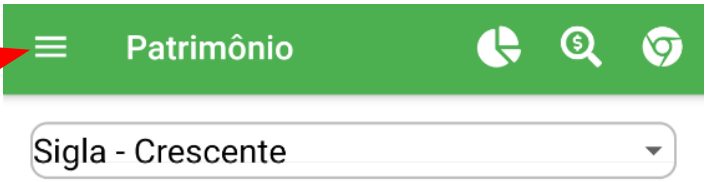
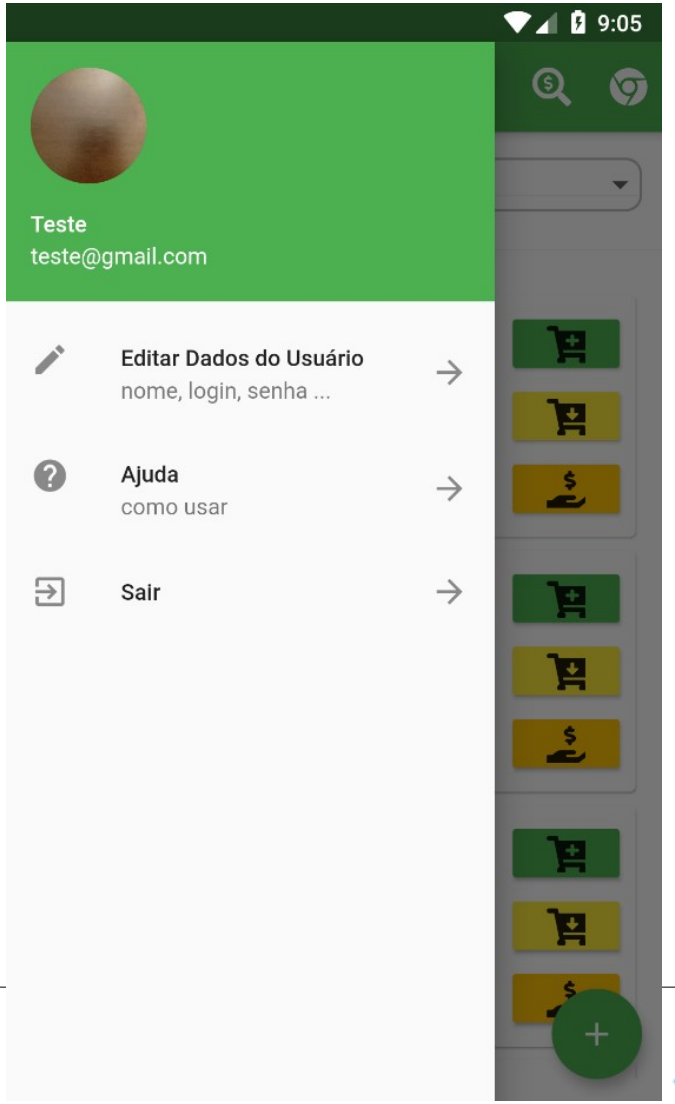
Menu Lateral (drawer) e FutureBuilder

Instituto Federal do Espirito Santo – Campus Colatina
Desenvolvimento para sistemas móveis - Flutter
Professor: Giovany Frossard Teixeira

Menu Lateral (drawer)

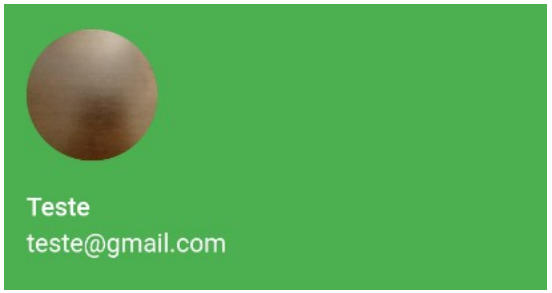


- O uso de menu lateral (drawer) é muito comum em aplicativos Android. Para facilitar sua implementação há um atributo chamado **drawer** nos **Scaffold** que recebe um widget que representa esse menu lateral.
- Esse menu lateral também funciona no iOS.
- O **MenuLateral** ao lado é apresentado para usuários do tipo “Padrao”.




```
@override
Widget build(BuildContext context) {
  return Scaffold(
    //...
    drawer: MenuLateral(),
  ); // Scaffold
}
```

MenuLateral




```dart
class MenuLateral extends StatelessWidget {
  Usuario usuario;

  UserAccountsDrawerHeader _header(ImageProvider imageProvider) {
    return UserAccountsDrawerHeader(
      accountName: Text(usuario.nome),
      accountEmail: Text(usuario.login),
      currentAccountPicture: CircleAvatar(
        backgroundImage: imageProvider,
      ), // CircleAvatar
    ); // UserAccountsDrawerHeader
  }
```

- MenuLateral é um **StatelessWidget**, ou seja, não permite mudança de estado (não possui initState, setState ou dispose).
- O método privado **_header** serve para desenhar o cabeçalho do **MenuLateral**.
- **accountName** é o texto maior do cabeçalho.
- **accountEmail** o texto que vem logo abaixo.
- **currentAccountPicture** é o widget que desenha a imagem. Nesse caso um **CircleAvatar** que tem como **backgroundImage** o parâmetro passado para essa função.

Flutter 2.0

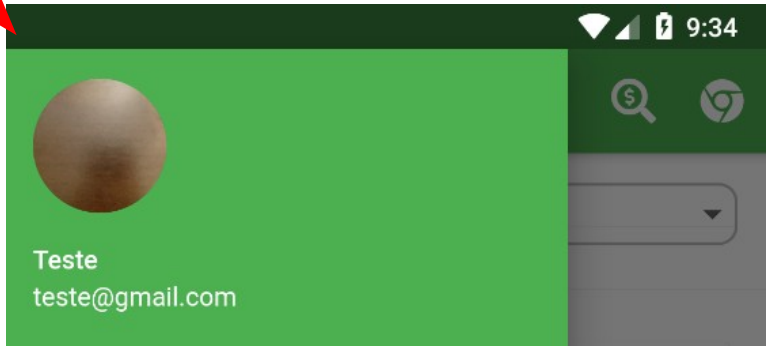
- O Flutter 2.0 o MenuLateral passou a ser um StatefulWidget. Isso ocorre pela necessidade do uso do initState para a obtenção do usuário.
- Em usuário teve-se que criar o método obterNaoNulo porque o **FutureBuilder** necessita que o future utilizado seja **Future<Usuario>** e não **Future<Usuario?>**.

```dart
class MenuLateral extends StatefulWidget {
  const MenuLateral({Key? key}) : super(key: key);

  @override
  _MenuLateralState createState() => _MenuLateralState();
}

class _MenuLateralState extends State<MenuLateral> {
  Usuario? usuario;
  Future<Usuario>? future;

  @override
  void initState() {
    super.initState();
    // A criação do future DEVE ser feita aqui
    // não pode ser feita no build
    // (https://api.flutter.dev/flutter/widgets/FutureBuilder-class.html)
    future = Usuario.obterNaoNulo();
  }

  UserAccountsDrawerHeader _header(ImageProvider imageProvider) {
    return UserAccountsDrawerHeader(
      accountName: Text(usuario!.nome!),
      accountEmail: Text(usuario!.login!),
      currentAccountPicture: CircleAvatar(
        backgroundImage: imageProvider,
      ),  // CircleAvatar
    );  // UserAccountsDrawerHeader
  }
```

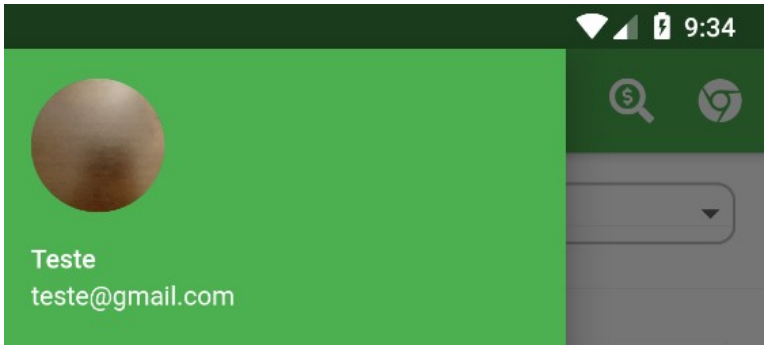

SafeArea e Drawer

- O widget **SafeArea** protege seu **child** de invasões do sistema operacional (barra de status na parte superior da tela) ou elementos físicos como Notch do iPhone X.
- O widget **Drawer** é quem efetivamente precisa ser desenhado como um **MenuLateral**. Ele terá um atributo **child** que efetivamente será responsável pelo desenho da tela. Nesse caso será desenhada uma **ListView** (o código interno será mostrado nos próximos slides).

```
@override
Widget build(BuildContext context) {
  Future<Usuario> future = Usuario.obter();
  return SafeArea(
    child: Drawer(
      child: ListView(
        children: <Widget>[

        ], // <Widget>[]
      ), // ListView
    ), // Drawer
  ); // SafeArea
}
```



FutureBuilder




```dart
@override
Widget build(BuildContext context) {
  Future<Usuario> future = Usuario.obter();
  return SafeArea(
    child: Drawer(
      child: ListView(
        children: <Widget>[

        ], // <Widget>[]
      ), // ListView
    ), // Drawer
  ); // SafeArea
}
```

- O **FutureBuilder** é parecido com o **StreamBuilder** porém mais simples. Enquanto o StreamBuilder está ligado a um valor que pode mudar com o tempo, o FutureBuilder é apenas uma resposta de um Future. Em outras palavras, um **FutureBuilder** é usado para uma **resposta única** enquanto um **StreamBuilder** pode **buscar os dados mais de uma vez**.
- Nesse exemplo, estamos obtendo um usuário utilizando **Shared Preferences**. Essa obtenção é assíncrona e o resultado dela precisa ser desenhado no cabeçalho do **Drawer**.
- De fato, precisaremos primeiro obter o usuário (Shared Preferences) e depois a imagem relativa a esse usuário (GerenciadoraArquivo), ambos procedimentos assíncronos.

Flutter 2.0

O future foi obtido no initState e não no próprio build

```
@override
Widget build(BuildContext context) {
  return SafeArea(
    child: Drawer(
      child: ListView(
        children: <Widget>[
          FutureBuilder<Usuario>(
            future: future,
            builder: (context, snapshot) {
              usuario = snapshot.data;
              if (usuario == null){
                return Container();
              }
              else if (usuario!.urlFoto != null){
                Future<File> future_arquivo = GerenciadoraArquivo.obterImagem(usuario!.urlFoto!);
```


FutureBuilder

```
Future<Usuario> future = Usuario.obter();

FutureBuilder<Usuario>(
  future: future,
  builder: (context, snapshot) {
    usuario = snapshot.data;
    if (usuario == null){
      return Container();
    }
    else if (usuario.urlFoto != null){
      Future<File> future_arquivo = GerenciadoraArquivo.obterImagem(usuario.urlFoto);
      return FutureBuilder<File>(
        future: future_arquivo,
        builder: (context, snapshot) {
          if(!snapshot.hasData){
            return Center(
              child: CircularProgressIndicator(),
              ); // Center
          }

          File imagem = snapshot.data;
          return _header(FileImage(imagem));
        }
      ); // FutureBuilder
    } else {
      return _header(AssetImage("assets/icon/icone_aplicacao.png"));
    }
  },
), // FutureBuilder
```

- O primeiro passo é passar o **future** que terá os dados a serem desenhados, nesse caso o future terá o usuário logado.
- No método **builder** temos **snapshot.data** com os dados do usuário (ou nulo, caso não tenham sido obtidos ainda). Caso nulo apenas será colocado um **Container** vazio.
- Caso haja usuário, mas ele não tenha foto, é passada para o método **_header** a imagem padrão da aplicação.
- Caso haja usuário, e ele tenha foto, é solicitado à **GerenciadoraArquivo** obter a imagem do usuário.
- Nesse contexto é construído um novo **FutureBuilder** que, enquanto não conseguir a imagem, irá apresentar um **CircularProgressIndicator**. Assim que a imagem for obtida o método **_header** é chamado para desenhar o cabeçalho.

Flutter 2.0

Igual exceto pelos símbolos do null-safe (safety)

```
FutureBuilder<Usuario>(
  future: future,
  builder: (context, snapshot) {
    usuario = snapshot.data;
    if (usuario == null){
      return Container();
    }
    else if (usuario.urlFoto != null){
      Future<File> future_arquivo = GerenciadoraArquivo.obterImagem(usuario.urlFo
      return FutureBuilder<File>(
        future: future_arquivo,
        builder: (context, snapshot) {
          if(!snapshot.hasData){
            return Center(
              child: CircularProgressIndicator(),
              ); // Center
            }

            File imagem = snapshot.data;
            return _header(FileImage(imagem));
          }
      ); // FutureBuilder
    } else {
      return _header(AssetImage("assets/icon/icone_aplicacao.png"));
    }
  },
), // FutureBuilder
```

```
FutureBuilder<Usuario>(
  future: future,
  builder: (context, snapshot) {
    usuario = snapshot.data;
    if (usuario == null){
      return Container();
    }
    else if (usuario!.urlFoto != null){
      Future<File> future_arquivo = GerenciadoraArquivo.obterImagem(usuario!.urlFoto!);
      return FutureBuilder<File>(
        future: future_arquivo,
        builder: (context, snapshot) {
          if(!snapshot.hasData){
            return Center(
              child: CircularProgressIndicator(),
              ); // Center
            }

            File? imagem = snapshot.data;
            return _header(FileImage(imagem!));
          }
      ); // FutureBuilder
    } else {
      return _header(AssetImage("assets/icon/icone_aplicacao.png"));
    }
  },
), // FutureBuilder
```

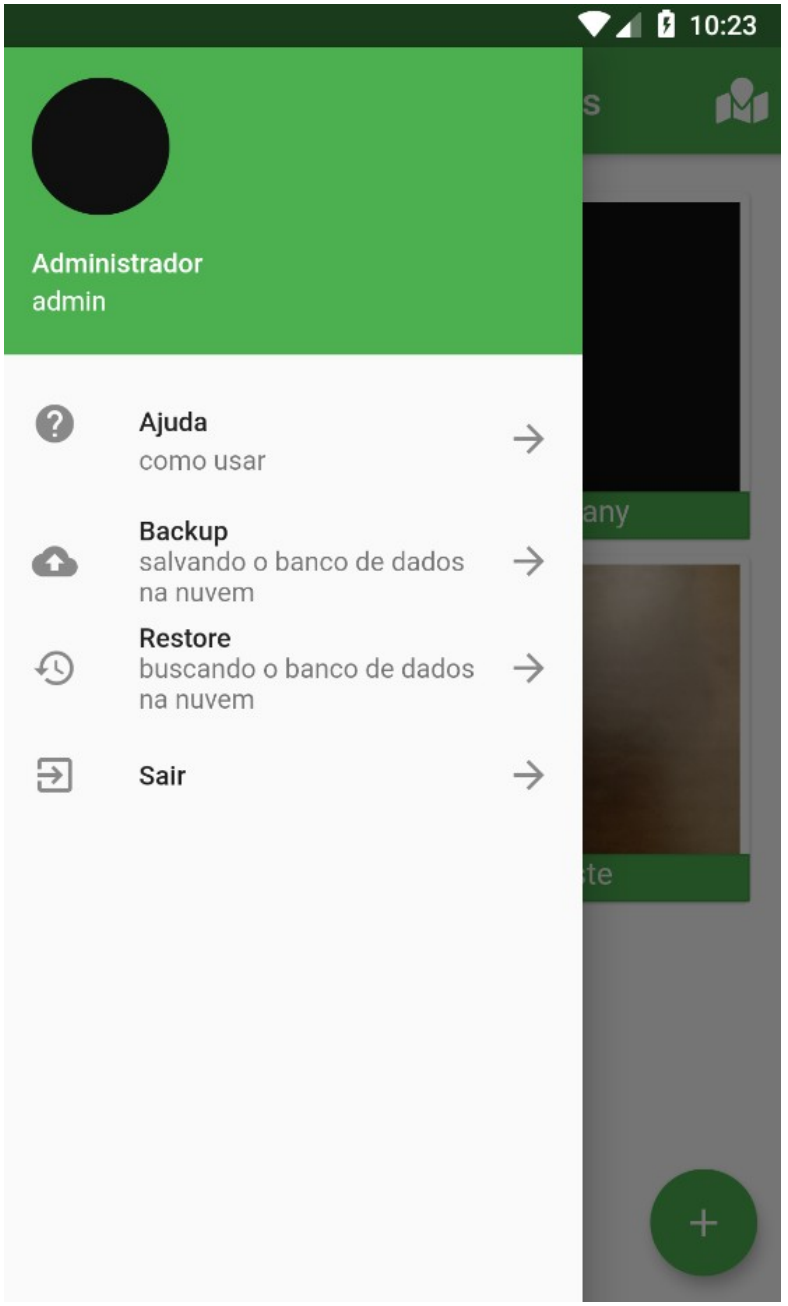

ListTitle

- Complementando o **ListView** do **Drawer** temos os 3 menus do tipo ListTitle.


```dart
ListTile(
  leading: Icon(Icons.edit),
  title: Text("Editar Dados do Usuário"),
  subtitle: Text("nome, login, senha ..."),
  trailing: Icon(Icons.arrow_forward),
  onTap: () async {
    // Fechando o menu lateral
    pop(context);

    push(context, TelaEdicaoUsuario(usuario));
  }
), // ListTile
ListTile(
  leading: Icon(Icons.help),
  title: Text("Ajuda"),
  subtitle: Text("como usar"),
  trailing: Icon(Icons.arrow_forward),
  onTap: () {
    // Fechando o menu lateral
    pop(context);

    push(context, TelaAjuda());
  },
), // ListTile
ListTile(
  leading: Icon(Icons.exit_to_app),
  title: Text("Sair"),
  trailing: Icon(Icons.arrow_forward),
  onTap: () {
    // Fechando o menu lateral
    pop(context);

    // Sobrescrevendo a tela de Login
    push(context, TelaLogin(), replace: true);

    // Retirando o usuário das Shared Preferences
    Usuario.limpar();
  },
) // ListTile
```

MenuLateralAdmin

- Para usuários do tipo “Administrador” o menu lateral é diferente do que vimos até agora. O **MenuLateralAdmin** possui inclusive diferenciação entre Android e iOS.
- Para iOS os menus “Backup” e “Restore” não existem pois essas funcionalidades não foram implementadas para iOS.



MenuLateralAdmin

```dart
class MenuLateralAdmin extends StatelessWidget {
  Usuario usuario;

  UserAccountsDrawerHeader _header(ImageProvider imageProvider) {
    return UserAccountsDrawerHeader(
      accountName: Text(usuario.nome),
      accountEmail: Text(usuario.login),
      currentAccountPicture: CircleAvatar(
        backgroundImage: imageProvider,
      ), // CircleAvatar
    ); // UserAccountsDrawerHeader
  }
```

- Poderíamos ter feito uma hierarquia de menus laterais e definido o método **_header** no ancestral, mas isso não foi feito para evitar acoplamento (se quiser mudar, por exemplo, **accountName** posso fazê-lo sem me preocupar se afetará outro MenuLateral).
- Entretanto essa abordagem pode ser mudada se o método **_header** for usado frequentemente. Uma boa prática aqui poderia ser definir um widget customizado de Drawer, deixar o método **_header** pré definido com parâmetros **accountName**, **accountEmail** além do parâmetro atual. Daí evitaríamos a necessidade de definir esse método para fazer o cabeçalho do Drawer.

MenuLateralAdmin

```
@override
Widget build(BuildContext context) {
  Future<Usuario> future = Usuario.obter();
  return SafeArea(
    child: Drawer(
      child: ListView(
        children: Platform.isAndroid
            ? <Widget>[
                _cabecalho(future),
                _ajuda(context),
                _backup(context),
                _restore(context),
                _sair(context)
              ] // <Widget>[]
            : <Widget>[
                _cabecalho(future),
                _ajuda(context),
                _sair(context),
              ], // <Widget>[]
      ), // ListView
    ), // Drawer
  ); // SafeArea
}
```

- Aqui foram criados métodos para cada um dos menus deixando o método **build** mais legível.
- A obtenção do usuário via **Shared Preferences** continua (no Flutter 2.0 o future do usuário é obtido no initState), mas os FutureBuilder foram para dentro do método **_cabecalho**.
- Usamos **Platform.isAndroid** para diferenciar se o dispositivo é Android ou iOS. Se for Android possui os menus de **Backup** e **Restore**, se for iOS esses menus não vão aparecer.

Dúvidas?