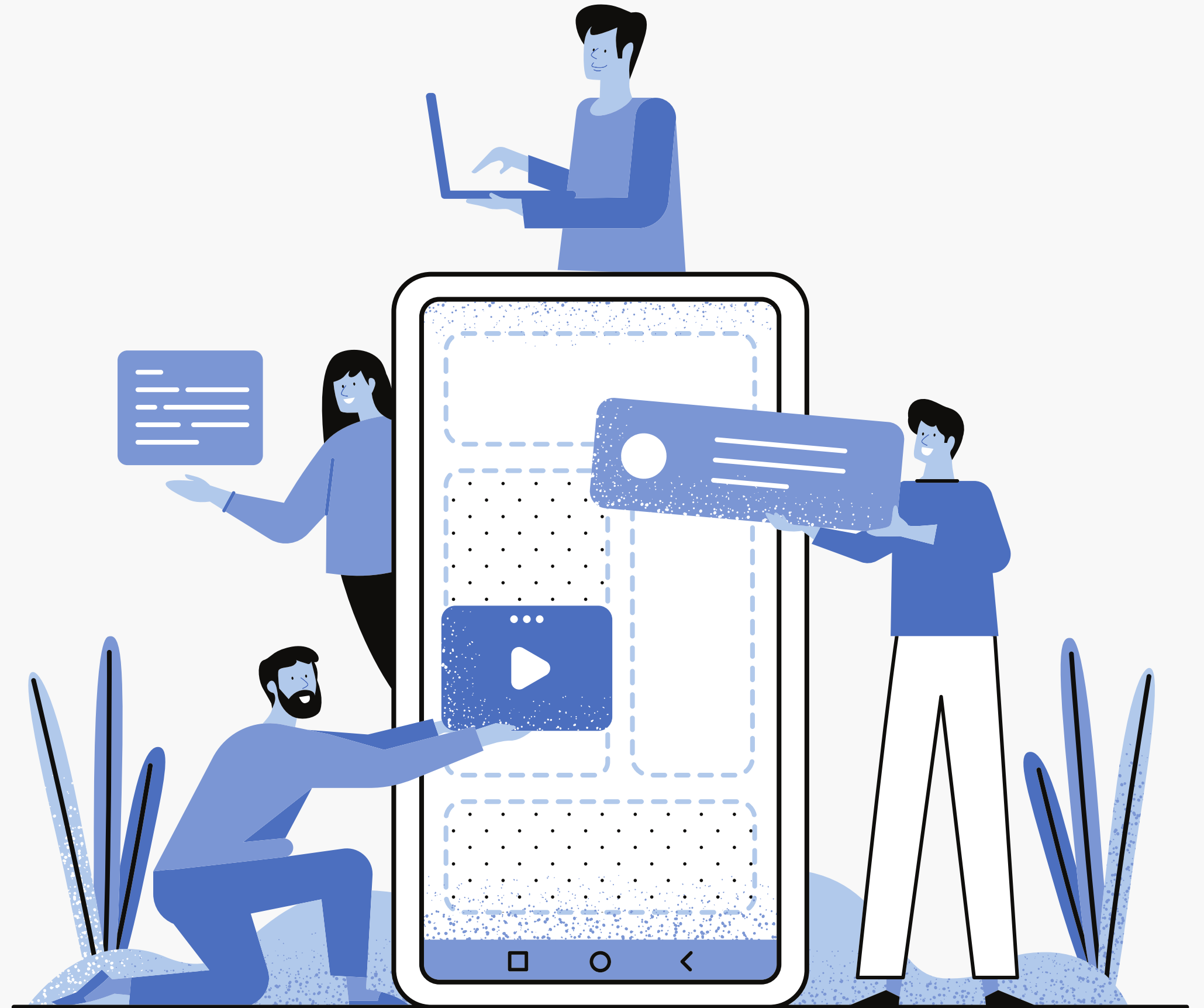


# Rotas

Curso Flutter de Verão



- Tratar roteamento das aplicações Flutter
- Definindo a estratégia de URL da web

# Objetivo da aula de hoje

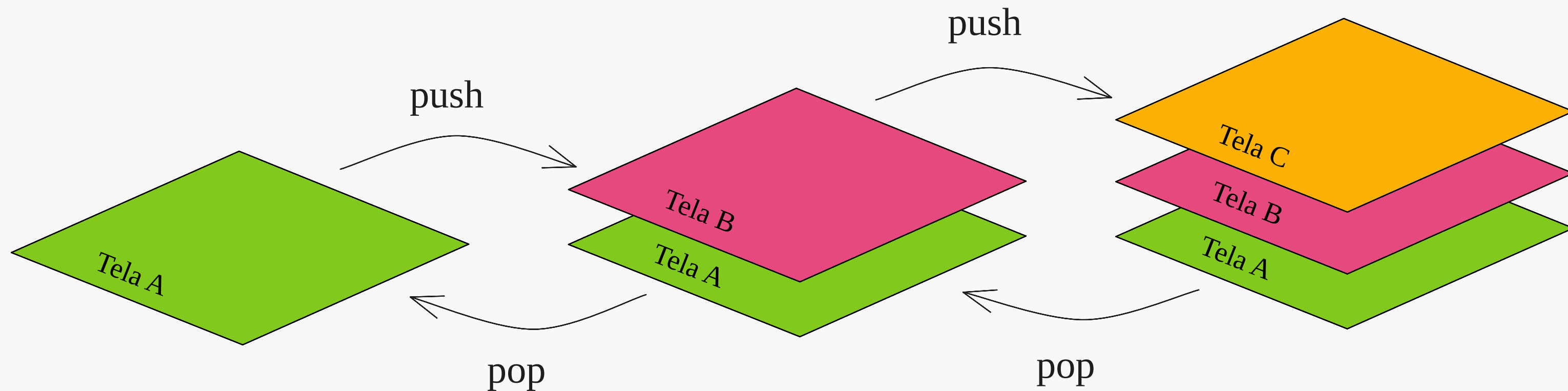



# Roteamento

É um mecanismo essencial no desenvolvimento de aplicativos que envolve a gestão das transições entre diferentes telas. Ele possibilita ao usuário navegar pelas diferentes partes do aplicativo de forma organizada. Isso inclui a exibição da tela inicial ao carregar o app, a abertura de novas telas e o controle da ação de retorno à tela anterior. Além das funcionalidades básicas, o roteamento requer um sistema robusto para organizar as telas, passar informações entre elas e gerenciar a pilha de navegação.

A implementação do roteamento envolve o uso da estrutura de dados chamada pilha (Stack), onde as páginas do aplicativo são empilhadas uma sobre a outra. O desenvolvedor pode adicionar páginas ao topo da pilha através da ação "push" e remover a página superior através da ação "pop".

A página no topo da pilha é a que o usuário está vendo, enquanto as outras páginas permanecem acessíveis e podem ser acessadas voltando pela pilha, normalmente usando o botão "Voltar".



A screenshot of a Flutter IDE window with a dark theme. The window has standard OS window controls (minimize, maximize, close) in the top right corner. The code is written in Dart and defines the main function for a Flutter application.

```
void main() {  
  runApp(const MaterialApp(  
    title: 'Minha Aplicação',  
    home: FirstRoute(),  
  ));  
}
```

```
{Widget? home}
```

Type: `Widget?`

The widget for the default route of the app (`[Navigator.defaultRouteName]`, which is `/`).

This is the route that is displayed first when the application is started normally, unless `[initialRoute]` is specified. It's also the route that's displayed if the `[initialRoute]` can't be displayed.

To be able to directly call `[Theme.of]`, `[MediaQuery.of]`, etc, in the code that sets the `[home]` argument in the constructor, you can use a `[Builder]` widget to get a `[BuildContext]`.

If `[home]` is specified, then `[routes]` must not include an entry for `/`, as `[home]` takes its place.


The `[Navigator]` is only built if routes are provided (either via `[home]`, `[routes]`, `[onGenerateRoute]`, or `[onUnknownRoute]`); if they are not, `[builder]` must not be null.



```
/// Navegar para uma nova página usando `push`  
Navigator.push(  
  context,  
  MaterialPageRoute(builder: (context) => const SecondRoute()),  
);  
  
/// Volte para a página anterior usando `pop`  
Navigator.pop(context);
```

# Rota nomeada

No desenvolvimento Flutter, para lidar com escalabilidade, é recomendado adotar a atribuição de nomes às rotas das páginas do aplicativo. Isso evita a necessidade de criar uma nova instância de rota a cada nova página, permitindo, em vez disso, empilhar uma página pelo seu nome associado. Cada nome de rota é vinculado a uma classe de rota específica. As rotas nomeadas no Flutter são facilmente implementadas através da definição de todas as rotas no parâmetro "routes" do widget MaterialApp, simplificando assim a gestão de navegação em um aplicativo.



```
void main() {  
  runApp(MaterialApp(  
    home: FirstRoute(), // Initial route named '/'  
    routes: <String, WidgetBuilder> {  
      '/second': (BuildContext context) => SecondRoute(),  
    },  
  ));  
}
```

```
/// Push route call '/second'  
Navigator.pushNamed(context, '/second');
```



# Definindo a estratégia de URL da web

O pacote "url\_strategy" é uma ferramenta útil para gerenciar a estratégia de URL em aplicações web desenvolvidas com Flutter. Ele oferece recursos avançados para controlar como as URLs são manipuladas no navegador, incluindo a capacidade de utilizar URLs mais amigáveis e significativas para os usuários. Ao adotar esse pacote, os desenvolvedores podem melhorar a experiência de navegação, permitindo que os URLs reflitam a estrutura e o conteúdo da aplicação.

# HashUrlStrategy vs PathUrlStrategy

A estratégia **HashUrlStrategy** usa o fragmento da URL (a parte após o **#**) para representar as rotas. Quando você navega entre as páginas, apenas o fragmento é alterado, e o navegador não recarrega a página inteira. Por exemplo, ao acessar **http://example.com/#/my-page**, o aplicativo irá interpretar **/my-page** como a rota.

A estratégia **PathUrlStrategy** usa o caminho da URL para representar as rotas. Isso significa que as informações de rota são incorporadas diretamente no caminho da URL após o domínio. Por exemplo, se você tiver um aplicativo em **http://example.com**, e a rota **/my-page** for acessada, a URL completa será **http://example.com/my-page**.



```
import 'package:url_strategy/url_strategy.dart';

void main() {
  // Aqui definimos a estratégia de URL para nosso aplicativo da
  web.
  // É seguro chamar esta função ao executar no celular ou no
  desktop também.
  setPathUrlStrategy();
  runApp(MyApp());
}
```

# go\_router

Um pacote de roteamento declarativo para Flutter que usa a API do roteador para fornecer uma API conveniente baseada em url para navegar entre telas diferentes.

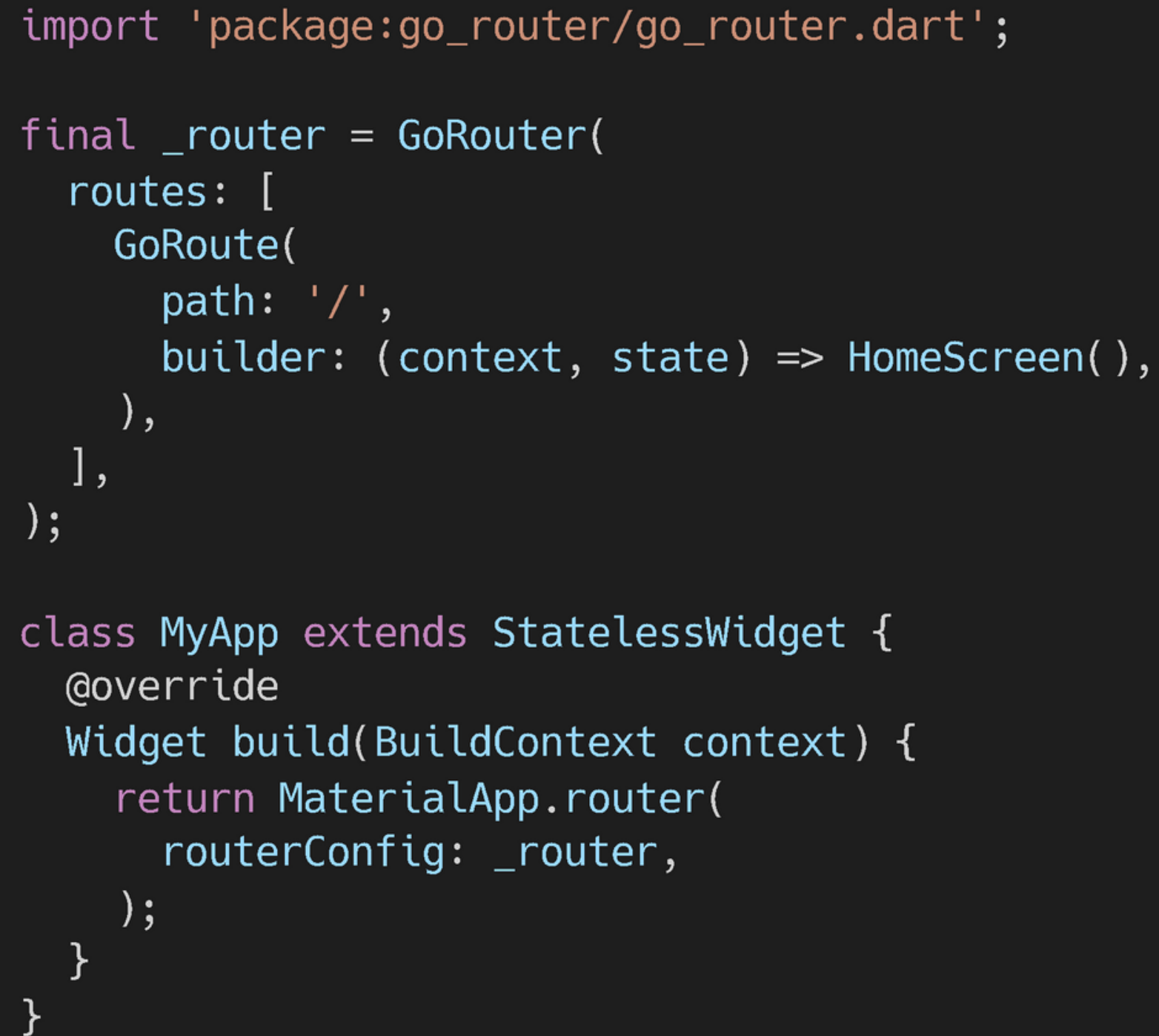
Você pode definir padrões de URL, navegar usando um URL, lidar com links diretos e vários outros cenários relacionados à navegação.

link: [https://pub.dev/packages/go\\_router](https://pub.dev/packages/go_router)

# Características

O GoRouter possui vários recursos para simplificar a navegação:

- Parâmetros de caminho e consulta de análise usando uma sintaxe de modelo (por exemplo, "user/:id")
- Exibição de várias telas para um destino (sub-rotas)
- Suporte de redirecionamento – você pode redirecionar o usuário para uma URL diferente com base no estado do aplicativo, por exemplo, para um login quando o usuário não está autenticado
- Suporte para vários navegadores via ShellRoute – você pode exibir um navegador interno que exibe suas próprias páginas com base na rota correspondente. Por exemplo, para exibir um BottomNavigationBar que fica visível na parte inferior da tela
- Suporte para aplicativos Material e Cupertino
- Compatibilidade com versões anteriores com Navigator API



```
import 'package:go_router/go_router.dart';

final _router = GoRouter(
  routes: [
    GoRoute(
      path: '/',
      builder: (context, state) => HomeScreen(),
    ),
  ],
);

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp.router(
      routerConfig: _router,
    );
  }
}
```