**Aula XVI e XVII - DevAppII**

**Navegação no Flutter**

**Trabalhando com rotas**

**Navegação básica**

**Transferir dados entre telas**

**Tutorial de Exemplo**

**Rotas nomeadas**

**@override e @immutable**

**Glossário da aula**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Fontes:***

<https://flutter.dev/>

<https://dev.to/henriquederosa/navegacao-com-flutter-14ki>

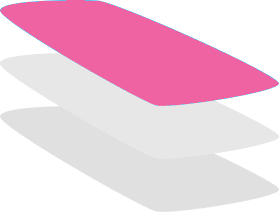
<https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html>

<https://medium.com/flutter-community/immutable-state-management-rebuilder-how-to-refactor-to-immutability-and-vice-versa-b9576d6744f2>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Navegação no Flutter**

A Navegação é uma parte essencial na maioria dos apps. Aqui vou abordar o que aprendi ao longo do desenvolvimento de alguns apps. Em aplicativos, a navegação é feita em **Stack,** isto é, pilha.



Por exemplo, temos uma tela chamada **Home**, a qual nos pode levar à próxima tela, **Configurações**, formando uma pilha. E esta, pode levar para Configurações de conta. Em cada tela nós podemos adicionar ou retirar uma tela da pilha.

**Navegação Básica**

Sabendo que ela acontece em Stack, vamos ver como adicionar ou remover telas dessa pilha de navegação.

**Para navegar, precisamos:**

1. **a tela inicial;**
2. **de uma ação para enviar o usuário à próxima tela;**
3. **a tela para a qual vamos direcionar o usuário.**

Depois disso, geralmente usamos os eventos: onPress ou onTap, para fazer uma ação sobre o Navigator.

| **Navigator.of(context).push(**  **MaterialPage Route(**  **builder: (\_) => NewPageScreen()**  **),**  **);** |
| --- |

Precisamos usar o **Navigator** (Navegador) do contexto, por isso Navigator.of(context). Ele possui os métodos necessários para controlar a **pilha (stack)**, alguns deles são: **push, pushNamed, pop**.

Nesse caso usamos o push, que precisa receber um **Route**. É possível criar o nosso próprio Route, mas o Material nos dá o **MaterialPageRoute, que recebe um parâmetro nomeado builder, onde podemos simplesmente retornar a página destino.**

**Transferir dados entre telas**

Pode ser que navegar entre telas não nos seja suficiente, mas podemos precisar enviar dados de uma tela para outra. **É possível fazer isso através dos construtores.** Desse modo, ao invés de retornarmos a tela sem parâmetros **NewScreen()**, nós podemos adicionar uma informação no seu construtor.

**Definição do construtor**

| **class NewScreen extends StatelessWidget {**  **const NewScreen(this.title);**  **final String title;**  **// [...]** |
| --- |

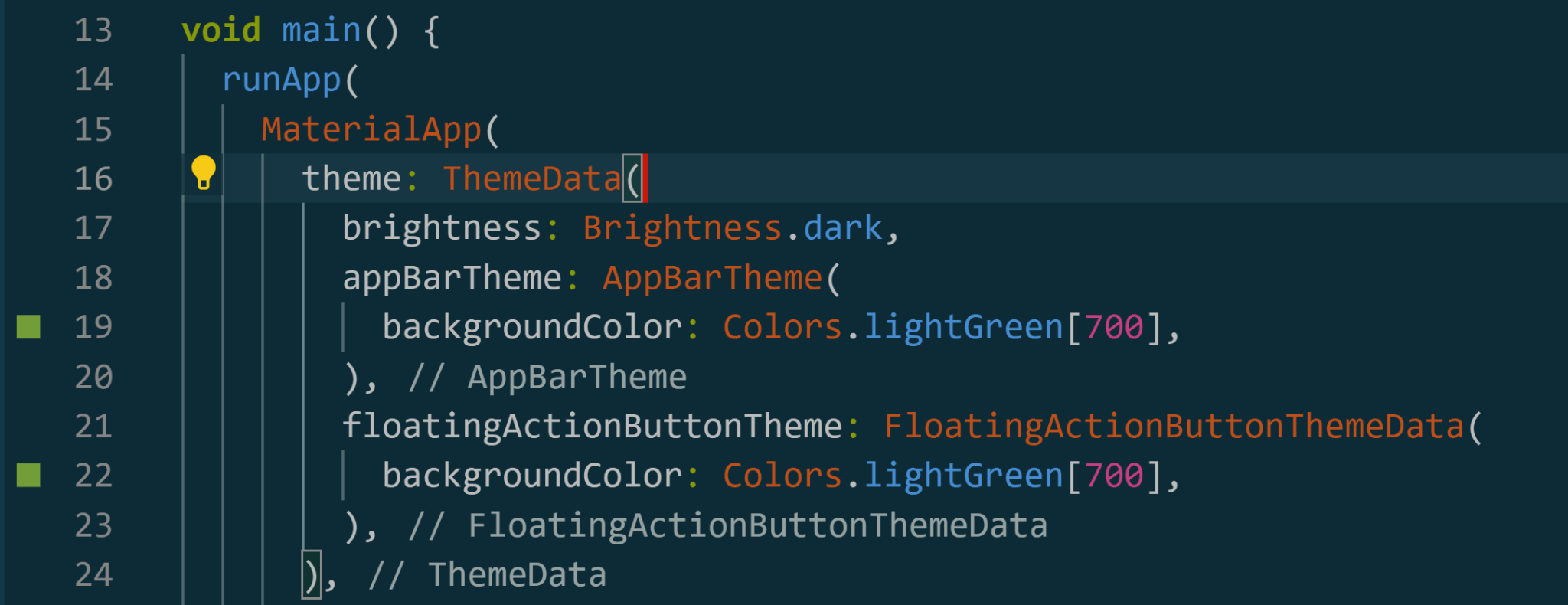
**Tutorial de Exemplo: Navegação e interação**

1. Vamos abrir um novo projeto Flutter, defina o nome desejado;
2. No arquivo main.dart, vamos declarar as constantes abaixo:

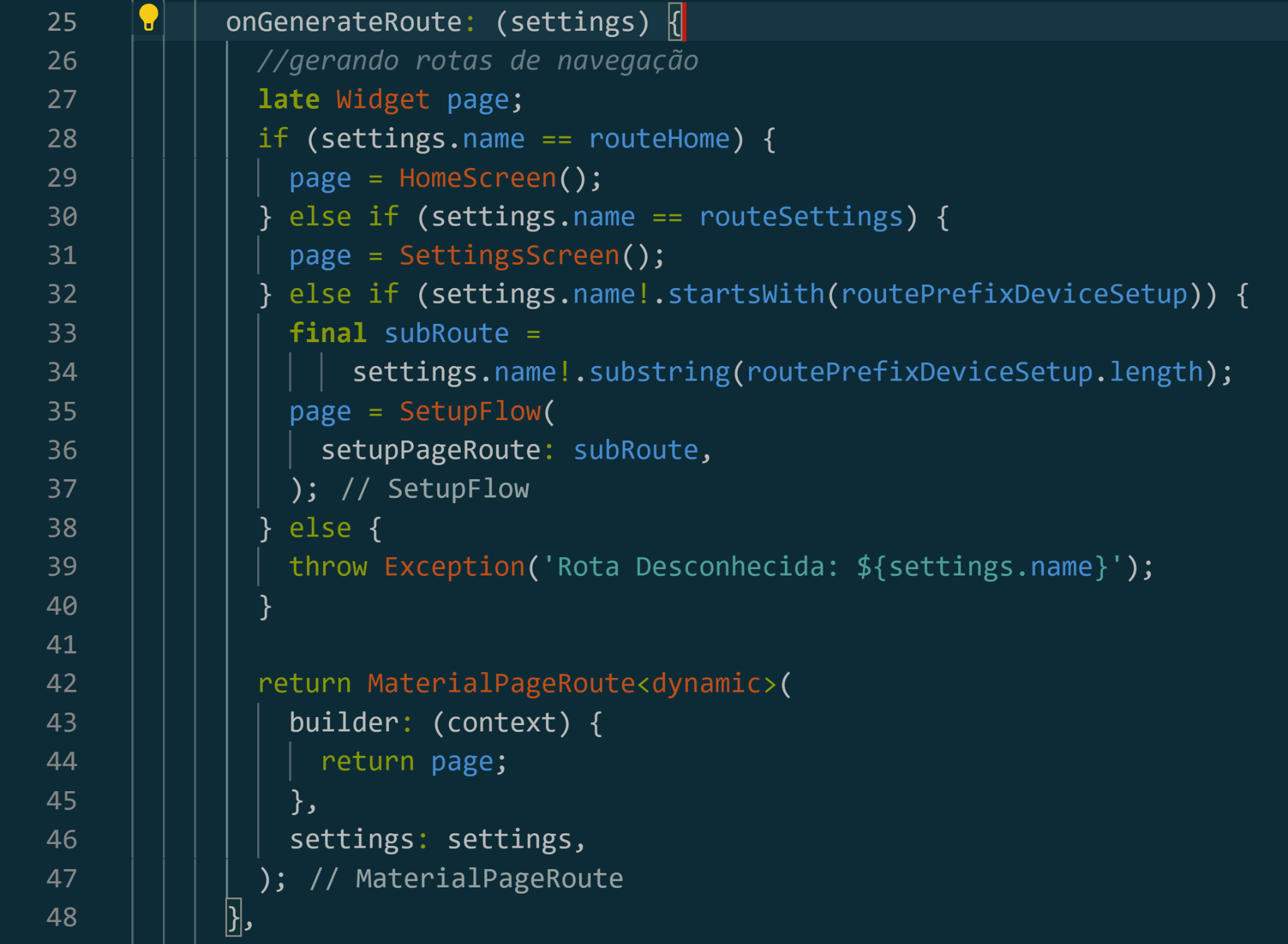


**Estas constantes servirão para armazenar as informações da nossa navegação**

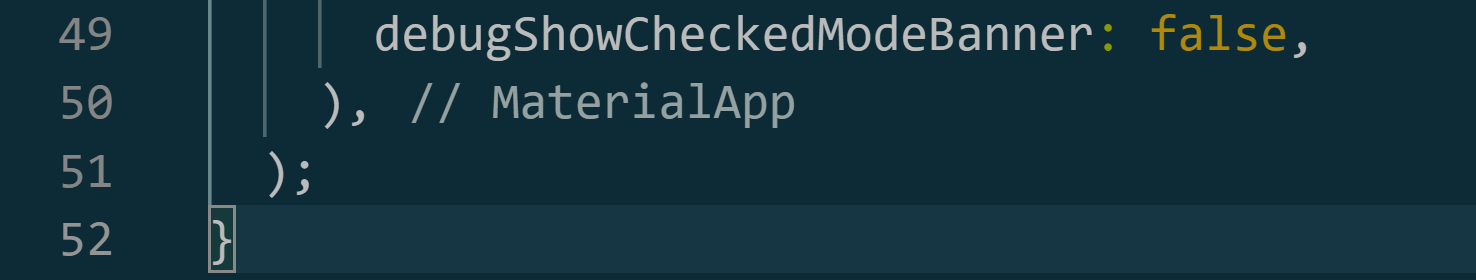
1. Vamos para a nossa main, nesta Main, já criaremos diretamente o nosso Theme - visto na aula passada. Cores, brilho e demais estilizações são livres:



1. A próxima etapa do nosso código será a criação nas nossas rotas utilizando as constantes que declaramos no início do código. Observe a lógica estabelecida para que o app faça as comparações. O “late” widget define uma sincronização dos widgets responsáveis pela exibição das Screens. A lógica da rota estabelecida neste tutorial é:
   1. Se a configuração do nome da rota é igual a configuração da rota do home, abre a primeira screen definida, senão se for igual a configuração do nome, abrirá outra screen e assim por diante. Portanto, temos uma estrutura condicional para fazer as comparações e consequentemente a navegação.



1. Finalizamos a nossa main e o MaterialApp:



1. Nesta parte do código iremos criar uma classe com StateFul, porém com a definição de **@immutable**. Desta forma, as instruções aqui colocadas tem as seguintes funcionalidades:
   1. são mais simples de construir, testar e usar
   2. aumenta a previsibilidade do código e dá garantia sobre seu estado a todos os objetos que dependem dele
   3. são muito mais fáceis de armazenar em cache
   4. esta classe está definindo as chaves e determina que irá sobrescrever as informações na criação de estado para ela.



1. **A próxima etapa do código são:**
   1. **configuração do estado inicial dos nossos widgets;**
   2. **configuração dos métodos;**
   3. **inserção dos elementos que pertencem a cada Screen;**
   4. **a própria declaração do MaterialPageRoute;**
   5. **a definição da navegação pela utilização do push, navigator**
   6. **inserção dos textos para navegação e experiência do usuário**

| class SetupFlowState extends State<SetupFlow> {  **final** \_navigatorKey = GlobalKey<NavigatorState>();  @**override**  **void** initState() {  super.initState();  }  **void** \_onDiscoveryComplete() {  \_navigatorKey.currentState!.pushNamed(routeDeviceSetupSelectDevicePage);  }  **void** \_onDeviceSelected(String deviceId) {  \_navigatorKey.currentState!.pushNamed(routeDeviceSetupConnectingPage);  }  **void** \_onConnectionEstablished() {  \_navigatorKey.currentState!.pushNamed(routeDeviceSetupFinishedPage);  }  Future<**void**> \_onExitPressed() async {  **final** isConfirmed = await \_isExitDesired();  if (isConfirmed && mounted) {  \_exitSetup();  }  }  Future<bool> \_isExitDesired() async {  return await showDialog<bool>(  context: context,  builder: (context) {  return AlertDialog(  title: Text('Você tem certeza que deseja sair?'),  content: Text(  'Se você sair da configuração do dispositivo, seus dados serão perdidos.'),  actions: [  TextButton(  onPressed: () {  Navigator.of(context).pop(true);  },  child: Text('Sair'),  ),  TextButton(  onPressed: () {  Navigator.of(context).pop(false);  },  child: Text('Continuar'),  ),  ],  );  }) ??  false;  }  **void** \_exitSetup() {  Navigator.of(context).pop();  }  @**override**  Widget build(BuildContext context) {  return WillPopScope(  onWillPop: \_isExitDesired,  child: Scaffold(  appBar: \_buildFlowAppBar(),  body: Navigator(  key: \_navigatorKey,  initialRoute: widget.setupPageRoute,  onGenerateRoute: \_onGenerateRoute,  ),  ),  );  }  Route \_onGenerateRoute(RouteSettings settings) {  **late** Widget page;  switch (settings.name) {  case routeDeviceSetupStartPage:  page = WaitingPage(  message: 'Localizando a versão mais próxima ...',  onWaitComplete: \_onDiscoveryComplete,  );  break;  case routeDeviceSetupSelectDevicePage:  page = SelectDevicePage(  onDeviceSelected: \_onDeviceSelected,  );  break;  case routeDeviceSetupConnectingPage:  page = WaitingPage(  message: 'Conectando...',  onWaitComplete: \_onConnectionEstablished,  );  break;  case routeDeviceSetupFinishedPage:  page = FinishedPage(  onFinishPressed: \_exitSetup,  );  break;  }  return MaterialPageRoute<dynamic>(  builder: (context) {  return page;  },  settings: settings,  );  }  PreferredSizeWidget \_buildFlowAppBar() {  return AppBar(  leading: IconButton(  onPressed: \_onExitPressed,  icon: Icon(Icons.chevron\_left),  ),  title: Text('Detectando...'),  );  }  } |
| --- |

1. A próxima classe é da segunda screen de navegação, onde após solicitar a inclusão de novo dispositivo, ele irá verificar a disponibilidade (hipoteticamente, porque aqui não temos conexão com data base).

class SelectDevicePage extends StatelessWidget {

**const** SelectDevicePage({

Key? key,

**required** this.onDeviceSelected,

}) : super(key: key);

**final** **void** Function(String deviceId) onDeviceSelected;

@**override**

Widget build(BuildContext context) {

return Scaffold(

body: Center(

child: Padding(

padding: **const** EdgeInsets.symmetric(horizontal: 24.0),

child: Column(

mainAxisSize: MainAxisSize.min,

children: [

Text(

'Selecionar a versão mais próxima:',

style: Theme.of(context).textTheme.headline6,

),

SizedBox(height: 24),

SizedBox(

width: double.infinity,

height: 54,

child: ElevatedButton(

style: ButtonStyle(

backgroundColor: MaterialStateColor.resolveWith((states) {

return **const** Color(0xFF222222);

}),

),

onPressed: () {

onDeviceSelected('22n483nk5834');

},

child: Text(

'AndyBug 22n483nk5834',

style: TextStyle(

fontSize: 24,

),

),

),

),

],

),

),

),

);

}

}

1. Criaremos a nossa WaitingPage em Stateful, observe as chaves e a definição dos objetos message e onWaiComplete e estamos sobrescrevendo o estado do widget.



1. Próxima classe, da Screen Waiting stage. Observe que cada screen possui a estrutura da tela que queremos exibir.

class \_WaitingPageState extends State<WaitingPage> {

@**override**

**void** initState() {

super.initState();

\_startWaiting();

}

Future<**void**> \_startWaiting() async {

await Future<dynamic>.delayed(**const** Duration(seconds: 3));

if (mounted) {

widget.onWaitComplete();

}

}

@**override**

Widget build(BuildContext context) {

return Scaffold(

body: Center(

child: Padding(

padding: **const** EdgeInsets.symmetric(horizontal: 24.0),

child: Column(

mainAxisSize: MainAxisSize.min,

children: [

CircularProgressIndicator(),

SizedBox(height: 32),

Text(widget.message),

],

),

),

),

);

}

}

1. Classe da Screen final

class FinishedPage extends StatelessWidget {

**const** FinishedPage({

Key? key,

**required** this.onFinishPressed,

}) : super(key: key);

**final** VoidCallback onFinishPressed;

@**override**

Widget build(BuildContext context) {

return Scaffold(

body: Center(

child: Padding(

padding: **const** EdgeInsets.symmetric(horizontal: 24.0),

child: Column(

mainAxisSize: MainAxisSize.min,

children: [

Container(

width: 250,

height: 250,

decoration: BoxDecoration(

shape: BoxShape.circle,

color: **const** Color(0xFF222222),

),

child: Center(

child: Icon(

Icons.android,

size: 175,

color: Colors.white,

),

),

),

SizedBox(height: 32),

Text(

'Bugdroid adicionado!',

textAlign: TextAlign.center,

style: TextStyle(

fontSize: 24,

fontWeight: FontWeight.bold,

),

),

SizedBox(height: 32),

ElevatedButton(

style: ButtonStyle(

padding: MaterialStateProperty.resolveWith((states) {

return **const** EdgeInsets.symmetric(

horizontal: 24, vertical: 12);

}),

backgroundColor: MaterialStateColor.resolveWith((states) {

return **const** Color(0xFF222222);

}),

shape: MaterialStateProperty.resolveWith((states) {

return StadiumBorder();

}),

),

onPressed: onFinishPressed,

child: Text(

'Finalizar',

style: TextStyle(

fontSize: 24,

),

),

),

],

),

),

),

);

}

}

1. Classe HomeScreen(). Novamente as configurações do Navigator e as ações desta navegação

@**immutable**

class HomeScreen extends StatelessWidget {

**const** HomeScreen({

Key? key,

}) : super(key: key);

@**override**

Widget build(BuildContext context) {

return Scaffold(

appBar: \_buildAppBar(context),

body: Center(

child: Padding(

padding: **const** EdgeInsets.symmetric(horizontal: 24.0),

child: Column(

mainAxisSize: MainAxisSize.min,

children: [

Container(

width: 250,

height: 250,

decoration: BoxDecoration(

shape: BoxShape.circle,

color: **const** Color(0xFF222222),

),

child: Center(

child: Icon(

Icons.android,

size: 175,

color: Theme.of(context).scaffoldBackgroundColor,

),

),

),

SizedBox(height: 32),

Text(

'Adicionar a versão do seu dispositivo',

textAlign: TextAlign.center,

style: TextStyle(

fontSize: 20,

fontWeight: FontWeight.bold,

),

),

],

),

),

),

floatingActionButton: FloatingActionButton(

onPressed: () {

Navigator.of(context).pushNamed(routeDeviceSetupStart);

},

child: Icon(Icons.add),

),

);

}

PreferredSizeWidget \_buildAppBar(BuildContext context) {

return AppBar(

title: Text('Configuração de dispositivo'),

actions: [

IconButton(

icon: Icon(Icons.settings),

onPressed: () {

Navigator.pushNamed(context, routeSettings);

},

),

],

);

}

}

1. Tela final de estilização.

class SettingsScreen extends StatelessWidget {

**const** SettingsScreen({

Key? key,

}) : super(key: key);

@**override**

Widget build(BuildContext context) {

return Scaffold(

appBar: \_buildAppBar(),

body: SingleChildScrollView(

child: Column(

mainAxisSize: MainAxisSize.min,

children: List.generate(8, (index) {

return Container(

width: double.infinity,

height: 54,

margin: **const** EdgeInsets.only(left: 16, right: 16, top: 16),

decoration: BoxDecoration(

borderRadius: BorderRadius.circular(8),

color: **const** Color(0xFF222222),

),

);

}),

),

),

);

}

PreferredSizeWidget \_buildAppBar() {

return AppBar(

title: Text('Settings'),

);

}

}

1. Tela final:



**Rotas nomeadas**

**Agora vamos falar das rotas nomeadas. Por que precisamos de rotas nomeadas?**

Bom, depende de cada caso, mas para muitas aplicações rotas nomeadas podem oferecer a vantagem de não ter código duplicado. Se quisermos navegar para a mesma tela em diferentes partes do app, do modo como fizemos até agora, teríamos que ter repetidas vezes **NewScreen().**