



Aula 4 – Criando Layouts

Desenvolvimento de aplicações híbridas com Flutter

22-23 de Outubro/21

Revisão do Desafio 1

Assets

Assets

Declarando Assets

Podemos declarar Assets no *pubspec.yaml*, especificando o caminho dos arquivos ou pastas.

Pubspec.yaml

flutter:

assets:

- assets/tree.png
- assets/flowers/

Assets

Asset Variants

Variantes definem situações diferentes em que uma mesma imagem pode ser carregada.

Pubspec.yaml

flutter:

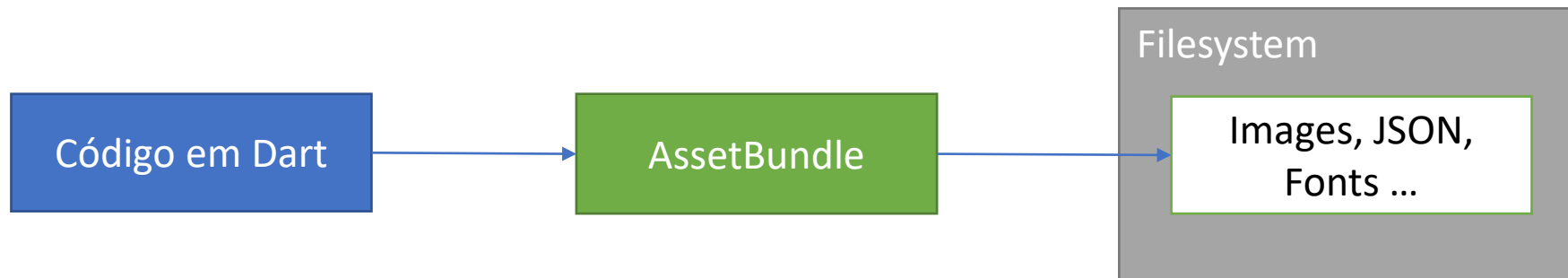
assets:

- assets/tree.png
- assets/3.0x/tree.png

Assets

Carregando Assets

- No Flutter, os assets no são empacotados em um **AssetBundle**.
- Usamos o AssetBundle para acessar os Assets em nosso código.



Assets

Carregando Assets

Todo app em flutter possui um AssetBundle global chamado de `rootBundle`:

```
import 'package:flutter/services.dart' show rootBundle;
```

```
Future<String> loadAsset() async {  
    return await rootBundle.loadString('assets/config.json');  
}
```

Podemos também obter um AssetBundle a partir do contexto com `DefaultAssetBundle.of()`

Assets

Carregando Assets de Imagem

O SDK disponibiliza o AssetImage para carregar imagens. Ele usa o AssetBundle por baixo dos panos, mas também trata as variantes de imagem por nós.

```
home: const Image(image: AssetImage('assets/tree.png'))
```


Temas

Temas

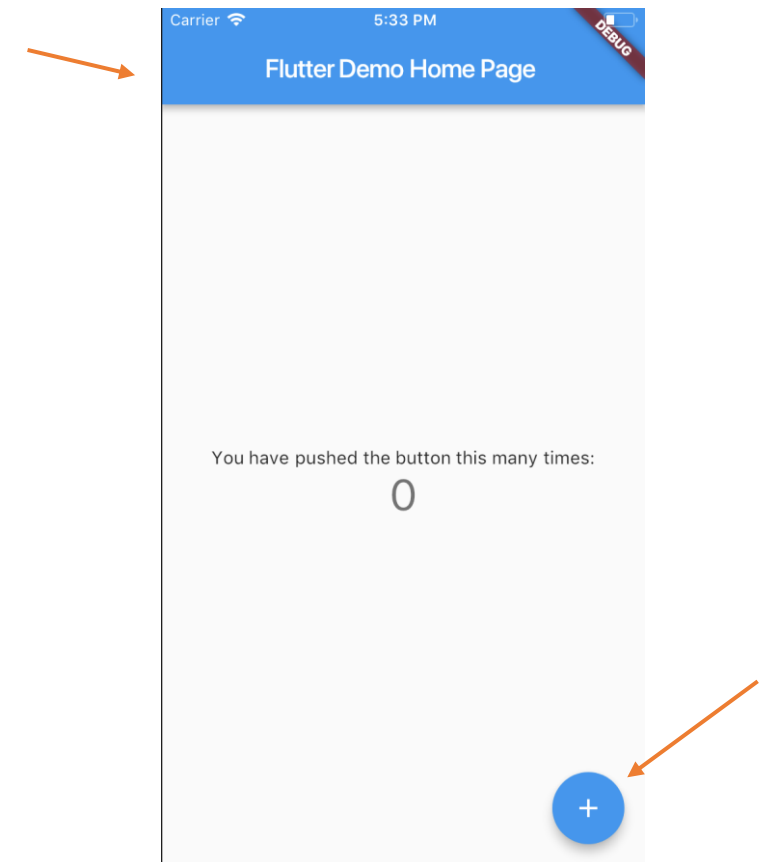
Temas definem o estilo dos componentes do app, desde a cor, até a fonte e tamanho dos elementos.

No Flutter, os temas são passados hereditariamente para os Widgets filhos, e eles podem acessar as propriedades do tema para definir seus parâmetros.

Temas

Ex:

O Floating Action Buttton e a AppBar decidem sua cor de acordo com a cor primaria do tema, definido no Widget `MaterialApp`

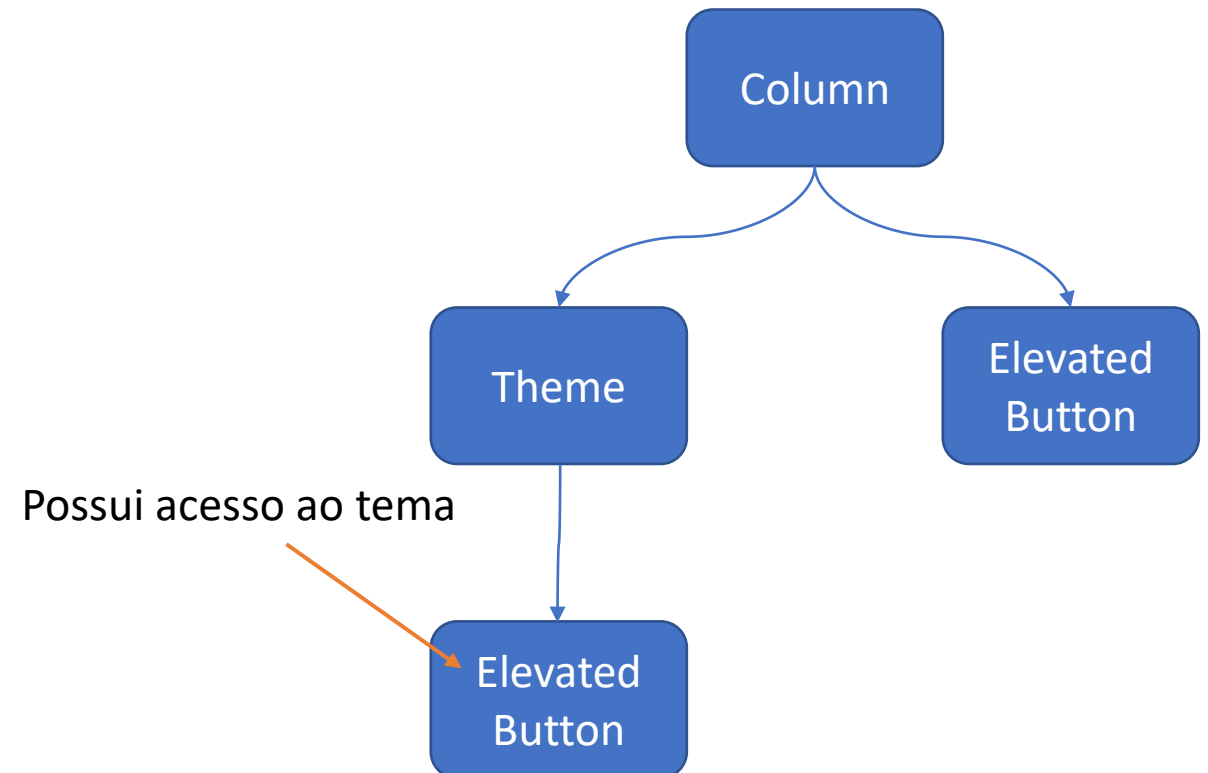


Temas

Criando um tema

Utilizamos a classe ThemeData para criar um tema e o Widget Theme, para aplicar o tema à todos os Widgets filhos.

```
Column(  
  children: <Widget> [  
    ElevatedButton(...),  
    Theme(  
      data: ThemeData(  
        primarySwatch: Colors.blue,  
        fontFamily: 'Georgia',  
      ),  
      child: ElevatedButton(...),  
    ),  
  ],  
)
```



Temas

Usando o tema em um Widget

Se quisermos mudar apenas uma propriedade do tema para os Widgets filhos:

```
// Dentro do Widget Theme
```

```
...
```

```
child: Container(color: Theme.of(context).colorScheme.primary)
```

Temas

Estendendo o tema pai

Se quisermos mudar apenas uma propriedade do tema pai:

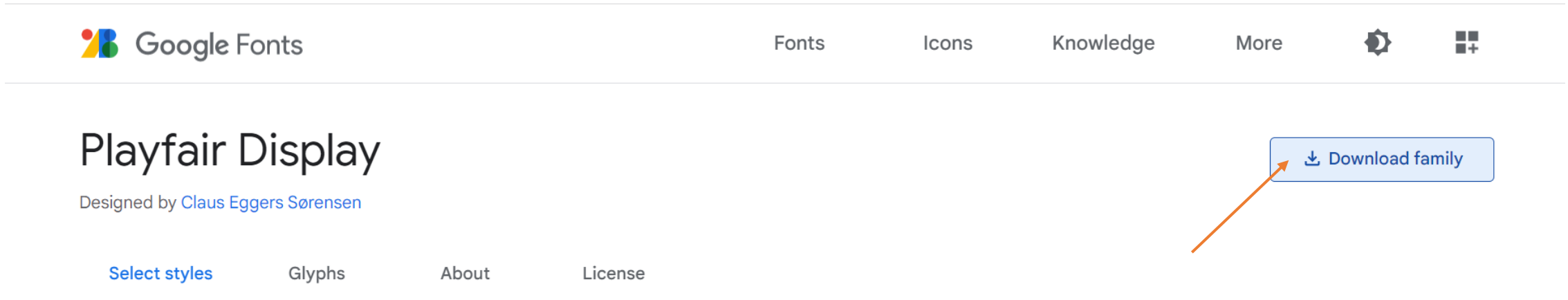
```
Theme(  
  data: Theme.of(context).copyWith(primaryColor: Colors.yellow),  
  child: ElevatedButton(onPressed: (){}, child: const Text("Button1")),  
)
```

Fontes

Fontes

Declarando fontes

1. Baixar o arquivo da fonte. (e.g Google Fonts)



Fontes

Declarando fontes

2. Declarar a fonte em `pubspec.yaml`.

```
flutter:  
  fonts:  
    - family: PlayfairDisplay  
      fonts:  
        - asset: assets/fonts/PlayfairDisplay-Regular.ttf
```

Fontes

Declarando fontes

2.1 Adicionar diferentes estilos de fonte.

```
flutter:  
  fonts:  
    - family: PlayfairDisplay  
      fonts:  
        - asset: assets/fonts/PlayfairDisplay-Regular.ttf  
        - asset: assets/fonts/PlayfairDisplay-Italic.ttf  
          style: italic
```

Fontes

Usando fontes

Agora que declaramos a fonte, ela estará disponível para usarmos no nosso código.

```
Center(  
  child: Text(  
    "Hello World",  
    style: TextStyle(  
      fontStyle: FontStyle.italic,  
      fontSize: 40,  
      fontFamily: 'PlayfairDisplay'  
    ),  
  ),  
)
```

Fontes

Google Fonts

1. Adicionar o pacote do [google_fonts](#)

```
$ flutter pub add google_fonts
```

```
$ flutter pub get
```

2. Adicionando a fonte no código

```
body: Center(  
  child: Text(  
    "Hello World",  
    style: GoogleFonts.lato()  
  )  
)
```

Navegação

Navegação

Navegação de uma app é um desafio no desenvolvimento de app, devido a:

- Navegação de telas do app mantendo o estado da aplicação correto;
- DeepLink;
- Notification:
 - Local Notification;
 - Push notification;

Navegação

Flutter tem um mecanismo imperativo de navegação, o ***Navigation*** Widget, e um declarativo roteamento de navegação, o ***Router*** Widget.

Inicialmente abordaremos o ***Navigation***, por ser mais simples, e já integrado no *Material App*.

Navegação

Navegando de uma pagina para outra

MaterialApp e **CupertinoApp** tem suporte ao API do **Navigator**, oferece as seguintes API para manipular a Stack:

```
Navigator.push(  
  context,  
  MaterialPageRoute(builder: (context) => const FirstScreen())  
);
```

```
Navigator.pop(context);
```


Navegação

Rotas Nomeadas

É possível definir as rotas e definir nomes para uma no parâmetro routes do **MaterialApp** ou **CupertinoApp**. O objetivo é pré definir as rotas:

```
MaterialApp(  
  initialRoute: '/home',  
  routes: {  
    '/home': (context) => const HomeScreen(),  
    '/first': (context) => const FirstScreen(),  
    '/second': (context) => const SecondScreen(),  
  }  
);
```

Navegação

Para navegar, segue a mesma forma de antes:

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

Para mover para uma rota pelo nome:

```
Navigator.pop(context);
```

Navegação

Para passar parâmetros em rotas nomeadas pode ser via o arguments:

```
Navigator.pushNamed(  
    context,  
    '/second',  
    arguments: SecondScreenDetail()  
);
```

```
class SecondScreenDetail {  
    // ...  
}
```

```
final args = ModalRoute.of(context)!.settings.arguments as SecondScreenDetail;
```

Navegação

Uma forma mais flexível em manipular rotas nomeadas é usar "*onGenerateRoute*", que permite manipular todas as rotas (adicionar regras de rotas)

```
MaterialApp(  
  initialRoute: "/",  
  onGenerateRoute: onGenerateRoute  
);
```

Navegação

```
Route<dynamic> onGenerateRoute(RouteSettings settings) {  
    if (settings.name == '/') {  
        return MaterialPageRoute(builder: (context) => const Home());  
    }  
    // Handle '/details/:id'  
    var uri = Uri.parse(settings.name!);  
    if (uri.pathSegments.length == 2 && uri.pathSegments.first == 'details')  
{  
        return MaterialPageRoute(  
            builder: (context) => DetailScreen(id: uri.pathSegments[1]));  
    }  
    return MaterialPageRoute(builder: (context) => const UnknownScreen());  
}
```

Navegação

Obtendo resultado de rotas

O resultado da página é retornado no método push ou pushNamed assincronamente.

```
_onBtnPressed(BuildContext context) async {  
  final result = await Navigator.pushNamed(  
    context,  
    '/second'  
  );  
}
```

```
_onBtnPressed(BuildContext context) {  
  Navigator.pop(context, "result");  
}
```

Formulários

Formulário


Usamos formulários quando queremos que o usuário insira múltiplos dados e que esses dados sejam validados de alguma forma.

Formulário

Criando um formulário

```
class UserFormState extends State<UserForm> {  
  
    final _formKey = GlobalKey<FormState>();  
  
    @override  
    Widget build(BuildContext context) {  
        return Form(  
            key: _formKey,  
            child: Column(  
                children: const [  
                    // Adicionar itens do formulário  
                ],  
            ),  
        );  
    }  
}
```

GlobalKey para
referenciar o estado do
Widget Form



Formulário

Definindo os campos

O TextFormField possui um validator, que pode ser usado quando queremos validar uma entrada

```
TextFormField(  
  validator: (value) {  
    if (value == null || value.isEmpty) {  
      return 'Please enter some text';  
    }  
    return null;  
  },  
)
```

Formulário

Validando o formulário

Podemos consultar programaticamente a GlobalKey para saber o estado atual do Formulário

```
_onConfirmPressed() {  
  if (_formKey.currentState!.validate()) {  
  
    ScaffoldMessenger.of(context).showSnackBar(  
      const SnackBar(content: Text('Processing Data')),  
    );  
  }  
}
```

Obrigado!