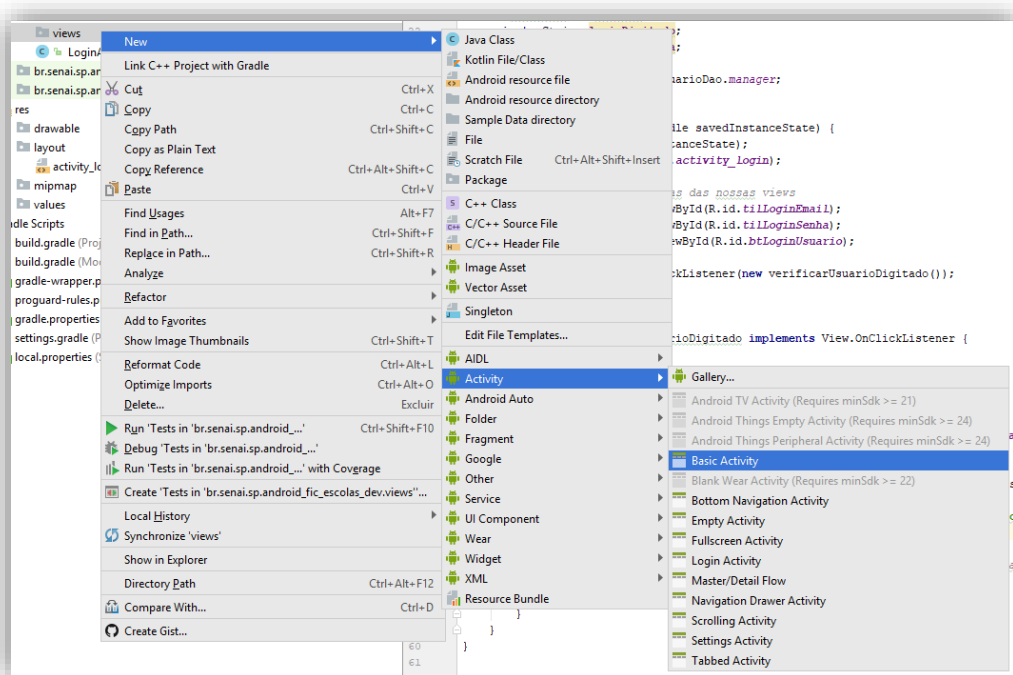


Figura 28 - Criando a MainActivity com o estilo Basic Activity

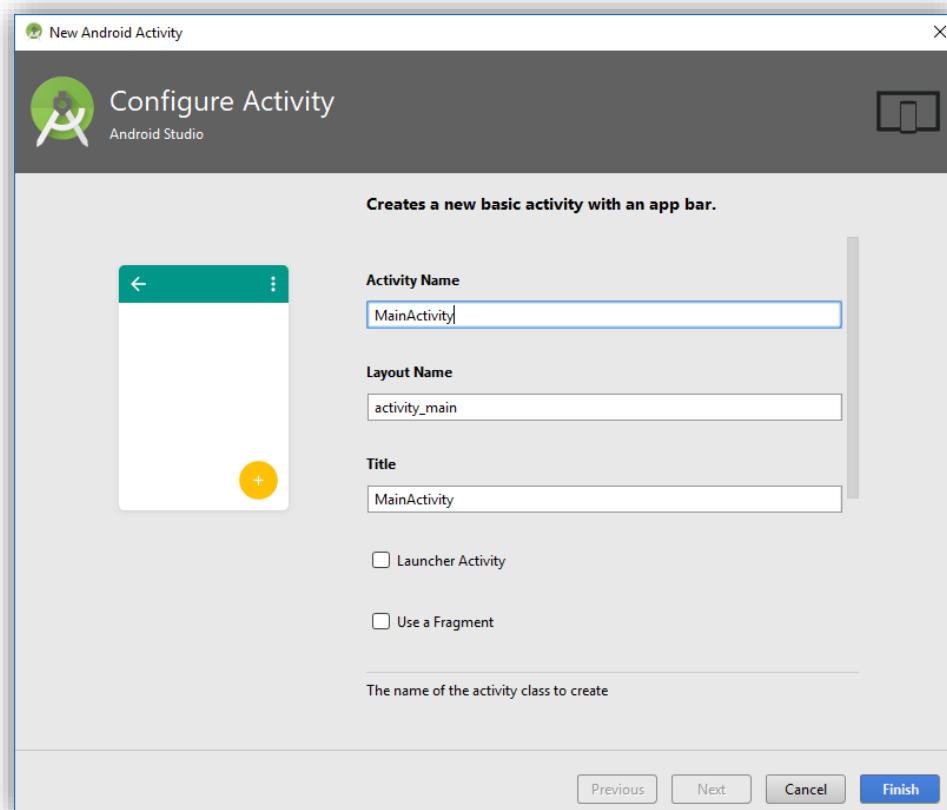


Figura 29 - MainActivity.java criada.

Clicar em 'Finish'.

Estrutura atual depois de criada nossa MainActivity.java.

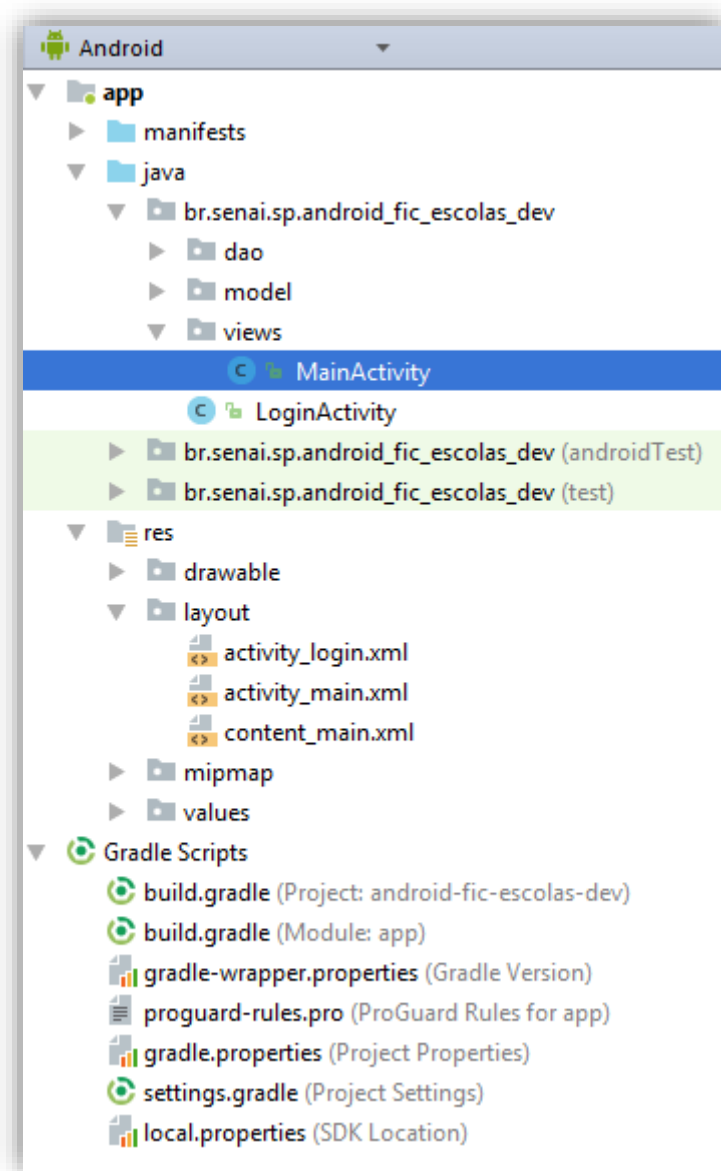


Figura 30 - Estrutura do projeto após criada a MainActivity.

Redirecionando o usuário para uma nova tela

O que precisamos fazer agora é: assim que o usuário digitar os valores corretos de entrada, ele tem a intenção de ir para uma nova tela, ou seja, ser redirecionado para a nossa tela principal.

```

if (dao.localizar(usuarioConferirNaLista) != null) {
    Intent intent = new Intent( packageContext: LoginActivity.this, MainActivity.class);
    startActivity(intent);
    LoginActivity.this.finish();
    // Toast.makeText(getApplicationContext(), "Login confere.", Toast.LENGTH_LONG).show();
} else {
    Toast.makeText(getApplicationContext(), text: "Usuário ou senha incorretos.", Toast.LENGTH_SHORT).show();
}

```

Figura 31 - Usuário sendo redirecionado após realizar login.

Copiável:

```

if (dao.localizar(usuarioConferirNaLista) != null) {
    Intent intent = new Intent(LoginActivity.this,
MainActivity.class);
    startActivity(intent);
    LoginActivity.this.finish();
    // Toast.makeText(getApplicationContext(), "Login confere.",
Toast.LENGTH_LONG).show();
} else {
    Toast.makeText(getApplicationContext(), "Usuário ou senha
incorretos.", Toast.LENGTH_SHORT).show();
}

```

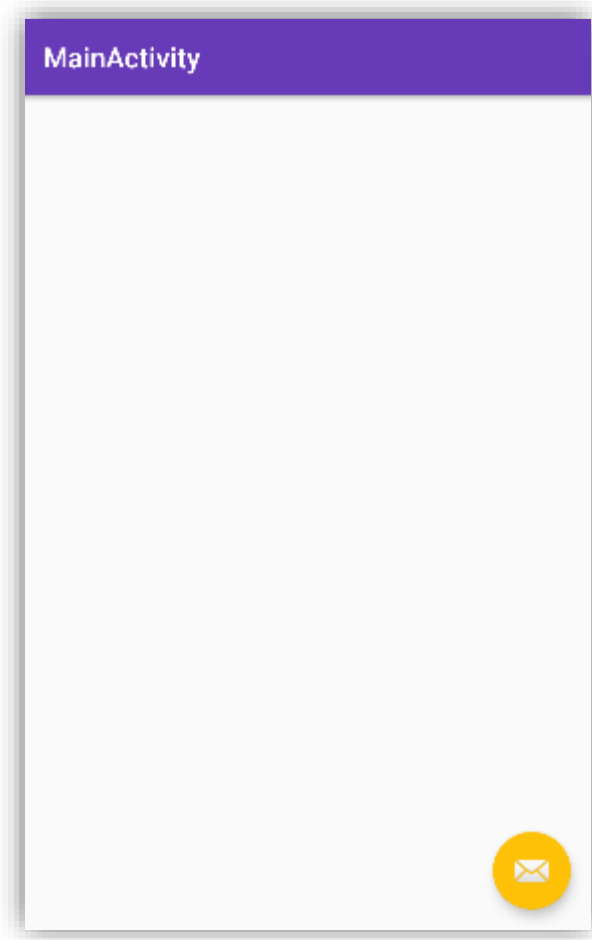


Figura 32 - Após o login ser realizado com sucesso, a Main irá aparecer.

Nós criamos uma Activity com o estilo Basic. A Base Activity é um estilo pré-definido do Android que gera um FloatingActionButton, facilitando o nosso trabalho.

As alterações que iremos fazer primeiramente, será feita na `content_main.xml`.

Mostrando a nossa primeira lista

Essa tela ficará responsável por carregar a lista de alunos cadastrados no nosso sistema. Primeiramente, vamos criar uma lista simples, que irá exibir o nome de todos os alunos.

Para adicionarmos uma nova lista em nossa tela, iremos criar uma view do tipo ListView. Vale lembrar que precisamos colocar um id para ela, para que em nossa classe Java, nós consigamos buscar sua referência.

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingView..."
    tools:context="br.senai.sp.android_fic_escolas_dev.views.MainActivity"
    tools:showIn="@layout/activity_main">

    <ListView
        android:id="@+id/lvListaAlunos"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</android.support.constraint.ConstraintLayout>

```

Figura 33 - Adicionando a ListView.

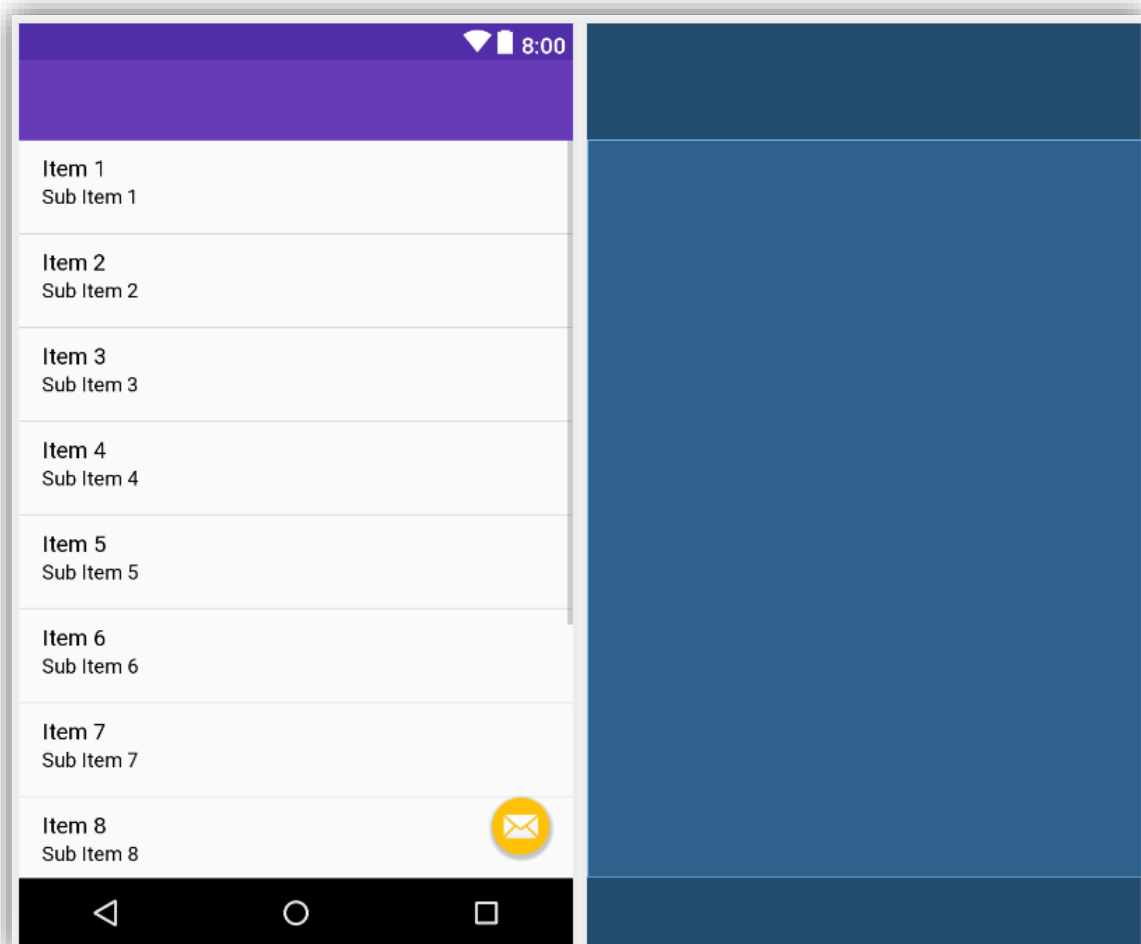


Figura 34 - Nossa lista criada, porém sem ainda dados para popular.

Uma vez que criamos nossa ListView, precisamos buscar a referência dessa view em nossa classe Java e carregarmos essa lista com os valores que desejamos.

```

13
14 public class MainActivity extends AppCompatActivity {
15
16     private ListView lvListaAlunos;
17
18     @Override
19     protected void onCreate(Bundle savedInstanceState) {
20         super.onCreate(savedInstanceState);
21         setContentView(R.layout.activity_main);
22         Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
23         setSupportActionBar(toolbar);
24         // getSupportActionBar().setTitle("Lista de Alunos");
25
26         FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
27         fab.setOnClickListener(new View.OnClickListener() {
28             @Override
29             public void onClick(View view) {
30                 Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
31                     .setAction("Action", null).show();
32             }
33         });
34
35         lvListaAlunos = findViewById(R.id.lvListaAlunos);
36
37         String[] listagemDeAlunos = new String[]{"Helena", "Thales", "Felipe"};
38
39         ArrayAdapter<String> adapter = new ArrayAdapter<>(getApplicationContext(), android.R.layout.simple_list_item_1, listagemDeAlunos);
40         lvListaAlunos.setAdapter(adapter);
41
42     }
43
44 }
45
46

```

Figura 35 - Populando a ListView.

Copiável:

```

package br.senai.sp.android_fic_escolas_dev.views;

import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import br.senai.sp.android_fic_escolas_dev.R;

public class MainActivity extends AppCompatActivity {

    private ListView lvListaAlunos;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        // getSupportActionBar().setTitle("Lista de Alunos");

        FloatingActionButton fab = (FloatingActionButton)
        findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action",
                Snackbar.LENGTH_LONG)

```



```

        .setAction("Action", null).show();
    }
});

lvListaAlunos = findViewById(R.id.lvListaAlunos);

String[] listagemDeAlunos = new String[]{"Helena", "Thales",
"Felipe"};

ArrayAdapter<String> adapter = new
ArrayAdapter<>(getApplicationContext(),
android.R.layout.simple_list_item_1, listagemDeAlunos);
lvListaAlunos.setAdapter(adapter);

}
}

```

Nesta etapa, nós estamos exibindo a nossa lista ainda sem nenhuma personalização. Isso por quê é possível personalizar a forma de exibir os itens da lista, porém, por enquanto nós apenas dissemos para o layout que gostaríamos de utilizar a própria forma de exibir os itens que o Android disponibiliza : a *simple_list_item_1*.

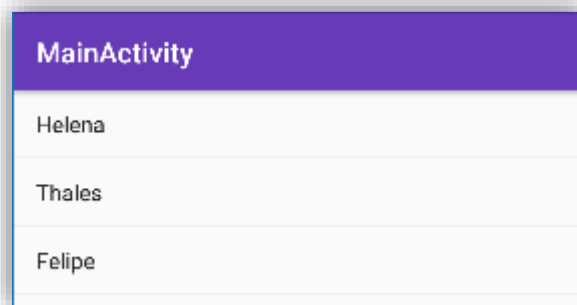


Figura 36 - Lista de Strings carregada na ListView.

Dica: para não ser necessário sempre ficar digitando os valores de login e senha, coloque os valores fixos em nossa LoginActivity.java.

```

tilLoginEmail.getText().setText("administrador");
tilLoginSenha.getText().setText("administrador");

```

Figura 37 - Dados fixos no login.

Nós faremos duas atividades em seguida. A primeira delas será não carregar os dados fixos de uma lista de Strings, porém, carregar os dados a partir dos alunos que forem cadastrados em nosso sistema. Primeiramente vamos criar o dao que posteriormente irá nos auxiliar no crud de nosso sistema.

A segunda delas será personalizar a nossa lista de acordo com o layout que desejamos.

Criando o nosso modelo de dados do Aluno

Nesta etapa, iremos modelar um aluno, de acordo com os dados que queremos que sejam exibidos. Lembrando que no início, quando falamos das etapas, nós dissemos que iríamos ter alguns tipos de dados diferentes para trabalharmos: imagem, nome, data de nascimento e endereço

O aluno terá então uma classe de modelo com os dados apresentados anteriormente e uma classe DAO, bem como o nosso usuário.

Nessa lista, por enquanto, estaremos exibindo alunos pré-cadastrados, ou seja, os dados não poderão ser manipulados pelo usuário

Aluno.java.

```

public class Aluno {

    private Long id;
    private String nome;
    private Date dataNascimento;
    private String endereco;
    private String fotoAluno;

+   public Aluno(Long id, String nome, Date dataNascimento, String endereco) {...}
+   public Aluno(Long id) { this.id = id; }
+   public Aluno() {...}
+   public Aluno(String nome, Date dataNascimento, String endereco) {...}
+   public String getFotoAluno() { return fotoAluno; }
+   public void setFotoAluno(String fotoAluno) { this.fotoAluno = fotoAluno; }
+   public Long getId() { return id; }
+   public void setId(Long id) { this.id = id; }
+   public String getNome() { return nome; }
+   public void setNome(String nome) { this.nome = nome; }
+   public Date getDataNascimento() { return dataNascimento; }
+   public void setDataNascimento(Date dataNascimento) { this.dataNascimento = dataNascimento; }
+   public String getEndereco() { return endereco; }
+   public void setEndereco(String endereco) { this.endereco = endereco; }

    @Override
+   public boolean equals(Object o) {...}

    @Override
+   public int hashCode() { return id != null ? id.hashCode() : 0; }

    @Override
+   public String toString() {...}
}

```

Figura 38 - Aluno.java

Copiável:

```

public class Aluno {

    private Long id;
    private String nome;
    private Date dataNascimento;
    private String endereco;
    private String fotoAluno;

    public Aluno(Long id, String nome, Date dataNascimento, String
endereco) {
        this.id = id;
        this.nome = nome;
        this.dataNascimento = dataNascimento;
        this.endereco = endereco;
    }
}

```

```

public Aluno(Long id) {
    this.id = id;
}

public Aluno() {
}

public Aluno(String nome, Date dataNascimento, String endereco) {
    this.nome = nome;
    this.dataNascimento = dataNascimento;
    this.endereco = endereco;
}

public String getFotoAluno() {
    return fotoAluno;
}

public void setFotoAluno(String fotoAluno) {
    this.fotoAluno = fotoAluno;
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
}

public Date getDataNascimento() {
    return dataNascimento;
}

public void setDataNascimento(Date dataNascimento) {
    this.dataNascimento = dataNascimento;
}

public String getEndereco() {
    return endereco;
}

public void setEndereco(String endereco) {
    this.endereco = endereco;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Aluno aluno = (Aluno) o;

```

```

        return id != null ? id.equals(aluno.id) : aluno.id == null;
    }

    @Override
    public int hashCode() {
        return id != null ? id.hashCode() : 0;
    }

    @Override
    public String toString() {
        return "Aluno{" +
            "id=" + id +
            ", nome='" + nome + '\'' +
            ", dataNascimento=" + dataNascimento +
            ", endereco='" + endereco + '\'' +
            ", fotoAluno=" + fotoAluno +
            '}';
    }
}

```

Agora iremos criar o nosso dao para realizar as ações de listar os alunos, cadastrar um novo aluno, remover e buscar apenas um aluno.

AlunoDao.java.

```

public class AlunoDao {

    public static AlunoDao manager = new AlunoDao();
    // Lista aonde serão armazenados os alunos
    private List<Aluno> lista;
    private long id = 1;

    private AlunoDao() {
        Date datel = null, date2 = null, date3 = null;
        try {
            SimpleDateFormat dateformat3 = new SimpleDateFormat( pattern: "dd/MM/yyyy");
            datel = dateformat3.parse( source: "17/07/1989");
            date2 = dateformat3.parse( source: "18/03/1993");
            date3 = dateformat3.parse( source: "01/11/1997");
        } catch (ParseException e) {
            e.printStackTrace();
        }

        lista = new ArrayList<>();
        lista.add(new Aluno(id++, nome: "Helena Strada", date2, endereco: "Av. Paulista, 1578 - Bela Vista, São Paulo - SP"));
        lista.add(new Aluno(id++, nome: "Felipe", datel, endereco: "Praça Roberto Gomes Pedrosa, 1 - Morumbi, São Paulo - SP"));
        lista.add(new Aluno(id++, nome: "Thales", date3, endereco: "Av. Miguel Ignácio Curi, 111 - Artur Alvim, São Paulo - SP"));
    }

    public List<Aluno> getLista() { return lista; }

    public void remover(Aluno Aluno) {
        lista.remove(Aluno);
        Log.d( tag: "Deletar: ", lista.toString());
    }

    public void salvar(Aluno obj) {
        if(obj.getId() == null) {
            obj.setId(id++);
            lista.add(obj);
        } else {
            lista.set(lista.indexOf(obj), obj);
        }
        Log.d( tag: "Salvar: ", lista.toString());
    }

    public Aluno localizar(Long id) { return lista.get(lista.indexOf(new Aluno(id))); }
}

```

Figura 39 - AlunoDao.java

Copiável:

```

public class AlunoDao {

    public static AlunoDao manager = new AlunoDao();
    // Lista aonde serão armazenados os alunos
    private List<Aluno> lista;
    private long id = 1;

    private AlunoDao() {
        Date datel = null, date2 = null, date3 = null;
        try {
            SimpleDateFormat dateformat3 = new
SimpleDateFormat("dd/MM/yyyy");
            datel = dateformat3.parse("17/07/1989");
            date2 = dateformat3.parse("18/03/1993");
            date3 = dateformat3.parse("01/11/1997");
        } catch (ParseException e) {
            e.printStackTrace();
        }

        lista = new ArrayList<>();
        lista.add(new Aluno(id++, "Helena Strada", date2, "Av.
Paulista, 1578 - Bela Vista, São Paulo - SP"));
    }
}

```

```

        lista.add(new Aluno(id++, "Felipe", date1, "Praça Roberto
Gomes Pedrosa, 1 - Morumbi, São Paulo - SP"));
        lista.add(new Aluno(id++, "Thales", date3, "Av. Miguel Ignácio
Curi, 111 - Artur Alvim, São Paulo - SP"));
    }

    public List<Aluno> getList() {
        return lista;
    }

    public void remover(Aluno Aluno) {
        lista.remove(Aluno);
        Log.d("Deletar: ", lista.toString());
    }

    public void salvar(Aluno obj) {
        if(obj.getId() == null) {
            obj.setId(id++);
            lista.add(obj);
        } else {
            lista.set(lista.indexOf(obj), obj);
        }
        Log.d("Salvar: ", lista.toString());
    }

    public Aluno localizar(Long id) {
        return lista.get(lista.indexOf(new Aluno(id)));
    }
}

```

Uma vez que deixamos a nossa estrutura pronta para realizar o crud de alunos que desejamos (ainda utilizando lista), não queremos mais carregar a lista de Strings em nosso adapter, agora queremos carregar uma lista de alunos que até então estão sendo inseridas na mão.

Para isto, além de trocarmos a lista que está sendo carregada em nossa MainActivity, vamos alterar também o tipo de visualização. Vamos criar uma visualização personalizada para os nossos itens. Não queremos mais utilizar a forma de visualização padrão que o Android disponibiliza para nós.

Personalizando nossa ListView

O que nós queremos fazer é personalizar a nossa lista. Ao invés de utilizarmos a lista que o Android não deixa nós manipularmos, vamos criar a nossa.

Vamos criar um novo pacote chamado adapter.

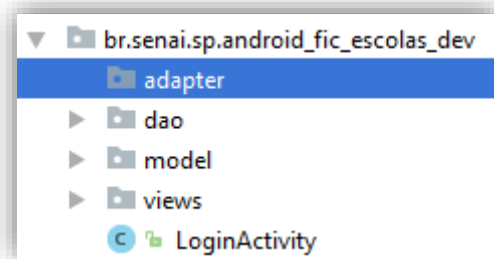


Figura 40 - Criando o pacote adapter.

Faltam três coisas para personalizarmos a nossa lista. Uma delas será criar o xml personalizado. Lembrando que: quando nos referenciamos a tela, trabalhamos com o xml.

A segunda etapa é criar em nosso Java, a lista personalizada para buscar os elementos da lista que desejamos e carregar os dados para nosso adapter personalizado. Não queremos mais somente a nossa lista de Strings.

E a terceira etapa é mudar em nossa MainActivity, para dizer que queremos utilizar nossa lista personalizada e nosso adapter, ao invés do Array de Strings.

Criando o layout personalizado

Dentro da pasta de layout, vamos criar um novo arquivo do tipo *layout resource file* com o nome de *item_simples_aluno_lista*.

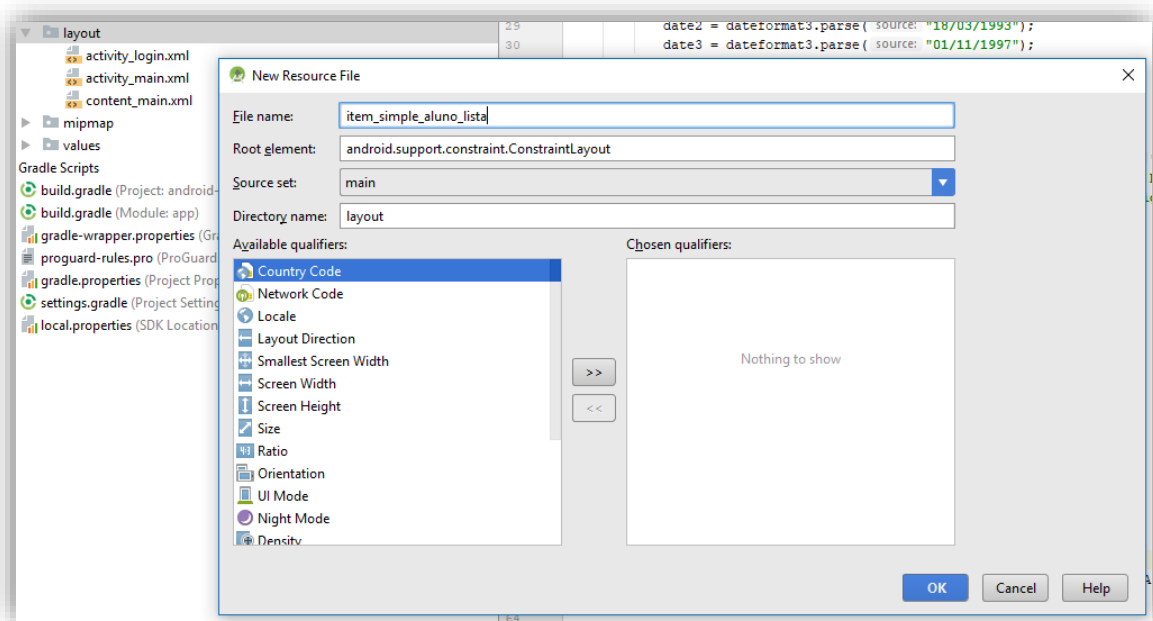


Figura 41 - Criando o layout do adapter.

Clique em 'Ok'.

Vamos criar a nossa lista personalizada que irá conter inicialmente o nome do aluno e a data de nascimento dele. Lembre-se sempre de fazer a referência com o id para buscarmos esse elemento em nosso Java.



Figura 42 - Criando o item da lista personalizado.

Copiável:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="?android:selectableItemBackground"
    android:clickable="true"
    android:focusable="true"
    android:foreground="?android:attr/selectableItemBackground"
    android:orientation="vertical"
    android:padding="8dp">

    <TextView
```

```

        android:id="@+id/tvNomeAluno"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Nome do Aluno"
        android:textSize="20dp"
        android:textAlign="center" />

<TextView
    android:textAlignment="center"
    android:textColor="#cccccc"
    android:id="@+id/tvDataNascimentoAluno"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Data de Nascimento"
    android:textSize="12dp" />

</LinearLayout>

```

Populando o adapter com os dados do aluno

Uma vez que definimos o nosso layout, vamos criar o nosso adapter personalizado para popular o layout com os dados que nós desejamos. Crie um arquivo chamado AlunoAdapter dentro do pacote adapter que criamos.

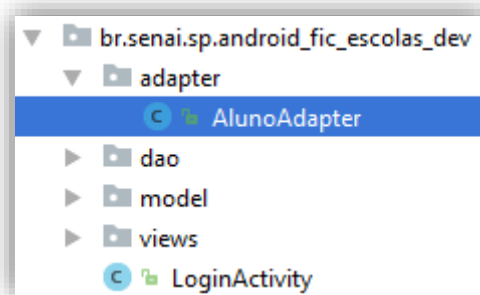


Figura 43 - Criando o AlunoAdapter.java.

Para criarmos um layout personalizado, nós precisamos estender a classe BaseAdapter.

```

public class AlunoAdapter extends BaseAdapter {

    @Override
    public int getCount() {
        return 0;
    }

    @Override
    public Object getItem(int i) {
        return null;
    }

    @Override
    public long getItemId(int i) {
        return 0;
    }

    @Override
    public View getView(int i, View view, ViewGroup viewGroup) {
        return null;
    }

}

```

Figura 44 - Estendendo a classe BaseAdapter.

Copiável:

```

public class AlunoAdapter extends BaseAdapter {

    @Override
    public int getCount() {
        return 0;
    }

    @Override
    public Object getItem(int i) {
        return null;
    }

    @Override
    public long getItemId(int i) {
        return 0;
    }

    @Override
    public View getView(int i, View view, ViewGroup viewGroup) {
        return null;
    }

}

```

Agora que criamos a nossa classe, vamos implementá-la com os dados que desejamos. Ou seja, popular o nosso adapter com a nossa lista de alunos.

```

public class AlunoAdapter extends BaseAdapter {

    private final List<Aluno> alunosAdapter;
    private final Activity activity;

    public AlunoAdapter(List<Aluno> alunosAdapter, Activity activity) {
        this.alunosAdapter = alunosAdapter;
        this.activity = activity;
    }

    @Override
    public int getCount() { return alunosAdapter.size(); }

    @Override
    public Object getItem(int i) { return alunosAdapter.get(i); }

    @Override
    public long getItemId(int i) { return 0; }

    @Override
    public View getView(int i, View view, ViewGroup viewGroup) {

        LinearLayout layout;

        if(view == null) {
            Context ctx = viewGroup.getContext();
            LayoutInflater svc = (LayoutInflater)ctx.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            layout = new LinearLayout(ctx);
            svc.inflate(R.layout.item_simple_aluno_lista, layout);
        } else {
            layout = (LinearLayout)view;
        }

        Aluno aluno = alunosAdapter.get(i);

        // buscando as referências das views
        TextView nomeAluno = layout.findViewById(R.id.tvNomeAluno);
        TextView dataNascimentoAluno = layout.findViewById(R.id.tvDataNascimentoAluno);

        // "populando" as views
        nomeAluno.setText(aluno.getNome());
        // formatando as datas
        dataNascimentoAluno.setText(DateFormat.getDateInstance(DateFormat.MEDIUM).format(aluno.getDataNascimento()));

        return layout;
    }
}

```

Figura 45 - AlunoAdapter com os métodos implementados.

Copiável:

```

public class AlunoAdapter extends BaseAdapter {

    private final List<Aluno> alunosAdapter;
    private final Activity activity;

    public AlunoAdapter(List<Aluno> alunosAdapter, Activity activity)
    {
        this.alunosAdapter = alunosAdapter;
        this.activity = activity;
    }

    @Override
    public int getCount() {
        return alunosAdapter.size();
    }

    @Override
    public Object getItem(int i) {

```

```

        return alunosAdapter.get(i);
    }

    @Override
    public long getItemId(int i) {
        return 0;
    }

    @Override
    public View getView(int i, View view, ViewGroup viewGroup) {

        LinearLayout layout;

        if(view == null) {
            Context ctx = viewGroup.getContext();
            LayoutInflater svc =
                (LayoutInflater) ctx.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            layout = new LinearLayout(ctx);
            svc.inflate(R.layout.item_simple_aluno_lista, layout);
        } else {
            layout = (LinearLayout) view;
        }

        Aluno aluno = alunosAdapter.get(i);

        // buscando as referências das views
        TextView nomeAluno = layout.findViewById(R.id.tvNomeAluno);
        TextView dataNascimentoAluno =
            layout.findViewById(R.id.tvDataNascimentoAluno);

        // "populando" as views
        nomeAluno.setText(aluno.getNome());
        // formatando as datas
        dataNascimentoAluno.setText(DateFormat.getDateInstance(DateFormat.MEDIUM).format(aluno.getDataNascimento()));

        return layout;
    }
}

```

Uma vez que definimos o layout que desejamos para o nosso adapter e implementamos os métodos em nosso adapter, fazendo referência ao layout que criamos, precisamos apenas alterar a nossa MainActivity. Ao invés de carregar a lista de strings com o adapter que o Android disponibiliza, iremos carregar a nossa lista personalizada com a lista de alunos.

```

public class MainActivity extends AppCompatActivity {

    private ListView lvListaAlunos;
    private AlunoDao dao = AlunoDao.manager;
    private List<Aluno> listagemDeAlunos;
    private AlunoAdapter alunoAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        // getSupportActionBar().setTitle("Lista de Alunos");

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
            }
        });

        // fazendo referência das views
        lvListaAlunos = findViewById(R.id.lvListaAlunos);

        // String[] listagemDeAlunos = new String[]{"Helena", "Thales", "Felipe"};
        // ArrayAdapter<String> adapter = new ArrayAdapter<>(getApplicationContext(), android.R.layout.simple_list_item_1, listagemDeAlunos);
        // lvListaAlunos.setAdapter(adapter);

        carregarListaDeAlunos();
    }

    private void carregarListaDeAlunos() {
        listagemDeAlunos = dao.getList();
        alunoAdapter = new AlunoAdapter(listagemDeAlunos, activity: this);
        lvListaAlunos.setAdapter(alunoAdapter);
    }
}

```

Figura 46 - MainActivity carregando a lista personalizada.

Copiável:

```

public class MainActivity extends AppCompatActivity {

    private ListView lvListaAlunos;
    private AlunoDao dao = AlunoDao.manager;
    private List<Aluno> listagemDeAlunos;
    private AlunoAdapter alunoAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        // getSupportActionBar().setTitle("Lista de Alunos");

        FloatingActionButton fab = (FloatingActionButton)
        findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action",
                Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
            }
        });

        // fazendo referência das views

```

```

        lvListaAlunos = findViewById(R.id.lvListaAlunos);

//        String[] listagemDeAlunos = new String[]{"Helena", "Thales",
"Felipe"};
//        ArrayAdapter<String> adapter = new
ArrayAdapter<>(getApplicationContext(),
android.R.layout.simple_list_item_1, listagemDeAlunos);
//        lvListaAlunos.setAdapter(adapter);

        carregarListaDeAlunos();

    }

    private void carregarListaDeAlunos() {
        listagemDeAlunos = dao.getLista();
        alunoAdapter = new AlunoAdapter(listagemDeAlunos, this);
        lvListaAlunos.setAdapter(alunoAdapter);
    }
}

```

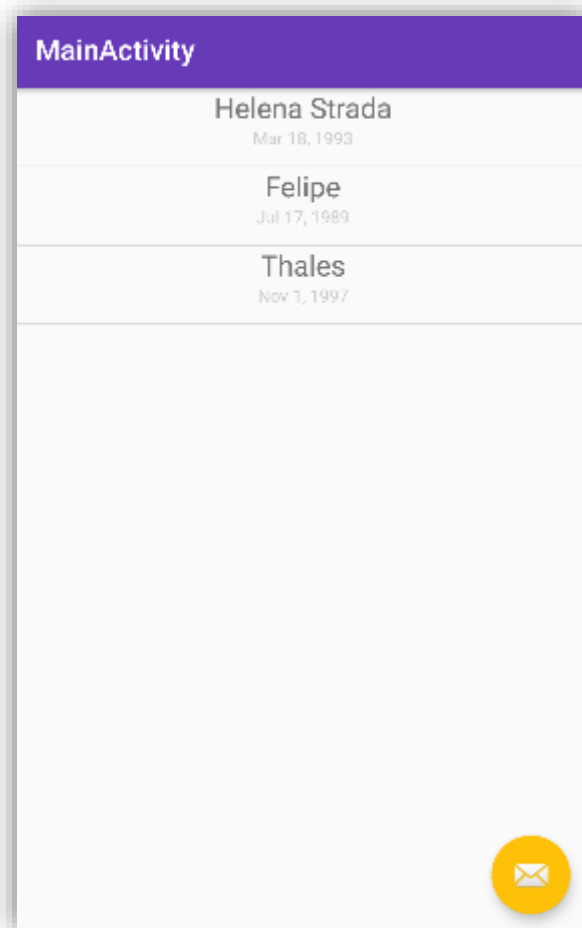


Figura 47 - Lista Personalizada carregada.

Arquivos

Resumo

Nesta segunda etapa do projeto, nós aprendemos a como criar uma lista personalizada através de alguns registros fixos em nossa lista. Na próxima etapa, vamos aprender a como incluir novos registros e uma que o registro está em nossa lista, como nós fazemos para alterá-lo.

Terceira Etapa

O que nós vimos até agora foi a criar a tela de login para o usuário e listar os alunos cadastrados através da lista. O que nós precisamos fazer agora é criar a nossa tela de cadastro de um novo aluno.

Criando a activity de formulário

Primeiramente iremos criar a nossa activity de formulário. Dentro da pasta views, vamos adicionar uma nova activity com o estilo *Empty Activity*.

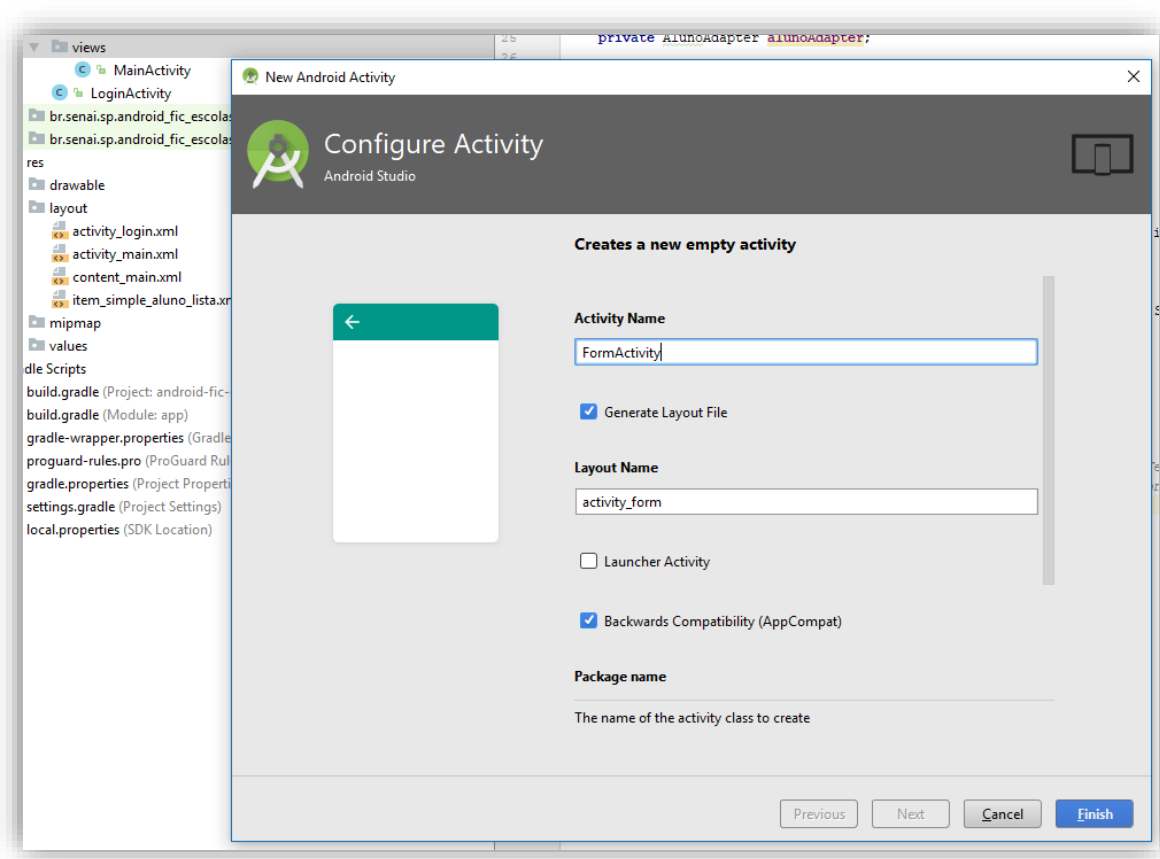


Figura 48 - Criando a tela de formulário para o aluno.

Clique em 'Finish'.

Uma vez criada essa tela, vamos realizar dois passos inicialmente. A primeira será redirecionar, ao clicar no botão de adicionar um novo aluno, para a tela de formulário. A segunda será criar o layout para essa tela.

Se olharmos a nossa MainActivity, temos um botão lateral. Ao clicarmos nesse botão, iremos abrir a nossa nova tela. Mas antes disso, vamos alterar o estilo dele.

Alterando o ícone do botão

Clique com o botão direito em drawable -> New -> Vector Asset para criar um novo recurso que iremos utilizar no nosso botão.

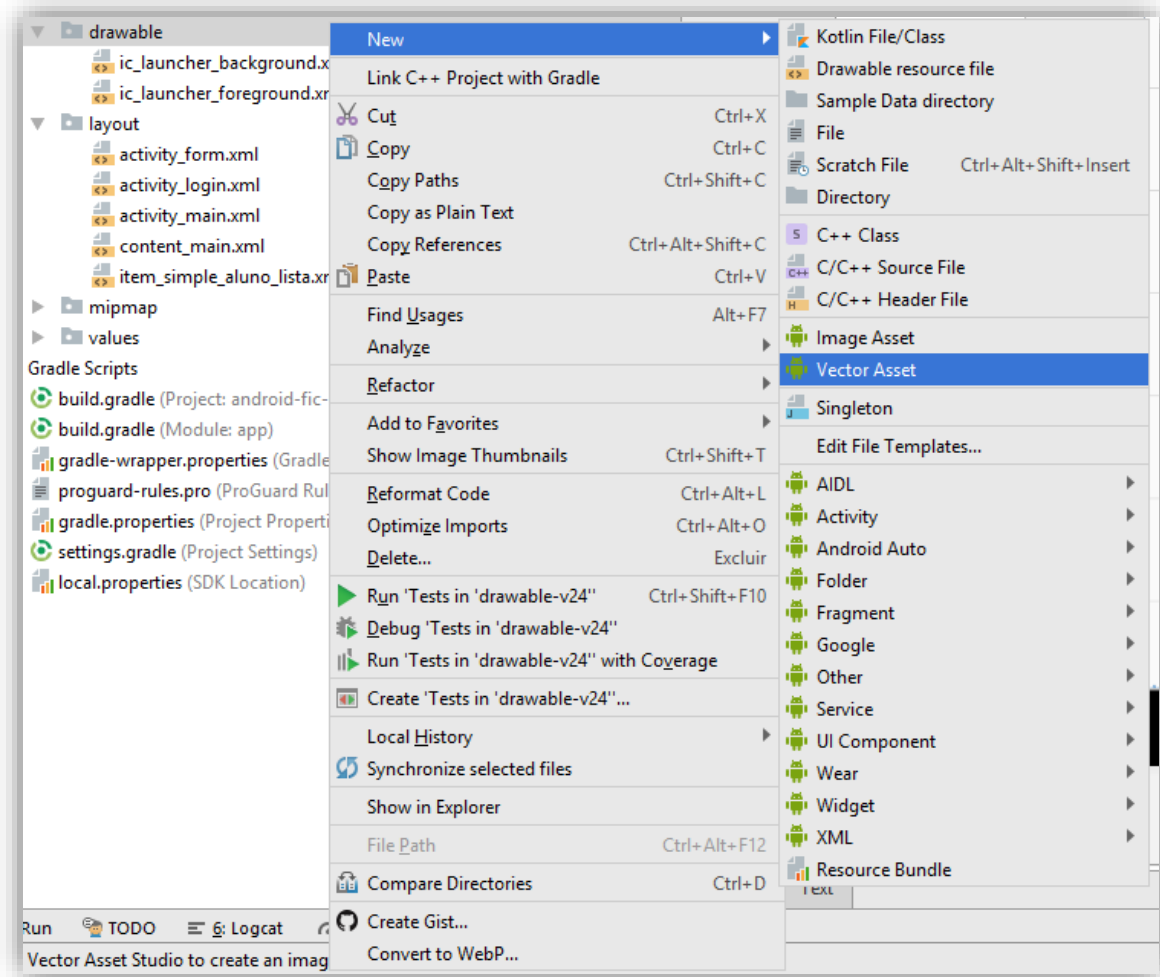


Figura 49 - Criando um novo recurso.



Clique no ícone para abrir e selecionar um novo item.

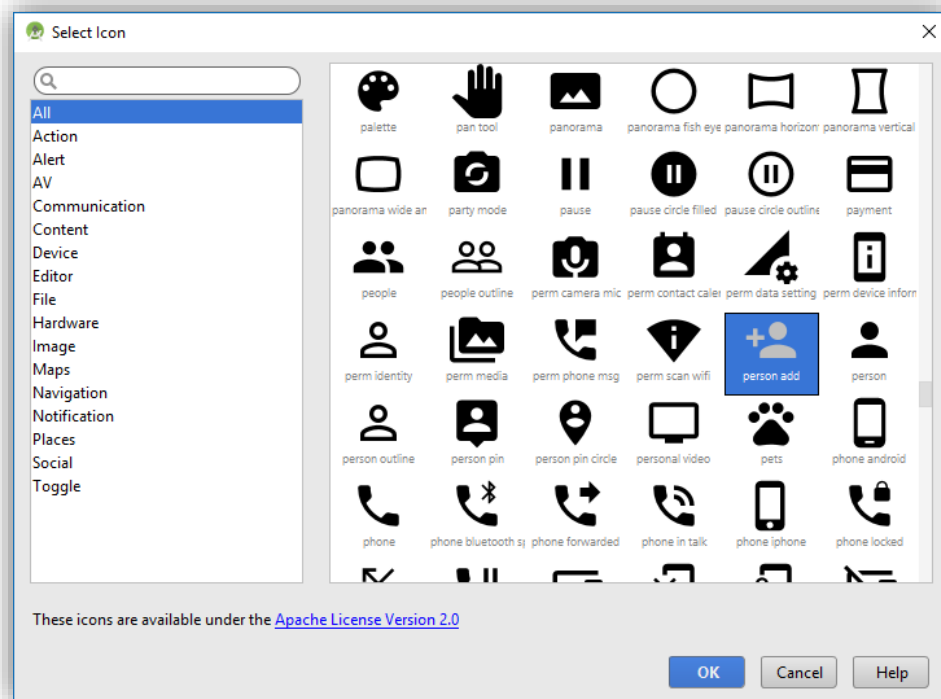


Figura 50 - Selecionando o ícone.

Clique em 'OK'.

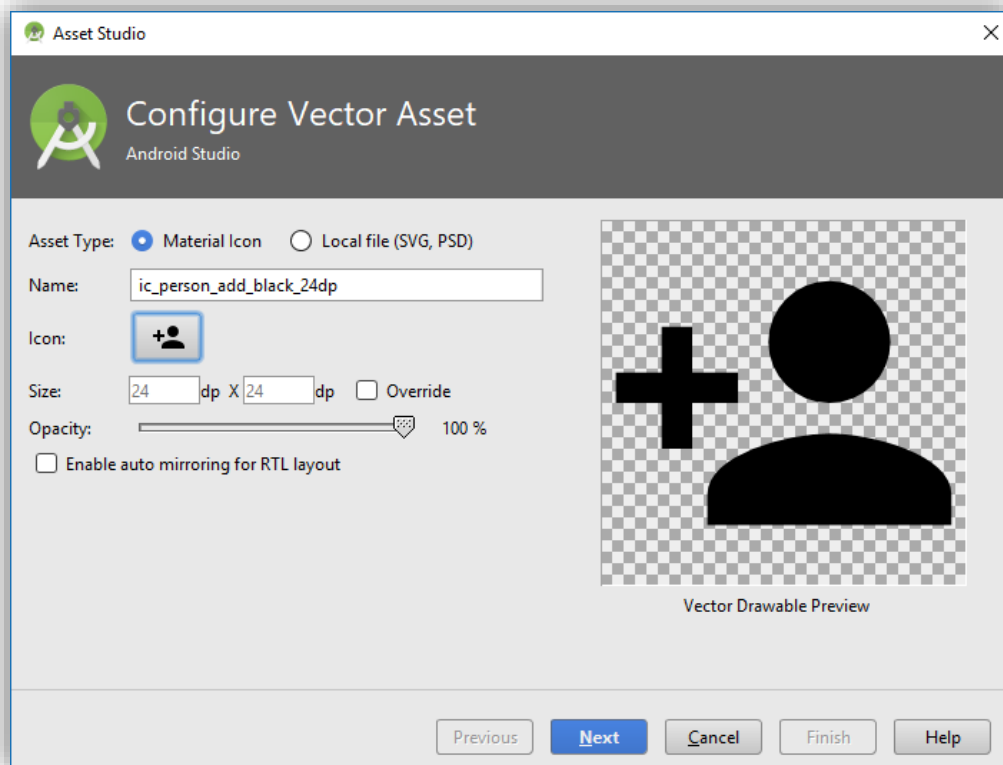


Figura 51 - Após selecionar o item desejado, finalize.

Clique em 'Next'.

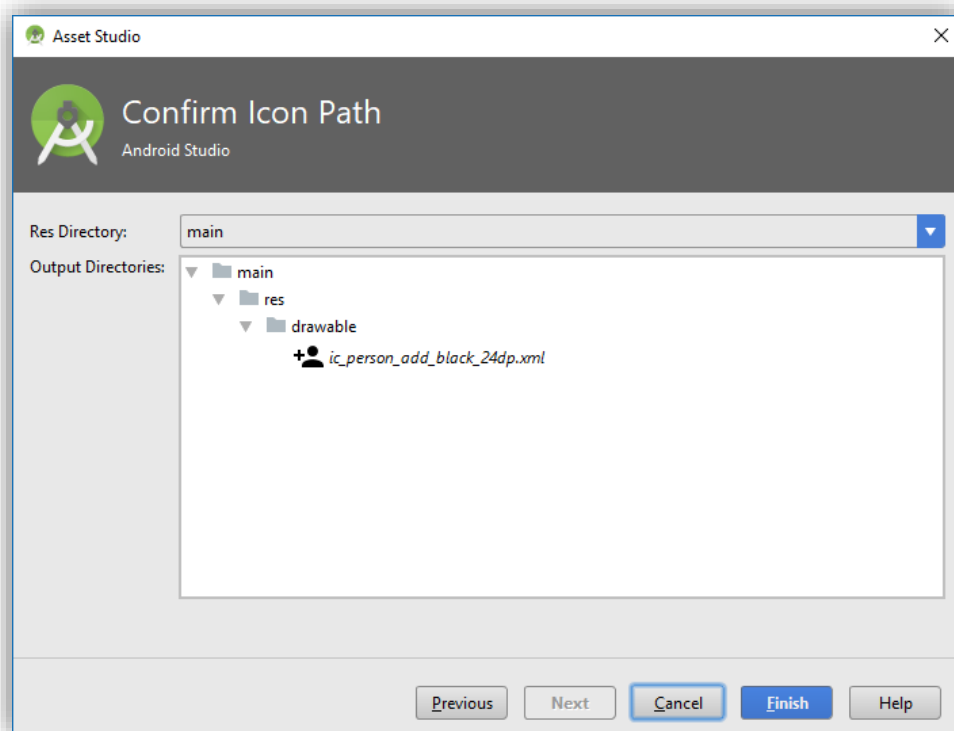


Figura 52 - O nome do arquivo pode ser alterado.

Clique em 'Finish'.

Após criar um novo ícone, basta alterá-lo na activity_main.xml.

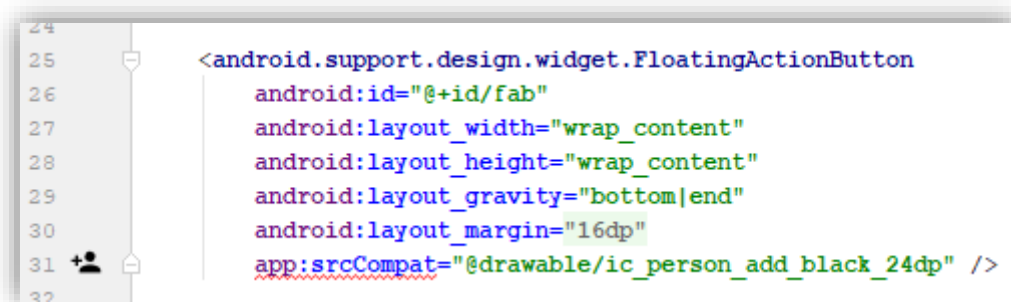


Figura 53 - Ícone do FloatingActionButton alterado.

Copiável:

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    android:layout_margin="@dimen/fab_margin"
    app:srcCompat="@drawable/ic_person_add_black_24dp" />
```

Falta uma pequena configuração. Para utilizarmos este recurso, precisamos colocar em nosso build.gradle, nível projeto, a seguinte configuração.

```
vectorDrawables.useSupportLibrary = true
```

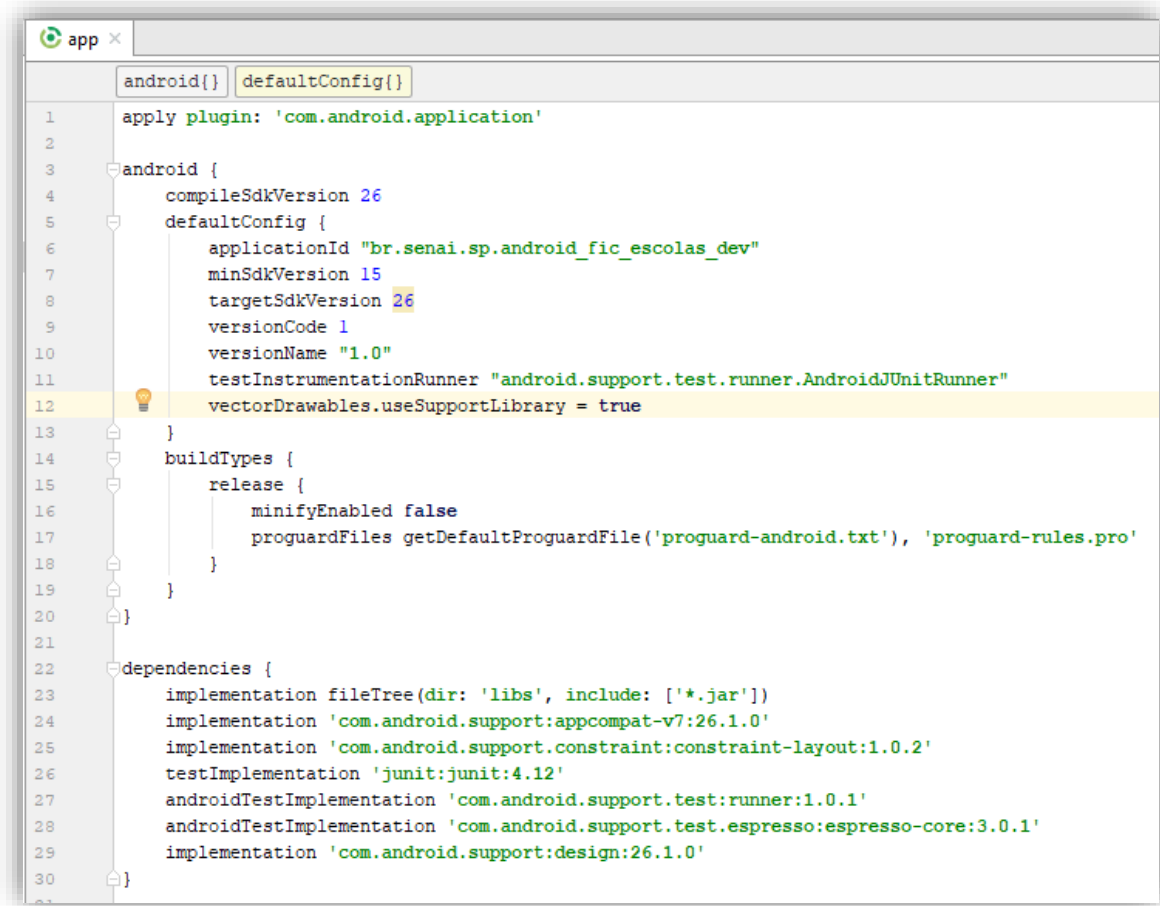


Figura 54 - Adicionando o suporte a biblioteca no build.gradle.

Criando uma ação para o floating button

Pronto, acabamos de estilizar o nosso botão. Vamos agora criar a nossa ação para irmos para uma nova tela.

```

public class MainActivity extends AppCompatActivity {

    private ListView lvListaAlunos;
    private AlunoDao dao = AlunoDao.manager;
    private List<Aluno> listagemDeAlunos;
    private AlunoAdapter alunoAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        // getSupportActionBar().setTitle("Lista de Alunos");

        FloatingActionButton fab = findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intentDeFormulario = new Intent( packageContext: MainActivity.this, FormActivity.class);
                startActivity(intentDeFormulario);
            }
        });

        // fazendo referência das views
        lvListaAlunos = findViewById(R.id.lvListaAlunos);

        // String[] listagemDeAlunos = new String[]{"Helena", "Thales", "Felipe"};
        // ArrayAdapter<String> adapter = new ArrayAdapter<>(getApplicationContext(), android.R.layout.simple_list_item_1, listagemDeAlunos);
        // lvListaAlunos.setAdapter(adapter);

        carregarListaDeAlunos();
    }

    private void carregarListaDeAlunos() {
        listagemDeAlunos = dao.getList();
        alunoAdapter = new AlunoAdapter(listagemDeAlunos, activity: this);
        lvListaAlunos.setAdapter(alunoAdapter);
    }
}

```

Figura 55 - Abrindo a tela de formulário através do floating button.

Copiável:

```

        FloatingActionButton fab = findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intentDeFormulario = new Intent(MainActivity.this,
                FormActivity.class);
                startActivity(intentDeFormulario);
            }
        });

```

Vamos alterar a FormActivity.java para alterar o título da nossa página.

```
public class FormActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_form);  
        getSupportActionBar().setTitle("Formulário");  
    }  
}
```

Figura 56 - Alterando o título da página do Formulário.

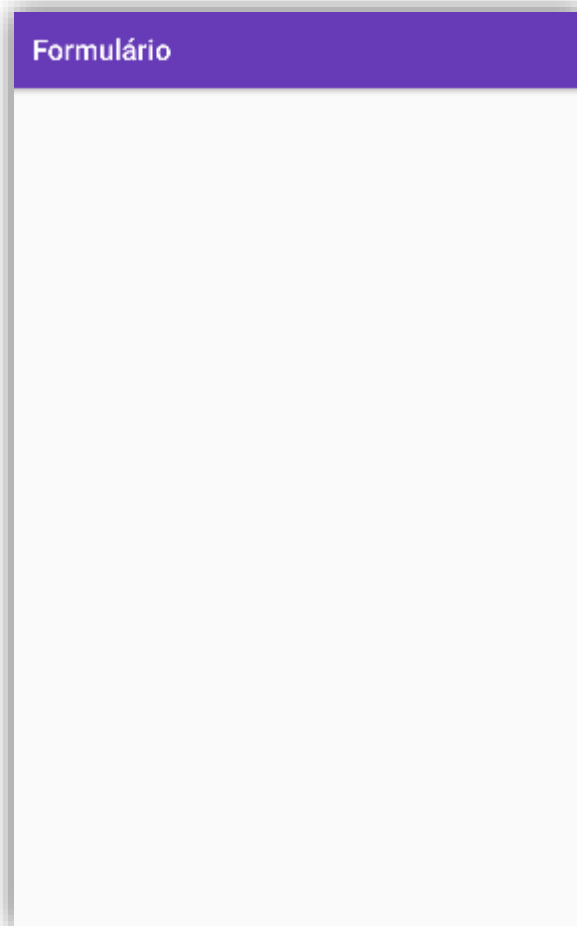


Figura 57 - Abrindo a tela de formulário.

Vamos criar o nosso layout para essa tela.

Copiável:

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
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:layout_width="match_parent"
android:layout_height="match_parent"

tools:context="br.senai.sp.android_fic_escolas_dev.views.FormActivity"
>

<ImageView
    android:layout_margin="8dp"
    android:id="@+id/ivFotoAluno"
    android:layout_width="100dp"
    android:layout_height="100dp"
    app:srcCompat="@mipmap/ic_launcher_round"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<android.support.design.widget.TextInputLayout
    android:id="@+id/tilNomeAluno"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:padding="4dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/ivFotoAluno">

    <EditText
        android:id="@+id/etNomeAluno"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Nome do Aluno" />

</android.support.design.widget.TextInputLayout>

<android.support.design.widget.TextInputLayout
    android:id="@+id/tilDataNascimentoAluno"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:padding="4dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tilNomeAluno">

    <android.support.design.widget.TextInputEditText
        android:id="@+id/etDataNascimentoAluno"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:focusableInTouchMode="false"
        android:hint="Data de Nascimento" />

</android.support.design.widget.TextInputLayout>

<android.support.design.widget.TextInputLayout
    android:id="@+id/tilEnderecoAluno"
    android:layout_width="0dp"
    android:padding="4dp"
    android:layout_height="wrap_content"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"

```



```

app:layout_constraintTop_toBottomOf="@+id/tilDataNascimentoAluno">

    <android.support.design.widget.TextInputEditText
        android:id="@+id/etEnderecoAluno"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Endereço" />
</android.support.design.widget.TextInputLayout>

<Button
    android:id="@+id/btnSalvarAluno"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="4dp"
    android:text="Salvar"
    android:background="@color/colorAccent"
    app:layout_constraintEnd_toEndOf="@+id/tilEnderecoAluno"
    app:layout_constraintTop_toBottomOf="@+id/tilEnderecoAluno" />

</android.support.constraint.ConstraintLayout>

```

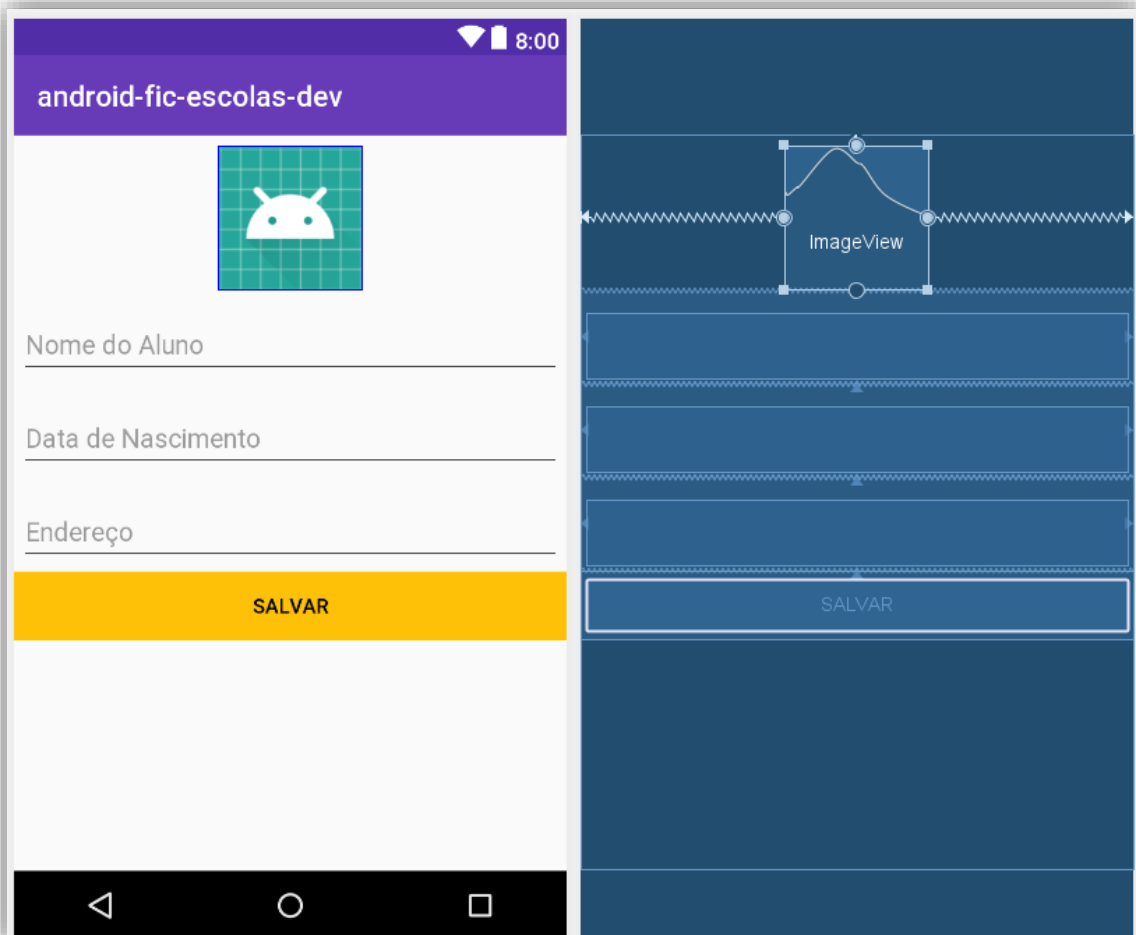


Figura 58 - Criando o layout para o formulário.

O que precisamos fazer agora é: fazer as referências das nossas views e ao clicarmos no botão de salvar, um novo item ser carregado em nossa lista com os dados que acabamos de incluir.

Abrindo o calendário e a câmera

Porém, precisamos adicionar alguns detalhes. Ao clicarmos na imagem, devemos abrir a câmera para tirar uma foto e ao clicarmos na data de nascimento, deverá aparecer um diálogo de data para selecionarmos a data que desejamos.

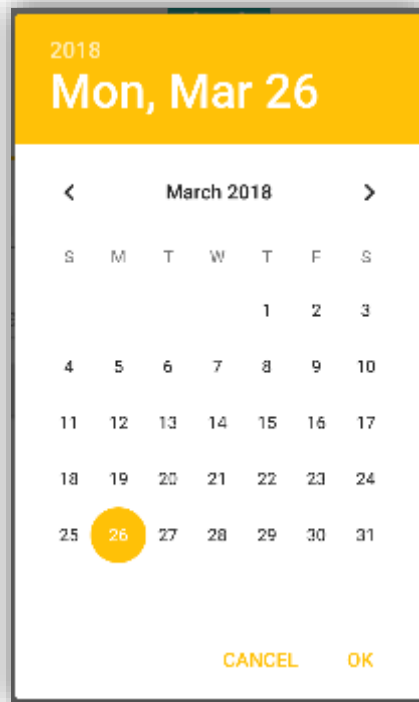


Figura 59 - Diálogo de data.

FormActivity com os dois códigos para abrir a câmera e abrir o diálogo adicionados. Além disso, ao carregar a imagem da câmera, colocar a imagem no ImageView uma vez que a foto foi tirada.

```
public class FormActivity extends AppCompatActivity {  
  
    private TextInputLayout tilNomeAluno;  
    private TextInputLayout tilDataNascimentoAluno;  
    private TextInputLayout tilEnderecoAluno;  
    private EditText etDataNascimentoAluno;  
    private Button btnSalvarAluno;  
    private ImageView ivFotoAluno;  
  
    // dados do calendário para montar a data  
    private int ano, mes, dia;  
}
```

```

// recursos para salvar a imagem
private byte[] b;
private String temp;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_form);
    getSupportActionBar().setTitle("Formulário");

    // Referências das views
    tilNomeAluno = findViewById(R.id.tilNomeAluno);
    tilDataNascimentoAluno =
findViewById(R.id.tilDataNascimentoAluno);
    tilEnderecoAluno = findViewById(R.id.tilEnderecoAluno);
    etDataNascimentoAluno =
findViewById(R.id.etDataNascimentoAluno);
    btnSalvarAluno = findViewById(R.id.btnSalvarAluno);
    ivFotoAluno = findViewById(R.id.ivFotoAluno);

    etDataNascimentoAluno.setOnClickListener(new
abrirCalendario());
    ivFotoAluno.setOnClickListener(new abrirCamera());
}

// abrir um calendário
private class abrirCalendario implements View.OnClickListener {
    @Override
    public void onClick(View view) {
        // calendário
        final Calendar calendar = Calendar.getInstance();
        ano = calendar.get(Calendar.YEAR);
        mes = calendar.get(Calendar.MONTH);
        dia = calendar.get(Calendar.DAY_OF_MONTH);

        DatePickerDialog datePickerDialog = new
DatePickerDialog(FormActivity.this,
            new DatePickerDialog.OnDateSetListener() {
                @Override
                public void onDateSet(DatePicker view, int
year,
int monthOfYear, int
dayOfMonth) {
                    etDataNascimentoAluno.setText((monthOfYear
+ 1) + "/" + dayOfMonth + "/" + year);
                }, ano, mes, dia);
        datePickerDialog.show();
    }
}

// abrir a camera
private class abrirCamera implements View.OnClickListener {
    @Override
    public void onClick(View view) {
        abrirIntentDeCamera();
    }
}

```

```

        static final int REQUEST_IMAGE_CAPTURE = 1;
        private void abrirIntentDeCamera() {
            Intent takePictureIntent = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);
            // starto uma nova activity para abrir a camera
            if (takePictureIntent.resolveActivity(getPackageManager()) !=
null) {
                startActivityForResult(takePictureIntent,
REQUEST_IMAGE_CAPTURE);
            }
        }

        @Override
        protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
            if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode ==
RESULT_OK) {
                // ao fechar a camera, eu preciso verificar se o status
foi ok e além disso, pegar a imagem que foi tirada
                Bundle extras = data.getExtras();
                Bitmap imageBitmap = (Bitmap) extras.get("data");
                // com a imagem, eu vou mostrar a imagem na nossa
imageview
                ivFotoAluno.setImageBitmap(imageBitmap);
                // uma vez que eu tenho essa imagem, preciso salvá-la
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                imageBitmap.compress(Bitmap.CompressFormat.PNG, 100,
baos);

                b = baos.toByteArray();
                temp = Base64.encodeToString(b, Base64.DEFAULT);
            }
        }
    }
}

```

Porém, no Android, para permitir que o aplicativo tenha acesso a câmera do usuário, é necessário solicitar essa permissão.

Solicitando permissão de acesso da câmera

Vamos abrir o AndroidManifest e solicitar as seguintes permissões para o usuário.

```

<!-- Salvar o arquivo da câmera -->
<uses-feature
    android:name="android.hardware.camera"
    android:required="true" />

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

```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.senai.sp.android_fic_escolas_dev">

    <!-- Salvar o arquivo da câmera -->
    <uses-feature
        android:name="android.hardware.camera"
        android:required="true" />

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="android-fic-escolas-dev"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".LoginActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".views.MainActivity"
            android:label="MainActivity"
            android:theme="@style/AppTheme.NoActionBar" />
        <activity android:name=".views.FormActivity"></activity>
    </application>
</manifest>
```

Figura 60 - AndroidManifest com permissão para acessar a câmera.

Uma vez que definimos as permissões, a primeira vez que formos executar o projeto, ele solicitará permissão para acessar os recursos que estamos solicitando.

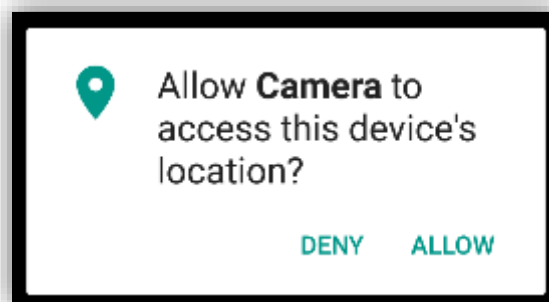


Figura 61 - Solicitando permissão de acesso para acessar a câmera.

Figura 62 - Diálogo de data e câmera para a tela de formulário.

Uma vez que conseguimos selecionar a imagem e a data, queremos completar o nome e o endereço e salvá-lo em nossa lista.

Salvando os dados na lista

Uma vez que preenchemos todos os dados de cadastro, queremos salvar os dados em nossa lista.

```
private AlunoDao dao = AlunoDao.manager;
```

```
btnSalvarAluno.setOnClickListener(new salvarDadosDoAluno());
```

```
private class salvarDadosDoAluno implements View.OnClickListener {
    @Override
    public void onClick(View view) {
        Aluno alunoNovo = new Aluno();

        alunoNovo.setNome(tilNomeAluno.getText().toString());
        alunoNovo.setDataNascimento(new
        Date(tilDataNascimentoAluno.getText().toString()));

        alunoNovo.setEndereco(tilEnderecoAluno.getText().toString());

        alunoNovo.setFotoAluno(temp);
        dao.salvar(alunoNovo);
        Log.d("NovoAluno: ", dao.getList().toString());
    }
}
```

```
}
}
```

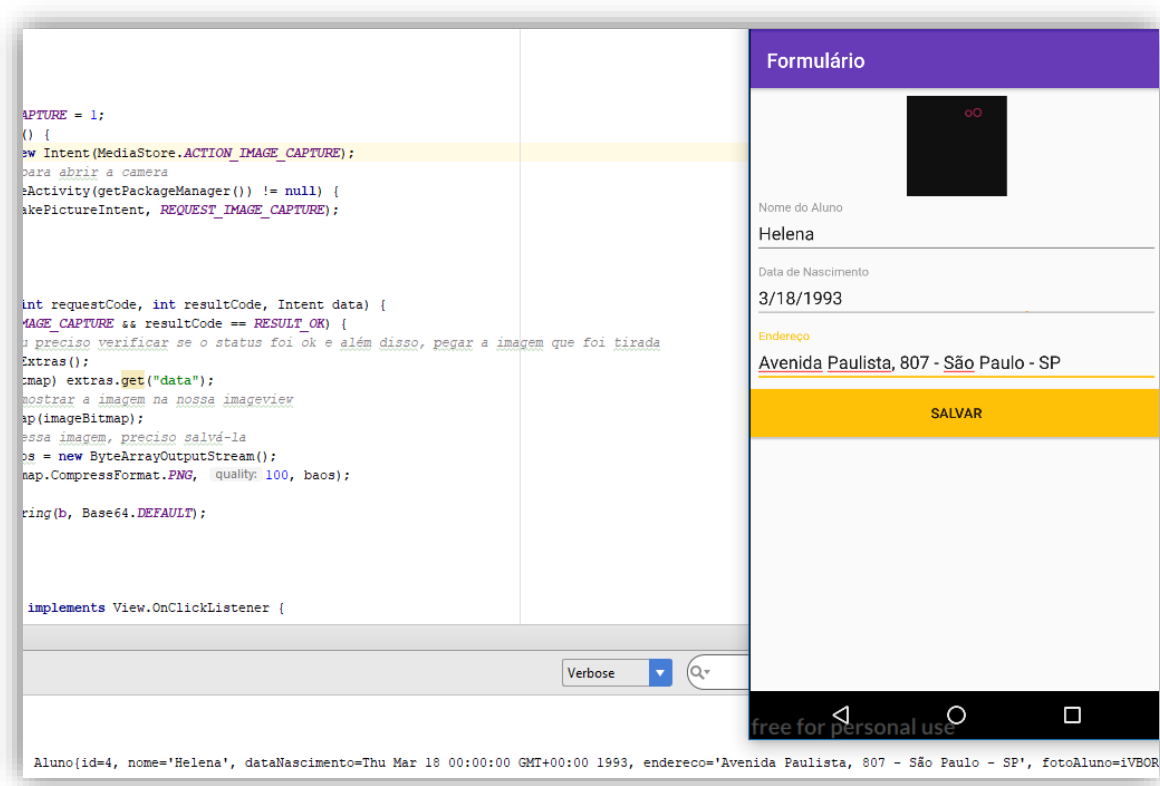


Figura 63 - Mostrando o usuário cadastrado no Log.

O que precisamos fazer agora, é retornar para a tela anterior, recarregar a nossa lista de alunos para mostrar o novo aluno cadastrado e seria interessante mostrar uma mensagem para o usuário informando que o mesmo foi cadastrado com sucesso.

Retornando para a Home

Vamos criar um método para o usuário ser redirecionado para home e recarregar a lista de alunos.

Algo que faremos diferente agora, será a mensagem. Vamos alterar também a nossa MainActivity. Teremos um método a mais. Uma vez que eu precisamos informar para o usuário alguma ação após a ação de salvar.

```

private class salvarDadosDoAluno implements View.OnClickListener {
    @Override
    public void onClick(View view) {
        Aluno alunoNovo = new Aluno();
        alunoNovo.setNome(tilNomeAluno.getText().getText().toString());
        alunoNovo.setDataNascimento(new Date(tilDataNascimentoAluno.getText().getText().toString()));
        alunoNovo.setEndereco(tilEnderecoAluno.getText().getText().toString());

        alunoNovo.setFotoAluno(temp);
        dao.salvar(alunoNovo);
        Log.d( tag: "NovoAluno: ", dao.getLista().toString());

        retornarParaTelaAnteriorAposSalvar();
    }
}

private void retornarParaTelaAnteriorAposSalvar() {
    Intent retornarParaActivityHome = new Intent();
    retornarParaActivityHome.putExtra( name: "result", value: 1);
    setResult(Activity.RESULT_OK, retornarParaActivityHome);
    finish();
}

```

Figura 64 - Adicionando a activity de retorno após cadastro.

Copiável:

```

private class salvarDadosDoAluno implements
View.OnClickListener {
    @Override
    public void onClick(View view) {
        Aluno alunoNovo = new Aluno();

        alunoNovo.setNome(tilNomeAluno.getText().getText().toString());
        alunoNovo.setDataNascimento(new
Date(tilDataNascimentoAluno.getText().getText().toString()));

        alunoNovo.setEndereco(tilEnderecoAluno.getText().getText().toStrin
g());

        alunoNovo.setFotoAluno(temp);
        dao.salvar(alunoNovo);
        Log.d("NovoAluno: ", dao.getLista().toString());

        retornarParaTelaAnteriorAposSalvar();
    }
}

private void retornarParaTelaAnteriorAposSalvar() {
    Intent retornarParaActivityHome = new Intent();
    retornarParaActivityHome.putExtra("result", 1);
    setResult(Activity.RESULT_OK, retornarParaActivityHome);
    finish();
}

```

Na nossa MainActivity.java.


```

FloatingActionButton fab = findViewById(R.id.fab);
fab.setOnClickListener((view) -> {
    Intent intentDeFormulario = new Intent( packageContext: MainActivity.this, FormActivity.class);
    startActivityForResult(intentDeFormulario, requestCode: 1);
});

```

Figura 65 - Adicionando um resultado a nossa activity.

```

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == 1) {
        if (resultCode == RESULT_OK) {
            carregarListaDeAlunos();
        }
    }
}

```

Figura 66 - Recarregando a lista de alunos após cadastro.

Copiável:

```

@Override
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
    if (requestCode == 1) {
        if (resultCode == RESULT_OK) {
            carregarListaDeAlunos();
        }
    }
}

```

Uma vez que adicionarmos um novo aluno, estaremos adicionando um código de retorno para a nossa MainActivity. O que ela faz? Verifica o status que colocamos. Caso a requisição seja ok, seja o mesmo valor que eu coloquei na "ida", ele recarrega a lista.

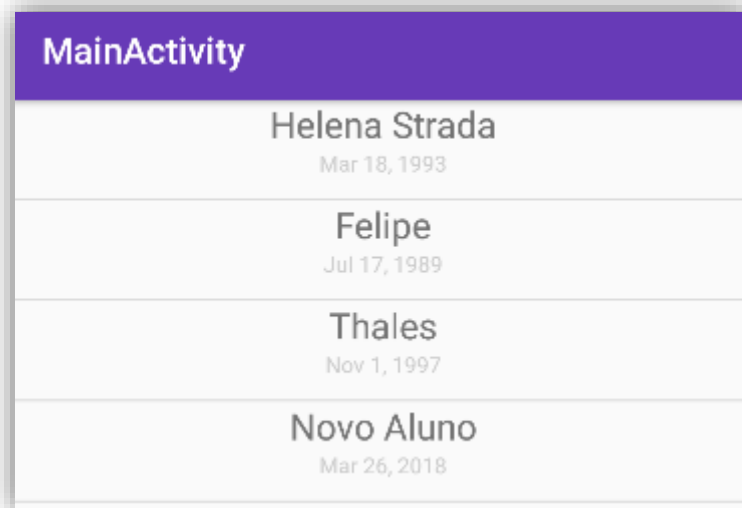


Figura 67 - Lista após nova inclusão de um novo aluno.

Adicionando uma mensagem de sucesso para o usuário

E se além disso, quisermos colocar uma mensagem para o usuário?

Simples: basta adicionarmos após recarregar a lista, a mensagem que deseja.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == 1) {
        if (resultCode == RESULT_OK) {
            carregarListaDeAlunos();
            Toast.makeText(getApplicationContext(), "Aluno inserido com sucesso.", Toast.LENGTH_LONG).show();
        }
    }
}
```

Figura 68 - Mensagem de sucesso após cadastro.

Arquivos

Resumo

Após realizar o login e a listagem de registros salvos em uma lista, nós aprendemos neste capítulo a como salvar dados a partir da entrada de informações que o usuário irá digitar. Além disso, a como acessar a câmera e criar um novo diálogo de data (calendário).

Quarta etapa

Nesta etapa do projeto, iremos aprender a como alterar um registro já cadastrado em nossa lista de alunos.

O que nós precisamos fazer para carregar os dados de um aluno que já foi cadastrado? Vamos lembrar que para cada aluno cadastrado, nós temos um identificador, correto? Através dele, nós podemos localizar o registro do aluno na lista, uma vez que ele é único.

Além disso, através do nosso ListView, quando nós clicarmos no item que desejamos, nós desejamos abrir a tela de cadastro com os dados populados do aluno. Por quê criarmos uma activity sendo que a tela de edição e cadastro possuem os mesmos campos? Como então identificamos se será um novo aluno a ser cadastrado ou se o aluno já existe na lista?

Vamos ao passo-a-passo.

Selecionando um item da lista

Nosso primeiro passo será selecionar um item da nossa lista. Ou seja, assim que o usuário clicar em cima de um item da lista, vamos imprimir na tela o aluno que foi clicado. Logo, a modificação que precisamos fazer será em nossa ListView.

Vamos para a nossa MainActivity e realizar a seguinte modificação.

```
// Selecionando um aluno da lista
lvListaAlunos.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        Aluno alunoSelecionado = (Aluno) adapterView.getAdapter().getItem(i);
        Toast.makeText(getApplicationContext(), alunoSelecionado.toString(), Toast.LENGTH_LONG).show();
    }
});
```

Figura 69 - Selecionando um aluno da lista.

Copiável:

```
// Selecionando um aluno da lista
lvListaAlunos.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int
i, long l) {
        Aluno alunoSelecionado = (Aluno)
adapterView.getAdapter().getItem(i);
        Toast.makeText(getApplicationContext(),
alunoSelecionado.toString(), Toast.LENGTH_LONG).show();
    }
});
```

```
}  
});
```

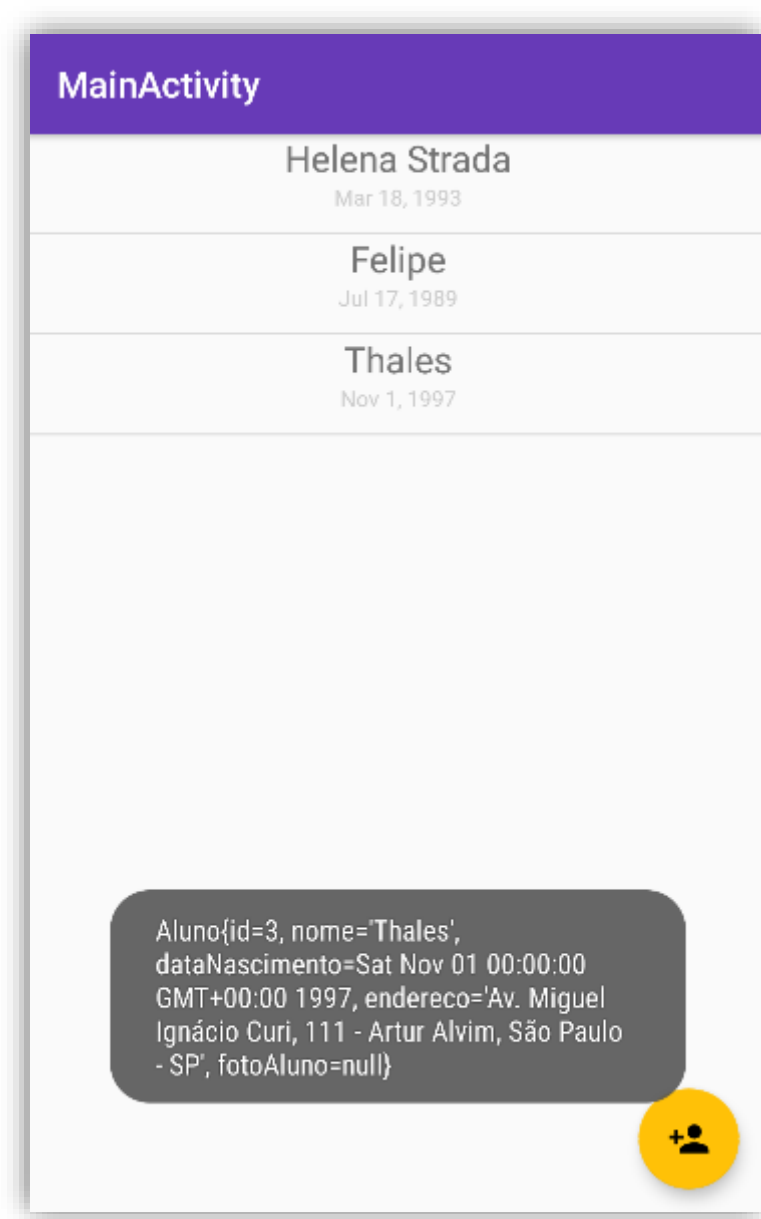


Figura 70 - Aluno após ser selecionando.

Já que conseguimos selecionar o ID do aluno clicado, o que precisamos fazer é: pegar o valor desse ID, levar o id para a próxima tela e buscar o aluno na próxima tela de formulário. O que isso significa? Significa que, quando eu carregar a tela de formulário, caso eu já tenha um ID, ou seja, caso eu tenha selecionado um aluno, os dados que eu vou carregar nessa tela, são os dados do aluno que eu selecionei. Porém, caso nenhum ID seja selecionado ou carregado nessa tela, o que irá acontecer é que o aluno será um novo registro.

Levando o ID do aluno para a próxima tela

Uma vez que conseguimos imprimir o aluno na mensagem, conseguimos buscar apenas a informação que desejamos que será o ID do aluno.

```
// Selecionando um aluno da lista
lvListaAlunos.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        Aluno alunoSelecionado = (Aluno) adapterView.getAdapter().getItem(i);
        // Toast.makeText(getApplicationContext(), alunoSelecionado.toString(), Toast.LENGTH_LONG).show();
        Intent intent = new Intent( packageContext: MainActivity.this, FormActivity.class);
        intent.putExtra( name: "idAluno", alunoSelecionado.getId());
        startActivityForResult(intent, requestCode: 1);
    }
});
```

Figura 71 - ID do Aluno para a tela de formulário.

Copiável:

```
// Selecionando um aluno da lista
lvListaAlunos.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int
i, long l) {
        Aluno alunoSelecionado = (Aluno)
adapterView.getAdapter().getItem(i);
        // Toast.makeText(getApplicationContext(),
alunoSelecionado.toString(), Toast.LENGTH_LONG).show();
        Intent intent = new Intent(MainActivity.this,
FormActivity.class);
        intent.putExtra("idAluno", alunoSelecionado.getId());
        startActivityForResult(intent, 1);
    }
});
```

Vamos realizar um teste simples na tela de formulário. Vamos verificar se eu possuo um id que está sendo carregado ou não.

```
final Bundle extras = getIntent().getExtras();
Long idDoAlunoCarregado = (extras != null) ? extras.getLong( key: "idAluno") : null;
Toast.makeText(getApplicationContext(), idDoAlunoCarregado.toString(), Toast.LENGTH_SHORT).show();
```

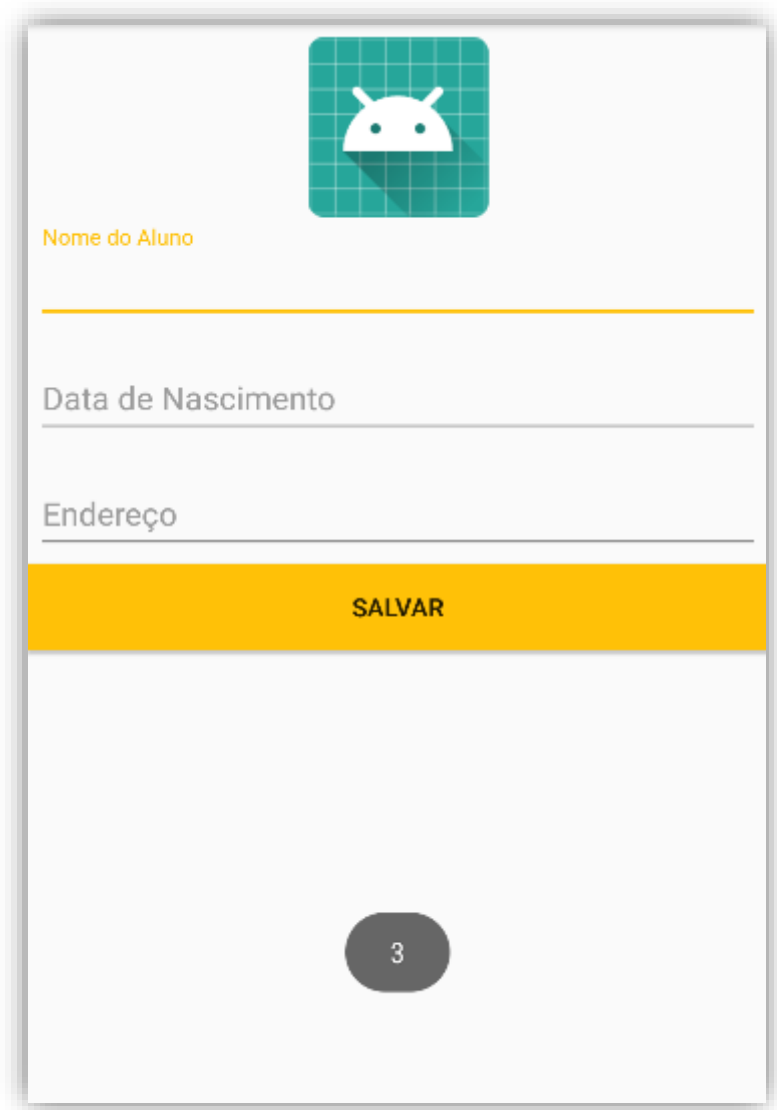
Figura 72 - Verificando na activity de formulário se o ID foi carregado.

Copiável:

```
final Bundle extras = getIntent().getExtras();
Long idDoAlunoCarregado = (extras != null) ? extras.getLong("idAluno")
: null;
```

```
Toast.makeText(getApplicationContext(), idDoAlunoCarregado.toString(),  
Toast.LENGTH_SHORT).show();
```

Ao clicar em algum item da nossa lista, na próxima tela, podemos perceber que o ID do aluno que selecionamos foi levado para a próxima tela.



Nome do Aluno

Data de Nascimento

Endereço

SALVAR

3

Figura 73 - ID do aluno selecionado na tela anterior.

Sempre que quisermos levar uma variável ou o próprio objeto, nós podemos resolver dessa maneira.

Mas pudemos perceber que os dados do aluno ainda não foram carregados. O que precisamos fazer agora?

Populando o formulário com os dados do aluno

Vamos criar um método na nossa FormActivity que irá verificar se ela estará recebendo o ID ou se será o cadastro de um aluno novo.

No método onCreate(), para organizar, irei chamar e criar um novo método chamado `carregarDadosDoAlunoCasoTenhaId()`;

```

private void carregarDadosDoAlunoCasoTenhaId() {
    final Bundle extras = getIntent().getExtras();
    Long idDoAlunoCarregado = (extras != null) ? extras.getLong( key: "idAluno") : null;

    if (idDoAlunoCarregado == null) {
        // se não tiver um aluno, eu irei realizar um novo cadastro
        Aluno aluno = new Aluno();
        getSupportActionBar().setTitle("Cadastrar Aluno");
    } else {
        getSupportActionBar().setTitle("Editar Aluno");
        Aluno alunoCarregado = new Aluno();
        alunoCarregado = dao.localizar(idDoAlunoCarregado);
        alunoCarregado.setId(idDoAlunoCarregado);
        tilNomeAluno.getEditText().setText(alunoCarregado.getNome());
        tilDataNascimentoAluno.getEditText().setText(DateFormat.getDateInstance(DateFormat.MEDIUM).format(alunoCarregado.getDataNascimento()));
        tilEnderecoAluno.getEditText().setText(alunoCarregado.getEndereco());

        // se o aluno possuir uma foto cadastrada, eu carrego essa imagem
        if (alunoCarregado.getFotoAluno() != null) {
            byte[] encodeByte = Base64.decode(alunoCarregado.getFotoAluno(), Base64.DEFAULT);
            Bitmap bitmap = BitmapFactory.decodeByteArray(encodeByte, 0, encodeByte.length);
            ivFotoAluno.setImageBitmap(bitmap);
            temp = alunoCarregado.getFotoAluno();
        }
        // se o aluno não possuir uma foto cadastrada, eu carrego uma imagem padrão
        else {
            ivFotoAluno.setImageResource(R.drawable.ic_launcher_background);
        }
    }
}

```

Figura 74 - Implementação do método de popular os dados do aluno.

Copiável:

```
carregarDadosDoAlunoCasoTenhaId();
```

```

private void carregarDadosDoAlunoCasoTenhaId() {
    final Bundle extras = getIntent().getExtras();
    Long idDoAlunoCarregado = (extras != null) ?
    extras.getLong("idAluno") : null;

    if (idDoAlunoCarregado == null) {
        // se não tiver um aluno, eu irei realizar um novo cadastro
        Aluno aluno = new Aluno();
        getSupportActionBar().setTitle("Cadastrar Aluno");
    } else {
        getSupportActionBar().setTitle("Editar Aluno");
        Aluno alunoCarregado = new Aluno();
        alunoCarregado = dao.localizar(idDoAlunoCarregado);
        alunoCarregado.setId(idDoAlunoCarregado);
        tilNomeAluno.getEditText().setText(alunoCarregado.getNome());

        tilDataNascimentoAluno.getEditText().setText(DateFormat.getDateInstance
e(DateFormat.MEDIUM).format(alunoCarregado.getDataNascimento()));

        tilEnderecoAluno.getEditText().setText(alunoCarregado.getEndereco());

        // se o aluno possuir uma foto cadastrada, eu carrego essa
        imagem
        if (alunoCarregado.getFotoAluno() != null) {
            byte[] encodeByte =
            Base64.decode(alunoCarregado.getFotoAluno(), Base64.DEFAULT);
            Bitmap bitmap = BitmapFactory.decodeByteArray(encodeByte,
            0, encodeByte.length);
            ivFotoAluno.setImageBitmap(bitmap);
            temp = alunoCarregado.getFotoAluno();
        }
    }
}


```



```
// se o aluno não possuir uma foto cadastrada, eu carrego uma
imagem padrão
else {
    ivFotoAluno.setImageResource(R.drawable.ic_launcher_background);
}

}
```

Uma vez que conseguimos passar valores de uma tela para outra, verificamos se este ID foi passado e caso seja este o caso, eu populamos os dados do meu aluno, com os dados que eu já havia cadastrado.



The screenshot shows a mobile application interface for editing a student. At the top is a purple header with the text "Editar Aluno". Below the header is a teal square placeholder for a photo. The form contains three input fields: "Nome do Aluno" with the value "Helena Strada", "Data de Nascimento" with the value "Mar 18, 1993", and "Endereço" with the value "Av. Paulista, 1578 - Bela Vista, São Paulo - SP". At the bottom of the form is a yellow button labeled "SALVAR".

Figura 75 - Dados populados do aluno.

Ajustando o método salvar

Eu consigo popular as informações do formulário com as informações do aluno que foram cadastradas. Porém, e o nosso método de salvar? Assim como o nosso método para carregar os dados do aluno, eu preciso verificar se o aluno já existe. Caso ele já exista, o que eu preciso fazer é atualizar apenas os registros dele e não criar um novo registro pra ele.

```

private class salvarDadosDoAluno implements View.OnClickListener {
    @Override
    public void onClick(View view) {
        cadastrarAluno();
    }
}

private void cadastrarAluno() {

    // Preciso verificar se já existi um aluno cadastrado
    final Bundle extras = getIntent().getExtras();
    // bundle.getString("idAluno");
    Long idDoAlunoSelecionado = (extras != null) ? extras.getLong( key: "idAluno") : null;

    // caso eu não tenha nenhum aluno selecionado, estarei criando um, portanto:
    if (idDoAlunoSelecionado == null) {
        Aluno alunoNovo = new Aluno();
        alunoNovo.setNome(tilNomeAluno.getText().getText().toString());
        alunoNovo.setDataNascimento(new Date(tilDataNascimentoAluno.getText().getText().toString()));
        alunoNovo.setEndereco(tilEnderecoAluno.getText().getText().toString());

        alunoNovo.setFotoAluno(temp);
        // dentro do método salvar, eu verifico se o aluno existe
        dao.salvar(alunoNovo);
    }
    else {
        Aluno alunoExistente = new Aluno();
        alunoExistente.setId(idDoAlunoSelecionado);
        alunoExistente.setNome(tilNomeAluno.getText().getText().toString());
        alunoExistente.setDataNascimento(new Date(tilDataNascimentoAluno.getText().getText().toString()));
        alunoExistente.setEndereco(tilEnderecoAluno.getText().getText().toString());

        alunoExistente.setFotoAluno(temp);

        dao.salvar(alunoExistente);
    }

    retornarParaTelaAnteriorAposSalvar();
}
}

```

Figura 76 - Salvando os dados do aluno.

Copiável:

```

private class salvarDadosDoAluno implements View.OnClickListener {
    @Override
    public void onClick(View view) {
        cadastrarAluno();
    }
}

private void cadastrarAluno() {

    // Preciso verificar se já existi um aluno cadastrado
    final Bundle extras = getIntent().getExtras();
    // bundle.getString("idAluno");
    Long idDoAlunoSelecionado = (extras != null) ?
    extras.getLong("idAluno") : null;

    // caso eu não tenha nenhum aluno selecionado, estarei criando um,
    portanto:

```

```

        if (idDoAlunoSelecionado == null) {
            Aluno alunoNovo = new Aluno();

alunoNovo.setNome(tilNomeAluno.getText().toString());
            alunoNovo.setDataNascimento(new
Date(tilDataNascimentoAluno.getText().toString()));

alunoNovo.setEndereco(tilEnderecoAluno.getText().toStrin
g());

            alunoNovo.setFotoAluno(temp);
            // dentro do método salvar, eu verifico se o aluno existe
            dao.salvar(alunoNovo);
        }
        else {
            Aluno alunoExistente = new Aluno();
            alunoExistente.setId(idDoAlunoSelecionado);

alunoExistente.setNome(tilNomeAluno.getText().toString()
);
            alunoExistente.setDataNascimento(new
Date(tilDataNascimentoAluno.getText().toString()));

alunoExistente.setEndereco(tilEnderecoAluno.getText().to
String());

            alunoExistente.setFotoAluno(temp);

            dao.salvar(alunoExistente);
        }

        retornarParaTelaAnteriorAposSalvar();
    }
}

```

Arquivos

Resumo

Nesta etapa, nós aprendemos a como editar um registro de um aluno de nossa lista. Além disso, a como salvar e/ou atualizar um registro existente.

Quinta Etapa

Esta etapa do projeto será menor. Na nossa lista de alunos, quando clicamos em um aluno, iremos abrir um menu de opções que conterá as seguintes informações: Edição e Deleção.

Criando o menu personalizado

Mas antes de colocarmos o código para implementar o clique com toque longo, precisamos criar o nosso menu personalizado.

Crie um 'Resource Directory' dentro da pasta res chamado menu.

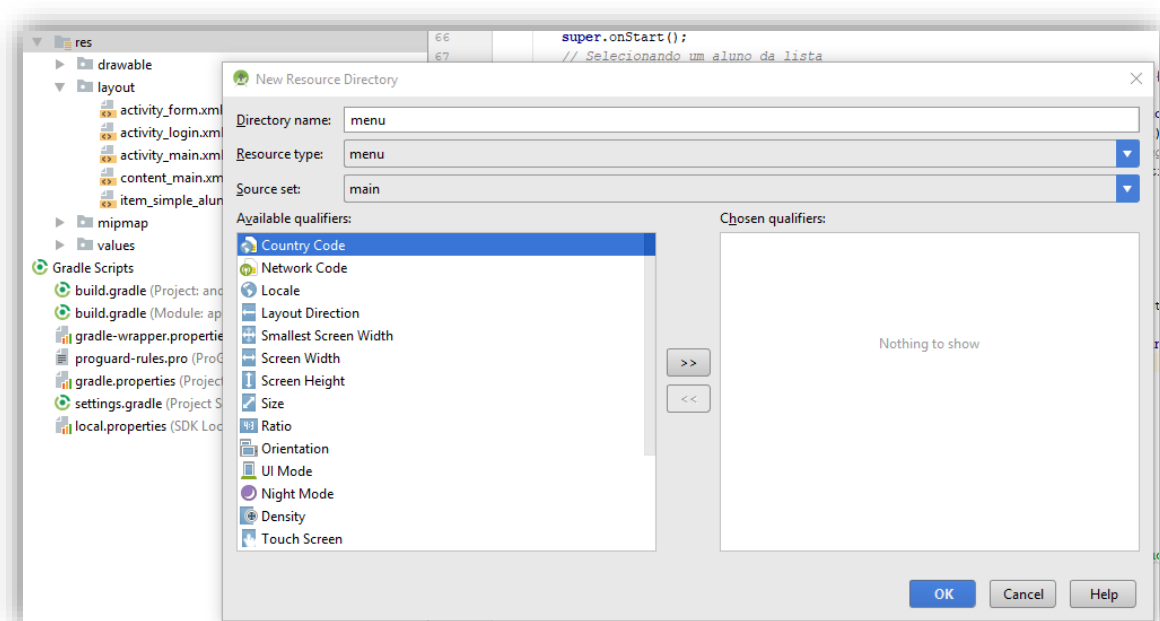


Figura 77 - Criando o novo diretório menu.

Clique em 'OK'.

Dentro da pasta de menu que acabamos de criar, crie um novo arquivo chamado 'menu_aluno_options.xml' que conterá as informações do xml que desejamos.

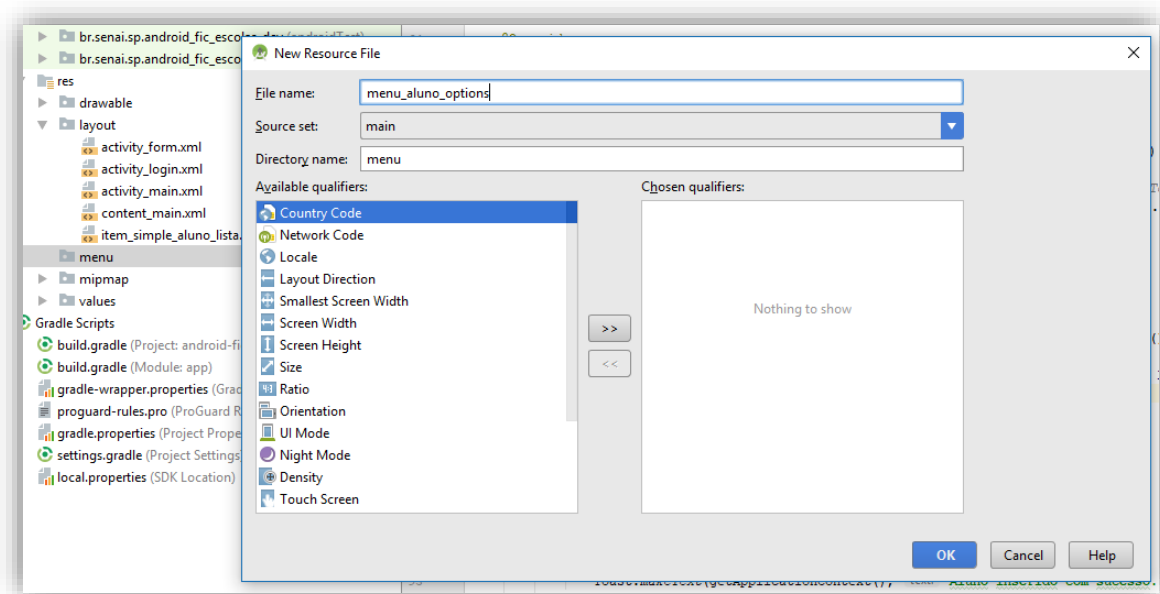


Figura 78 - Criando o menu personalizado.

Clique em 'OK'.

Dentro do xml, vamos colocar as atuais informações que desejamos.



Figura 79 - Colocando as opções no menu.

Uma vez que criamos o nosso menu personalizado, precisamos informar na nossa lista que ao selecionarmos com um clique longo, um item da lista, esse menu irá aparecer. Além disso que, ao clicar no item para editar ou remover, alguma ação aconteça.

Habilitando o menu personalizado

Para habilitar o menu que desejamos habilitar, precisamos trabalhar com a classe PopupMenu.

```
// Selecionando um aluno da lista com longclick
lvListaAlunos.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public boolean onItemClick(final AdapterView<?> adapterView, View view, final int i, long l) {
        PopupMenu popupMenu = new PopupMenu(view.getContext(), view);
        popupMenu.getMenuInflater().inflate(R.menu.menu_aluno_options, popupMenu.getMenu());

        popupMenu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
            public boolean onMenuItemClick(MenuItem item) {

                switch (item.getItemId()) {
                    case R.id.menuAlunoEditar:
                        Aluno alunoSelecionado = (Aluno) adapterView.getAdapter().getItem(i);
                        Intent intent = new Intent( packageContext: MainActivity.this, Normalizer.Form.class);
                        intent.putExtra( name: "idAluno", alunoSelecionado.getId());
                        startActivityForResult(intent, requestCode: 1);
                        break;

                    case R.id.menuAlunoDeletar:

                        alunoSelecionado = (Aluno) adapterView.getAdapter().getItem(i);
                        dao.remove(alunoSelecionado);
                        carregarListaDeAlunos();
                        break;

                }
                return true;
            }
        });

        popupMenu.show();
        return false;
    }
});
```

Figura 80 - Código para o PopupMenu.

Copiável:

```
// Selecionando um aluno da lista com longclick
lvListaAlunos.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    @Override
    public boolean onItemClick(final AdapterView<?>
adapterView, View view, final int i, long l) {
        PopupMenu popupMenu = new PopupMenu(view.getContext(),
view);

        popupMenu.getMenuInflater().inflate(R.menu.menu_aluno_options,
popupMenu.getMenu());

        popupMenu.setOnMenuItemClickListener(new
PopupMenu.OnMenuItemClickListener() {
            public boolean onMenuItemClick(MenuItem item) {

                switch (item.getItemId()) {
                    case R.id.menuAlunoEditar:
                        Aluno alunoSelecionado = (Aluno)
adapterView.getAdapter().getItem(i);
```

```

Intent intent = new
Intent(MainActivity.this, Normalizer.Form.class);
intent.putExtra("idAluno",
alunoSelecionado.getId());

startActivityForResult(intent, 1);
break;

case R.id.menuAlunoDeletar:

alunoSelecionado = (Aluno)
adapterView.getAdapter().getItem(i);
dao.remove(alunoSelecionado);
carregarListaDeAlunos();
break;

}

return true;

}

});

popupMenu.show();
return false;

}

});

}

```

Uma vez que clicarmos na opção de Editar, iremos ser redirecionados para a tela de edição.

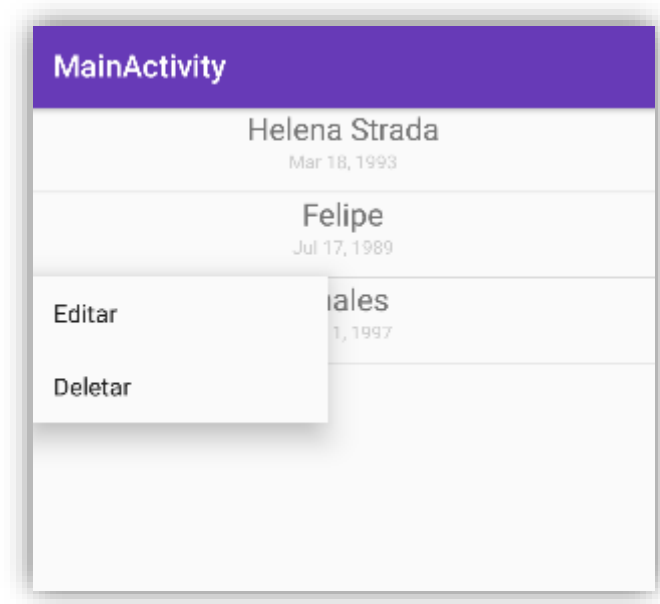


Figura 81 - Mostrando o menu personalizado.

E se for clicado em deletar, o aluno é removido da lista.

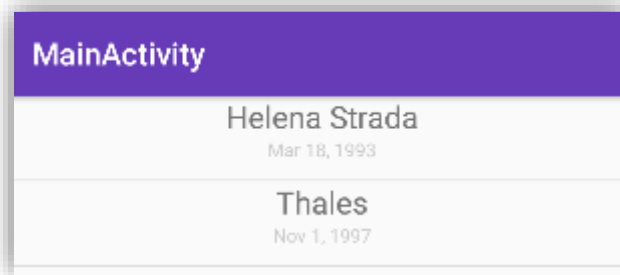


Figura 82 - Removendo um aluno da lista.

Abrindo um menu de decisão

Vamos melhorar um pouco essa deleção. Não seria interessante mostrar para o usuário que está deletando um aluno, se ele tem certeza que deseja deletar o aluno selecionado?

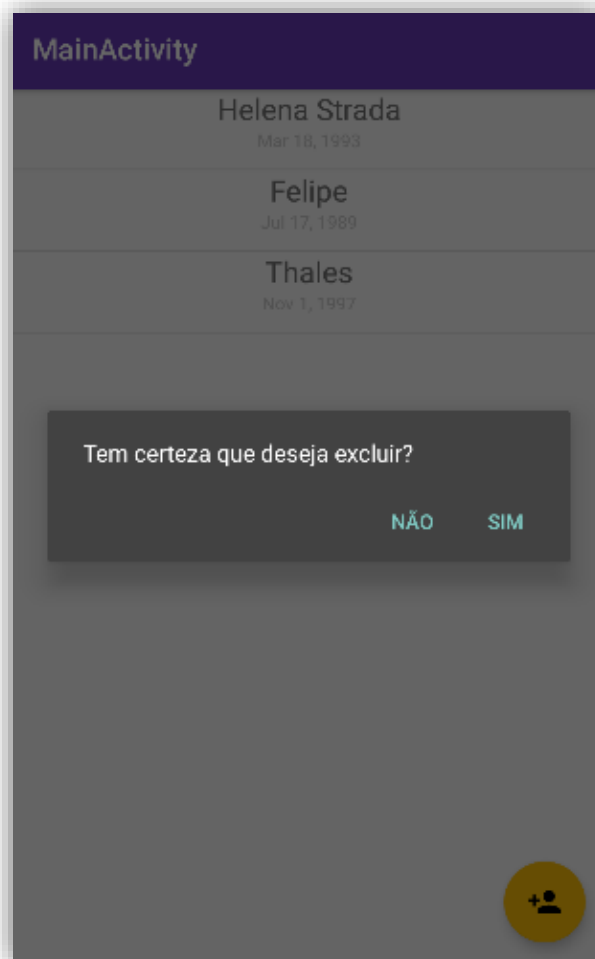


Figura 83 - Diálogo de decisão.

Vamos adicionar uma nova interface para o nosso diálogo.

Antes disso, precisamos criar um novo estilo para o nosso diálogo.

```
<!-- Estilo para o diálogo de decisão -->  
<style name="dialogoDecisao" parent="Theme.AppCompat.Dialog">  
|   <item name="android:windowNoTitle">true</item>  
</style>
```

Figura 84 - Adicionando um novo estilo para o diálogo.

Abra o arquivo styles.xml e cole este código lá.

```
<!-- Estilo para o diálogo de decisão -->  
<style name="dialogoDecisao" parent="Theme.AppCompat.Dialog">  
  <item name="android:windowNoTitle">true</item>  
</style>
```

Após adicionar o estilo ao styles.xml, chame o novo código, para abrir um diálogo. Ele somente irá excluir o registro se ele clicar em Sim.

```

// Selecionando um aluno da lista com longclick
lvListaAlunos.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public boolean onItemClick(final AdapterView<?> adapterView, View view, final int i, long l) {
        PopupMenu popupMenu = new PopupMenu(view.getContext(), view);
        popupMenu.getMenuInflater().inflate(R.menu.menu_aluno_options, popupMenu.getMenu());

        popupMenu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
            public boolean onMenuItemClick(MenuItem item) {
                switch (item.getItemId()) {
                    case R.id.menuAlunoEditar:
                        Aluno alunoSelecionado = (Aluno) adapterView.getAdapter().getItem(i);
                        Intent intent = new Intent( packageContext: MainActivity.this, Normalizer.Form.class);
                        intent.putExtra( name: "idAluno", alunoSelecionado.getId());
                        startActivityForResult(intent, requestCode: 1);
                        break;
                    case R.id.menuAlunoDeletar:
                        alunoSelecionado = (Aluno) adapterView.getAdapter().getItem(i);
                        criarDialogoComAlunoSelecionado(getApplicationContext(), alunoSelecionado);
                        break;
                }
                return true;
            }
        });

        popupMenu.show();
        return false;
    }
});

private void criarDialogoComAlunoSelecionado(Context context, final Aluno alunoSelecionado) {
    DialogInterface.OnClickListener dialogClickListener = (dialog, which) -> {
        switch (which){
            case DialogInterface.BUTTON_POSITIVE:
                dao.remover(alunoSelecionado);
                carregarListaDeAlunos();
                break;
            case DialogInterface.BUTTON_NEGATIVE:
                break;
        }
    };

    AlertDialog.Builder builder = new AlertDialog.Builder(new ContextThemeWrapper( base: this, R.style.dialogoDecisao));
    builder.setMessage("Tem certeza que deseja excluir?").setPositiveButton( text: "Sim", dialogClickListener)
        .setNegativeButton( text: "Não", dialogClickListener).show();
}

```

Figura 85 - Adicionando um diálogo de decisão.

Copiável:

```

// Selecionando um aluno da lista com longclick
lvListaAlunos.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    @Override
    public boolean onItemClick(final AdapterView<?>
adapterView, View view, final int i, long l) {
        PopupMenu popupMenu = new PopupMenu(view.getContext(),
view);

        popupMenu.getMenuInflater().inflate(R.menu.menu_aluno_options,
popupMenu.getMenu());

        popupMenu.setOnMenuItemClickListener(new
PopupMenu.OnMenuItemClickListener() {
            public boolean onMenuItemClick(MenuItem item) {

                switch (item.getItemId()) {
                    case R.id.menuAlunoEditar:

```

```

        Aluno alunoSelecioneado = (Aluno)
adapterView.getAdapter().getItem(i);
        Intent intent = new
Intent(MainActivity.this, Normalizer.Form.class);
        intent.putExtra("idAluno",
alunoSelecioneado.getId());
        startActivityForResult(intent, 1);
        break;

        case R.id.menuAlunoDeletar:

            alunoSelecioneado = (Aluno)
adapterView.getAdapter().getItem(i);

            criarDialogoComAlunoSelecioneado(getApplicationContext(),
alunoSelecioneado);

            break;
        }
        return true;
    }
});

popupMenu.show();
return false;
}
});
}

private void criarDialogoComAlunoSelecioneado(Context context, final
Aluno alunoSelecioneado) {
    DialogInterface.OnClickListener dialogClickListener = new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            switch (which){
                case DialogInterface.BUTTON_POSITIVE:
                    dao.remove(alunoSelecioneado);
                    carregarListaDeAlunos();
                    break;

                case DialogInterface.BUTTON_NEGATIVE:
                    break;
            }
        }
    };

    AlertDialog.Builder builder = new AlertDialog.Builder(new
ContextThemeWrapper(this, R.style.dialogoDecisao));
    builder.setMessage("Tem certeza que deseja
excluir?").setPositiveButton("Sim", dialogClickListener)
        .setNegativeButton("Não", dialogClickListener).show();
}

```

Arquivos

Resumo

Sexta Etapa

Uma vez que salvamos o nosso aluno com o endereço dele, nós precisamos mostrar de alguma maneira em nosso aplicativo.

Adicionando um novo item no menu

Quando demos um clique longo no aluno, nós precisamos, além de mostrar somente a edição ou a deleção. Vamos criar um novo item no menu para mostrar o aluno no mapa.



Figura 86 - Adicionando mais um item da lista.

Copiável:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/menuAlunoEditar"
        android:title="Editar"/>

    <item
        android:id="@+id/menuAlunoMostrarMapa"
        android:title="Visualizar Mapa" />

    <item
        android:id="@+id/menuAlunoDeletar"
        android:title="Deletar"/>

</menu>
```

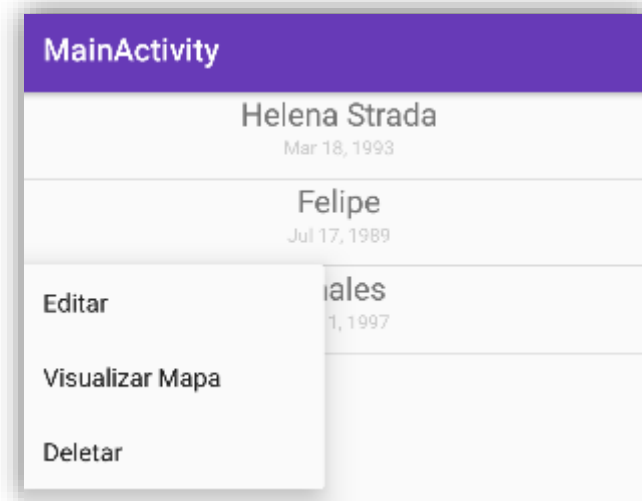


Figura 87 - Adicionando um novo item no menu.

Agora precisamos informar para a nossa aplicação, que desejamos trabalhar com os recursos de mapas do google.

Trabalhando com mapas

Vamos adicionar uma dependência no nosso build.gradle.

```
implementation 'com.google.android.gms:play-services-maps:11.8.0'
```

```
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    implementation 'com.android.support:appcompat-v7:26.1.0'  
    implementation 'com.android.support.constraint:constraint-layout:1.0.2'  
    testImplementation 'junit:junit:4.12'  
    androidTestImplementation 'com.android.support.test:runner:1.0.1'  
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.1'  
    implementation 'com.android.support:design:26.1.0'  
  
    // mapas  
    implementation 'com.google.android.gms:play-services-maps:11.8.0'  
}
```

Figura 88 - Adicionando o maps no nosso gradle.

Para mostrar o mapa, vamos mostrar uma nova activity para o mapa.

Na pasta views, vamos criar uma nova activity do tipo: Google -> MapsActivity.

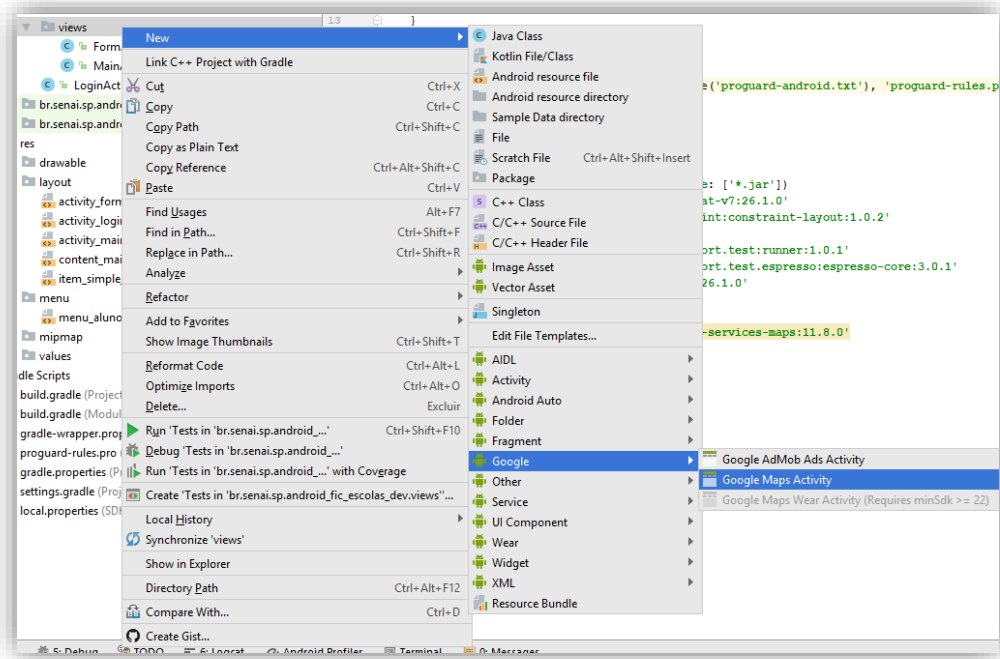


Figura 89 - Criando uma nova activity do tipo MapsActivity

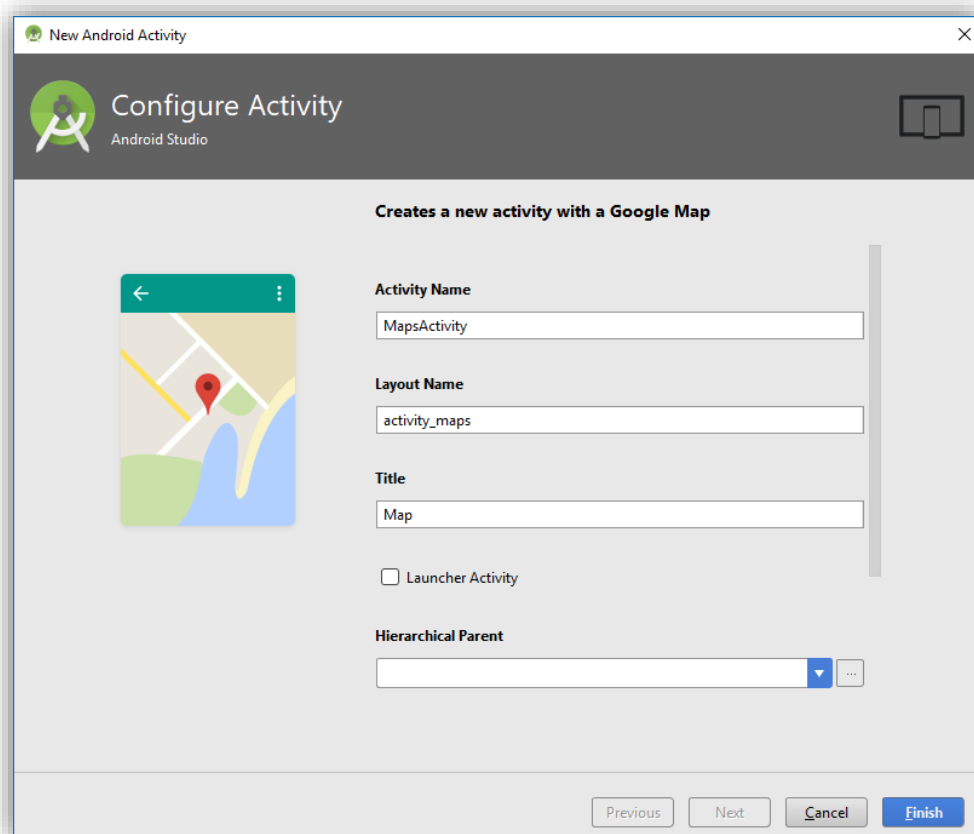


Figura 90 - Adicionando a nova activity de mapa.

Clique em 'Finish'.

Criando uma nova chave

<https://developers.google.com/maps/documentation/javascript/get-api-key?hl=pt-br>

Vamos entrar no site para obter uma nova chave para trabalharmos com mapas.

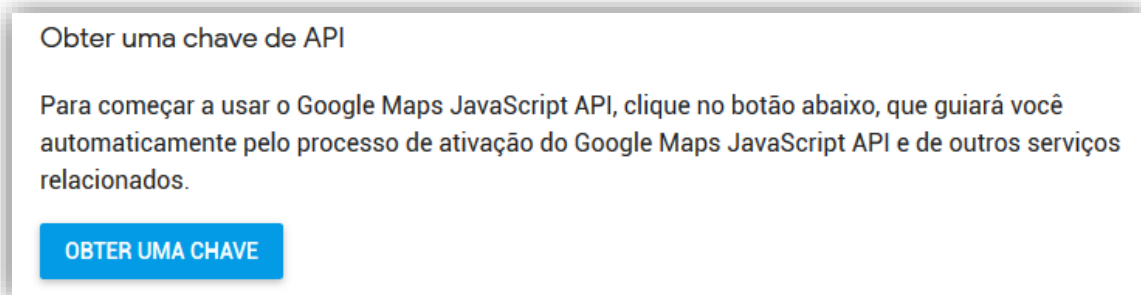


Figura 91 - Obtendo uma nova chave pelo site.

Clique para obter uma nova chave. Selecione um projeto existente ou crie um novo.

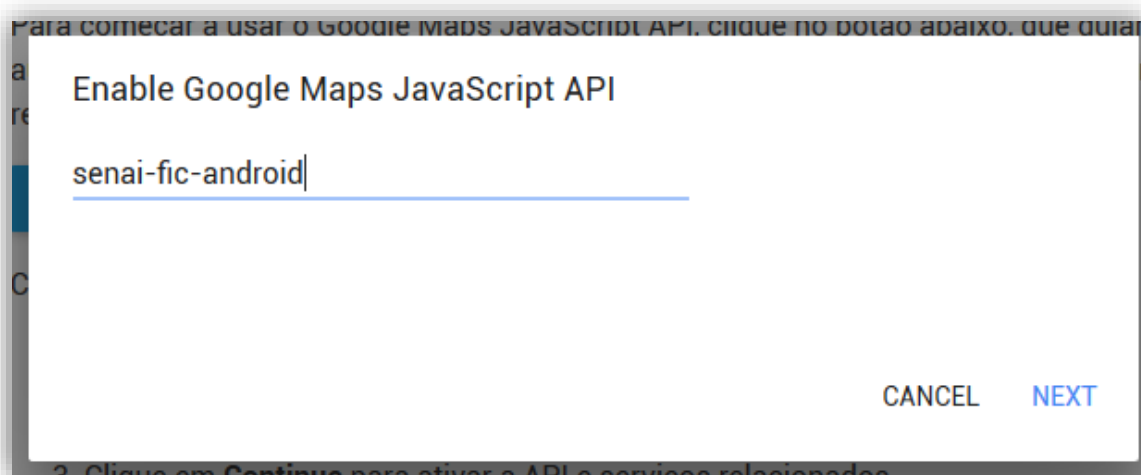


Figura 92 - Escolha o nome do projeto.

Copie e salve as informações da chave.

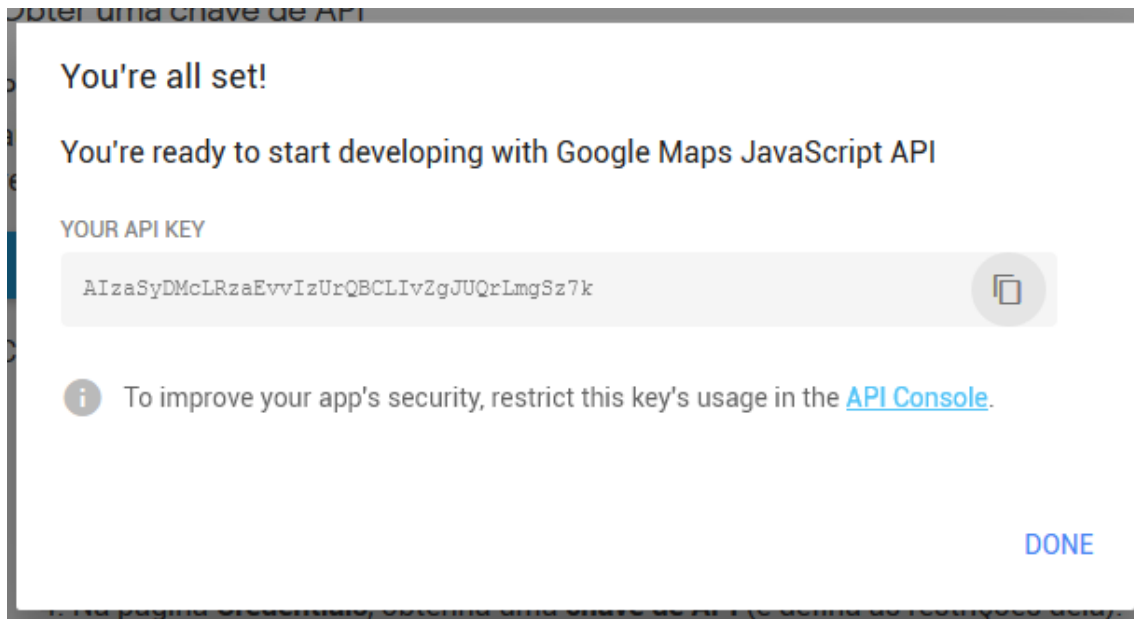


Figura 93 - Copie e salve a chave que foi gerada.

Colocando a chave no projeto

Vamos alterar e colocar a chave em alguns locais do projeto.

Coloque a chave dentro do arquivo: google_maps_api.xml.

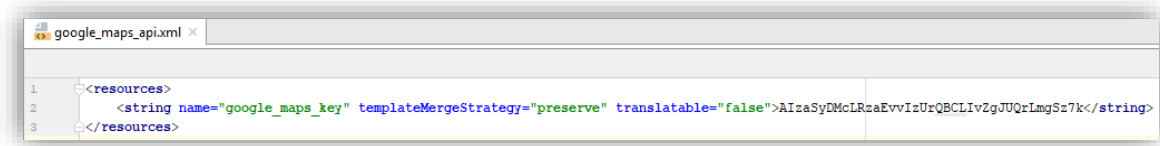


Figura 94 - Colocando a chave no modo debug.

No arquivos strings.xml, crie uma nova string com o valor: google_maps_api_key.

```
<string  
name="google_maps_api_key">AIzaSyCuXMy2XVMaVsJmecLc0sEYYXdRWMdVhHI</st  
ring>
```




Figura 95 - Criando uma nova string para o mapa.

Adicione mais as informações que seguem no AndroidManifest.xml.

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="@string/google_maps_api_key" />
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.senai.sp.android_fic_escolas_dev">

    <!-- Salvar o arquivo da câmera -->
    <uses-feature
        android:name="android.hardware.camera"
        android:required="true" />

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

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="android-fic-escolas-dev"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".LoginActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".views.MainActivity"
            android:label="MainActivity"
            android:theme="@style/AppTheme.NoActionBar" />
        <activity android:name=".views.FormActivity" />

        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_api_key" />
        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />

        <activity
            android:name=".views.MapsActivity"
            android:label="Map"></activity>
    </application>
</manifest>

```

Figura 96 - AndroidManifest depois de adicionado a chave da api.

Testando a primeira parte do mapa

Após colocarmos todas as chaves no projeto, quando criamos uma activity do tipo MapsActivity, ele nos dá um ponto fixo de referência no mapa para testes. Vamos testar nossa aplicação e ver o que está ocorrendo?



Figura 97 - Testando a visualização no mapa com ponto fixo.

Após testarmos e visualizarmos que o mapa funciona, bem como as permissões de acesso que solicitamos ao usuário, ao invés de carregarmos esse ponto fixo no mapa, o que desejamos fazer é colocar no mapa, o pin com o endereço do aluno que ele for adicionado. Para isto, faremos as seguintes alterações.

Colocando o endereço do aluno no mapa

O que faremos agora será alterar o ponto fixo do mapa para o endereço do aluno. Mas, além de alterarmos o ponto no mapa, primeiro precisamos saber qual é o aluno que desejamos "levar" para a próxima tela.

```
case R.id.menuAlunoMostrarMapa:
    alunoSelecionado = (Aluno) adapterView.getAdapter().getItem(i);
    intent = new Intent( packageContext: MainActivity.this, MapsActivity.class);
    intent.putExtra( name: "idAluno", alunoSelecionado.getId());
    startActivity(intent);
```

Figura 98 - Levando o id do aluno para a próxima tela.

Copiável:

```
alunoSelecionado = (Aluno) adapterView.getAdapter().getItem(i);
intent = new Intent(MainActivity.this, MapsActivity.class);
intent.putExtra("idAluno", alunoSelecionado.getId());
startActivity(intent);
```

Transformando endereço em latitude e longitude

No MapsActivity precisamos realizar os seguintes passos. Uma vez que eu carregar os dados do aluno com o endereço, para colocarmos o marker no mapa, eu preciso da latitude e longitude. Então vamos criar um conversor para isso.

```
public LatLng buscarLocalizacaoAtravesDoEndereco(Context context, String enderecoRecebido) {

    Geocoder coder = new Geocoder(context);
    List<Address> address;
    LatLng pl = null;

    try {
        address = coder.getFromLocationName(enderecoRecebido, maxResults: 5);
        if (address == null) {
            return null;
        }
        Address localizacao = address.get(0);
        localizacao.getLatitude();
        localizacao.getLongitude();

        pl = new LatLng(localizacao.getLatitude(), localizacao.getLongitude() );
    } catch (IOException e) {
        e.printStackTrace();
    }

    return pl;
}
```

Figura 99 - Buscando a latitude e longitude do endereço.

Copiável:

```
public LatLng buscarLocalizacaoAtravesDoEndereco(Context context,
String enderecoRecebido) {

    Geocoder coder = new Geocoder(context);
    List<Address> address;
    LatLng p1 = null;

    try {
        address = coder.getFromLocationName(enderecoRecebido, 5);
        if (address == null) {
            return null;
        }
        Address localizacao = address.get(0);
        localizacao.getLatitude();
        localizacao.getLongitude();

        p1 = new LatLng(localizacao.getLatitude(),
        localizacao.getLongitude() );

    } catch (IOException e) {
        e.printStackTrace();
    }

    return p1;
}
```

O código completo da nossa activity de mapas ficará assim:

```
package br.senai.sp.android_fic_escolas_dev.views;

import android.content.Context;
import android.location.Address;
import android.location.Geocoder;
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

import java.io.IOException;
import java.util.List;

import br.senai.sp.android_fic_escolas_dev.R;
import br.senai.sp.android_fic_escolas_dev.dao.AlunoDao;
import br.senai.sp.android_fic_escolas_dev.model.Aluno;

public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback {

    private GoogleMap mMap;
    private AlunoDao dao = AlunoDao.manager;
    private String enderecoDoAluno;

    @Override
```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_maps);

    // verifica as informações do aluno
    final Bundle extras = getIntent().getExtras();
    Long idDoAlunoCarregado = (extras != null) ?
    extras.getLong("idAluno") : null;
    // localizo o aluno através do id dele
    Aluno alunoCarregado = new Aluno();
    alunoCarregado = dao.localizar(idDoAlunoCarregado);
    enderecoDoAluno = alunoCarregado.getEndereco();

    SupportMapFragment mapFragment = (SupportMapFragment)
    getSupportFragmentManager()
        .findFragmentById(R.id.map);
    mapFragment.getMapAsync(this);
}

@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;

    LatLng localizacaoDoAluno =
    buscarLocalizacaoAtravesDoEndereco(getApplicationContext(),
    enderecoDoAluno);

    LatLng localizacao = new LatLng(localizacaoDoAluno.latitude,
    localizacaoDoAluno.longitude);
    mMap.addMarker(new
    MarkerOptions().position(localizacao).title("Casa do Aluno"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(localizacao));
}

public LatLng buscarLocalizacaoAtravesDoEndereco(Context context,
String enderecoRecebido) {

    Geocoder coder = new Geocoder(context);
    List<Address> address;
    LatLng p1 = null;

    try {
        address = coder.getFromLocationName(enderecoRecebido, 5);
        if (address == null) {
            return null;
        }
        Address localizacao = address.get(0);
        localizacao.getLatitude();
        localizacao.getLongitude();

        p1 = new LatLng(localizacao.getLatitude(),
        localizacao.getLongitude());

    } catch (IOException e) {
        e.printStackTrace();
    }

    return p1;
}
}

```



Figura 100 - Endereço no mapa.

Adicionando um zoom no mapa

Esta é apenas uma dica: caso queira realizar um zoom no mapa, para o usuário visualizar mais de perto a sua localização, comente a linha que segue e altere pela nova abaixo.

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;

    LatLng localizacaoDoAluno = buscarLocalizacaoAtravesDoEndereco(getApplicationContext(), enderecoDoAluno);

    LatLng localizacao = new LatLng(localizacaoDoAluno.latitude, localizacaoDoAluno.longitude);
    mMap.addMarker(new MarkerOptions().position(localizacao).title("Casa do Aluno"));
    // mMap.moveCamera(CameraUpdateFactory.newLatLng(localizacao));
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(localizacao, 10));
}
```

Figura 101 - Alterando o zoom no mapa.

Copiável:

```
// mMap.moveCamera(CameraUpdateFactory.newLatLng(localizacao));  
mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(localizacao, 10));
```



Figura 102 – Resultado do mapa com zoom.

Sétima Etapa

Fizemos todo o nosso projeto. Desenvolvemos a tela de login, a home com a listagem de alunos e o cadastro/edição de alunos. Porém, você percebeu que até agora, toda vez que reiniciamos o aplicativo, nossos dados são perdidos?

O que faremos agora será alterar a nossa aplicação para que os dados sejam armazenados no banco de dados local da nossa aplicação.

Conhecendo o armazenamento interno da aplicação

O SQLite é uma biblioteca de software que implementa um Banco de Dados SQL transacional autônomo, sem servidor e sem nenhuma configuração.

Introdução ao SQLite

O SQLite possui as características de um banco de dados relacional. O SQLite armazena as tabelas, views e índices em apenas um arquivo do disco, e é nesse arquivo que a leitura e a escrita serão realizadas. Com o SQLite, se você reiniciar o celular, ou a aplicação, não iremos perder as informações que foram previamente inseridas. Essa informação só será removida do celular se a aplicação for desinstalada.

Porém, por ser um pouco mais simples do que bancos de dados como o MySQL ou SQL Server, ele possui algumas limitações. Por exemplo, os tipos de dados podem ser somente do tipo *Text*, *Integer*, *Real*, *Blob* e *Null*. Além disso, ele não valida se o tipo de dado que está sendo inserido, corresponde ao tipo de dado da coluna.

SQLite no android

O SQLite é incorporado em todos os dispositivos Android. Como dito anteriormente, não precisamos de nenhuma configuração a mais.

Armazenamento no Android

Por padrão, nosso arquivo de banco de dados será salvo no diretório:

DATA/data/<Nome-Applicacao>/databases/<Nome-BD>

DATA: caminho que o método `Environment.getDataDirectory()` retorna;

Nome-Applicacao: é o nome do seu aplicativo;

Nome-BD: nome especificado para o seu banco de dados.

Conhecendo um pouco mais sobre banco de dados

Um banco de dados é uma maneira estrutura de armazenar dados de forma persistente em formato de tabelas.

A tabela é composta por colunas e linhas. As colunas representam os tipos de dados que serão inseridos e as linhas representam os dados em si que foram armazenados. Pensamos em uma tabela do excel.

int	String	string
ID	Nome	Fabricante
1	God of War	Sony

Se fizermos a comparação para a nossa programação orientada a objetos, cada coluna irá representar o nosso atributo da classe e cada registro representaria uma instância específica deste objeto.

```
Public class Jogo {  
    private Long id;  
    private String nome;  
    private String fabricante;  
    // getters e setters  
}  
  
Jogo jogo = new Jogo();  
jogo.setId(2);  
jogo.setNome("GTA");  
jogo.setFabricante("Saint Monica Studios");
```

Para o nosso banco de dados, o registro que seria adicionado a nossa base, seria o registro deste jogo que criamos com suas respectivas propriedades.

ID	Nome	Autor
1	God of War	Sony
2	GTA	Saint Monica Studios

A grande situação aqui é que para manipularmos do Java ou de outra linguagem de programação para o nosso banco de dados, nós temos comandos específicos para o banco de dados. Ou seja, se precisamos inserir um novo registro no banco de dados por exemplo, nós temos um comando específico para essa ação.

Eu quero *inserir* uma nova linha na tabela de *jogos* com os respectivos dados de *id* = 3, *nome* = "FIFA", *fabricante* = "EA".

No SQL:

Insert into *jogos* (id, nome, fabricante) values (3, 'FIFA', 'EA');

INSERT: para inserir um novo registro no bd;

UPDATE: para atualizar um registro já existente no bd;

SELECT: para selecionar um registro no bd;

DELETE: para deletar um registro existente no bd.

Para que estes dados sejam manipulados, precisamos criar as nossas tabelas com as respectivas colunas e os tipos de dados que queremos armazenar. Para isto, temos comandos específicos também do SQL. Neste caso, utilizaremos o CREATE.

SQLite na Prática

Salvar nossos dados em um banco de dados é ideal para dados estruturados e para uma grande quantidade de informações que deverão ser armazenadas.

Esquema e Contrato

Um dos princípios mais importante de bancos de dados é o esquema: é a declaração de como o nosso banco de dados será organizado. Além disso, é recomendado criar um classe de contrato que irá especificar a estrutura do esquema.

Essa classe de contrato é aonde colocaremos as definições globais para o banco de dados, como por exemplo, nossas propriedades para o nosso Aluno.

Criando nosso banco de dados

Uma vez que definimos a estrutura das nossas tabelas, podemos de fato criá-las. Para criar nosso banco, vamos utilizar um conjunto útil de APIs que está disponível para nós na classe SQLiteOpenHelper.

Assim, podemos chamar somente quando necessitamos das informações e não durante a inicialização do aplicativo. Utilizaremos o `getWritableDatabase()` e o `getReadableDatabase()`. Além disso, devemos ter em mente que as chamadas podem ser de longa duração, devemos assim utilizar `AsyncTask`, por exemplo.

Para nós criarmos um novo banco de dados em nossa aplicação, a primeira coisa que devemos fazer é criar uma classe que herde de SQLiteOpenHelper para que nós tenhamos acesso a dois métodos principais, onCreate e onUpgrade.

onCreate nós definimos qual o script de criação das nossas tabelas. onUpgrade nós podemos atualizar uma versão do nosso banco de dados (atualizando estrutura, alterando versão).

Nesta classe, definimos também o construtor padrão. Ela irá definir em qual banco de dados queremos realizar todo o nosso script.

Alterando nossa estrutura

Vamos criar uma classe que irá definir o nosso esquema de criação do banco de dados. Para não alterarmos a nossa estrutura atual e mantermos o nosso projeto com os arquivos antigos, vamos criar um novo pacote chamado bd e criar duas classes Java dentro dele: AlunoDbHelper e AlunoDaoDB.

Criando nosso helper

```
public class AlunoDbHelper extends SQLiteOpenHelper {

    // propriedades para o nosso banco de dados
    private static final String NOME_BANCO = "alunos.db";
    public static final String TABELA = "alunos";
    public static final String ID = "_id";
    public static final String NOME = "nome";
    public static final String ENDERECO = "endereço";
    public static final String FOTO = "foto";
    public static final String DATANASCIMENTO = "dt_nascimento";
    private static final int VERSAO = 1;

    // definimos o nome do nosso banco e qual script queremos executar no início da nossa aplicação
    public AlunoDbHelper(Context context) { super(context, NOME_BANCO, factory: null, VERSAO); }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        // script para a criação da nossa tabela
        String criarBD = "CREATE TABLE " + TABELA + " ("
            + ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"
            + NOME + " TEXT,"
            + ENDERECO + " TEXT,"
            + DATANASCIMENTO + " TEXT,"
            + FOTO + " TEXT)";
        sqLiteDatabase.execSQL(criarBD);
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int il) {
        // script para atualização
        sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + TABELA);
        onCreate(sqLiteDatabase);
    }
}
```

Figura 103 - AlunoDbHelper - definindo o esquema do nosso banco de dados.

Copiável:

```
public class AlunoDbHelper extends SQLiteOpenHelper {

    // propriedades para o nosso banco de dados
    private static final String NOME_BANCO = "alunos.db";
    public static final String TABELA = "alunos";
    public static final String ID = "_id";
    public static final String NOME = "nome";
    public static final String ENDERECO = "endereço";
    public static final String FOTO = "foto";
    public static final String DATANASCIMENTO = "dt_nascimento";
    private static final int VERSAO = 1;

    // definimos o nome do nosso banco e qual script queremos executar no início da nossa aplicação
    public AlunoDbHelper(Context context) {
        super(context, NOME_BANCO, null, VERSAO);
    }
}
```

```

@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
    // script para a criação da nossa tabela
    String criarBD = "CREATE TABLE " + TABELA + " ("
        + ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"
        + NOME + " TEXT,"
        + ENDERECO + " TEXT,"
        + DATANASCIMENTO + " TEXT,"
        + FOTO + " TEXT)";
    sqLiteDatabase.execSQL(criarBD);
}

@Override
public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int
i1) {
    // script para atualização
    sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + TABELA);
    onCreate(sqLiteDatabase);
}
}

```

Uma vez que definimos o esquema do nosso banco de dados, precisamos definir os nossos métodos de inserção, seleção de um item. O que difere é que ao invés de salvarmos os dados em nossa lista e perdermos estes dados sempre que reiniciarmos o app, será que salvaremos ele no sqlite.

Definindo os métodos

```

public class AlunoDaoDB {

    private SQLiteDatabase db;
    private AlunoDbHelper dbo;

    public AlunoDaoDB (Context context) { dbo = new AlunoDbHelper(context); }

    public void salvar(Aluno aluno) {...}

    public List<Aluno> getList() {...}

    public Aluno localizar(Long id) {...}

    public void remover(Aluno aluno) {...}

}

```

Figura 104 - Métodos do nosso dao com SQLite.

Copiável:

```

public class AlunoDaoDB {

    private SQLiteDatabase db;
    private AlunoDbHelper dbo;

    public AlunoDaoDB (Context context) {
        dbo = new AlunoDbHelper(context);
    }

    public void salvar(Aluno aluno) {

        SQLiteDatabase db = dbo.getWritableDatabase();

        /* long resultado;
        ContentValues values = new ContentValues();
        values.put(JogoDbHelper.NOME, jogo.getNome());
        values.put(JogoDbHelper.FABRICANTE, jogo.getFabricante());
        resultado = db.insert(JogoDbHelper.TABELA,
            null,
            values);
        if (resultado == -1) {
            Log.d("Erro ao inserir", "Erro");
            return "Erro ao inserir registro";
        } else {
            Log.d("Sucesso ao inserir", "Sucesso");
            return "Registro inserido com sucesso";
        } */

        if (aluno.getId() == null) {
            String inserir = "insert into "
                + AlunoDbHelper.TABELA
                + " (nome, endereco, foto, dt_nascimento) values
                (?, ?, ?, ?)";
            db.execSQL(inserir, new Object[]{aluno.getNome(),
aluno.getEndereco(), aluno.getFotoAluno(),
aluno.getDataNascimento()});
            // try {
            //     db.execSQL(inserir, new Object[]{aluno.getNome(),
aluno.getEndereco(), aluno.getFotoAluno().getBytes("UTF-8"),
aluno.getDataNascimento()});
            // } catch (UnsupportedEncodingException e) {
            //     e.printStackTrace();
            // }
        } else {
            String update = "update " + AlunoDbHelper.TABELA + " set
            nome = ?, endereco = ?, foto = ?, dt_nascimento = ? where _id = ?";
            db.execSQL(update, new Object[]{aluno.getNome(),
aluno.getEndereco(), aluno.getFotoAluno(), aluno.getDataNascimento(),
aluno.getId()});
            // try {
            //     db.execSQL(update, new Object[]{aluno.getNome(),
aluno.getEndereco(), aluno.getFotoAluno().getBytes("UTF-8"),
aluno.getDataNascimento(), aluno.getId()});
            // } catch (UnsupportedEncodingException e) {
            //     e.printStackTrace();
            // }
        }

        db.close();
    }
}

```

```

    public List<Aluno> getLista() {
        List<Aluno> alunos = new LinkedList<>();
        String rawQuery = "SELECT _id, nome, endereco, foto,
dt_nascimento FROM " + AlunoDbHelper.TABELA;
        SQLiteDatabase db = dbo.getReadableDatabase();
        Cursor cursor = db.rawQuery(rawQuery, null);
        Aluno aluno = null;
        if (cursor.moveToFirst()) {
            do {
                aluno = new Aluno();
                aluno.setId(cursor.getLong(0));
                aluno.setNome(cursor.getString(1));
                aluno.setEndereco(cursor.getString(2));
                aluno.setFotoAluno(cursor.getString(3));
                aluno.setDataNascimento(new
Date(cursor.getString(4)));
                alunos.add(aluno);
            } while (cursor.moveToNext());
        }
        return alunos;
    }

    public Aluno localizar(Long id) {
        SQLiteDatabase db = dbo.getWritableDatabase();
        String query = "SELECT _id, nome, endereco, foto,
dt_nascimento FROM " + AlunoDbHelper.TABELA + " WHERE _id = ?";
        Cursor cursor = db.rawQuery(query, new
String[]{String.valueOf(id)});
        cursor.moveToFirst();
        Aluno alunoA = new Aluno();
        alunoA.setId(cursor.getLong(0));
        alunoA.setNome(cursor.getString(1));
        alunoA.setEndereco(cursor.getString(2));
        alunoA.setFotoAluno(cursor.getString(3));
        alunoA.setDataNascimento(new Date(cursor.getString(4)));
        db.close();
        return alunoA;
    }

    public void remover(Aluno aluno) {
        SQLiteDatabase db = dbo.getWritableDatabase();

        /* String where = JogoDbHelper.ID + "=" + jogo.getId();
        db.delete(JogoDbHelper.TABELA, where, null); */
        String deletar = "delete from " + AlunoDbHelper.TABELA + "
where _id = ?";
        db.execSQL(deletar, new Object[]{aluno.getId()});
        db.close();
    }
}

```

Instanciando

Precisamos mudar e alterar as nossas classes que estão chamando os nossos Dao's, para que, ao invés de chamar o nosso dao da lista, o nosso dao do banco de dados, seja chamado.

```
private AlunoDaoDB dao = new AlunoDaoDB(this);
```

Conteúdo Extra

Evolução das listas

O *RecyclerView* é uma “evolução” das listas *ListView* e *GridView*.

Uma das vantagens do *RecyclerView* é que ele é constantemente atualizado. Porém, como ela não está presente diretamente na SDK do Android (como no caso da *ListView*, por exemplo), nós precisamos incluir essa biblioteca no nosso *gradle*.

Como o nome do componente sugere (*Recycler* é Reciclar em inglês) quando o usuário descer/subir a lista o componente identifica as *views* que não estão mais visíveis para o usuário e as **reutiliza** colocando novos valores. Fazendo isso evita-se criar novas *views* para cada novo conteúdo. O objetivo é reduzir o tempo e custo, reaproveitando objetos.

O usuário pode também definir sua própria animação ao remover ou adicionar um novo conteúdo na lista. Além disso, podemos definir nossa lista horizontal, vertical, grade sem a necessidade de recriar todo o nosso *RecyclerView*.

Sendo assim, olhando o cenário, temos os seguintes componentes:

RecyclerView: irá posicionar a lista na tela (interface) para o usuário.

LayoutManager: Identificamos se os itens serão mostrados horizontalmente, verticalmente.

Adapter: associar o conteúdo à view. Exibição dos nossos itens na lista.

ViewHolder: precisamos mostrar cada item da nossa lista.

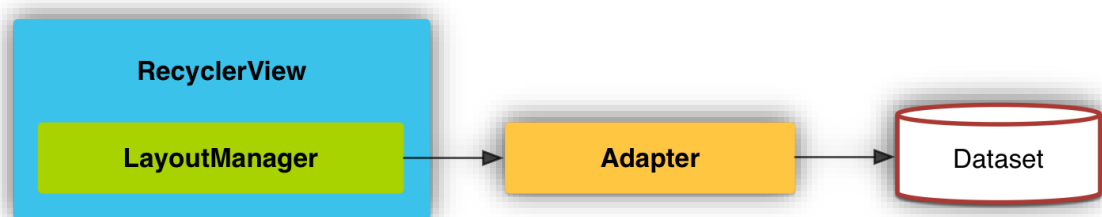


Figura 105 - Estrutura do *RecyclerView*.

<https://developer.android.com/training/material/lists-cards.html>

Passo-a-passo

Precisamos de alguns passos chaves para utilizarmos o *RecyclerView*:

- Adicionar o suporte da biblioteca do *RecyclerView* em nosso projeto (arquivo do *gradle*);

- Definirmos as nossas classes de modelo (model, dao, db - listas);
- Adicionar o nosso RecyclerView em nossa activity para mostrarmos os itens da lista anterior;
- Criarmos uma linha personalizada no nosso layout XML para visualizarmos o item da lista;
- Criarmos o nosso RecyclerView.Adapter e o nosso ViewHolder para renderizarmos o item;
- Vincularmos o nosso adapter a nossa fonte de dados para popularmos o RecyclerView.

build.gradle

Primeiramente vamos incluir a nossa dependência no build.gradle uma vez que ela não faz parte da nossa SDK.

```
// recyclerView
compile 'com.android.support:recyclerview-v7:26.1.0'
```

Como os nossos modelos de dados já foram definidos anteriormente, o que queremos realizar agora é trocar a nossa ListView para a RecyclerView.

Portanto, vamos alterar a nossa content_main.xml.



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingView..."
    tools:context="br.senai.sp.android.fic_escolas_dev.views.MainActivity"
    tools:showIn="@layout/activity_main">

    <!-- <ListView
        android:id="@+id/lvListaAlunos"
        android:layout_width="match_parent"
        android:layout_height="match_parent" /> -->

    <android.support.v7.widget.RecyclerView
        android:id="@+id/rvListaAlunos"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</android.support.constraint.ConstraintLayout>
```

Figura 106 - Alterando do ListView para o RecyclerView em nossa lista.

Copiável:

```
<!-- <ListView
    android:id="@+id/lvListaAlunos"
    android:layout_width="match_parent"
    android:layout_height="match_parent" /> -->
```

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/rvListaAlunos"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Nós iremos utilizar o mesmo layout que criamos para a nossa ListView. Porém, no caso da RecyclerView nós precisamos tanto do Adapter quanto do ViewHolder.

Layout personalizado

Item_simple_aluno_lista_rv.xml

Copiável:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="?android:selectableItemBackground"
    android:clickable="true"
    android:focusable="true"
    android:foreground="?android:attr/selectableItemBackground"
    android:orientation="vertical"
    android:padding="8dp">

    <TextView
        android:id="@+id/tvNomeAlunoRv"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Nome do Aluno"
        android:padding="4dp"
        android:textAlignment="center"
        android:textSize="20dp" />

    <TextView
        android:id="@+id/tvDataNascimentoAlunoRv"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Data de Nascimento"
        android:padding="12dp"
        android:textAlignment="center" />

</LinearLayout>
```

Adapter e ViewHolder

Precisamos criar o nosso adapter que irá de fato popular os dados para o RecyclerView. No entanto, ele requer a existência de um objeto “ViewHolder” que irá descrever e fornecer o acesso a cada item da nossa lista.

Apenas para organizar o nosso projeto, vamos criar um novo pacote chamado rv (de RecyclerView) para diferenciar das classes que já criamos.

E dentro deste pacote, mais dois. ‘adapter’ e ‘holder’.

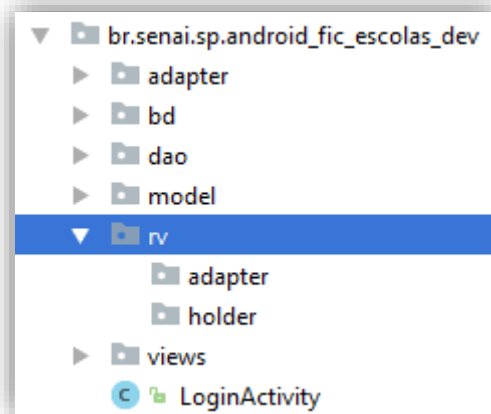


Figura 107 - Ajustando os pacotes do RecyclerView.

Vamos criar mais duas classes novas dentro de cada pacote.

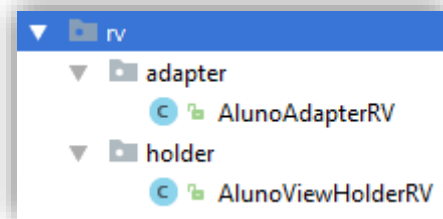


Figura 108 - Criando as classes correspondentes para o RecyclerView.

Adapter

Todo adapter possui três métodos principais: o `onCreateViewHolder` para "inflar" o item que desejamos. `onBindViewHolder` para "settar" os atributos da view baseado nos dados de cada filme. E o `getItemCount` para determinar o número de itens.

```

public class AlunoAdapterRV extends RecyclerView.Adapter {

    private List<Aluno> alunos;
    private Context context;

    public AlunoAdapterRV(List<Aluno> alunos, Context context) {
        this.alunos = alunos;
        this.context = context;
    }

    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(context)
            .inflate(R.layout.item_simple_aluno_lista, parent, attachToRoot: false);
        AlunoViewHolderRV holder = new AlunoViewHolderRV(view, adapter: this);
        return holder;
    }

    @Override
    public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {
        AlunoViewHolderRV viewHolder = (AlunoViewHolderRV) holder;

        Aluno aluno = alunos.get(position);

        ((AlunoViewHolderRV) holder).preencher(aluno);
    }

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

```

Figura 109 - Arrumando o adapter do RecyclerView.

Copiável:

```

public class AlunoAdapterRV extends RecyclerView.Adapter {

    private List<Aluno> alunos;
    private Context context;

    public AlunoAdapterRV(List<Aluno> alunos, Context context) {
        this.alunos = alunos;
        this.context = context;
    }

    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup
parent, int viewType) {
        View view = LayoutInflater.from(context)
            .inflate(R.layout.item_simple_aluno_lista, parent,
false);
        AlunoViewHolderRV holder = new AlunoViewHolderRV(view, this);
        return holder;
    }
}

```

```

    }

    @Override
    public void onBindViewHolder(RecyclerView.ViewHolder holder, int
position) {
        AlunoViewHolderRV viewHolder = (AlunoViewHolderRV) holder;

        Aluno aluno = alunos.get(position);

        ((AlunoViewHolderRV) holder).preencher(aluno);

    }

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

```

ViewHolder

```

public class AlunoViewHolderRV extends RecyclerView.ViewHolder {

    public final TextView nomeAluno;
    public final TextView dataNascimentoAluno;
    private Long alunoId;
    public final AlunoAdapterRV adapter;

    public AlunoViewHolderRV(View itemView, AlunoAdapterRV adapter) {
        super(itemView);
        this.adapter = adapter;
        nomeAluno = itemView.findViewById(R.id.tvNomeAluno);
        dataNascimentoAluno = itemView.findViewById(R.id.tvDataNascimentoAluno);
    }

    public void preencher(Aluno aluno) {
        alunoId = aluno.getId();
        nomeAluno.setText(aluno.getNome());
        dataNascimentoAluno.setText(DateFormat.getDateInstance(DateFormat.MEDIUM).format(aluno.getDataNascimento()));
    }
}

```

Figura 110 - Ajustando o ViewHolder do nosso RecyclerView.

Copiável:

```

public class AlunoViewHolderRV extends RecyclerView.ViewHolder {

    public final TextView nomeAluno;
    public final TextView dataNascimentoAluno;
    private Long alunoId;
    public final AlunoAdapterRV adapter;

    public AlunoViewHolderRV(View itemView, AlunoAdapterRV adapter) {
        super(itemView);
        this.adapter = adapter;
        nomeAluno = itemView.findViewById(R.id.tvNomeAluno);
        dataNascimentoAluno =

```

```

itemView.findViewById(R.id.tvDataNascimentoAluno);
    }

    public void preencher(Aluno aluno) {
        alunoId = aluno.getId();
        nomeAluno.setText(aluno.getNome());

        dataNascimentoAluno.setText(DateFormat.getDateInstance(DateFormat.MEDIUM).format(aluno.getDataNascimento()));
    }
}

```

Precisamos apenas alterar a nossa MainActivity para que, ao invés de carregarmos o nosso ListView que comentamos, para que ele trabalhe com o nosso RecyclerView.

Selecionando um item da lista

Holder

```

package br.senai.sp.android_fic_escolas_dev.rv.holder;

import android.app.Activity;
import android.content.Intent;
import android.support.v7.widget.RecyclerView;
import android.view.MenuItem;
import android.view.View;
import android.widget.PopupMenu;
import android.widget.TextView;

import java.text.DateFormat;

import br.senai.sp.android_fic_escolas_dev.R;
import br.senai.sp.android_fic_escolas_dev.bd.AlunoDaoDB;
import br.senai.sp.android_fic_escolas_dev.model.Aluno;
import br.senai.sp.android_fic_escolas_dev.rv.adapter.AlunoAdapterRV;
import br.senai.sp.android_fic_escolas_dev.views.FormActivity;
import br.senai.sp.android_fic_escolas_dev.views.MainActivity;
import br.senai.sp.android_fic_escolas_dev.views.MapsActivity;

/**
 * Created by Helena Strada on 10/04/2018.
 */

public class AlunoViewHolderRV extends RecyclerView.ViewHolder
implements View.OnClickListener, View.OnLongClickListener {

    public final TextView nomeAlunoRv;
    public final TextView dataNascimentoAlunoRv;
    private Long alunoId;
    public final AlunoAdapterRV adapter;

    public AlunoViewHolderRV(View itemView, AlunoAdapterRV adapter) {

```



```

        super(itemView);
        this.adapter = adapter;

        itemView.setOnClickListener(this);
        itemView.setOnLongClickListener(this);

        nomeAlunoRv = itemView.findViewById(R.id.tvNomeAluno);
        dataNascimentoAlunoRv =
itemView.findViewById(R.id.tvDataNascimentoAluno);
    }

    public void preencher(Aluno aluno) {
        alunoId = aluno.getId();
        nomeAlunoRv.setText(aluno.getNome());

        dataNascimentoAlunoRv.setText(DateFormat.getDateInstance(DateFormat.MEDIUM).format(aluno.getDataNascimento()));
    }

    @Override
    public void onClick(View view) {
        final Activity context = (Activity) view.getContext();
        final Intent intent = new Intent(context, FormActivity.class);
        intent.putExtra("idAluno", alunoId);
        context.startActivityForResult(intent, 1);
    }

    @Override
    public boolean onLongClick(View view) {
        PopupMenu popup = new PopupMenu(view.getContext(), view);
        popup.getMenuInflater().inflate(R.menu.menu_aluno_options,
popup.getMenu());

        final Activity context = (Activity) view.getContext();

        popup.setOnMenuItemClickListener(new
PopupMenu.OnMenuItemClickListener() {
            public boolean onMenuItemClick(MenuItem item) {

                switch (item.getItemId()) {

                    case R.id.menuAlunoEditar:
                        final Intent intent = new Intent(context,
FormActivity.class);
                        intent.putExtra("idAluno", alunoId);
                        context.startActivityForResult(intent, 1);
                        break;

                    case R.id.menuAlunoDeletar:
                        AlunoDaoDB dao = new AlunoDaoDB(context);
                        dao.remover(dao.localizar(alunoId));
                        deletarAluno();
                        break;

                    case R.id.menuAlunoMostrarMapa:
                        Intent intentMapa = new Intent(context,
MapsActivity.class);
                        intentMapa.putExtra("idAluno", alunoId);
                        context.startActivity(intentMapa);

                }
            }
        });
    }

```

```

        return true;
    }
});

popup.show();
return false;
}

public void deletarAluno() {
    adapter.remove(getAdapterPosition());
}
}

```

Adapter

```

package br.senai.sp.android_fic_escolas_dev.rv.adapter;

import android.content.Context;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import java.util.List;

import br.senai.sp.android_fic_escolas_dev.R;
import br.senai.sp.android_fic_escolas_dev.model.Aluno;
import br.senai.sp.android_fic_escolas_dev.rv.holder.AlunoViewHolderRV;

/**
 * Created by Helena Strada on 10/04/2018.
 */

public class AlunoAdapterRV extends RecyclerView.Adapter {

    private List<Aluno> alunos;
    private Context context;

    public AlunoAdapterRV(List<Aluno> alunos, Context context) {
        this.alunos = alunos;
        this.context = context;
    }

    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(context)
            .inflate(R.layout.item_simple_aluno_lista, parent,
false);
        AlunoViewHolderRV holder = new AlunoViewHolderRV(view, this);
        return holder;
    }
}

```

```

@Override
public void onBindViewHolder(RecyclerView.ViewHolder holder, int
position) {
    AlunoViewHolderRV viewHolder = (AlunoViewHolderRV) holder;

    Aluno aluno = alunos.get(position);

    ((AlunoViewHolderRV) holder).preencher(aluno);

}

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

public void remove(int position) {
    alunos.remove(position);
    notifyItemRemoved(position);
    notifyItemRangeChanged(position, getItemCount());
}
}

```

Introdução WebServices

É utilizado quando precisamos transferir dados através de protocolos de comunicação em diferentes plataformas. Identificado através de uma URL. A sua comunicação é via protocolo de internet – HTTP.

Vantagens

Integração entre sistemas: independente da linguagem de programação;

Segurança: Regras de Negócio e acesso ao banco de dados;

Implementação: Mais simples de implementar.

REST

Representational State Transfer

Em 2000, Roy Fielding apresentou como tese de doutorado um conjunto de regras a ser aplicada à arquitetura de sistemas distribuídos.

Protocolo de comunicação.

Recursos

Definidas através de URL's.

<http://localhost:8080/usuario/helena>

<http://localhost:8080/usuario/1>

Representação

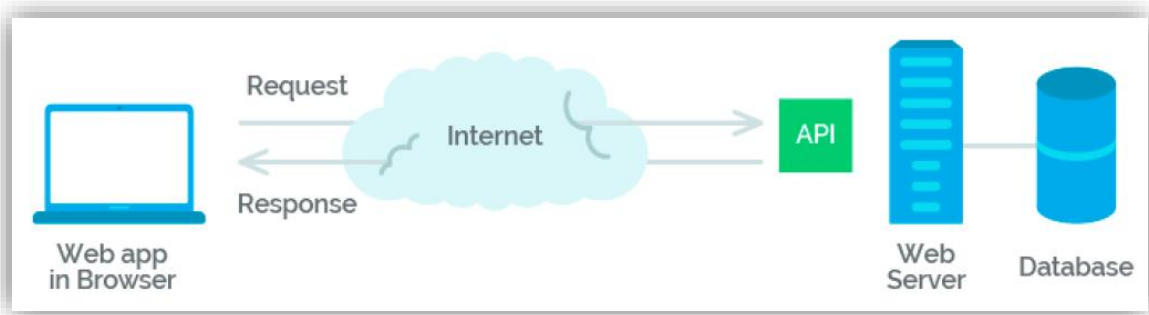
Irá transferir um JSON ou XML para representar os objetos e atributos.

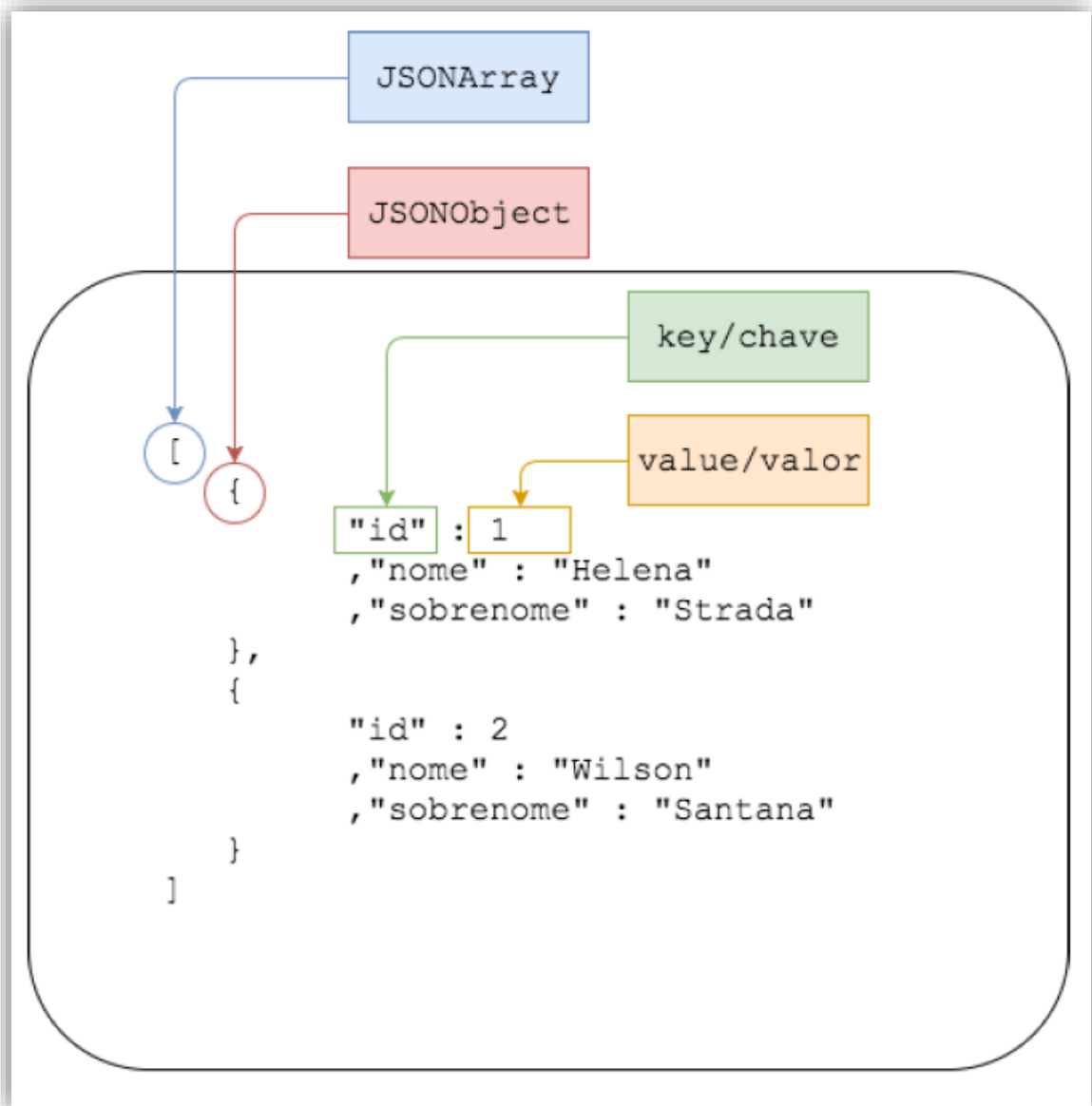
Mensagens

Você pode aplicar algumas ações sobre os recursos (URL's). Métodos HTTP.

GET, PUT, DELETE, POST.

Padrão





Retrofit

O Retrofit é uma biblioteca *open source* que facilita o processo de realização de requisições. Não precisamos fazer manualmente a serialização dos objetos. Necessidade de definir apenas a nossa conexão.

<http://square.github.io/retrofit/>

Inclua no build.gradle no nível da aplicação, a biblioteca do Retrofit.

```
// retrofit library  
compile 'com.google.code.gson:gson:2.8.2'
```

```
compile 'com.squareup.retrofit2:retrofit:2.3.0'  
compile 'com.squareup.retrofit2:converter-gson:2.3.0'  
  
// okhttp  
compile 'com.squareup.okhttp3:okhttp:3.9.1'
```

Vamos criar um novo pacote chamado config e dentro dele um subpacote chamado service.

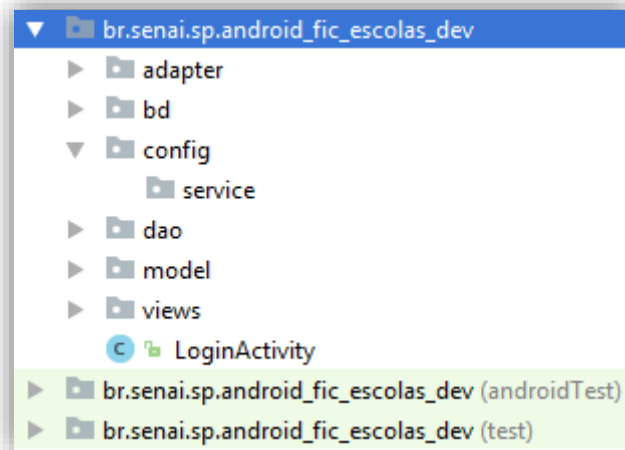


Figura 111 - Pacotes para colocarmos nossas classes de configuração do REST.

Vamos criar uma classe de configuração do Retrofit, e outra classe que ficará responsável por 'determinar', organizar, as nossas chamadas à API.

A classe do RetrofitConfig ficará dentro do pacote config e a nossa interface ficará dentro do pacote do rest.

```

public class RetrofitConfig {

    // atributo
    private final Retrofit retrofit;
    private String urlConexao = "http://10.0.2.2:8080/api/";

    // Nossa configuração será feita no construtor
    public RetrofitConfig() {

        // Precisamos construir um objeto do tipo retrofit
        this.retrofit = new Retrofit.Builder()
            // definimos a url base da nossa aplicação
            .baseUrl(urlConexao)
            // precisamos transformar a nossa resposta que vem em JSON para String
            .addConverterFactory(GsonConverterFactory.create())
            // precisamos de fato criá-lo
            .build();
    }

    public RetrofitConfig(OkHttpClient client) {
        // Precisamos construir um objeto do tipo retrofit
        this.retrofit = new Retrofit.Builder()
            // definimos a url base da nossa aplicação
            .baseUrl(urlConexao)
            // caso seja necessário colocar um interceptor
            .client(client)
            // precisamos transformar a nossa resposta que vem em JSON para String
            .addConverterFactory(GsonConverterFactory.create())
            // precisamos de fato criá-lo
            .build();
    }

    public RestInterface getRestInterface() {
        return this.retrofit.create(RestInterface.class);
    }
}

```

Figura 112 - Classe de configuração do Retrofit.

Copiável:

```

/*
 *
 * Essa classe ficará responsável por configurar e instanciar o
 * Retrofit
 *
 * */
public class RetrofitConfig {

    // atributo
    private final Retrofit retrofit;
    private String urlConexao = "http://10.0.2.2:8080/api/";

    // Nossa configuração será feita no construtor
    public RetrofitConfig() {

```

```

        // Precisamos construir um objeto do tipo retrofit
        this.retrofit = new Retrofit.Builder()
            // definimos a url base da nossa aplicação
            .baseUrl(urlConexao)
            // precisamos transformar a nossa resposta que vem em
JSON para String
            .addConverterFactory(GsonConverterFactory.create())
            // precisamos de fato criá-lo
            .build();
    }

    public RetrofitConfig(OkHttpClient client) {
        // Precisamos construir um objeto do tipo retrofit
        this.retrofit = new Retrofit.Builder()
            // definimos a url base da nossa aplicação
            .baseUrl(urlConexao)
            // caso seja necessário colocar um interceptor
            .client(client)
            // precisamos transformar a nossa resposta que vem em
JSON para String
            .addConverterFactory(GsonConverterFactory.create())
            // precisamos de fato criá-lo
            .build();
    }

    public RestInterface getRestInterface() {
        return this.retrofit.create(RestInterface.class);
    }
}

```

```

public interface RestInterface {

    @POST("auth/login")
    Call<ResponseBody> buscarLogin(@Body Usuario usuario);

    @GET("alunos")
    Call<List<Aluno>> listarClientes();

    @POST("alunos")
    Call<Aluno> salvarCliente(@Body Aluno aluno);

    @PUT("alunos")
    Call<Aluno> atualizarCliente(@Body Aluno aluno);

    @GET("alunos/{id}")
    Call<Aluno> buscarCliente(@Path("id") Long id);
}

```

Figura 113 - Classe RestInterface.

Copiável:

```
import okhttp3.ResponseBody;
import retrofit2.Call;
import retrofit2.http.Body;
import retrofit2.http.GET;
import retrofit2.http.POST;
import retrofit2.http.PUT;
import retrofit2.http.Path;

public interface RestInterface {

    @POST("auth/login")
    Call<ResponseBody> buscarLogin(@Body Usuario usuario);

    @GET("alunos")
    Call<List<Aluno>> listarClientes();

    @POST("alunos")
    Call<Aluno> salvarCliente(@Body Aluno aluno);

    @PUT("alunos")
    Call<Aluno> atualizarCliente(@Body Aluno aluno);

    @GET("alunos/{id}")
    Call<Aluno> buscarCliente(@Path("id") Long id);

}
```

A diferença agora, é que precisamos alterar o nosso modelo para que as características sejam iguais as do JSON que estamos enviando e que as chamadas, não sejam feitas mais para o SQLite (nosso bd local) e sim para o endpoint da API.

```
@SerializedName("username")
private String email;

@SerializedName("password")
private String senha;
```

Figura 114 - Colocando o valor igual ao que chamamos na API.

```

Call<ResponseBody> call = new RetrofitConfig().getRestInterface().buscarLogin(usuarioConferirNaLista);
call.enqueue(new Callback<ResponseBody>() {
    @Override
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
        if (response.isSuccessful()) {
            Intent intent = new Intent( packageContext: LoginActivity.this, MainActivity.class);
            startActivity(intent);
            LoginActivity.this.finish();
        }
    }

    @Override
    public void onFailure(Call<ResponseBody> call, Throwable t) {
        Toast.makeText(getApplicationContext(), text: "Usuário ou senha incorretos.", Toast.LENGTH_SHORT).show();
    }
});

```

Figura 115 - Fazendo a chamada ao login.

Referências

https://www.androidpro.com.br/android-sdk/#SDK_Tools

<https://developer.android.com/samples/BasicSyncAdapter/res/values/dimen.html>