

Telas de Abertura e Login – Navegação entre telas



Tela de abertura

- Veremos como fazer uma tela de abertura como ao lado:
- A ideia que será utilizada é sempre ter um controlador de interação para cada tela. Deixando no código da tela o que for necessário para desenhar a tela, com os demais comportamentos indo para essa classe controladora.
- Isso se relaciona com o padrão BloC, tendo objetivo final similar. No nosso caso, é o clássico modelo de divisão em camadas.



Flutter 2.0

Por ora vou optar por mostrar o suporte a Flutter 2.0 e a código legado (antes do null safety)

- Com o Flutter 2.0 veio o suporte a null safety e uma série de atualizações de bibliotecas. Especificamente para uso do Firebase faz-se necessário o seguinte código:

```
// Para chamar firebase.initializeApp()
import 'package:firebase_core/firebase_core.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}
```



Tela de abertura

```
import 'package:flutter/material.dart';
import 'package:nalista/telas/controle_interacao/controle_tela_abertura.dart';

class TelaAbertura extends StatefulWidget {
  @override
  _TelaAberturaState createState() => _TelaAberturaState();
}

class _TelaAberturaState extends State<TelaAbertura> {
  ControleTelaAbertura _controle = ControleTelaAbertura();

  @override
  void initState() {
    // TODO: implement initState
    super.initState();
    _controle.inicializarAplicacao(context);
  }
}
```

- Na primeira parte do código temos as importações e a declaração de um objeto de controle (**_controle**).
- A ideia é que cada tela dos apps que fizemos terá seu respectivo controlador.
- Temos também a chamada ao método **inicializarAplicacao**. Esse método dará o comportamento da tela de abertura, ou seja, enquanto o **build** (próximo slide) desenha a tela, o **_controle** faz o trabalho de avançar para outras telas através do método **inicializarAplicacao**.



build

- No método build temos o uso de uma Stack. Aqui usamos o **StackFit.expand** (para fazer a Stack ocupar todo espaço disponível) e **BoxFit.contain** (vai ocupar o maior espaço possível sem deformar a imagem).
- Na sequência temos um **Text** com um **TextStyle** simples, mas que precisou ser feito (o padrão ficava não muito agradável).
- Notar também o uso de um **CircularProgressIndicator** no centro da tela e o uso de **padding** e **alignment** para posicionar o **Text** na parte inferior da tela.

```
@override
Widget build(BuildContext context) {
  return Container(
    color: Colors.green[200],
    alignment: Alignment.center,
    child: Stack(
      fit: StackFit.expand,
      children: <Widget>[
        Image.asset("assets/icon/icone_aplicacao.png", fit: BoxFit.contain),
        Container(
          alignment: Alignment.bottomCenter,
          padding: EdgeInsets.only(bottom: 100),
          child: Text(
            "NaLista",
            style: TextStyle(
              fontSize: 20,
              color: Colors.white,
              // Sem linha abaixo do texto
              decoration: TextDecoration.none,
            ), // TextStyle
          ), // Text
        ), // Container
        Center(child: CircularProgressIndicator()),
      ], // <Widget>[]
    ), // Stack
  ); // Container
}
```



```

import 'package:flutter/cupertino.dart';
import 'package:nalista/ dominio/usuario.dart';
import 'package:nalista/telas/tela_login.dart';
import 'package:nalista/telas/tela_principal.dart';
import 'package:nalista/ util/nav.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

class ControleTelaAbertura{
  void inicializarAplicacao(BuildContext context) {
    // Dando um tempo para exibição da tela de abertura
    Future futureA = Future.delayed(Duration(seconds: 3));

    // Obtendo o Usuário (caso já esteja logado)
    Future<FirebaseUser> futureB = FirebaseAuth.instance.currentUser();

    // Agurandando as 2 operações terminarem
    // Quando terminarem a aplicação ou vai para a tela de login
    // ou para a tela principal
    Future.wait([futureA, futureB]).then((List values) {
      FirebaseUser firebaseUser = values[1];
      if(firebaseUser == null)
        push(context, TelaLogin(), replace: true);
      else{
        Usuario usuario;
        Firestore.instance.collection('usuarios').
          where("email", isEqualTo: "${firebaseUser.email}").snapshots().
          listen((data) {
            usuario = Usuario.fromMap(data.documents[0].data);
            usuario.id = data.documents[0].documentID;
            push(context, TelaPrincipal(usuario), replace: true);
          });
      }
    });
  }
}

```

ControleTelaAbertura

- Criamos 2 objetos do tipo Future (**futureA** e **futureB**).
- O método **delayed** nos possibilitará aguardar 3 segundos para ter seu retorno. Na prática, se algo precisa do resultado desse Future deverá aguardar 3 segundos.
- Já o **futureB** recebe o usuário caso esteja logado.
- O método **wait** possibilita aguardar que ambos Future terminem, ou seja, **then** só será chamado quando a **futureA** e **futureB** terminarem.
- Em **values** teremos os retornos de cada um dos Futures quando o método **wait** terminar.
- Na posição 1 teremos o usuário logado (caso haja um). Caso não haja, esse valor será **null** e a **TelaLogin** será acionada. Caso haja um usuário logado, seus dados serão obtidos do **Firestore** e a **TelaPrincipal** será chamada (estudaremos o Firestore em outra aula).



Flutter 2.0

- Para deixar a tela de abertura visível mantivemos o `delayed`.
- Aqui temos funcionalidades do próprio Firebase para tratar a busca pelo usuário (por debaixo dos panos está sendo feito o uso de Streams ao invés dos Futures feitos no slide anterior).
- Ocorreram também algumas mudanças cosméticas (`documents[0]` virou `docs[0]`, por exemplo).
- A lógica em si se mantém praticamente a mesma.

```
class ControleTelaAbertura {  
  void inicializarAplicacao(BuildContext context) {  
    // Dando um tempo para exibição da tela de abertura  
    Future future = Future.delayed(Duration(seconds: 2));  
  
    future.then((value) => {  
      // Obtendo o Usuário (caso já esteja logado)  
      FirebaseAuth.instance.authStateChanges().listen((User? user) {  
        if (user == null) {  
          push(context, TelaLogin(), replace: true);  
        } else {  
          Usuario usuario;  
          FirebaseFirestore.instance  
            .collection('usuarios')  
            .where("email", isEqualTo: "${user.email}")  
            .snapshots()  
            .listen((data) {  
  
            usuario = Usuario.fromMap(data.docs[0].data());  
            usuario.id = data.docs[0].id;  
            push(context, TelaPrincipal(usuario), replace: true);  
          });  
        }  
      });  
    });  
  }  
}
```



Navegação entre telas – nav.dart

```
import 'package:flutter/material.dart';

Future push(BuildContext context, Widget page, {bool replace = false}) {
  if (replace){
    return Navigator.pushReplacement(context, MaterialPageRoute(builder: (BuildContext context) {
      return page;
    }));
  }
  return Navigator.push(context, MaterialPageRoute(builder: (BuildContext context) {
    return page;
  }));
}

void pop(BuildContext context, {String mensagem = null}){
  if (mensagem == null)
    Navigator.of(context).pop();
  else
    Navigator.pop(context, mensagem);
}
```

```
push(context, TelaPrincipal(usuario), replace: true);
```

```
push(context, TelaLogin(), replace: true);
```

push empilha telas
enquanto **pop** desempilha.

O uso do **replace: true** é
para sobrescrever a tela
anterior, ou seja,
não existirá a seta de voltar
na tela destino.

Por padrão, mantém-se a
seta de voltar, ou seja,
replace = false.



Obtendo o usuário logado

- É importante notar que o usuário do FirebaseAuth e do Firestore não são a mesma coisa.
- Basicamente utilizamos o e-mail cadastrado em FirebaseAuth como chave para buscar os dados do Usuário em Firestore.

// Obtendo o Usuário (caso já esteja logado)

```
Future<FirebaseUser> futureB = FirebaseAuth.instance.currentUser();
```

...

```
FirebaseUser firebaseUser = values[1];
```

...

```
Usuario usuario;
```

```
Firestore.instance.collection('usuarios').
```

```
where("email", isEqualTo: "${firebaseUser.email}").snapshots().
```

```
listen((data) {
```

```
    usuario = Usuario.fromMap(data.documents[0].data);
```

```
    usuario.id = data.documents[0].documentID;
```

```
    push(context, TelaPrincipal(usuario), replace: true);
```

```
});
```

Obtendo o usuário em FirebaseAuth

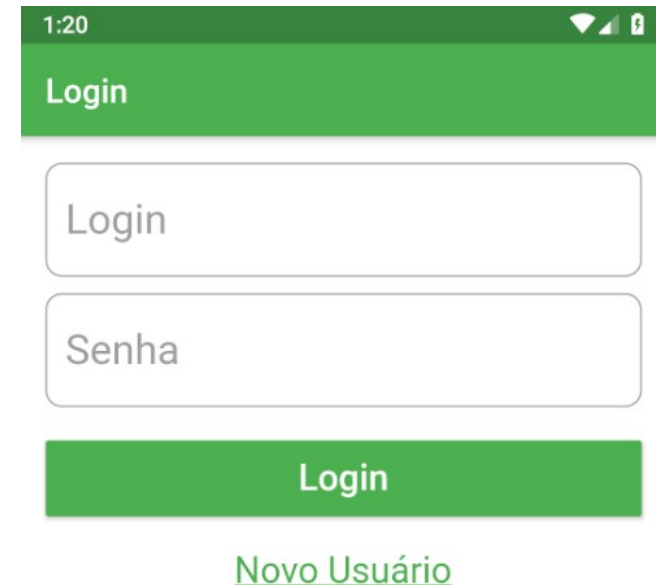
Obtendo um objeto de Usuario em Firestore usando o e-mail do firebaseUser.



TelaLogin

- Na tela de login utilizaremos um widget chamado **InkWell**. Esse widget dá o efeito de link a um **Text**, além de ser possível atribuir comportamento a esse clique através do parâmetro **onTap()**.

```
Container(  
  height: 46,  
  margin: EdgeInsets.only(top: 20),  
  child: InkWell(  
    onTap: (){  
      _controle.cadastrar(context);  
    },  
    child: Text(  
      "Novo Usuário",  
      textAlign: TextAlign.center,  
      style: TextStyle(  
        fontSize: 22,  
        color: Colors.green,  
        decoration: TextDecoration.underline,  
      ), // TextStyle  
    ), // Text  
  ), // InkWell  
) // Container
```



TelaLogin

- Na tela de login também faremos uso de um objeto controlador de interação (**_controle**).
- Nesse controlador colocaremos a **key** do formulário e todos os controladores de campos de edição.
- Os comportamentos acionados por cliques também chamarão funcionalidades do controlador.

```
class _TelaLoginState extends State<TelaLogin> {  
  ControleTelaLogin _controle;  
  
  @override  
  void initState() {  
    // TODO: implement initState  
    super.initState();  
    _controle = ControleTelaLogin();  
  }  
}
```

```
_body() {  
  return Form(  
    key: _controle.formkey,  
    child: Container(  
      // Margem padrão no Material Design  
      padding: EdgeInsets.all(16),  
      child: ListView(  
        children: <Widget>[  
          CampoEdicao(  
            "Login",  
            texto_dica: "Digite o Login",  
            controlador: _controle.controlador_login,  
            teclado: TextInputType.emailAddress,  
            receptor_foco: _controle.foco_senha,  
          ), // CampoEdicao
```

```
        child: InkWell(  
          onTap: () {  
            _controle.cadastrar(context);  
          },
```

```
          Botao(  
            texto: "Login",  
            cor: Colors.green,  
            ao_clicar: () {  
              _controle.logar(context);  
            },  
            marcador_foco: _controle.foco_botao,  
          ), // Botao
```

ControleTelaLogin

```
class ControleTelaLogin {  
  // Controles de edição do login e senha  
  final controlador_login = TextEditingController();  
  final controlador_senha = TextEditingController();  
  
  // Controlador de formulário (para fazer validações)  
  final formkey = GlobalKey<FormState>();  
  
  // Controladores de foco  
  final focus_senha = FocusNode();  
  final focus_botao = FocusNode();  
  
  // Autenticação  
  final FirebaseAuth _auth = FirebaseAuth.instance;  
  
  CollectionReference get _collection_usuarios => Firestore.instance.collection('usuarios');
```

- Aqui temos a definição dos controladores de edição, do formkey e dos controladores de foco.
- Temos também a obtenção do objeto de autenticação do FirebaseAuth.
- Por fim, temos também a obtenção da Collection de usuários. Uma Collection está para o Firestore como uma tabela está para um banco relacional.



Flutter 2.0

- Aqui podemos ver que o tipo de **_collection_usuarios** mudou:


```
CollectionReference ==> CollectionReference<Map<String, dynamic>>
```

```
class ControleTelaLogin {  
  // Controles de edição do login e senha  
  final controlador_login = TextEditingController();  
  final controlador_senha = TextEditingController();  
  
  // Controlador de formulário (para fazer validações)  
  final formkey = GlobalKey<FormState>();  
  
  // Controladores de foco  
  final focus_senha = FocusNode();  
  final focus_botao = FocusNode();  
  
  // Autenticação  
  final FirebaseAuth _auth = FirebaseAuth.instance;  
  
  CollectionReference<Map<String, dynamic>> get _collection_usuarios => FirebaseFirestore.instance.collection('usuarios');
```



ControleTelaLogin

```
void logar(BuildContext context) async{  
  if (formkey.currentState.validate()){  
    String login = controlador_login.text.trim();  
    String senha = controlador_senha.text.trim();  
  
    // Logando  
    try{  
      AuthResult result = (await _auth.signInWithEmailAndPassword(email: login, password: senha));  
      final FirebaseUser user = result.user;  
      print("login ${user.email} ");  
      _irParaTelaPrincipal(user, context);  
    } catch (error){  
      MensagemAlerta("Erro: Login inválido ou senha incorreta");  
    }  
  }  
}
```

- Aqui temos o método **logar**. Esse método chama a validação do formulário e se tudo estiver **ok** busca os valores dos campos de **login** e **senha**.
- Na sequência temos a chamada ao método do **_auth** para verificar o e-mail e senha na base do Firebase (na web). 
- Após obtido o FirebaseUser (user) o método **_irTelaPrincipal** é chamado para acionarmos a tela principal do app.
- Caso não ocorra a validação uma mensagem de alerta é enviada para o usuário do app.



Flutter 2.0

```
void login(BuildContext context) async{
  if (formkey.currentState!.validate()){
    String login = controlador_login.text.trim();
    String senha = controlador_senha.text.trim();

    try {
      // Logando
      UserCredential userCredential = await _auth.signInWithEmailAndPassword(
        email: login,
        password: senha
      );
      _irParaTelaPrincipal(userCredential.user, context);
    } on FirebaseAuthException catch (e) {
      if (e.code == 'user-not-found') {
        MensagemAlerta("Erro: Usuário não encontrado para o email informado");
      } else if (e.code == 'wrong-password') {
        MensagemAlerta("Erro: Password inválido!!!");
        print('Wrong password provided for that user.');
```

- Aqui ao invés de **AuthResult** temos **UserCredential**.
- Além disso aproveitamos e acrescentamos itens ao tratamento de exceção para ter mensagens mais completas.



ControleTelaLogin

```
void _irParaTelaPrincipal(FirebaseUser user, BuildContext context) {  
  // Buscando o usuário no serviço de armazenamento e chamando a tela Principal  
  _collection_usuarios.  
    where("email", isEqualTo: "${user.email}").snapshots().  
    listen((data) {  
      Usuario usuario = Usuario.fromMap(data.documents[0].data);  
      usuario.id = data.documents[0].documentID;  
      push(context, TelaPrincipal(usuario), replace: true);  
    });  
}
```

- Nesse método temos a busca do usuário na Collection de usuários através de seu e-mail.
- Uma vez encontrado o documento de usuário (documento em uma Collection é o equivalente a um registro/linha numa tabela) é criado a partir dele um objeto de Usuario através do construtor nomeado **fromMap**.
- Por fim acionamos a tela principal da aplicação passando o usuário obtido.



Flutter 2.0

```
void _irParaTelaPrincipal(FirebaseUser user, BuildContext context) {  
  // Buscando o usuário no serviço de armazenamento e chamando a tela Principal  
  _collection_usuarios.  
    where("email", isEqualTo: "${user.email}").snapshots().  
    listen((data) {  
      Usuario usuario = Usuario.fromMap(data.documents[0].data);  
      usuario.id = data.documents[0].documentID;  
      push(context, TelaPrincipal(usuario), replace: true);  
    });  
}
```

```
void _irParaTelaPrincipal(User? user, BuildContext context) {  
  // Buscando o usuário no serviço de armazenamento e chamando a tela Principal  
  _collection_usuarios.  
    where("email", isEqualTo: "${user!.email}").snapshots().  
    listen((data) {  
      Usuario usuario = Usuario.fromMap(data.docs[0].data());  
      usuario.id = data.docs[0].id;  
      push(context, TelaPrincipal(usuario), replace: true);  
    });  
}
```



ControleTelaLogin

```
void cadastrar(BuildContext context) async{
  if (formkey.currentState.validate()){
    String login = controlador_login.text.trim();
    String senha = controlador_senha.text.trim();

    // Criando o usuário
    try{
      // No serviço de autenticação
      AuthResult result = (await _auth.createUserWithEmailAndPassword(email: login, password: senha));
      final FirebaseUser user = result.user;
      print("login ${user.email} ");
      // No serviço de armazenamento
      DocumentReference docRef = _collection_usuarios.document();
      Future<void> future = docRef.setData({'email': user.email});
      future.then( (value){
        _irParaTelaPrincipal(user, context);
      });
    } catch (error){
      MensagemAlerta("Erro: Não foi possível criar o usuário");
    }
  }
}
```

value terá void, que é o retorno de **setData** feito através do **future**.

```
Future<void> setData(Map<String, dynamic> data, {bool merge = false}) {
```

- Aqui temos a tentativa de criação de um novo usuário.
- Primeiramente ocorre a validação dos dados do formulário.
- Na sequência é chamado o método para tentar criar o usuário (existem regras como por exemplo o número de caracteres da senha, se isso não for obedecido esse método dispara uma exceção).
- Criado esse novo usuário em FirebaseAuth é necessário criá-lo em Firestore. Isso é feito através de **collection_usuario.document()**.
- Em seguida colocamos o e-mail no usuário criado em Firestore (esse usuário terá apenas um id que é gerado pelo Firestore e o e-mail). Quando essa atualização terminar o método **_irParaTelaPrincipal** será acionado.



Flutter 2.0

```
void cadastrar(BuildContext context) async{
  if (formkey.currentState!.validate()){
    String login = controlador_login.text.trim();
    String senha = controlador_senha.text.trim();

    try {
      UserCredential userCredential = await FirebaseAuth.instance.createUserWithEmailAndPassword(
        email: login,
        password: senha
      );

      // No serviço de armazenamento
      _collection_usuarios.add({
        'email': login,
      }).then((value) => _irParaTelaPrincipal(userCredential.user, context))
        .catchError((error) => print("Falha ao adicionar o usuário: $error"));

    } on FirebaseAuthException catch (e) {
      if (e.code == 'weak-password') {
        MensagemAlerta("Erro: A senha fornecida é muito fraca");
      } else if (e.code == 'email-already-in-use') {
        MensagemAlerta("Erro: Já existe conta com o email informado");
      }
    } catch (e) {
      print(e);
    }
  }
}
```



Dúvidas?

