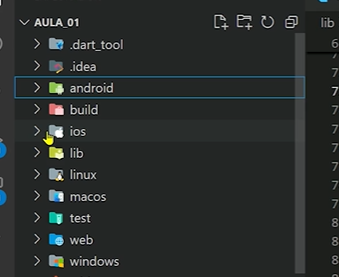
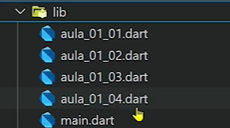
Flamework utilizado na disciplina é o Flutter com linguagem dart.

**Estrutura de um projeto:**

Quando crio uma aplicação flutter ele já cria:

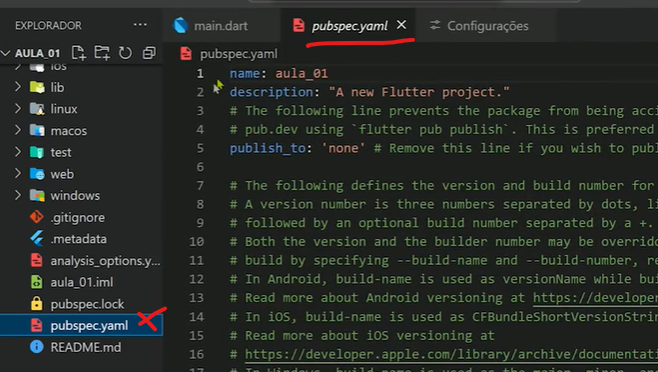


No qual dentro do **LIB** fica os executaveis:

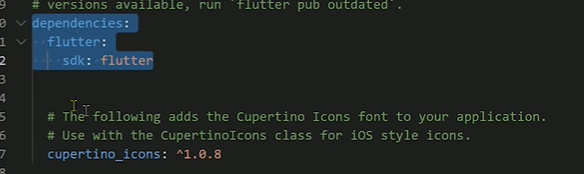


No qual o main.dart é o principal.

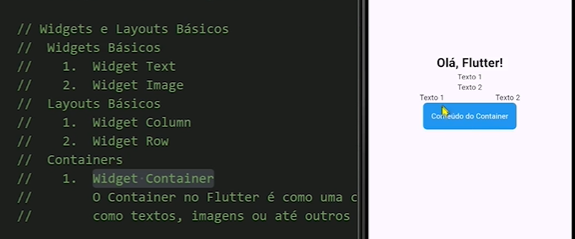
**Pubspec:**



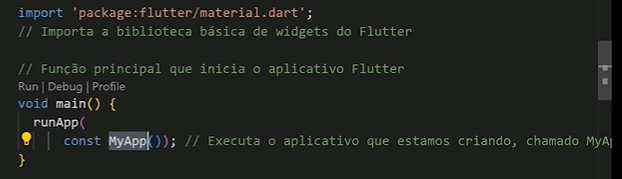
O flutter para ser executado ele precisa de varias APIs que são disponibilizado pela comunidade. E elas são cadastradas na pubspec.



**Widgets:** tudo no flutter é um widgets! No qual possui uma hierarquia.



Todas as vezes que vamos fazer uma aplicação flutter precisamos import ‘package:flutter/material.dart’; Pois possui todas as bibliotecas basicas do flutter.



Com o ambiente configurado, é possível criar o primeiro projeto Flutter, que apresentará uma estrutura de diretórios predefinida.

**O diretório "lib" armazena o código-fonte do aplicativo, sendo o arquivo "main.dart" o ponto de entrada principal.**

O diretório "android" contém os arquivos específicos para a plataforma Android, enquanto o diretório "ios" armazena os arquivos para a plataforma iOS.

**O arquivo "pubspec.yaml" desempenha um papel fundamental na configuração do projeto, especificando as dependências, assets e outras configurações relevantes**.

As dependências são bibliotecas externas que fornecem funcionalidades adicionais ao aplicativo, como acesso a APIs, widgets personalizados e recursos de interface do usuário.

No desenvolvimento com Flutter, **tudo é um widget**, desde elementos básicos como texto e imagens até layouts complexos e telas inteiras. Os widgets são organizados em uma estrutura de árvore, formando a interface do usuário do aplicativo.

O Flutter oferece uma ampla variedade de widgets pré-construídos, além da possibilidade de criar widgets personalizados. Dois tipos principais de widgets são utilizados no Flutter: StatelessWidget e StatefulWidget.

**StatelessWidget,** como o nome sugere, representam widgets que não possuem estado interno, ou seja, sua aparência e comportamento não se alteram ao longo do tempo.

**StatefulWidgets** possuem estado interno que pode ser modificado, refletindo as mudanças na interface do usuário.

A interação do usuário com o aplicativo é um aspecto crucial no desenvolvimento mobile. O Flutter fornece widgets específicos para lidar com eventos de interação, como toques na tela, cliques em botões e entrada de texto.

O widget GestureDetector permite detectar uma variedade de gestos, como toques, arrastos e movimentos de pinça.

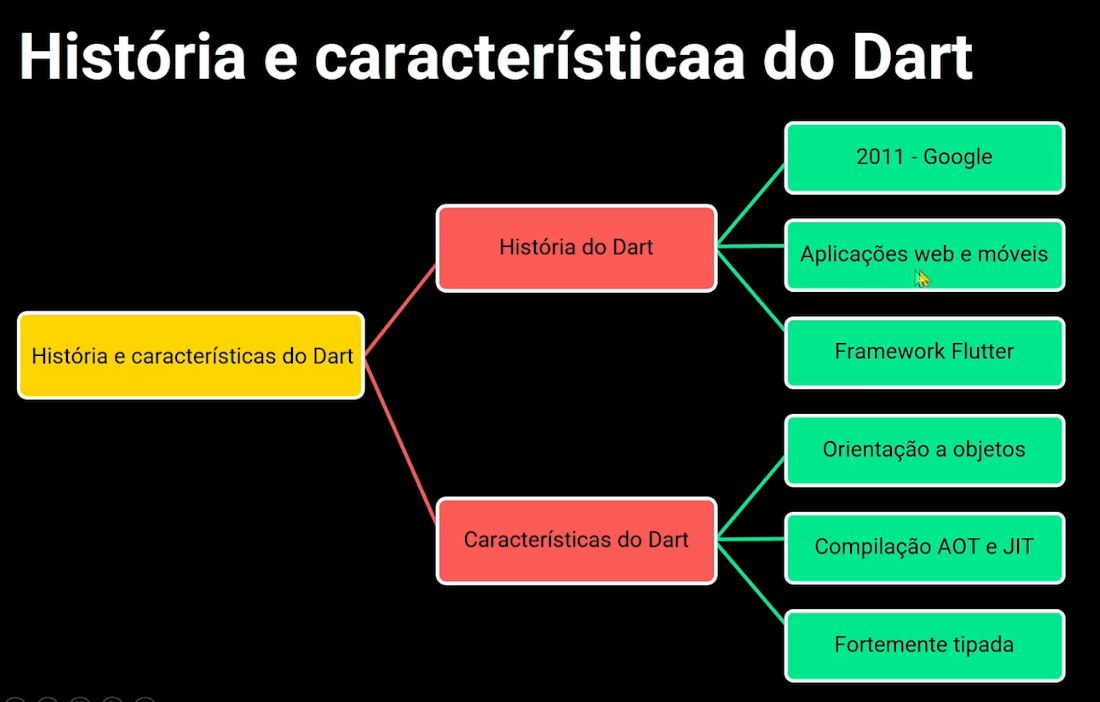
O RaisedButton é um widget que representa um botão com elevação, respondendo a cliques do usuário.

Já o TextField permite a entrada de texto pelo usuário, capturando o valor digitado para processamento posterior.

O Hot Reload é um recurso poderoso do Flutter que acelera o desenvolvimento, permitindo visualizar as alterações no código em tempo real, sem a necessidade de reiniciar o aplicativo. Essa funcionalidade agiliza o processo de desenvolvimento e facilita a identificação e correção de erros.

Em resumo, o desenvolvimento mobile com Flutter oferece uma abordagem moderna e eficiente para a criação de aplicativos multiplataforma. A curva de aprendizado pode parecer desafiadora inicialmente, mas a estrutura organizada do framework, a linguagem Dart e a vasta gama de recursos disponíveis tornam o Flutter uma escolha poderosa para desenvolvedores de todos os níveis de experiência.

**Dart:**



**Características do Dart**

- **Orientado a Objetos**: Dart é uma linguagem orientada a objetos, onde tudo é um objeto.

- **Compilação AOT e JIT**: Suporta compilação Ahead-of-Time (AOT) para desempenho e Just-in-Time (JIT) para desenvolvimento rápido.

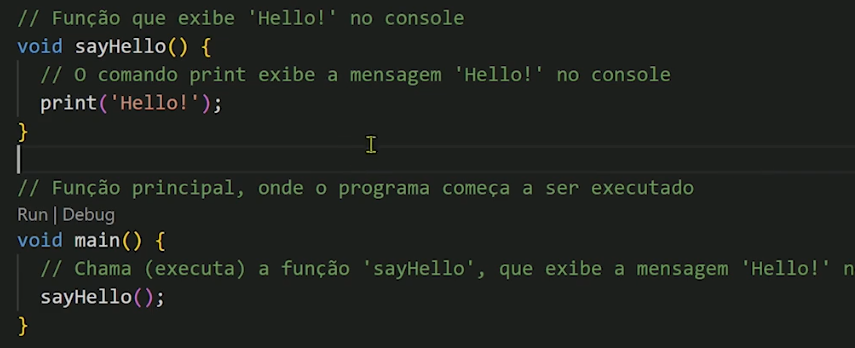
- **Fortemente Tipada**: Suporta tipagem estática com inferência de tipos.

Sempre que for começar com uma aplicação Dart tem que colocar: **void main(){ }**

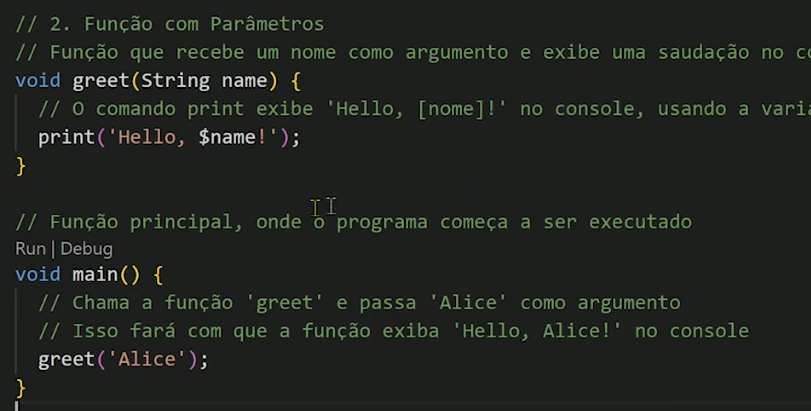
**Variaveis**: String, int, double e bool.

**Funções**: O código dart só executa o main! Então lembre de chamar a outra função:

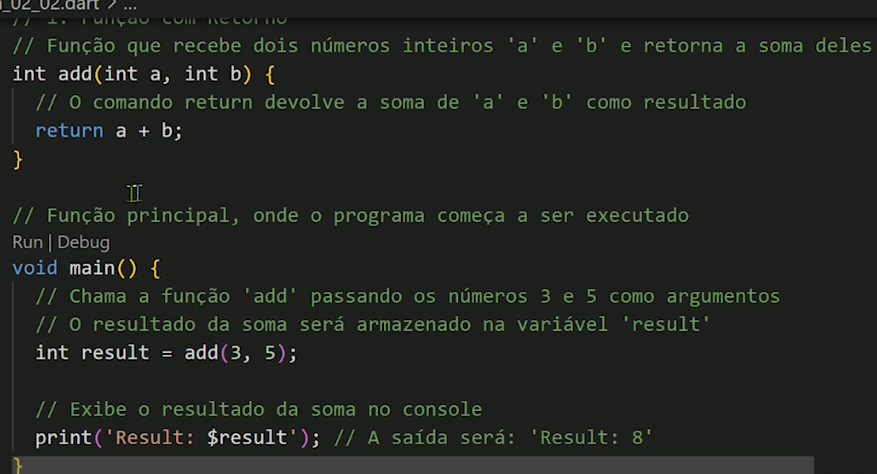
**Função simples:**



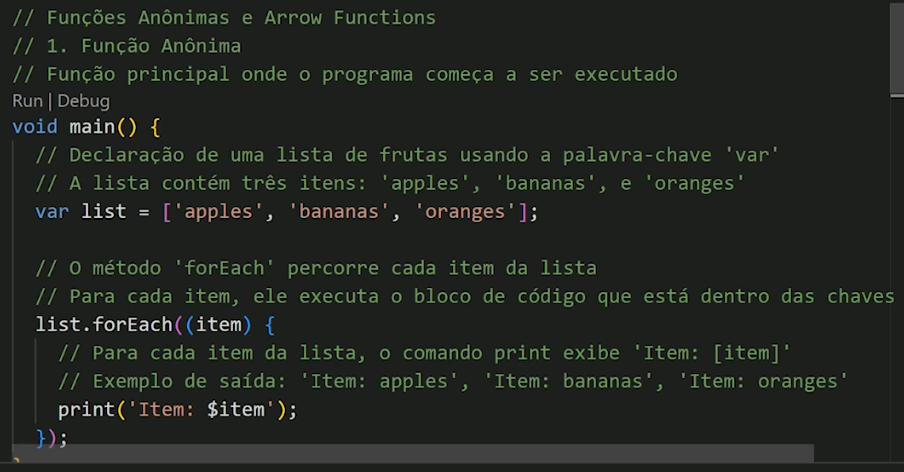
**Função com parâmetro:**



**Função com retorno:**

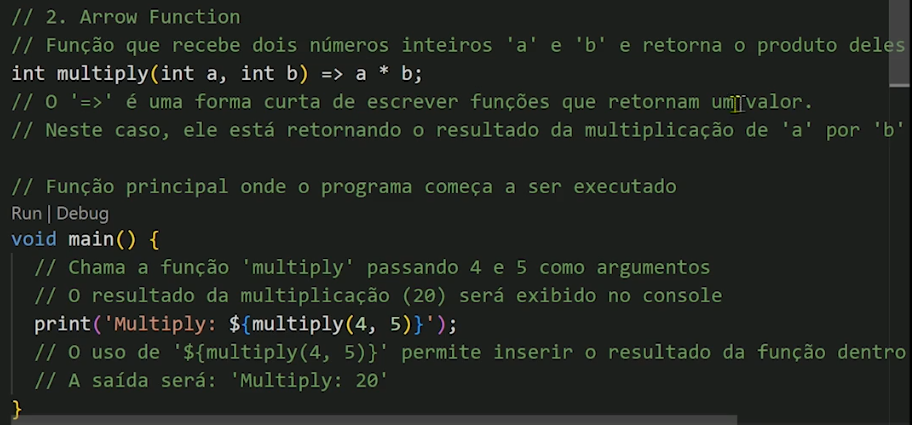


**Função anonima: A declaração é como uma lista.**



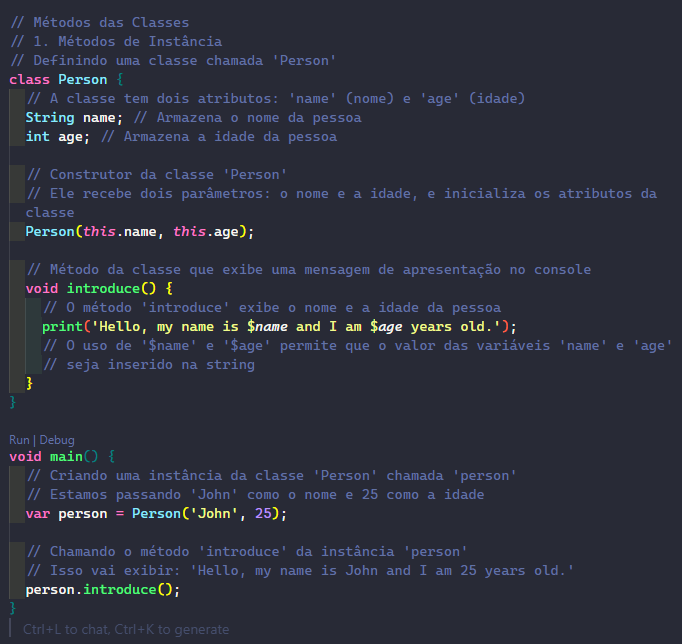
Uma função anônima é uma função sem nome que pode ser passada como parâmetro.

**Arrow Function: Função simples que retorna valor**



A sintaxe de arrow function é uma maneira concisa de definir funções de uma linha.

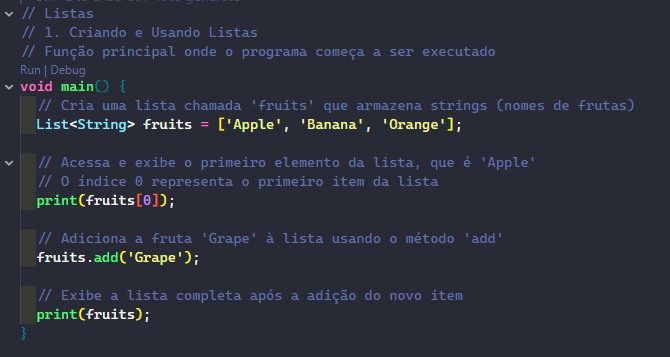
**Metodos de uma classe:**

****

Métodos de instância são funções definidas dentro de uma classe e operam sobre instâncias dessa classe.

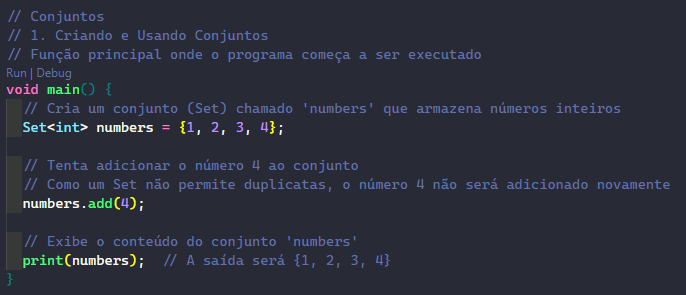
**Coleções: Lista, Conjuntos, Mapas e Operações comuns.**

**Listas:**



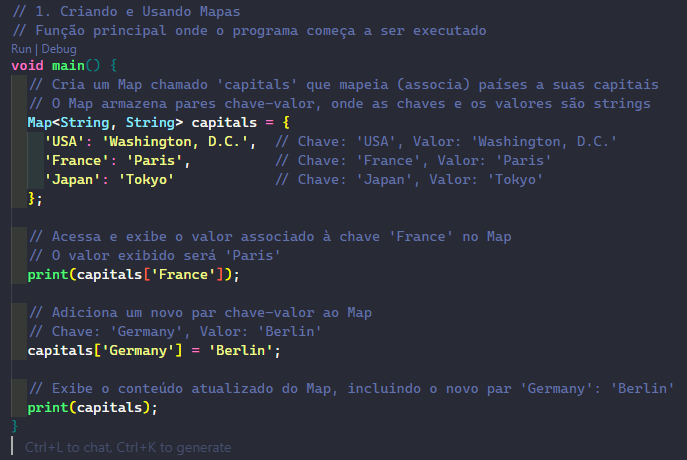
Listas são coleções ordenadas que permitem duplicatas. Podem ser manipuladas usando vários métodos, como add.

**Conjuntos:**



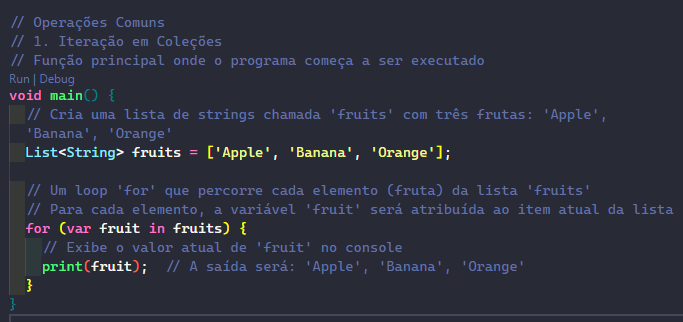
Conjuntos são coleções desordenadas que não permitem duplicatas.

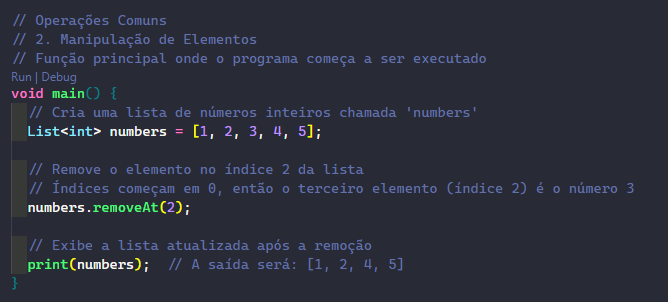
**Mapas:**



Mapas são coleções de pares chave-valor, onde cada chave é única.

**Operações:**

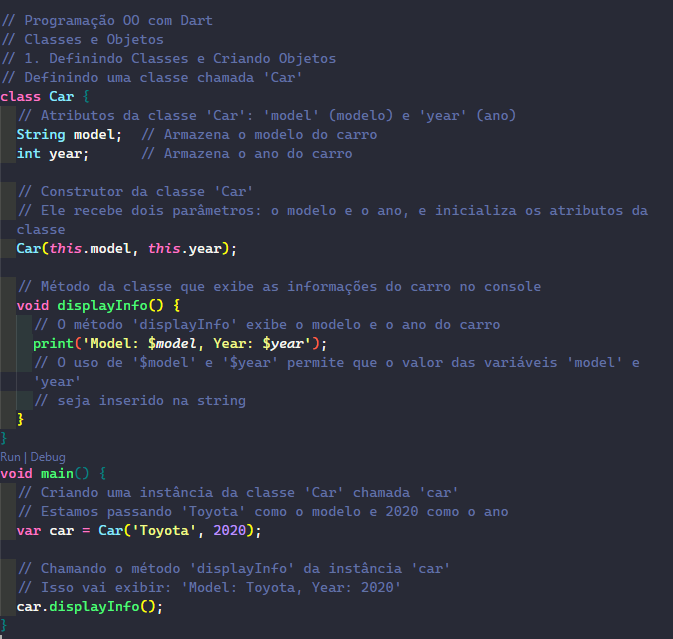




**Programação orientada a objetos.**

Classe e objetos, herança, interfaces e mixins, construtores.

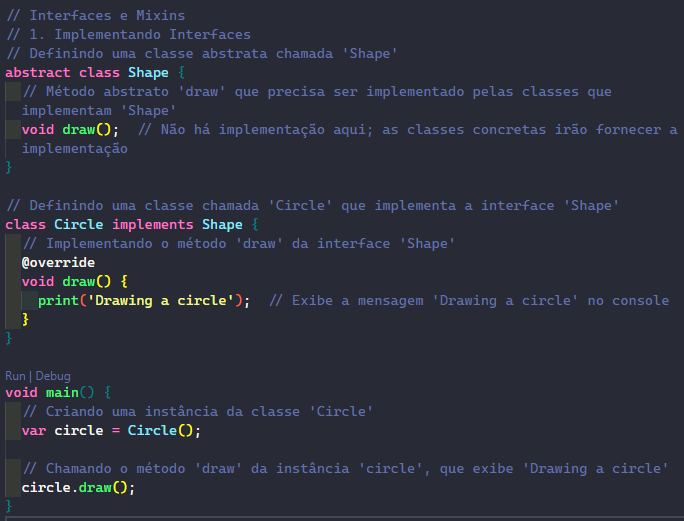
**Classe e objeto:**



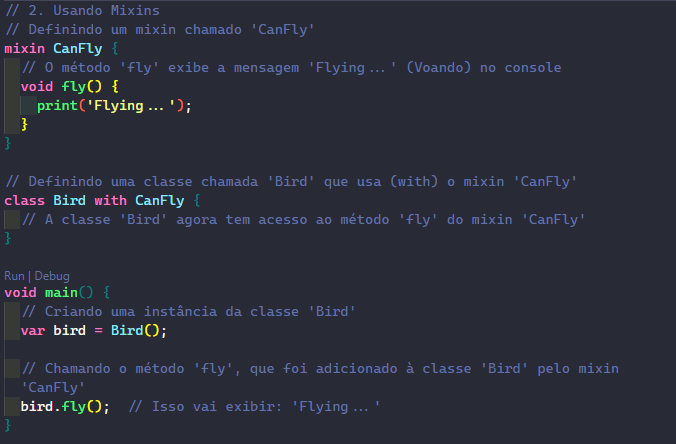
**Herança:**



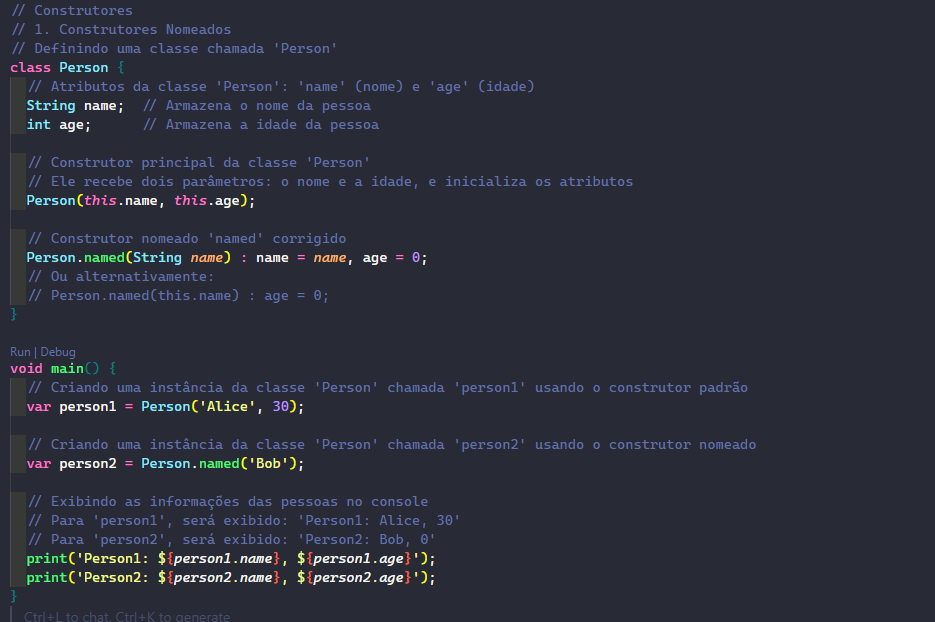
**Interfaces:**



**Mixins:**

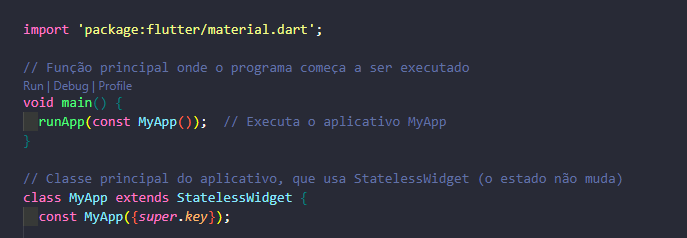


**Construtores:**

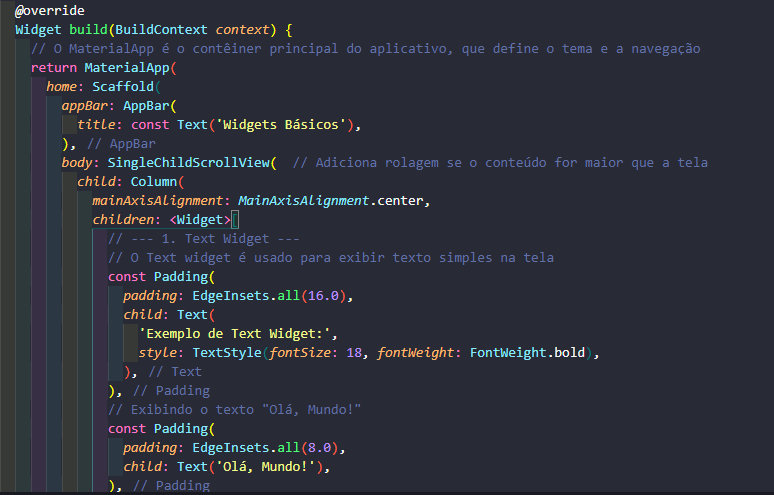


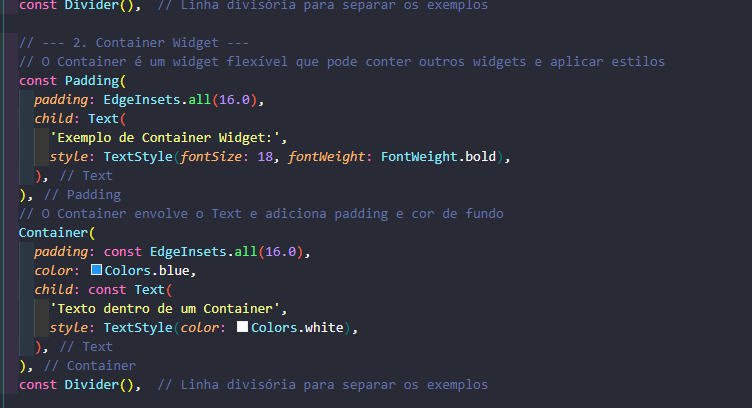
**Interfaces com Widgets:**

Sempre vou ter:



**Text Widget:** O Text widget é usado para exibir texto na tela.

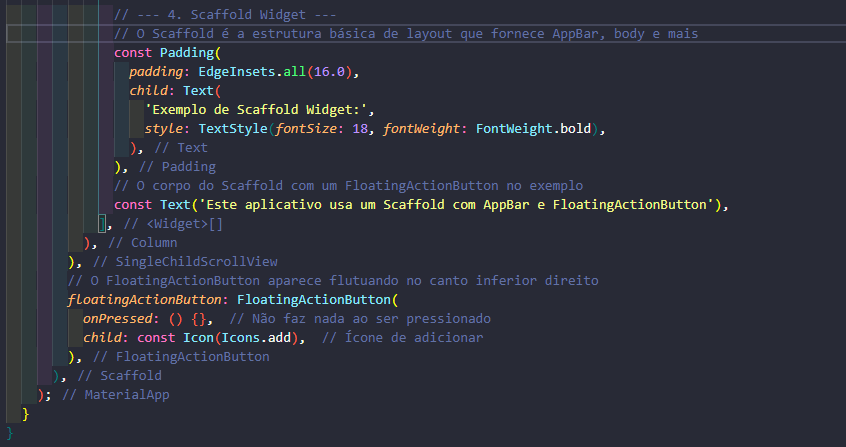




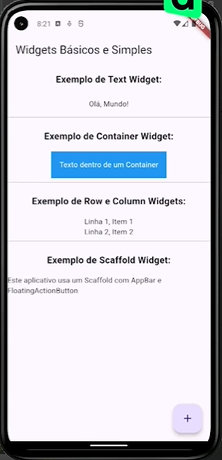
O Container é um widget flexível usado para criar caixas com propriedades específicas, como margens, padding e alinhamento.



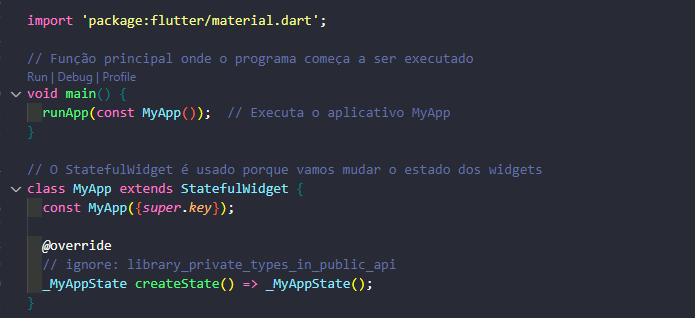
Row e Column são usados para organizar widgets em linhas e colunas, respectivamente. Usamos Column para organizar as linhas verticalmente e Row para organizar os textos horizontalmente. Isso ajuda a estruturar a interface do usuário de forma ordenada

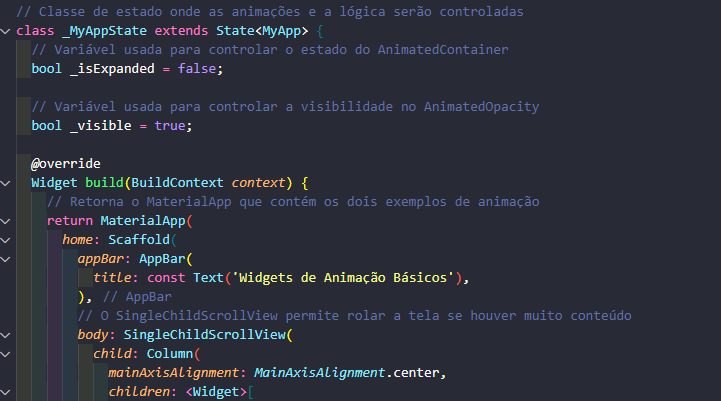


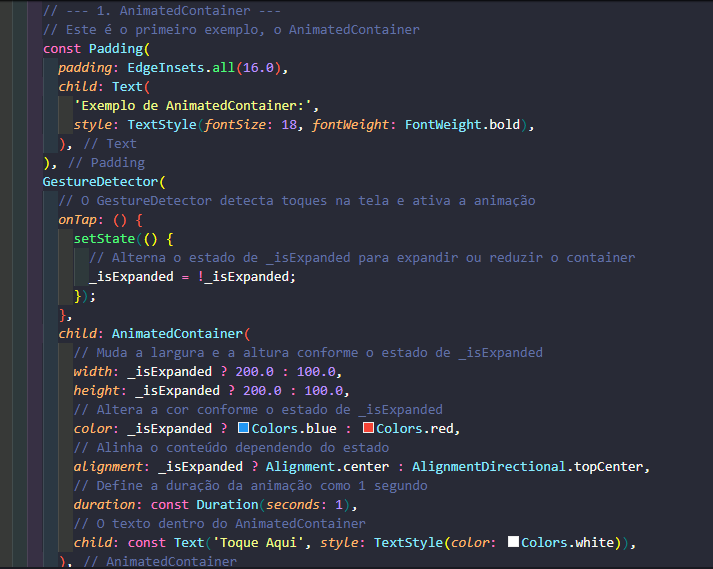
O Scaffold é um widget que fornece uma estrutura básica de layout para um aplicativo.O Scaffold fornece uma estrutura visual com AppBar no topo, body no meio e um FloatingActionButton flutuante. Isso facilita a criação de uma interface padronizada.



**Animações:**



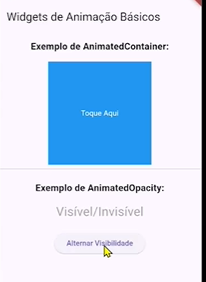




O AnimatedContainer permite animar mudanças em suas propriedades, como tamanho, cor e borda.AnimatedContainer muda de tamanho e cor ao ser tocado. A mudança é animada com a duração de 1 segundo, proporcionando uma transição suave.



O AnimatedOpacity permite animar mudanças na opacidade de um widget.AnimatedOpacity permite que a opacidade de um widget mude suavemente entre visível e invisível. O botão alterna a visibilidade do texto com uma animação de 1 segundo.



**Integração de elementos multimídia:**

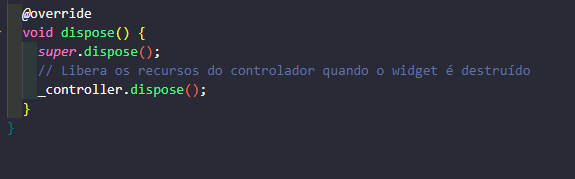




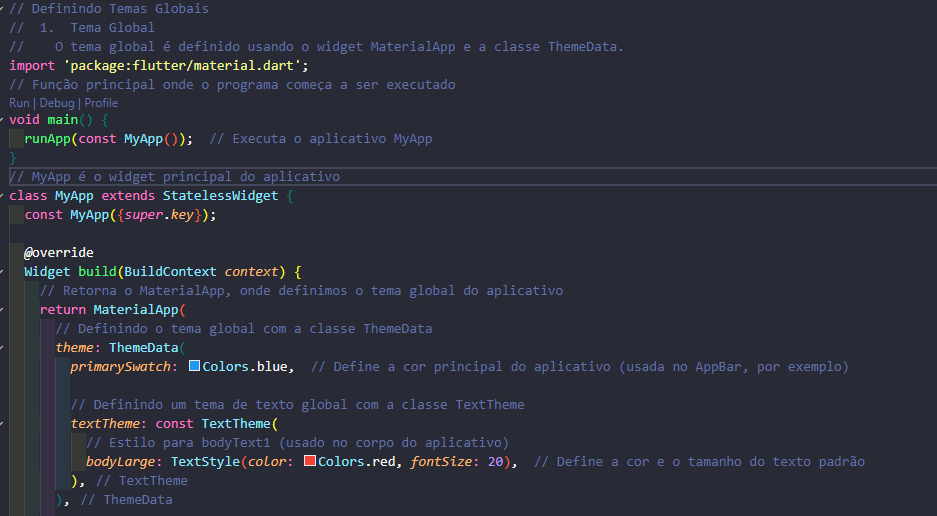


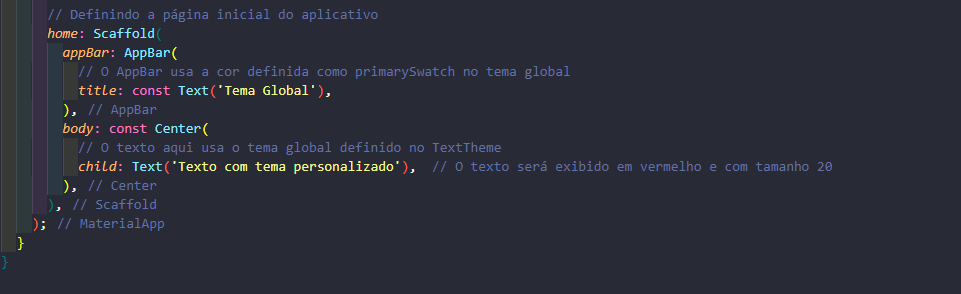


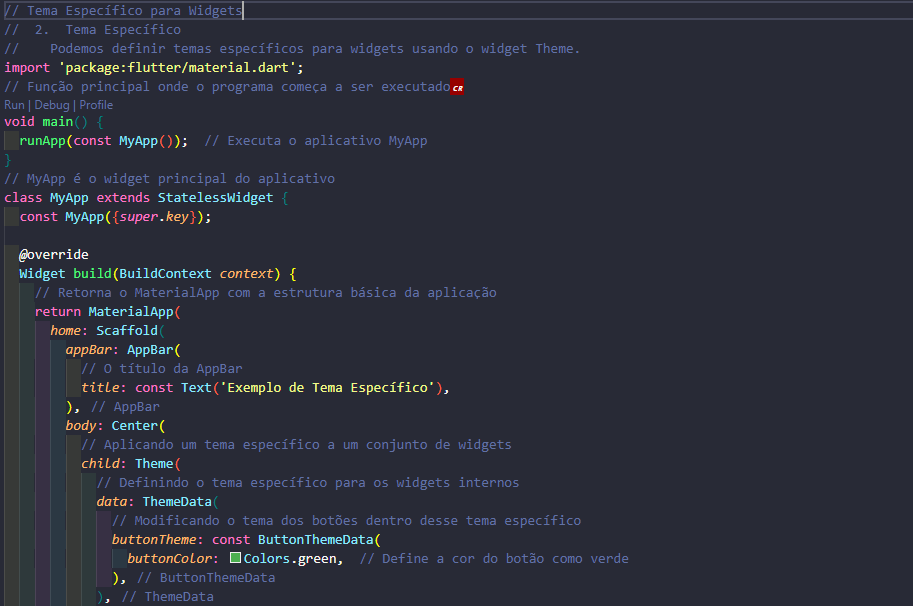


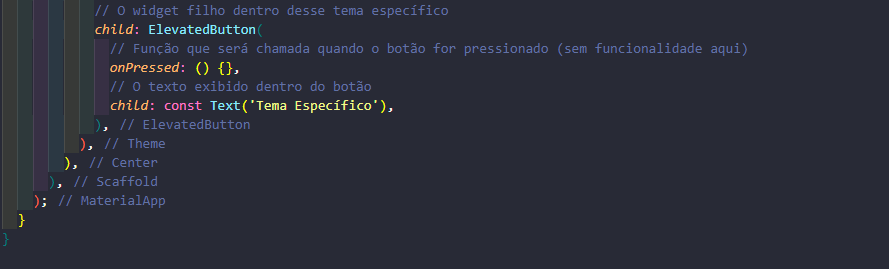


**Customização e temas:**



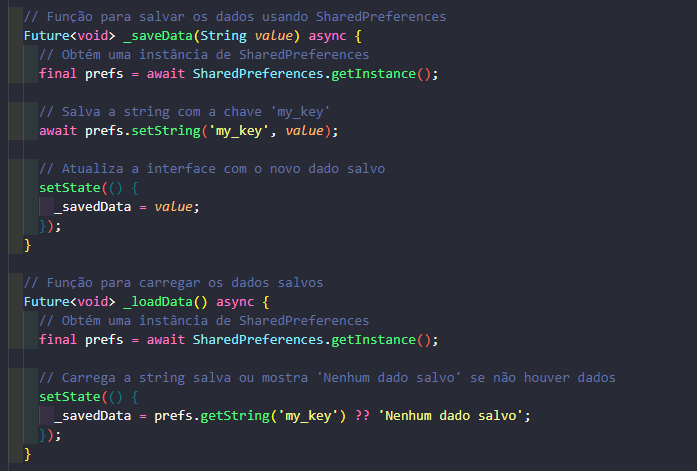
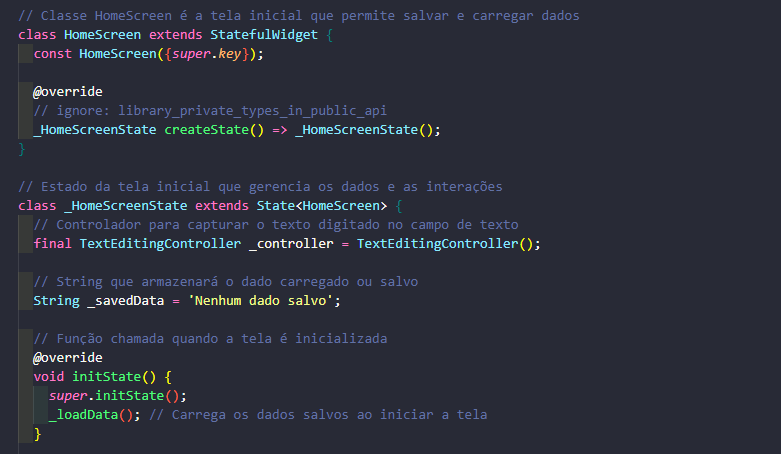
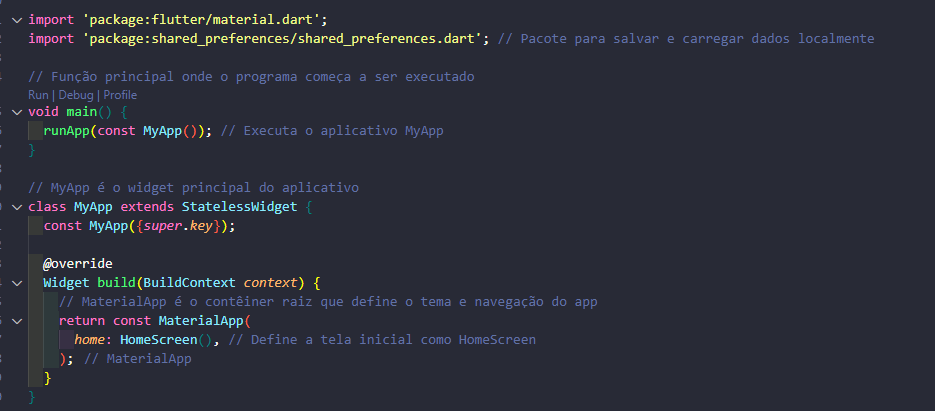


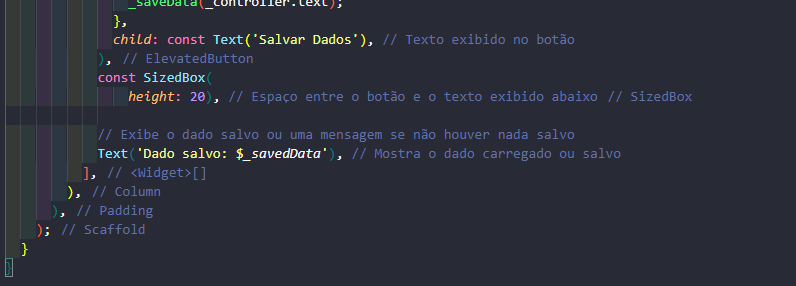




**Persistência de Objetos:**

A persistência de objetos refere-se ao armazenamento de dados gerados por uma aplicação, permitindo que informações não sejam perdidas. O aprendizado abrange o uso de APIs, SQLite e operações CRUD, além de melhores práticas em persistência de dados.





**Salvando o conteudo com SQLite com flutter:**

(Aula04\_02)

**Operações CRUD com SQLite:**

(Aula04\_03)

**Melhores Práticas em Persistência de Dados:**

// Estrutura e Organização do Código

// 1. Separação de Responsabilidades

// Mantenha a lógica de persistência separada da lógica de apresentação. Use classes de serviço ou repositório para interagir com o banco de dados.

// 2. Uso de Models

// Crie modelos (models) para representar seus dados de forma estruturada.

// 3. Gerenciamento de Estado

// Utilize pacotes de gerenciamento de estado como provider ou bloc para manter o estado da aplicação consistente.

//

// Segurança e Performance

// 1. Evite o Uso de Strings Literais para Consultas

// Use parâmetros nomeados para evitar injeção de SQL.

// 2. Compactar o Banco de Dados

// Periodicamente, compacte o banco de dados para otimizar o espaço.

// 3. Criptografia

// Para dados sensíveis, considere usar criptografia para proteger informações armazenadas.

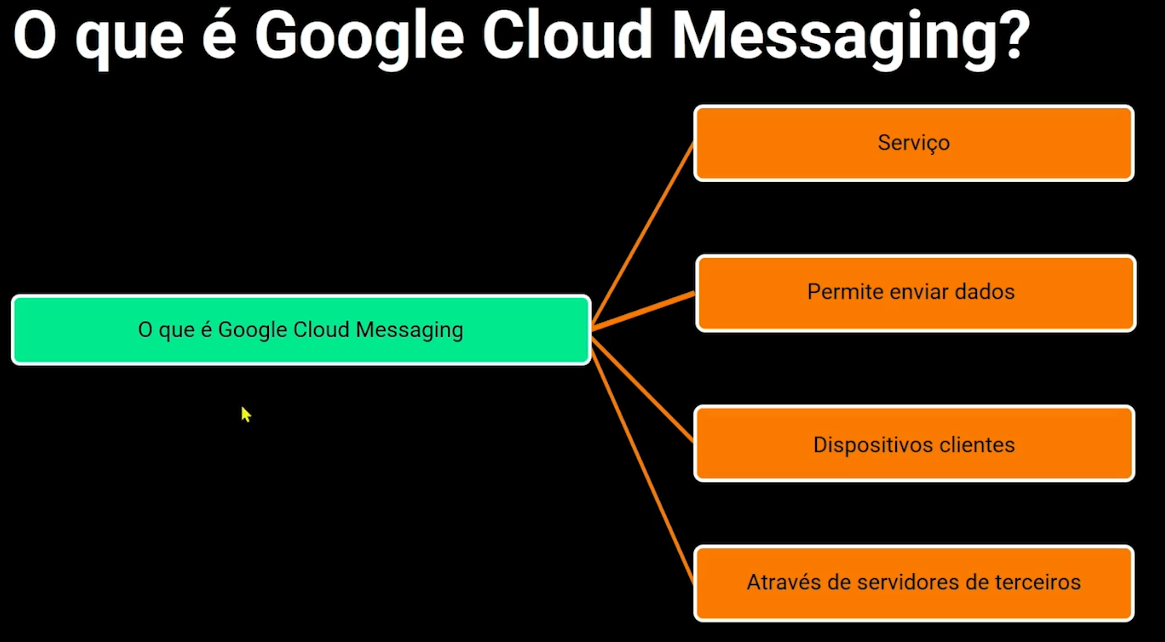
// 4. Fechar Conexões

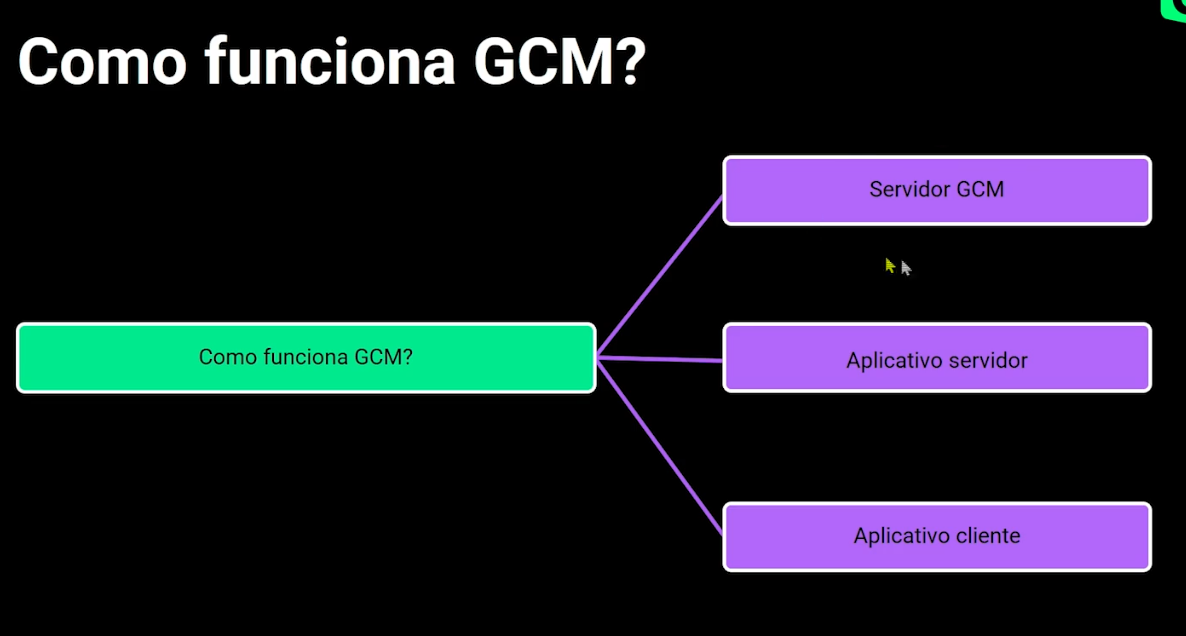
// Certifique-se de fechar as conexões com o banco de dados para liberar recursos.

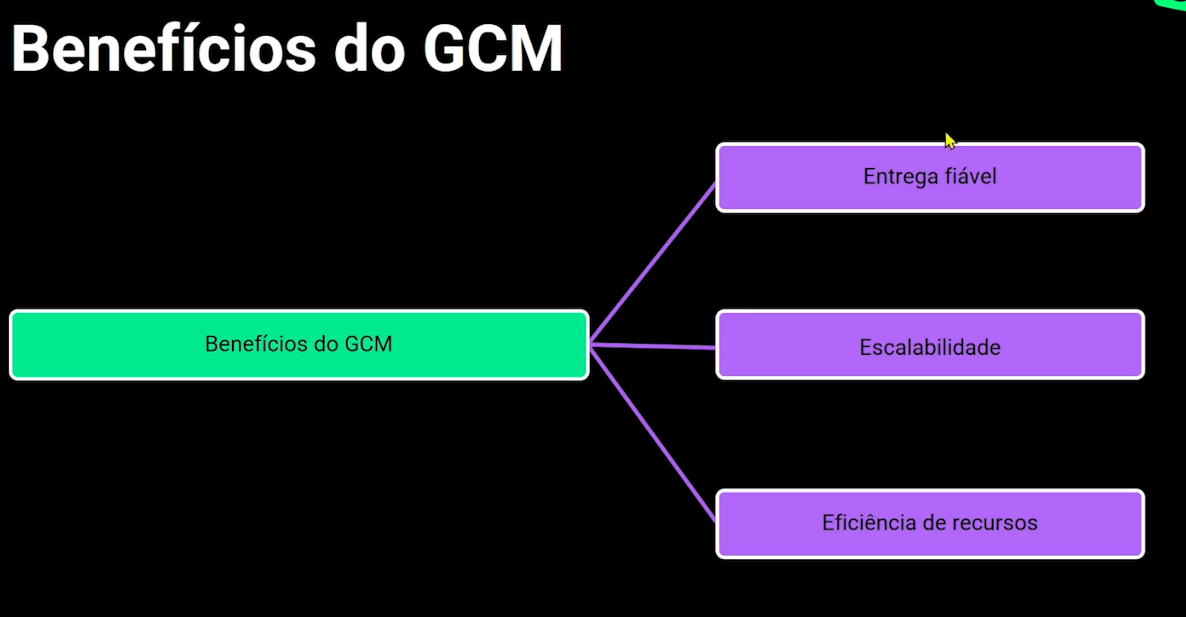
**Google Cloude Messaging:**

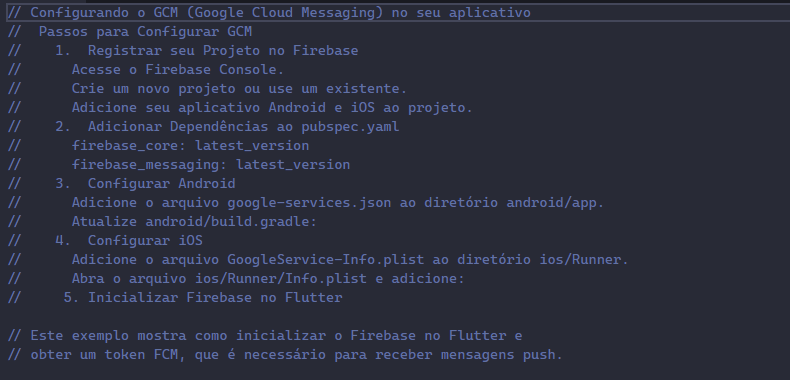
O Google Cloud Message foi renomeado para Firebase Cloud Message, mantendo as mesmas funcionalidades. Este serviço permite enviar e receber mensagens Push para dispositivos móveis, facilitando a comunicação entre aplicativos e usuários.

O Firebase Cloud Message funciona através da criação de um servidor e aplicativos que enviam e recebem mensagens. Seus benefícios incluem entrega confiável, escalabilidade e eficiência, permitindo uma comunicação eficaz entre dispositivos



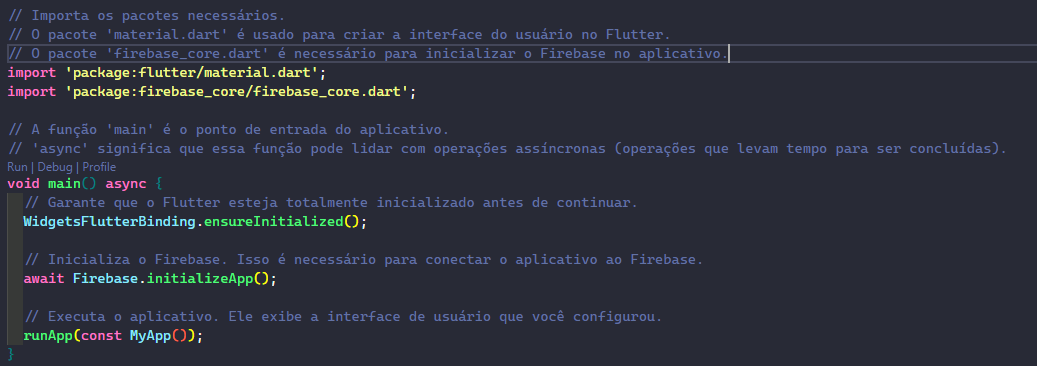


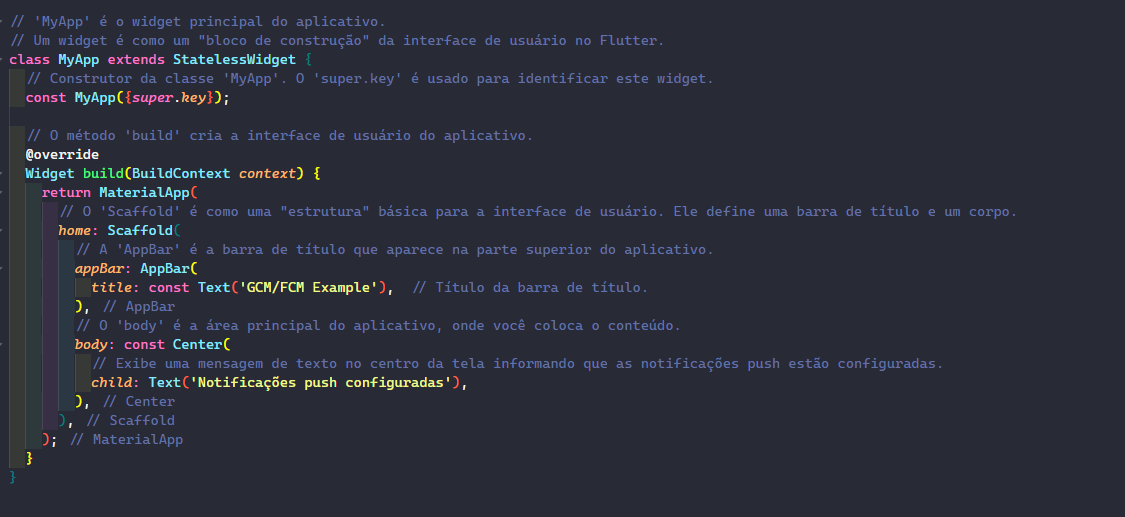




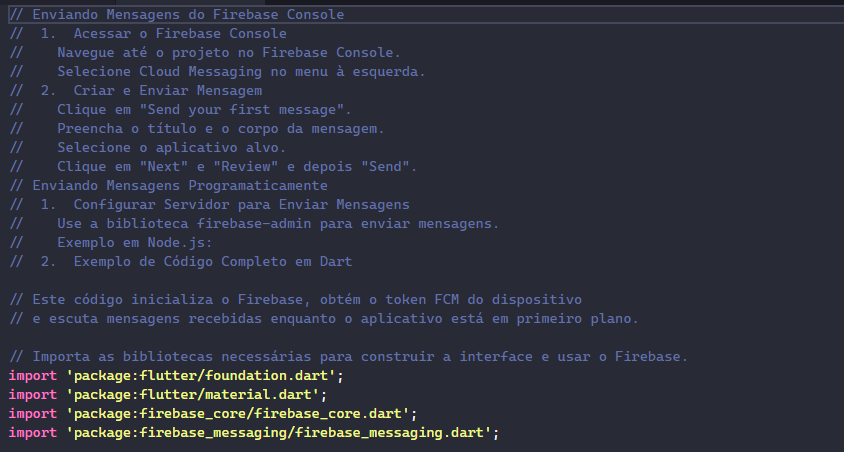
O Firebase Cloud Messaging (FCM) é a evolução do Google Cloud Messaging, mantendo funcionalidades semelhantes. A configuração envolve registrar o projeto no Firebase Console, adicionar dependências no PubSpec e configurar o Android com o arquivo Google Services.json.

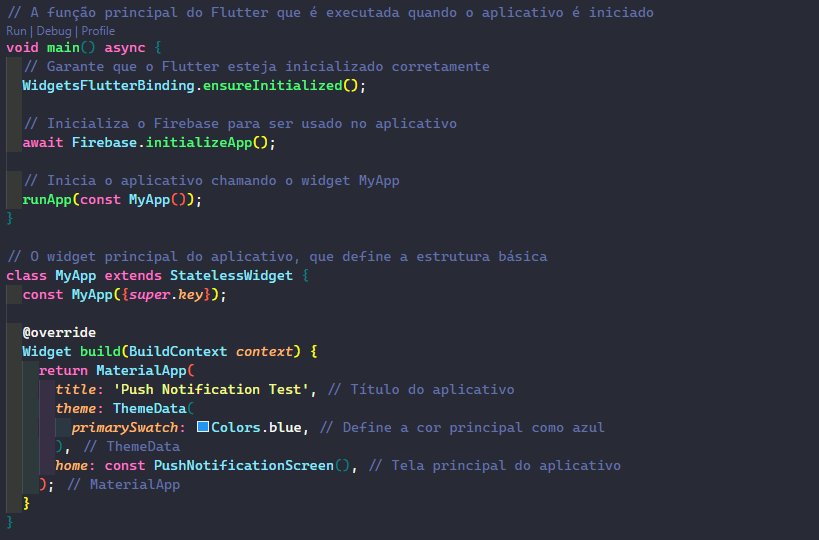
A inicialização do Firebase no Flutter requer a importação de bibliotecas essenciais, como material.dart e Firebase Core. O ponto de entrada do aplicativo deve garantir que o Flutter esteja totalmente inicializado antes de conectar ao Firebase, permitindo a troca de mensagens de push.

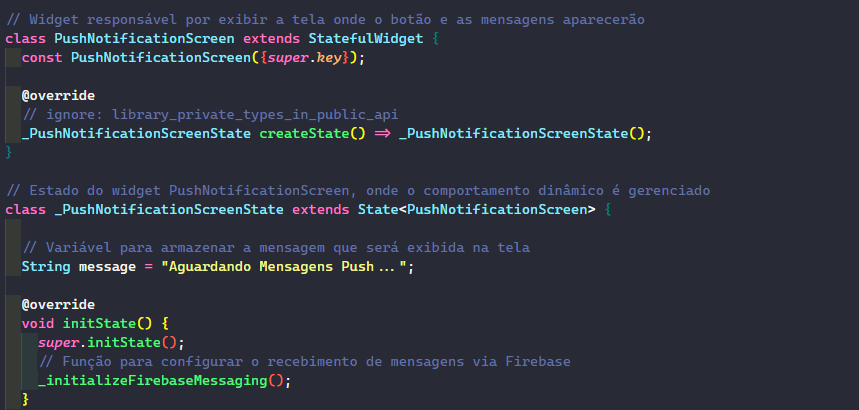




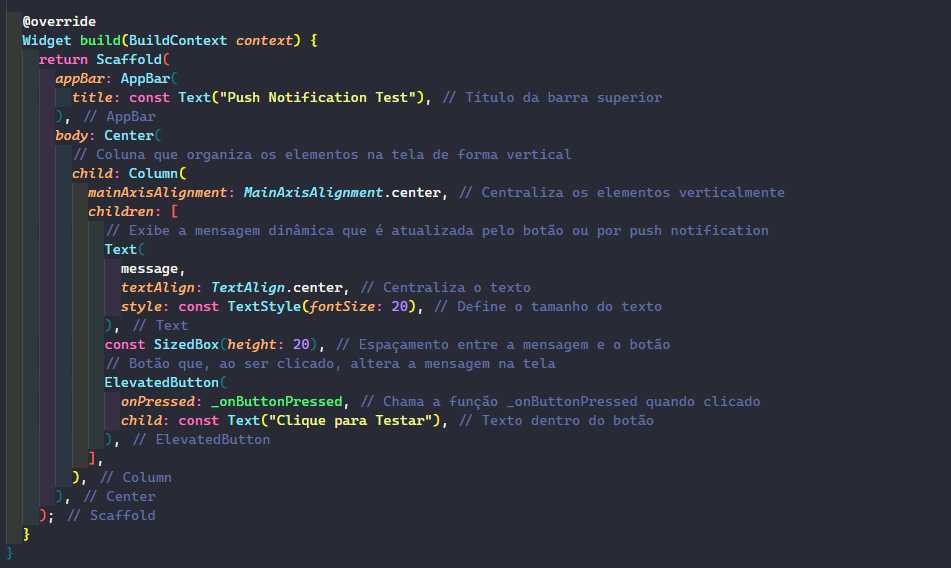
**Enviando mensagens:**











**Responder as mensagens:**

