UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL

FACULDADE DE COMPUTAÇÃO

RELATÓRIO DE ATIVIDADE ORIENTADA DE ENSINO

Matheus Biesdorf

Orientadora: Profª Dra. Ana Karina D. Salina de Oliveira

PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS:

Um manual para iniciantes em desenvolvimento mobile

CAMPO GRANDE

2026

1. Introdução

O desenvolvimento de aplicativos móveis tornou-se uma das áreas mais relevantes da computação moderna, impulsionado pelo crescimento contínuo do ecossistema Android. Para que os estudantes adquiram autonomia nesse ambiente, é fundamental compreender não apenas as ferramentas utilizadas, mas também a estrutura lógica por trás de um projeto Android. Este manual foi elaborado com o objetivo de fornecer uma orientação clara, objetiva e sequencial sobre os principais passos para iniciar o desenvolvimento na plataforma.

Ao longo deste documento, o leitor encontrará instruções sobre a instalação e configuração do Android Studio , criação de projetos, organização de pastas, utilização de recursos de interface e gerenciamento de dependências. Também será apresentada uma introdução simples ao uso do banco de dados Room, construindo um aplicativo simples para melhor entendimento a respeito do fluxo de funcionamento de um app. Por fim, você aprenderá a rodar para fazer testes manuais.

Este material se destina a iniciantes no desenvolvimento Android, especialmente alunos da área de Computação, e busca servir como referência introdutória para práticas de laboratório, estudos individuais e atividades orientadas.

2. Instalação do Android Studio

2.1 Requisitos de sistema

O Android Studio [1] é compatível com os principais sistemas operacionais atuais, incluindo Windows, Linux, Android e macOS. Neste manual será utilizada a distribuição Ubuntu, uma distribuição do Linux gratuita, muito popular e amigável para iniciantes.

Para utilizar o Android Studio, o computador deve atender a alguns requisitos mínimos e recomendados de sistema, são eles:

Requisito	**Mínimo**	**Recomendado**
Sistema Operacional	Qualquer distribuição de Linux de 64 bits com suporte para Gnome, KDE ou Unity DE e com a Biblioteca C do GNU (glibc) 2.31 ou mais recente	Versão mais recente do Linux de 64 bits
RAM (Memória)	Studio:8 GB Studio e emulador:16 GB	32 GB de RAM ou mais
CPU (Processador)	Suporte à virtualização necessário (Intel VT-x ou AMD-V, ativado no BIOS). Microarquitetura de CPU após 2017 Intel Core de 8ª geração i5 / AMD Zen Ryzen (por exemplo, Intel i5-8xxx, Ryzen 1xxx).	Suporte à virtualização necessário (Intel VT-x ou AMD-V, ativado no BIOS). Microarquitetura de CPU mais recente: procure CPUs das séries Intel Core i5, i7 ou i9 e/ou os sufixos H/HK/HX para laptop ou S/F/K para computador desktop, ou as séries AMD Ryzen 5, 6, 7 ou 9. Os processadores Intel Core das séries N e U não são recomendados devido à performance insuficiente.
GPU (Placa de vídeo)	Studio:nenhuma Studio e emulador:GPU com 4 GB de VRAM, como Nvidia Geforce série 10 ou mais recente, ou AMD Radeon RX 5000 ou mais recente com os drivers mais recentes	GPU com 8 GB de VRAM, como Nvidia Geforce série 20 ou mais recente, ou AMD Radeon RX 6600 ou mais recente com os drivers mais recentes
Resolução de tela	1280 x 800	1920 x 1080
Espaço em disco	Studio:8 GB de espaço livre. Studio e emulador:16 GB de espaço livre.	Unidade de estado sólido (SSD) com 32 GB ou mais

2.2 Download e instalação do Android Studio

Para realizar o download do software você pode acessar o site oficial https://developer.android.com/studio?hl=pt-br, realizar o download e instalar manualmente.
A versão utilizada do Android Studio foi a Narwhal 3 Feature Drop 2025.1.3.

Para que funcione corretamente no Windows, vc deve instalar com a conta de Administrador.
Para o Linux, um jeito mais simples de instalar:

1. Abra o terminal

2. Digite o seguinte comando:

```
sudo snap install android-studio --classic
```

3. Tecle Enter (Talvez o ubuntu peça uma senha ou permissão de administrador)

Feito isso, o Android Studio já estará instalado no seu computador.

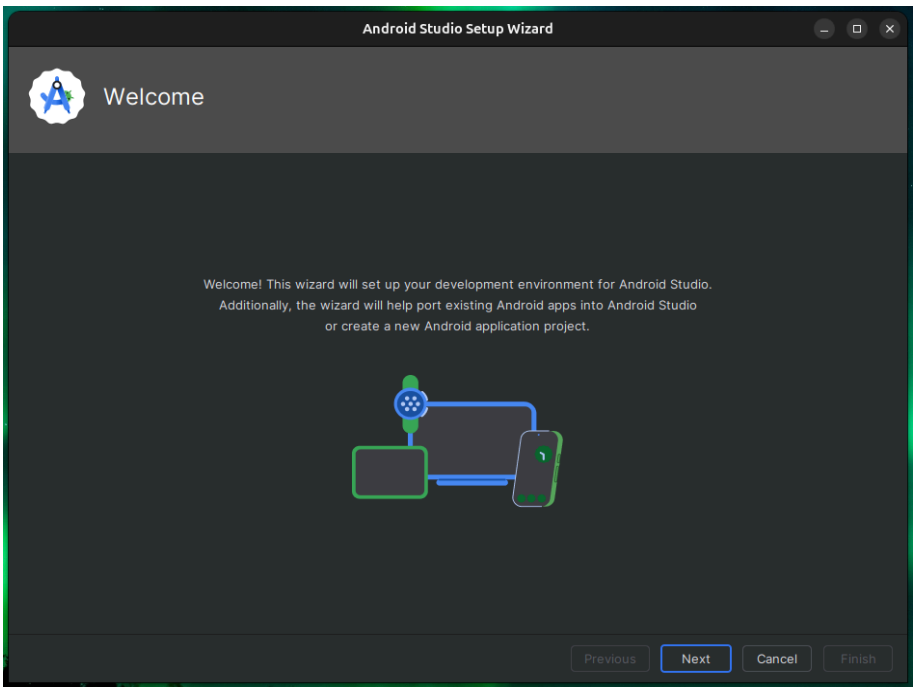
Para abrir o software, você pode procurar nos seus aplicativos ou digitar um dos seguintes comandos no terminal:

```
android-studio
```

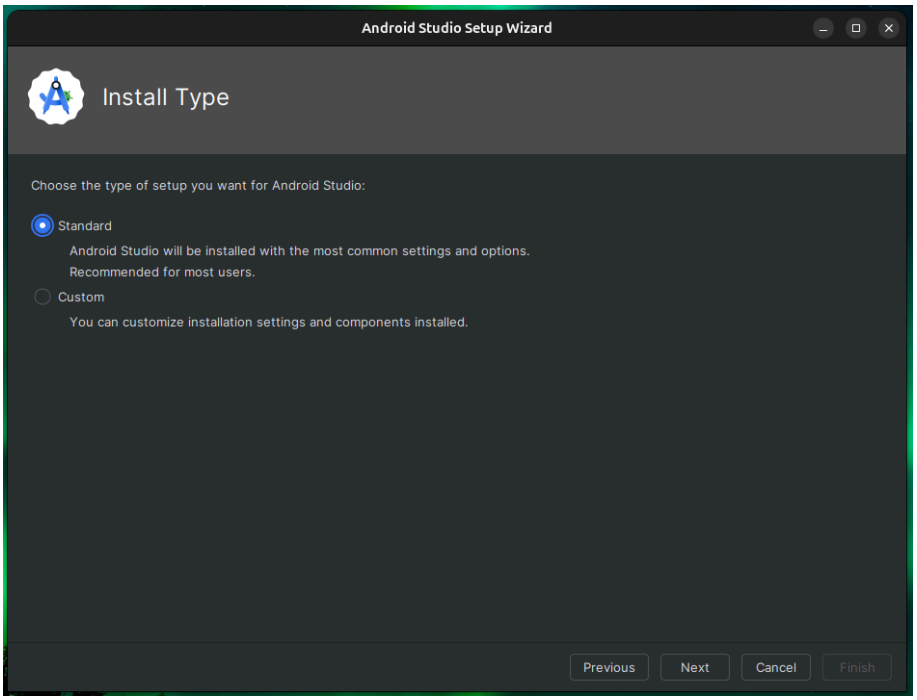
```
snap run android-studio
```

2.3 Configuração inicial

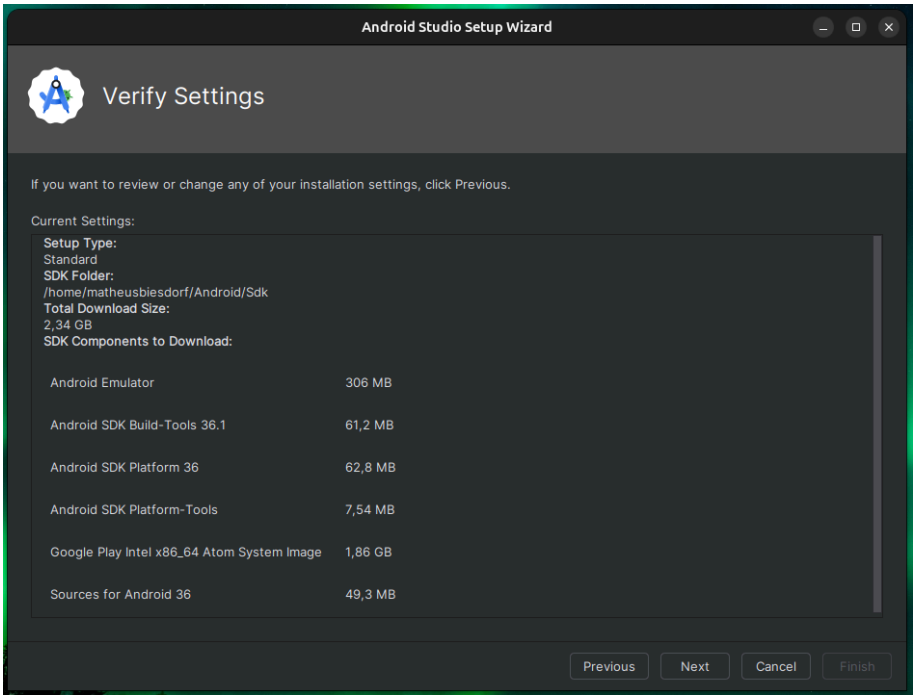
Ao abrir o Android Studio pela primeira vez, nos deparamos com o Setup Wizard, que faz algumas etapas essenciais



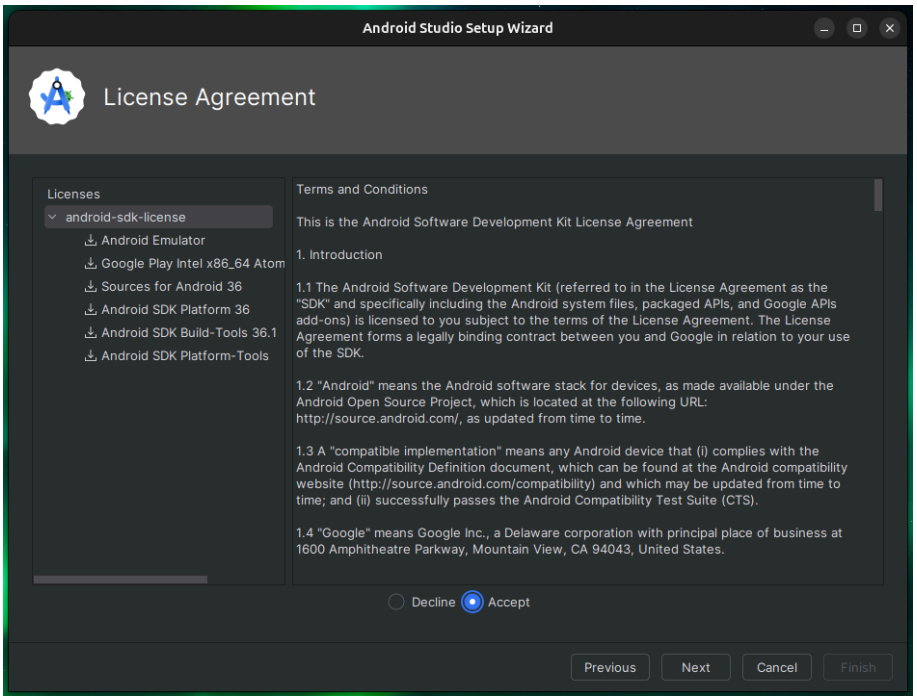
Nesta tela de boas vindas, apenas clique em Next.



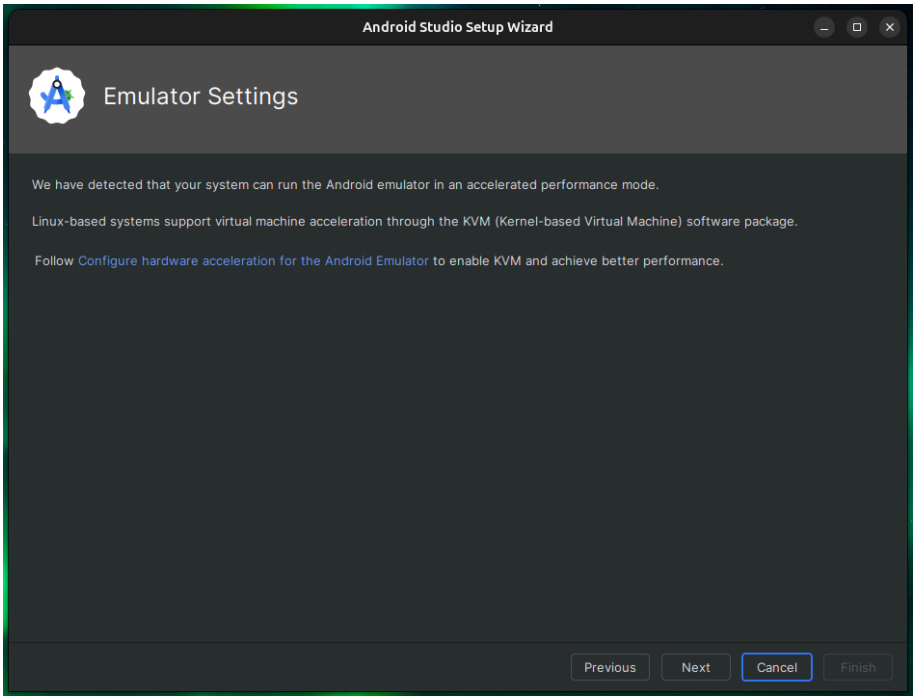
Nesta tela é possível selecionar o tipo de instalação desejado. Selecione Standart (Padrão) e clique em Next.



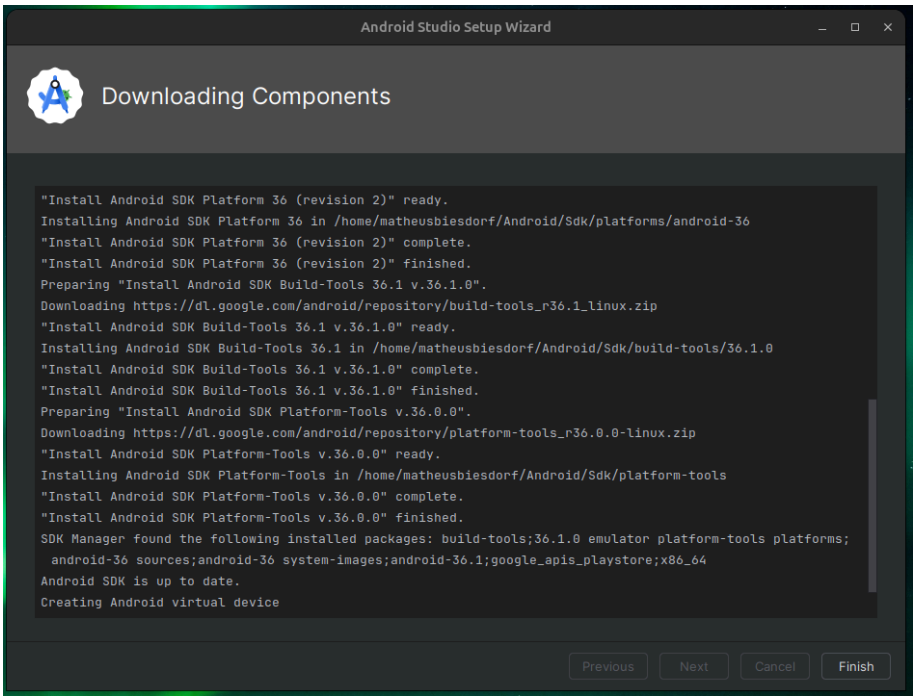
Algumas informações serão exibidas, você pode conferir na tela e seguir com Next.



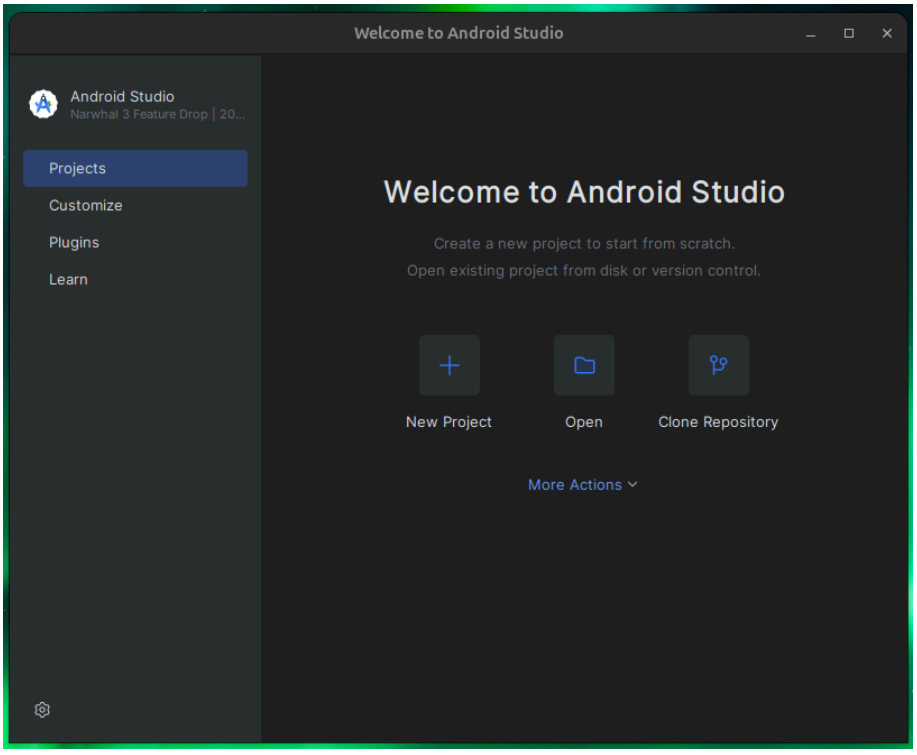
Nesta tela o Android Studio baixará automaticamente o Android SDK, ferramentas de build, emulador Android (AVD) e imagens do sistema para o emulador. Selecione a opção Accept e clique em Next.



Algumas vezes, dependendo do seu hardware, esta tela pode ser exibida, mas você pode ignorá-la por enquanto e aperte em Next.



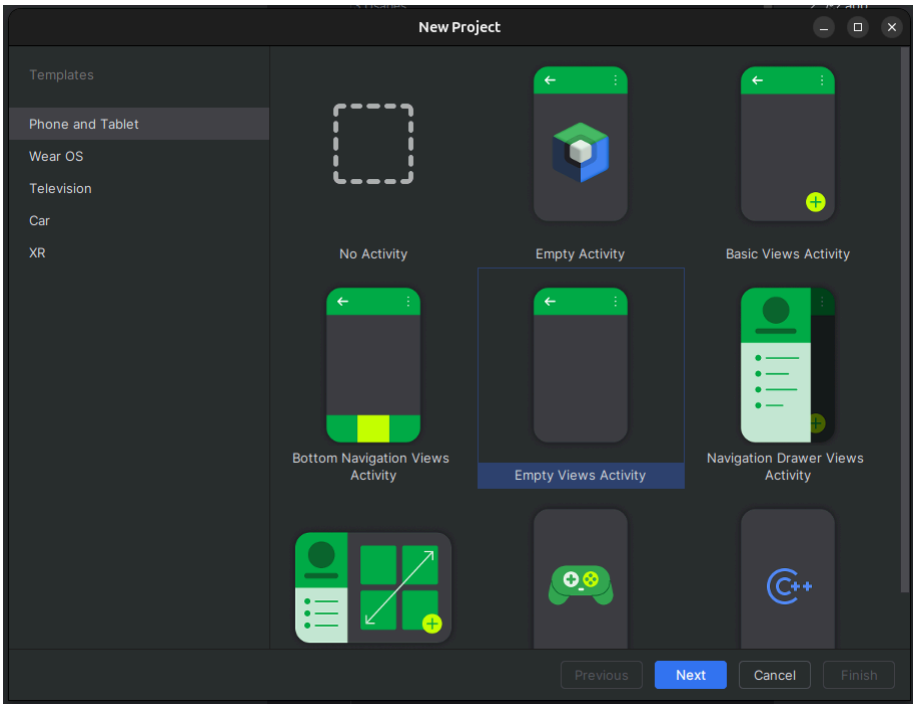
O Android Studio começará a fazer os downloads, espere até que ele termine de baixar tudo e clique em Finish.



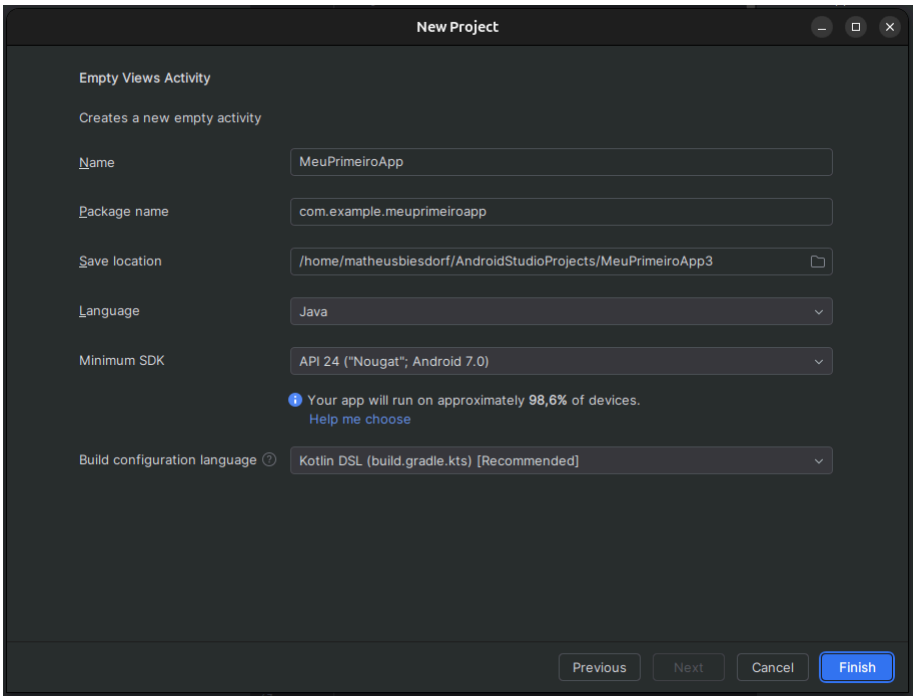
Concluída a instalação, você será direcionado para a tela inicial do Android Studio.

3. Criando um novo projeto Android

Na tela inicial, clique em New Project e você será levado para a seguinte tela:



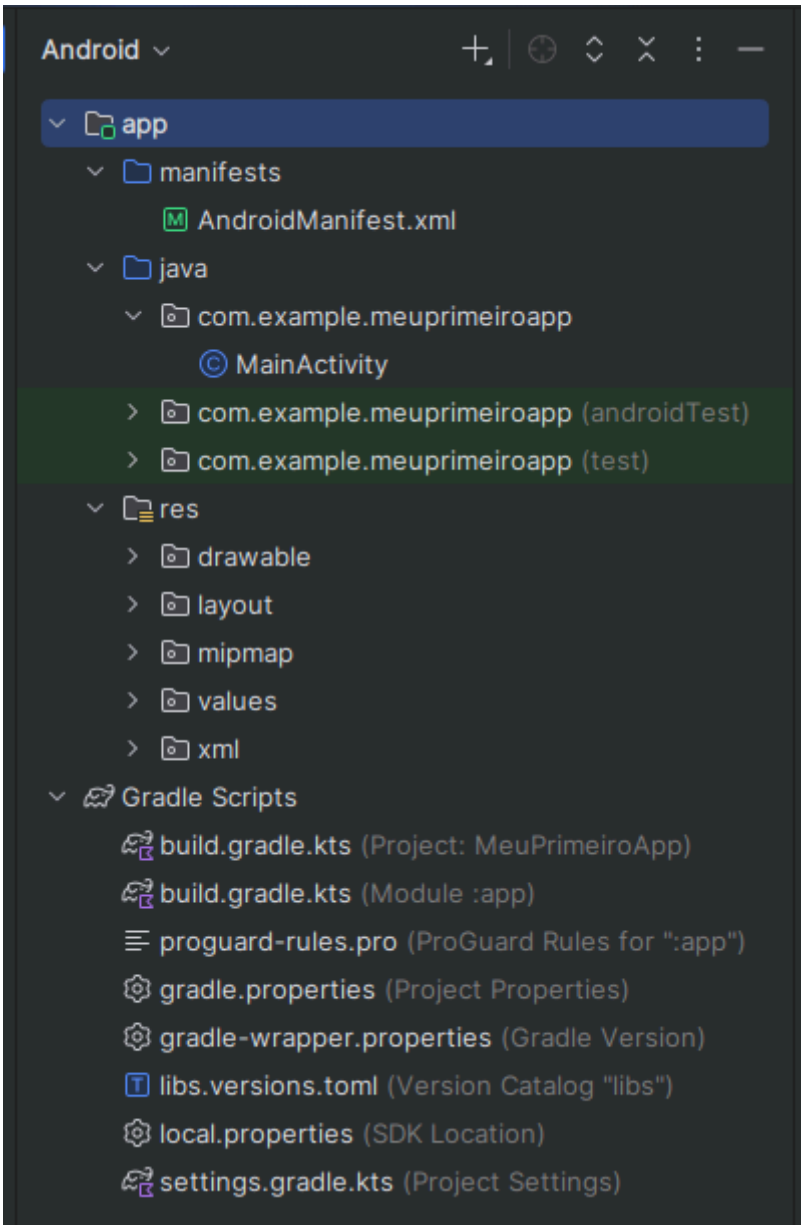
Selecione o template Empty Views Activity e clique em Next.



Escolha um nome para o seu projeto como por exemplo “MeuPrimeiroApp”, selecione Java como a linguagem e clique em Finish. Depois disso, o Android Studio vai criar toda a estrutura do projeto, configurar e sincronizar tudo. Apenas aguarde, se aparecer alguma sugestão de atualização, aceite. Essa etapa pode demorar um pouco, dependendo do seu hardware e conexão com a internet.

4. Estrutura básica de um projeto de desenvolvimento android

Ao criar um novo projeto no Android Studio, uma estrutura de pastas é criada da seguinte forma:



4.1 Pasta manifests/

O arquivo AndroidManifest.xml contém informações essenciais sobre o aplicativo, como:

- Activities declaradas
- Permissões necessárias (Ex: acesso à internet)
- Nome do aplicativo
- Configurações gerais de inicialização

O Android utiliza esse arquivo para saber como seu app deve funcionar em nível do sistema.

4.2 Pasta java/ (ou kotlin+java/)

Aqui ficam todos os arquivos do código-fonte, incluindo:

- **Activities:** Classes que representam telas do aplicativo (Ex: MainActivity.java)
- **Fragments:** Componentes de interface reutilizáveis que funcionam dentro de uma Activity (Ex: HomeFragment.java)
- **Modelos (Models):** Representam dados do aplicativo. São classes usadas para guardar informações. (Exemplo: um usuário, um produto, um item de lista)
- **Controladores (Controllers):** Coordenam ações e regras do aplicativo. Eles recebem ações do usuário, decidem o que fazer e chamam funções de outras classes.
- **Adaptadores (Adapters):** Conectam listas de dados com componentes visuais, como RecyclerView. O adapter recebe uma lista de dados, cria as células (itens visuais) e diz como cada item deve aparecer na tela. Sem um Adapter, listas dinâmicas não funcionam, por exemplo.
- **Repositórios (Repositories):** São classes responsáveis por acessar dados. Eles podem buscar informações de banco de dados (Como o Room), API online ou arquivos internos. O repositório separa a lógica de dados da lógica da interface, deixando o projeto mais organizado.
- **Pastas de teste:** O Android Studio cria automaticamente duas pastas: androitTest (Testes instrumentados - rodam no dispositivo) e test (Testes unitários - rodam na JVM)

4.3 Pasta /res (resources)

Contém todos os recursos gŕaficos e textuais do app.

É um dos diretórios mais importantes do projeto. Dentro dela temos:

- **drawable:** Imagens e gráficos do aplicativo. Pode conter PNG, WEBP, SGV, Sharpes XML, etc.
- **layout:** Arquivos XML que definem a interface das telas. Exemplo: activity_main.xml ou fragment_home.xml. Nelas, você adiciona botões, textos, campos, etc.
- **minimap:** Ícones do aplicativo (launcher icons) em várias resoluções. O Android usa automaticamente a densidade correta para cada tela.
- **values:** Armazena arquivos XML com valores reutilizáveis.
    - colors.xml: Define cores do app
    - strings.xml: Armazena textos usados na interface
    - themes.xml: Define o tema visual do aplicativo (cores primárias, fontes, modo escuro, etc.)
- **xml:** Configurações diversas, como permissões do backup, navigation graphs, etc.

4.4 Arquivos Gradle

O Android usa o Gradle para gerenciar dependências e compilar o projeto.

- build.gradle (Project): Configurações gerais, versão do Kotlin, repositórios, etc.
- build.gradle (Module: app): Onde você adiciona dependências do seu aplicativo. É o arquivo mais usado por iniciantes quando precisam adicionar bibliotecas.

5. Banco de dados Room

Para melhor entendimento do funcionamento do banco de dados Room [5], vamos criar um mini-projeto com um banco de dados. Inicialmente, precisamos deixar nosso banco de dados funcionando direitinho.

5.1 Adicionando dependências ao projeto

Para utilizar o banco de dados room é necessário adicionar algumas dependências na pasta raiz do projeto.

Em Gradle Scripts > build.gradle.kts (Module :app), dentro de dependencies {...} adicione as seguintes linhas de código:

```
implementation("androidx.room:room-runtime:2.6.1")
annotationProcessor("androidx.room:room-compiler:2.6.1")
```

Depois disso, sincronize novamente o projeto

5.2 Estrutura básica do Room: Entity, DAO e Database

Após adicionar as dependências, o próximo passo é configurar a estrutura fundamental do Room, composta por três elementos principais: Entity, DAO e Database. Esses componentes definem como os dados serão armazenados, acessados e gerenciados dentro do aplicativo.

Todos esses arquivos podem ser criados dentro da pasta:

app/java/com.example.meuprimeiroapp/

Uma Entity representa uma tabela do banco de dados. Cada objeto dessa classe corresponde a um registro salvo.

Nota.java

```java
package com.example.meuprimeiroapp;

import androidx.room.ColumnInfo;
import androidx.room.Entity;
import androidx.room.PrimaryKey;

no usages
@Entity(tableName = "notas")
public class Nota {

    no usages
    @PrimaryKey(autoGenerate = true)
    public int id;

    1 usage
    @ColumnInfo(name = "texto")
    public String texto;

    no usages
    public Nota(String texto) {
        this.texto = texto;
    }
}
```

O DAO (Data Access Object) define os métodos usados para inserir, excluir e consultar informações.

Nota.java NotaDao.java

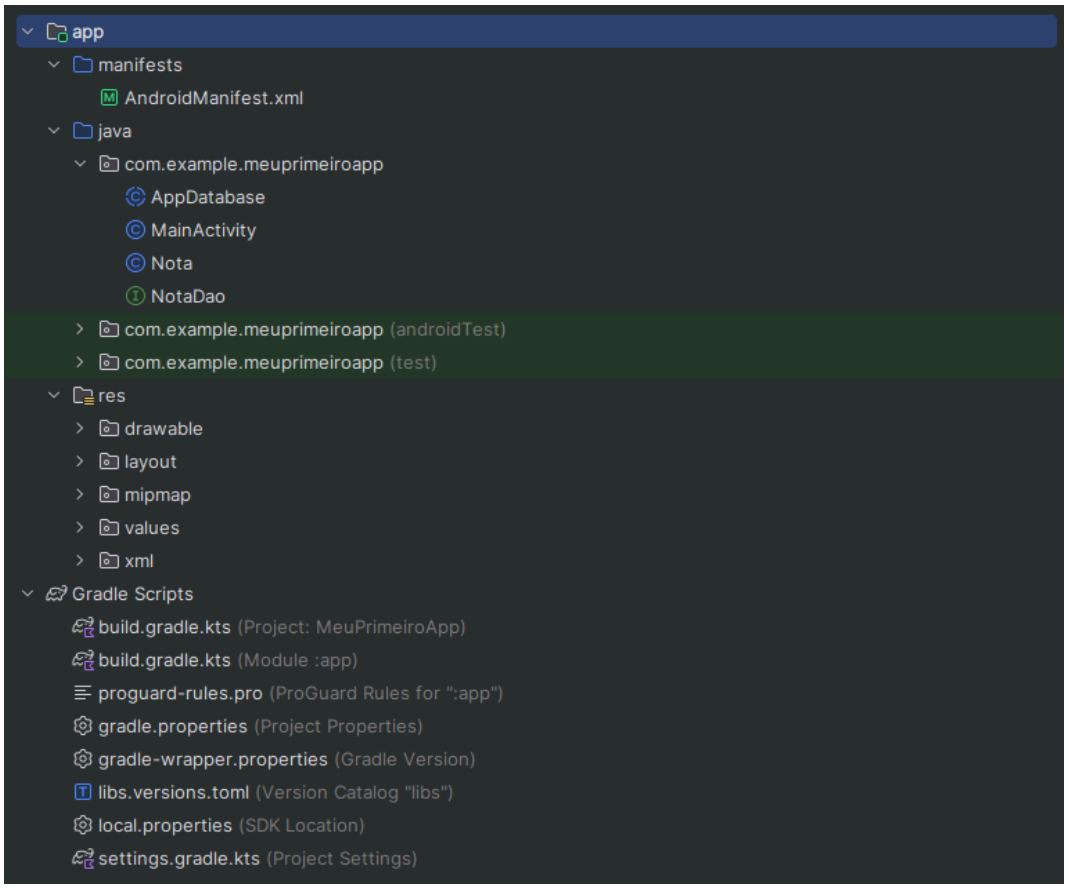
```java
package com.example.meuprimeiroapp;

import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;

import java.util.List;

no usages
@Dao
public interface NotaDao {

    no usages
    @Insert
    void inserir(Nota nota);

    no usages
    @Query("SELECT * FROM notas ORDER BY id DESC")
    List<Nota> listarNotas();
}
```

Por fim, a classe Database conecta tudo, criando a instância do Room que será utilizada no restante do aplicativo para acessar os dados.

Nota.java NotaDao.java AppDatabase.java

```java
package com.example.meuprimeiroapp;

import android.content.Context;

import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;

3 usages
@Database(entities = {Nota.class}, version = 1, exportSchema = false)
public abstract class AppDatabase extends RoomDatabase {

    3 usages
    private static AppDatabase instance;

    no usages
    public abstract NotaDao notaDao();

    no usages
    public static synchronized AppDatabase getInstance(Context context) {
        if (instance == null) {
            instance = Room.databaseBuilder(
                    context.getApplicationContext(),
                    AppDatabase.class,
                    name: "notas_db"
            )
            .allowMainThreadQueries()
            .build();
        }
        return instance;
    }
}
```

A estrutura do projeto ficará mais ou menos da seguinte maneira:



Com isso, o banco de dados Room está configurado e pronto para integrar e usar no resto do projeto.

6. Activity, Fragments e layout xml

Depois de configurar o banco de dados, precisamos criar as telas que exibem e consomem informações do banco de dados.

6.1 Criando arquivos XML de interface

No Android, a interface gráfica normalmente é construída usando arquivos XML dentro da pasta res/layout. Cada tela do aplicativo ou parte dela possui um arquivo XML correspondente, que descreve como os elementos visuais devem ser organizados. [2]

Esses arquivos não contêm lógica de programação. Eles apenas definem como a interface deve aparecer. Em seguida, a Activity ou o Fragment "inflam" esse layout para exibi-lo na tela e manipular seus elementos via código Java.

Para o nosso aplicativo simples de notas, utilizaremos três layouts XML. Eles podem ser criados na pasta **app/res/layout**:

activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/fragmentContainerView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:defaultNavHost="false" />

```

Responsável por definir o espaço onde os fragments serão exibidos. Ele não contém elementos de interface do usuário, apenas um FragmentContainerView.

fragment_lista_notas.xml

```xml
<?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="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/textoNotas"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Nenhuma nota cadastrada"
        android:textSize="16sp" />

    <Button
        android:id="@+id/botaoNovaNota"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Adicionar nova nota"
        android:layout_marginTop="16dp" />

</LinearLayout>

```

Layout usado pelo fragmento que exibe a lista de notas. Para manter a simplicidade, usaremos apenas um TextView que mostrará o conteúdo das notas armazenadas.

fragment_nova_nota.xml

```xml
<?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="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <EditText
        android:id="@+id/editTextoNota"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Digite sua nota aqui" />

    <Button
        android:id="@+id/botaoSalvar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Salvar"
        android:layout_marginTop="16dp" />

</LinearLayout>
```

Layout utilizado para criar novas notas. Contém um EditText onde o usuário digita a nota e um Button que, ao ser clicado, salva essa nota no banco de dados.

Esses três arquivos compõem a base visual do aplicativo e permitem que a Activity e os Fragments exibam conteúdo na tela e interajam com o usuário.

6.2 Criando Activity e Fragments

A MainActivity é a Activity [3] principal do aplicativo, ou seja, a primeira tela lógica do app que foi definida no Manifest. No nosso projeto, ela não exibe conteúdo diretamente. Em vez disso, ela funciona como um contêiner que hospeda os Fragments [4]. Ela faz duas coisas importantes:

1. Carrega o layout activity_main.xml, que contém o FragmentContainerView, com o método setContentView.
2. Abre automaticamente o primeiro fragment (ListaNotasFragment), logo quando o app inicia, com o método replace() do getSupportFragmentManager().

Ou seja, a Activity é o “esqueleto” e os Fragments são as telas que aparecem dentro dela. Isso deixa o app modular, mais organizado e muito didático.

MainActivity.java

```java
package com.example.meuprimeiroapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

        if (savedInstanceState == null) {
            getSupportFragmentManager() FragmentManager
                    .beginTransaction() FragmentTransaction
                    .replace(R.id.fragmentContainerView, new ListaNotasFragment())
                    .commit();
        }
    }
}
```

Vamos criar agora o ListaNotasFragment.java, o fragmento responsável por:

- Exibir todas as notas salvas no banco consultando o NotaDao e mostrando as notas dentro do TextView, com o método db.notaDao().listarNotas().
- Permitir que o usuário vá para a tela de nova nota. Quando o usuário clicar no botão “Adicionar nova rota”, o fragment troca a tela para o fragmento responsável por criar notas, com o método relace (...NovaNotaFragment()).
- O layout correspondente é fragment_lista_notas.xml

```java
package com.example.meuprimeiroapp;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import java.util.List;

public class ListaNotasFragment extends Fragment {

    3 usages
    private TextView textoNotas;
    2 usages
    private Button botaoNovaNota;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
                             @Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {

        return inflater.inflate(R.layout.fragment_lista_notas, container, attachToRoot: false);
    }

    9 usages
    @Override
    public void onViewCreated(@NonNull View view,
                              @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        textoNotas = view.findViewById(R.id.textoNotas);
        botaoNovaNota = view.findViewById(R.id.botaoNovaNota);

        carregarNotas();

        botaoNovaNota.setOnClickListener( View v -> {
            requireActivity() FragmentActivity
                    .getSupportFragmentManager() FragmentManager
                    .beginTransaction() FragmentTransaction
                    .replace(R.id.fragmentContainerView, new NovaNotaFragment())
                    .addToBackStack( name: null)
                    .commit();
        });
    }

    1 usage
    private void carregarNotas() {
        AppDatabase db = AppDatabase.getInstance(requireContext());

        List<Nota> lista = db.notaDao().listarNotas();

        if (lista.isEmpty()) {
            textoNotas.setText("Nenhuma nota cadastrada");
            return;
        }

        StringBuilder sb = new StringBuilder();
        for (Nota nota : lista) {
            sb.append("- ").append(nota.texto).append("\n");
        }

        textoNotas.setText(sb.toString());
    }
}
```

Por fim, criamos o NovaNotaFragment.java, um fragmento responsável por:

- Exibir um campo de texto para o usuário digitar uma nova nota.
- Salvar essa nota no banco de dados Room quando o botão for clicado, com o método db.notaDao().inserir(nota).
- Voltar para a tela de lista (ListaNotasFragment) depois de salvar

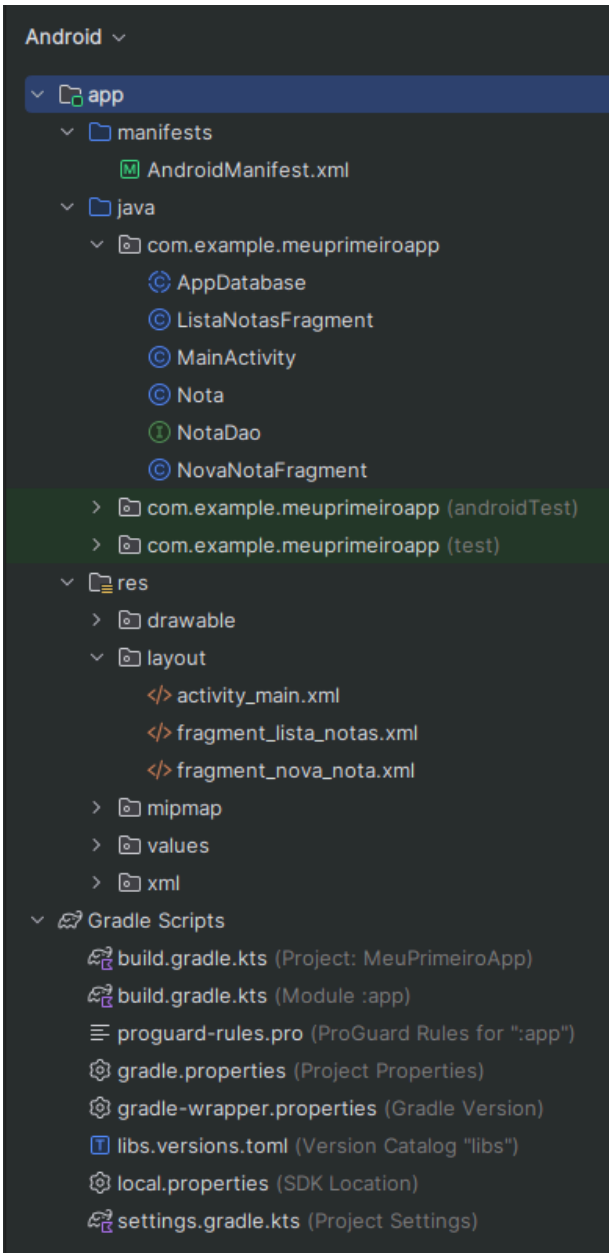
Ele usa o layout fragment_nova_nota.xml, que já criamos com um EditText e um Button.

NovaNotaFragment.java

```java
package com.example.meuprimeiroapp;

import android.os.Bundle;
import android.text.TextUtils;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;

public class NovaNotaFragment extends Fragment {

    2 usages
    private EditText editTextoNota;
    2 usages
    private Button botaoSalvar;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
                             @Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {

        return inflater.inflate(R.layout.fragment_nova_nota, container, attachToRoot: false);
    }

    9 usages
    @Override
    public void onViewCreated(@NonNull View view,
                              @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        editTextoNota = view.findViewById(R.id.editTextoNota);
        botaoSalvar = view.findViewById(R.id.botaoSalvar);

        botaoSalvar.setOnClickListener( View v -> salvarNota());
    }

    1 usage
    private void salvarNota() {
        String texto = editTextoNota.getText().toString().trim();

        if (TextUtils.isEmpty(texto)) {
            Toast.makeText(requireContext(), text: "Digite uma nota antes de salvar", Toast.LENGTH_SHORT).show();
            return;
        }

        AppDatabase db = AppDatabase.getInstance(requireContext());
        Nota nota = new Nota(texto);
        db.notaDao().inserir(nota);

        Toast.makeText(requireContext(), text: "Nota salva com sucesso", Toast.LENGTH_SHORT).show();

        requireActivity() FragmentActivity
                .getSupportFragmentManager() FragmentManager
                .popBackStack();
    }
}
```

6.3 Estrutura de pastas do projeto usando banco de dados Room

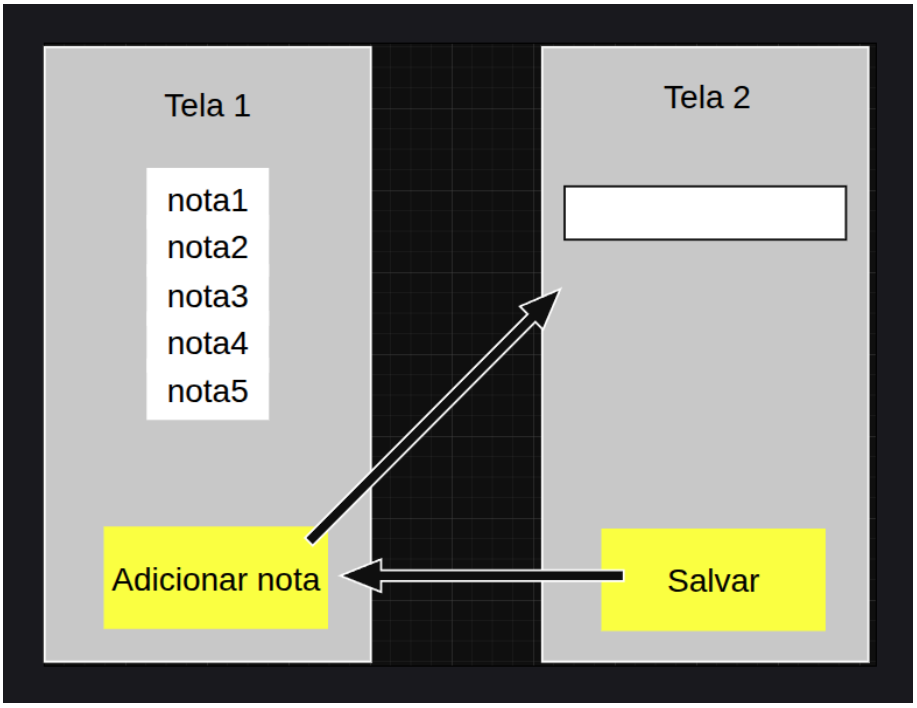
A estrutura do app ficou disposta da seguinte forma:



7. Funcionamento do app

7.1 Fluxo de dados e funcionamento do app

Após configurar o banco de dados Room, as activities, fragments e XML, nosso app ficou da seguinte forma:



1. O app inicia mostrando a lista de notas

2. O usuário pode navegar para a tela de Adicionar nota

3. A nova nota é salva no banco de dados

4. O app volta para a lista e exibe os dados atualizados

Activity carrega Fragment > Fragment lê/escreve no banco Room > Banco devolve dados > Fragment exibe dados > Activity coordena a navegação

7.2 MainActivity

Arquivo: **MainActivity.java**

Layout: **activity_main.xml**

Função: É a tela principal do app, mas não mostra conteúdo próprio. Serve apenas como contêiner para carregar os fragments.

Fluxo: Ao iniciar, a Activity carrega automaticamente o fragmento ListaNotasFragment dentro do FragmentContainerView.

7.3 Fragmento de lista de notas

Arquivo: **ListaNotasFragment.java**

Layout: **fragment_lista_notas.xml**

Função: Mostrar todas as notas já cadastradas no banco. Permite também navegar para a tela de criação de nota.

O que ele faz:

- Usa AppDatabase.getInstance() para acessar o banco Room
- Chama notaDao().listarNotas() para buscar todas as notas
- Mostra as notas dentro do TextView
- Quando o usuário clica em “Adicionar nova nota”, troca para o NovaNotaFragment

7.4 Fragmento de criar nova nota

Arquivo: **NovaNotaFragment.java**

Layout: **fragment_nova_nota.xml**

Função: Permite que o usuário digite um texto e salve como nota no banco.

O que ele faz:

- Captura o texto do EditText
- Cria um objeto Nota
- Usa AppDatabase.getInstance().notaDao().inserir(nota) para inserir no banco de dados
- Mostra uma mensagem de sucesso
- Usa popBackStack() para voltar ao ListaNotasFragment

7.5 Banco de Dados Room

Nota.java (Entity) representa a tabela “notas”. Cada objeto da classe vira um registro no banco.

NodaDao.java (DAO) define as operações permitidas: inserir(Nota nota) e listarNotas(). É a interface responsável por definir as operações que você pode fazer no banco de dados. Ele atua como uma ponte entre comunicação de alto nível, abstraindo o código SQLite.

AppDatabase.java (Database) cria e gerencia o banco Room. Fornece o notaDao() para acessar os dados. Usamos Room.databaseBuilder() para criar o banco.

7.6 Layouts XML

activity_main.xml contém apenas o FragmentContainerView (É onde os fragmentos aparecem).

fragment_lista_notas.xml possui um TextView que exibe as notas e um Button para ir à tela de nova nota.

fragment_nova_nota.xml contém um EditText para escrever a nota e um Button para salvar no banco.

8. Testando o app

8.1 Testes no emulador do Android Studio

O emulador no Android Studio permite executar o aplicativo sem precisar de um celular físico. [6]

Abra o Device Manager

No Android Studio, clique em Tools > Device Manager

Criar um dispositio virtual

Clique em Create Device

Escolha um modelo, por exemplo Pixel 5

Clique em Next

Selecione uma versão do Android (API recomendada pelo Android Studio).

Clique em Download se ainda não estiver instalada.

Após o download, clique em Next e depois em Finish.

Selecionar o emulador para executar o app

No topo da janela do Android Studio, ao lado do botão Run, selecione o dispositivo virtual criado (exemplo Pixel 5 API 34).

Executar o aplicativo

Clique no botão verde Run (ícone de play).

Aguarde o emulador iniciar.

O aplicativo será instalado e aberto automaticamente no emulador.

8.2 Testes em um dispositivo físico

Também é possível testar o aplicativo diretamente em um celular Android. [7]

Ativar o modo desenvolvedor no celular

Abra as configurações do aparelho.

Entre em Sobre o telefone.

Toque várias vezes em Número da versão ou Número da compilação até aparecer a mensagem dizendo que o modo desenvolvedor foi ativado.

Ativar a depuração USB

Volte para as configurações.

Acesse Opções do desenvolvedor ou Programador.

Ative a opção Depuração USB.

Conectar o celular ao computador

Conecte o celular ao computador usando um cabo USB.

No aparelho, aceite a mensagem perguntando se deseja permitir a depuração USB.

Selecionar o dispositivo no Android Studio

No topo do Android Studio, ao lado do botão Run, selecione o nome do seu dispositivo físico (em vez do emulador).

Executar o aplicativo

Clique no botão Run.

O Android Studio vai instalar o app no celular.

O aplicativo será aberto automaticamente no dispositivo físico.

9. Referências

[1] Developer workflow basics https://developer.android.com/studio/workflow. Acesso em 10 dez. 2025.

[2] Layouts em visualizações https://developer.android.com/develop/ui/views/layout/declaring-layout?hl=pt-br. Acesso em 10 dez. 2025.

[3] Introdução às atividades https://developer.android.com/guide/components/activities/intro-activities?hl=pt-br. Acesso em 10 dez. 2025.

[4] Fragmentos https://developer.android.com/guide/fragments?hl=pt-br. Acesso em 10 dez. 2025.

[5] Salvar dados em um banco de dados local usando o Room https://developer.android.com/guide/fragments?hl=pt-br. Acesso em 10 dez. 2025.

[6] Executar apps no Android Emulator https://developer.android.com/studio/run/emulator?hl=pt-br. Acesso em 10 dez. 2025.

[7] Executar apps em um dispositivo de hardware. https://developer.android.com/studio/run/device?hl=pt-br. Acesso em 10 dez. 2025.