



Etec
Juscelino Kubitschek
de Oliveira
Diadema



Programação de Aplicativos Mobile

Introdução ao Flutter

Prof. Dr. Alexandre Garcez Vieira

Ciência da Computação

Lattes: <http://lattes.cnpq.br/1502755860213625>

<https://www.linkedin.com/in/garcezv/>



Profissional PCD-TEA | CID 10 - F84.5 | Transtorno do Espectro Autista



Neste módulo usaremos 3 plataformas on-line para conhecer e aprender o Flutter

<https://www.flutterflow.io/>

Tabela Comparativa Rápida

<https://zapp.run/new>

	Plataforma	Ideal Para	Suporta Múltiplos Arquivos?	Suporta Pacotes Externos?	Custo
https://dartpad.dev/	FlutterFlow	Construir apps completos (visual/código)	✓ Sim	✓ Sim	Gratuito com planos pagos
	DartPad	Testar trechos de código e aprender	✗ Não	✗ Não	Totalmente Gratuito
	Zapp!	Prototipar apps completos (foco no código)	✓ Sim	✓ Sim	Totalmente Gratuito

A linguagem de programação usada no Flutter é o **Dart**.



Dart

Por que o Google escolheu o Dart para o Flutter?

Como seu professor, acho importante que você não apenas saiba "qual" é a linguagem, mas "por que" ela é perfeita para o trabalho. O Dart tem superpoderes que tornam o Flutter tão especial:

Otimizado para Interfaces de Usuário (UI): A sintaxe do Dart é clara e concisa, o que facilita a criação e organização de árvores de Widgets (os blocos de construção visuais do Flutter).

Extremamente Performático: O Dart pode ser compilado de duas formas geniais:

Compilação JIT (Just-In-Time): Durante o desenvolvimento, o Dart compila "em tempo real". É isso que permite a funcionalidade mágica do **Hot Reload** (⚡), onde você vê suas alterações de código refletidas no app em menos de um segundo.

Compilação AOT (Ahead-of-Time): Quando você vai publicar seu aplicativo, o Dart é compilado diretamente para **código de máquina nativo** (ARM para Android/iOS, por exemplo). Isso garante que seu aplicativo final seja incrivelmente rápido e tenha uma performance comparável a de um app nativo.

Segurança de Nulos (Null Safety): O Dart possui um sistema moderno que ajuda a evitar um dos erros mais comuns na programação: os "erros de nulo" (tentar usar algo que não tem valor). Isso torna seu código mais robusto e com menos bugs.

Fácil de Aprender: Se você já teve algum contato com linguagens como Java, C#, JavaScript ou C++, você se sentirá em casa com o Dart. A sintaxe é muito familiar e considerada limpa e fácil de ler.

FlutterFlow

Ideal para: Criar aplicativos completos, do iniciante ao profissional, com uma abordagem visual (low-code).

[FlutterFlow](#) é a plataforma mais poderosa e completa para desenvolver Flutter no navegador. Ela vai muito além de um simples editor de código.

Como funciona: Você constrói a interface do seu aplicativo arrastando e soltando componentes visuais (Widgets). Para adicionar lógica, interações com banco de dados (como o Firebase), e funcionalidades customizadas, você pode usar a interface visual ou escrever seu próprio código Dart/Flutter.

Vantagens:

Extremamente Rápido: Permite criar aplicativos complexos em uma fração do tempo.

Visual e Intuitivo: Ótimo para iniciantes que ainda não dominam toda a estrutura de código.

Recursos Integrados: Conexão com Firebase, APIs, autenticação de usuários, tudo de forma simplificada.

Exportação do Código: O mais importante é que você não fica "preso" na plataforma. A qualquer momento, você pode visualizar e exportar todo o código-fonte Flutter, limpo e bem estruturado, para continuar trabalhando localmente no VS Code ou Android Studio.

Limitações: Possui um plano gratuito generoso, mas recursos mais avançados (como exportação do código e APK/App Bundle) exigem um plano pago.

Resumindo: Se você quer construir um aplicativo real e funcional de forma online, com uma interface visual poderosa e a opção de mergulhar no código quando necessário, **FlutterFlow é a melhor escolha.**

DartPad

Ideal para: Testar trechos de código Dart e Flutter de forma rápida e simples.

O [DartPad](#) é a ferramenta oficial do time do Google para experimentação. Pense nele como um "playground" ou um "bloco de rascunho".

Como funciona: É um editor de código minimalista onde você pode escrever código Dart ou snippets de Flutter e ver o resultado instantaneamente em uma tela ao lado.

Vantagens:

Oficial e Sempre Atualizado: Mantido pela equipe do Google.

Extremamente Leve e Rápido: Abre instantaneamente, sem necessidade de login.

Ótimo para Aprender: Perfeito para testar um widget específico, entender um conceito da linguagem Dart ou compartilhar um exemplo de código com alguém.

Limitações: Não foi feito para criar projetos completos. Você não pode adicionar múltiplos arquivos, importar pacotes externos (bibliotecas) ou construir um aplicativo de verdade.

Resumindo: Use o DartPad para estudar, testar pequenas ideias e compartilhar exemplos de código. Ele é o canivete suíço para testes rápidos.

Zapp! (by Codemagic)

Ideal para: Trabalhar em projetos Flutter completos, com múltiplos arquivos e pacotes, em um ambiente que simula o VS Code.

[Zapp!](#) é uma ferramenta criada pela Codemagic (uma popular plataforma de automação e CI/CD para Flutter). Ele oferece um ambiente de desenvolvimento online muito similar ao que você teria localmente.

Como funciona: Ele te dá um ambiente completo no navegador onde você pode criar arquivos, importar pacotes do pub.dev e rodar um projeto Flutter web. Ele é muito parecido com o VS Code online.

Vantagens:

Suporte a Pacotes: Você pode usar bibliotecas da comunidade, o que é uma grande vantagem sobre o DartPad.

Ambiente Completo: Permite criar e gerenciar múltiplos arquivos e pastas, como em um projeto real.

Baseado em Projetos do GitHub: Você pode importar e rodar projetos diretamente de um repositório do GitHub.

Limitações: O foco principal é a execução de aplicativos para a web. Embora o código seja Flutter puro, a visualização e teste são no formato web.

Resumindo: Se o DartPad é muito simples para você e você precisa de um ambiente mais robusto para prototipar um app com várias telas e dependências, o Zapp! é a escolha ideal.

Tabela Comparativa Rápida

Plataforma	Ideal Para	Suporta Múltiplos Arquivos?	Suporta Pacotes Externos?	Custo
FlutterFlow	Construir apps completos (visual/código)	<input checked="" type="checkbox"/> Sim	<input checked="" type="checkbox"/> Sim	Gratuito com planos pagos
DartPad	Testar trechos de código e aprender	<input type="checkbox"/> Não	<input type="checkbox"/> Não	Totalmente Gratuito
Zapp!	Prototipar apps completos (foco no código)	<input checked="" type="checkbox"/> Sim	<input checked="" type="checkbox"/> Sim	Totalmente Gratuito

O que é Flutter e FlutterFlow?

Antes de mergulharmos nos passos, uma explicação rápida:

Flutter é uma ferramenta gratuita do Google para criar aplicativos bonitos que funcionam em celulares Android, iOS, web e até desktops. É como um kit de construção para apps.

FlutterFlow é uma plataforma online (no site <https://www.flutterflow.io/>) que torna o Flutter mais fácil. Em vez de escrever código todo o tempo, você usa um editor visual: arrasta e solta elementos na tela, como se estivesse montando um quebra-cabeça. É perfeito para iniciantes porque é "low-code" (pouco código) ou até "no-code" (sem código). Com ela, você constrói apps rápidos para Android.

Vamos criar um app simples: um "To-Do List" básico, onde o usuário pode adicionar tarefas e ver uma lista delas. É um exemplo clássico para iniciantes. Você precisará de um computador com internet e um navegador (como Chrome).

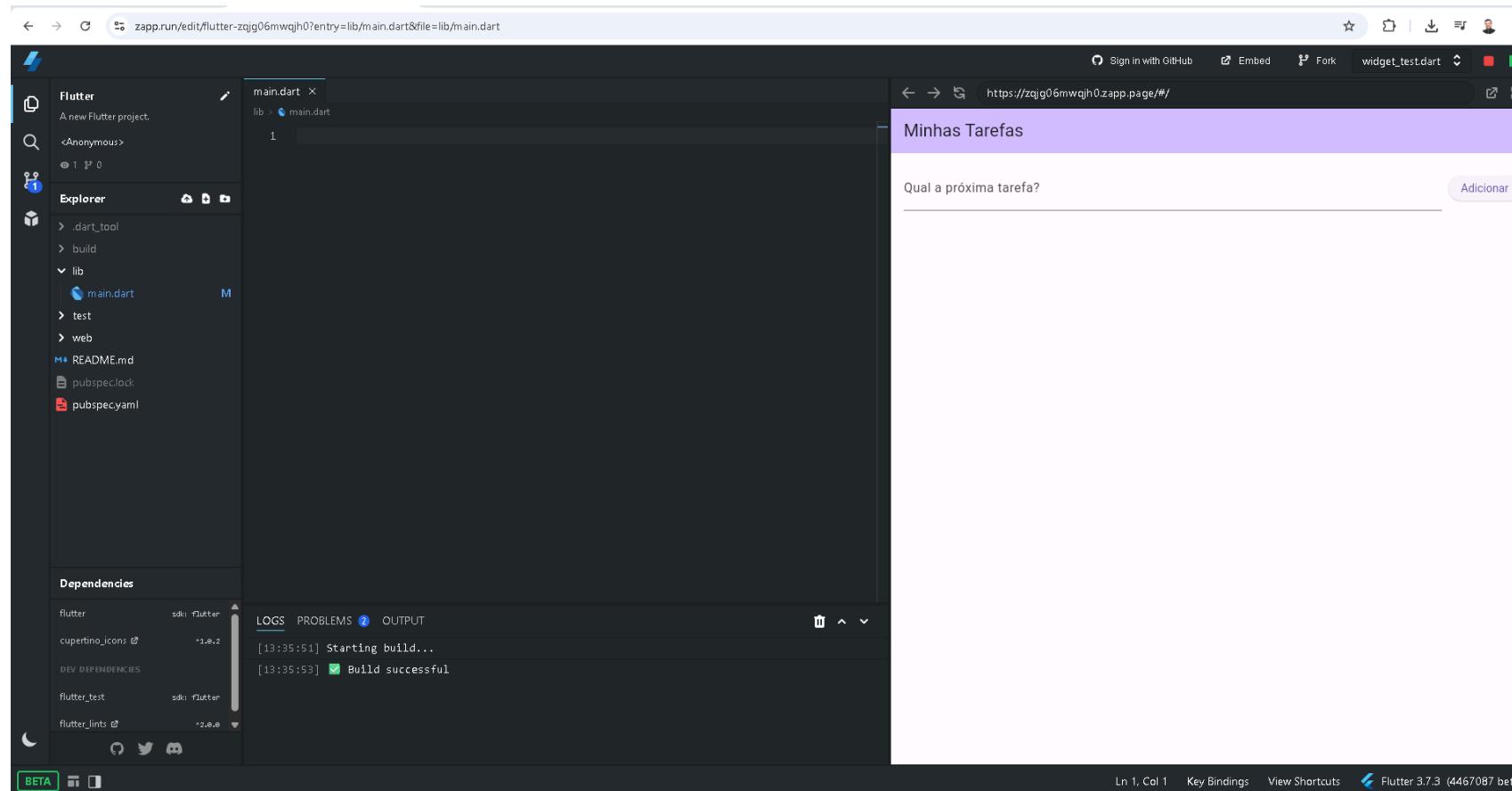


Passo 1: Acesse a Plataforma

Abra o Zapp.run no seu navegador. Você pode usar o link direto para o editor Flutter:

<https://zapp.run/edit/flutter>

Você verá uma janela de código à esquerda (o editor) e uma pré-visualização do aplicativo à direita (o *Canvas*).



Passo 2: O Código Completo

- A plataforma Zapp.run foca no código. Portanto, em vez de arrastar e soltar widgets, vamos escrever o código que os representa.
- Copie todo o código abaixo e cole-o no editor do Zapp.run (substituindo o código de exemplo que já vem nele).
- O aplicativo "Minhas Tarefas" aparecerá e funcionará instantaneamente na tela de pré-visualização.

The screenshot shows the Zapp.run editor interface. The title bar says "zapp.run/edit/flutter-zqjg06mwqh0?entry=lib/main.dart&file=lib/main.dart". The main area displays the code for the `main.dart` file:

```
1
```

The code is annotated with large white text: "main.dart" above the first line and "em anexo" below it.

The left sidebar shows a project structure for a Flutter project:

- Flutter
- <Anonymous>
- 1 0
- Explorer
- .dart_tool
- build
- lib
- main.dart
- test
- web
- README.md
- pubspec.lock
- pubspec.yaml

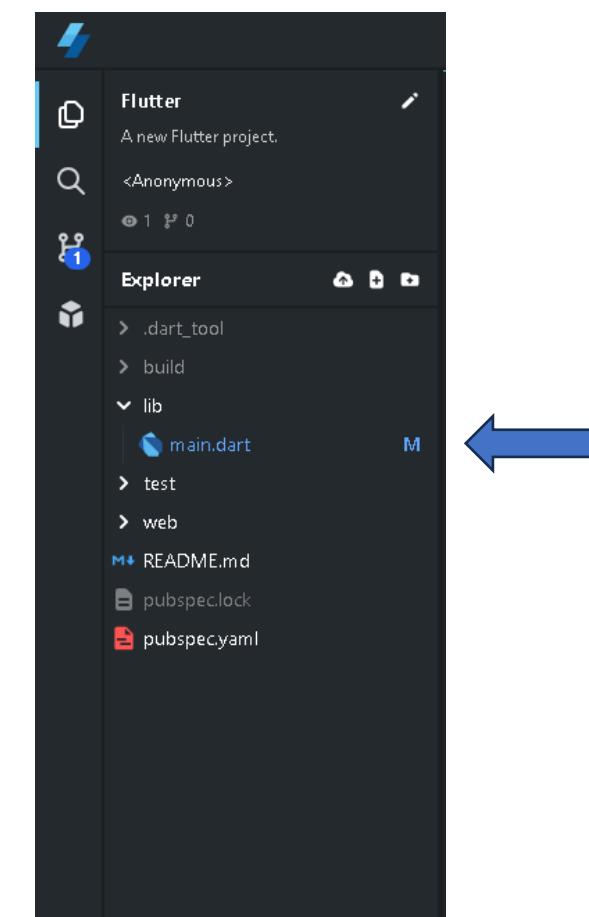
The bottom section shows dependencies:

flutter	sdk: flutter
cupertino_icons	*3.0.2
DEV_DEPENDENCIES	
flutter_test	sdk: flutter
flutter_lints	*2.0.0

Logs at the bottom show:

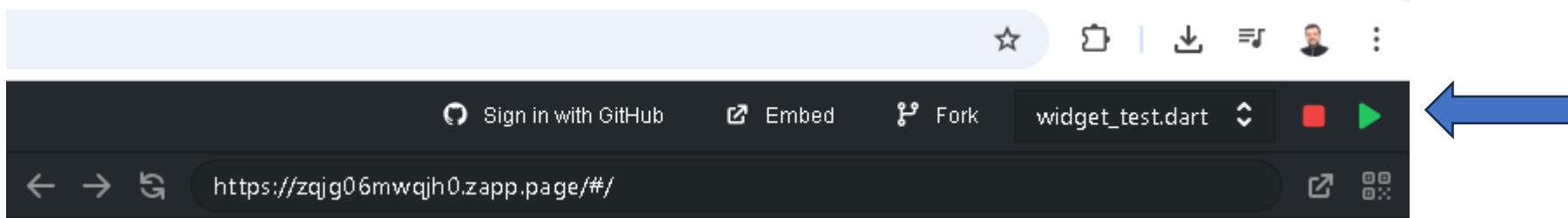
```
[13:35:51] Starting build...
[13:35:53] Build successful
```

Bottom right corner: BETA



Passo 3: Testando sua Criação

- (O "Test Mode")No Zapp.run, o teste é instantâneo.
- Assim que você cola o código:A pré-visualização à direita irá compilar e exibir o aplicativo.
- Digite uma tarefa no campo "Qual a próxima tarefa?".
- Clique no botão "Adicionar".
- A "mágica" acontecerá: sua tarefa aparecerá na lista abaixo, assim como planejado.



Adicionando um Botão de Excluir Tarefas

Para excluir uma tarefa, precisamos de duas coisas:

Um botão (vamos usar um ícone de lixeira) ao lado de cada tarefa.

Uma "instrução" (uma nova função) que diz ao aplicativo: "Quando este botão for clicado, pegue exatamente esta tarefa e a remova do nosso 'caderno de anotações' (`a listaDeTarefas`)".

Passo 1: Criando a "Lógica" para Remover

Primeiro, vamos criar a função que faz o trabalho de remoção.

Encontre um espaço livre dentro da classe `_TelaPrincipalTarefasState`, logo abaixo da função `_adicionarTarefa` que já criamos.

Copie e cole este novo bloco de código lá:

```
// ... (aqui fica a sua função _adicionarTarefa) ...

// 3. A "Ação" para REMOVER
// Função que será chamada quando o ícone de lixeira for clicado
void _removerTarefa(int index) {
    // "setState" avisa o Flutter que a tela precisa ser atualizada
    setState(() {
        // Remove o item da lista que está na posição "index"
        _listaDeTarefas.removeAt(index);
    });
}

// 4. Construindo a "Cara" do App
// (Este é o seu método @override build(BuildContext context) ...)
```

Entendendo este código:

`void _removerTarefa(int index):`

Criamos uma função chamada `_removerTarefa`.

A parte mais importante é o `(int index)`. Isso significa que, para a função funcionar, ela *precisa* receber um número (o `index`, ou seja, a "posição" do item na lista) para saber qual item exato ela deve apagar.

`setState(() { ... }):`

Assim como no "adicionar", usamos `setState` para avisar o Flutter que algo mudou e que ele precisa redesenhar a tela.

`_listaDeTarefas.removeAt(index):`

Este é o comando mágico.

Ele vai até o nosso "caderno de anotações" (`_listaDeTarefas`) e remove o item que está na posição (`index`) que recebemos.

Passo 2: Adicionando o "Visual" (O Botão de Lixeira)

Agora que temos a função que sabe como excluir, precisamos adicionar o botão que a chama.

Vá até o final do seu código, onde você constrói o `ListView.builder`.

Você encontrará o `ListTile` que usamos para mostrar a tarefa.

```
// ... (código do ListView.builder) ...
    itemBuilder: (context, index) {
        final tarefa = _listaDeTarefas[index];

        // O "ListTile" para cada item
        return ListTile(
            title: Text(tarefa), // Define o título
        );
    },
// ... (resto do código) ...
```

Nós vamos modificar esse `ListTile` para adicionar um botão nele.

O `ListTile` tem uma propriedade especial chamada `trailing`, que serve exatamente para colocar um ícone ou botão no canto direito do item.

```
// ... (código do ListView.builder) ...
itemBuilder: (context, index) {
    final tarefa = _listaDeTarefas[index];

    // O "ListTile" para cada item
    return ListTile(
        title: Text(tarefa), // Define o título

        // NOVO! Adicionando o ícone de lixeira
        trailing: IconButton(
            icon: const Icon(Icons.delete), // O ícone de lixeira
            color: Colors.red, // Deixa o ícone vermelho

            // Ação ao clicar no ícone
            onPressed: () {
                // Chama nossa função de remover!
                _removerTarefa(index);
            },
        ),
    );
}
// ... (resto do código) ...
```

- **trailing: IconButton(...)**: Estamos dizendo ao ListTile para colocar um IconButton (um botão de ícone) no espaço trailing (à direita).
- **icon: const Icon(Icons.delete)**: Estamos escolhendo o ícone de lixeira da biblioteca do Flutter.
- **onPressed: () { ... }**: Esta é a "ação de clique" do nosso novo botão.
- **_removerTarefa(index)**: Esta é a parte crucial! Quando o botão é pressionado, ele chama a função _removerTarefa que criamos no Passo 1. E o mais importante: ele passa o index (a posição exata da tarefa em que o usuário clicou) para que a função saiba qual item apagar.

Código Completo (Para Referência)

Caso tenha se perdido, aqui está o arquivo `main.dart` completo com as novas alterações.

Você pode copiar e colar tudo no Zapp.run para ver funcionando.

Arquivo: `main2.dart` em anexo.

Etec
Juscelino Kubitschek
de Oliveira
Diadema



maio de 2025