* import 'package:flutter/material.dart';: Importa o pacote Flutter Material, que fornece widgets básicos para a criação de interfaces de usuário.
* import 'package:botao/login\_page.dart';: Importa o arquivo login\_page.dart, que contém a tela de login.

**Função** main**:**

* void main() {: A função main é o ponto de entrada do aplicativo.
* runApp(MyApp());: Executa o widget MyApp.

**Classe** MyApp**:**

* class MyApp extends StatelessWidget {: Define a classe MyApp como um widget sem estado.
* @override: Indica que o método build está sobrescrevendo o método da classe base.
* Widget build(BuildContext context) {: O método build constrói o widget.

**Construindo o widget:**

* return MaterialApp(: Retorna um widget MaterialApp.
* home: Scaffold(: Define o widget home como um widget Scaffold.
* appBar: AppBar(: Define a barra de aplicativos.
* title: Text('Exemplo de Botão'),: Define o título da barra de aplicativos.
* body: Builder(: Usa um widget Builder para garantir que o contexto fornecido inclua um Navigator.
* builder: (context) => Center(: Constrói um widget Center.
* child: ElevatedButton(: Constrói um widget ElevatedButton.
* onPressed: () {: Define a ação a ser executada quando o botão é pressionado.
* Navigator.push(: Navega para a tela de login.
* context,: O contexto do widget atual.
* MaterialPageRoute(: Define a rota para a tela de login.
* builder: (context) => LoginPages(),: Constrói a tela de login.
* ),: Fecha o widget MaterialPageRoute.
* ),: Fecha o widget ElevatedButton.
* ),: Fecha o widget Center.
* ),: Fecha o widget Builder.
* ),: Fecha o widget Scaffold.
* ),: Fecha o widget MaterialApp.
* );: Fecha o método build.

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

- Esta linha importa o pacote material.dart do Flutter, que é necessário para construir interfaces de usuário no estilo Material Design.

**class LoginPages extends StatefulWidget {**

const LoginPages({Key? key}) : super(key: key);

- Aqui é definida a classe LoginPages, que é um StatefulWidget. Isso significa que pode ter seu estado alterado durante a execução do aplicativo. A classe tem um construtor nomeado `LoginPages`, que aceita um parâmetro key, usado para identificar este widget de forma única.

@override

State<LoginPages> createState() => \_LoginPagesState();

- Este método sobrescrito cria o estado para o widget LoginPages. Ele retorna uma instância de \_LoginPagesState, que é a classe de estado associada a LoginPages.

class \_LoginPagesState extends State<LoginPages> {

- Aqui é definida a classe `\_LoginPagesState`, que é o estado para o widget `LoginPages`. Esta classe é responsável por gerenciar o estado interno do widget e construir sua interface de usuário.

TextEditingController emailController = TextEditingController();

TextEditingController passwordController = TextEditingController();

- Duas instâncias de `TextEditingController` são criadas para controlar os campos de texto para o email e a senha. Estes controladores são usados para acessar e manipular o texto inserido nos campos de texto.

@override

Widget build(BuildContext context) {

- Este método `build` é chamado sempre que o estado do widget precisa ser reconstruído. Aqui é onde a interface de usuário do widget é construída.

return Material(

child: SizedBox(

width: double.infinity,

height: double.infinity,

- Um `Material` widget é usado como o widget raiz. O `Material` widget implementa o layout de material design. Um `SizedBox` é usado para definir um tamanho fixo para a tela de login, ocupando toda a largura e altura disponíveis.

child: Padding(

padding: const EdgeInsets.all(20.0),

- Um `Padding` widget é usado para adicionar espaço ao redor do conteúdo da tela de login.

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

- Um `Column` widget é usado para organizar os widgets em uma coluna vertical. `mainAxisAlignment` define como os widgets filhos devem ser alinhados verticalmente, neste caso, no centro da tela.

children: [

// Campo de entrada de e-mail

TextField(

controller: emailController,

- Um `TextField` widget é usado para permitir que os usuários insiram texto. O `controller` atribuído a ele é o `emailController`, que controla o texto inserido neste campo.

decoration: InputDecoration(

hintText: 'E-mail',

labelText: 'E-mail',

border: OutlineInputBorder(),

),

),

- `InputDecoration` é usado para adicionar uma aparência visual ao campo de entrada de texto, como um texto de sugestão (`hintText`) e uma etiqueta (`labelText`). Um `OutlineInputBorder` é usado para adicionar uma borda ao redor do campo de texto.

SizedBox(height: 20), // Espaçamento entre os campos

```

- Um `SizedBox` com uma altura de 20 pixels é usado para adicionar espaço entre os campos de texto.

// Campo de entrada de senha

TextField(

controller: passwordController,

obscureText: true, // Para ocultar a senha

decoration: InputDecoration(

hintText: 'Senha',

labelText: 'Senha',

border: OutlineInputBorder(),

),

),

- Outro `TextField` é usado para a entrada de senha. O `controller` atribuído a ele é o `passwordController`, e `obscureText` é definido como `true` para ocultar o texto da senha.

SizedBox(height: 20), // Espaçamento entre os campos

- Outro `SizedBox` é usado para adicionar espaço entre os campos de texto.

// Botão de login

ElevatedButton(

onPressed: () {

- Um `ElevatedButton` é usado para criar um botão de login. `onPressed` é o callback chamado quando o botão é pressionado.

// Valores fornecidos

final String usuarioFornecido = 'italo';

final String senhaFornecida = 'Teste@1234';

// Valores inseridos pelo usuário

final String usuarioInserido = emailController.text;

final String senhaInserida = passwordController.text;

- Aqui, as credenciais de login fornecidas pelo usuário e as credenciais esperadas são definidas.

// Verifica se os valores inseridos são iguais aos valores fornecidos

if (usuarioInserido == usuarioFornecido &&

senhaInserida == senhaFornecida) {

// Navega para a próxima tela se as credenciais estiverem corretas

Navigator.push(

context,

MaterialPageRoute(

builder: (context) => NextScreen(),

),

);

- Aqui é feita a verificação se as credenciais inseridas pelo usuário correspondem às credenciais esperadas. Se sim, a próxima tela (`NextScreen`) é exibida usando a função `Navigator.push`.

} else {

// Exibe uma mensagem de erro se as credenciais estiverem incorretas

showDialog(

context: context,

builder: (context) => AlertDialog(

title: Text('Erro de Autenticação'),

content: Text('Usuário ou senha incorretos.'),

actions: [

TextButton(

onPressed: () {

Navigator.of(context).pop(); // Fecha o diálogo

},

child: Text('OK'),

),

],

),

);

}

},

child: Text('Login'),

),

],

),

),

),

);

}

}

```

import 'package:flutter/material.dart';

import 'package:botao/login\_page.dart'; // Importe o arquivo login\_pages.dart

- Essas linhas importam os pacotes necessários do Flutter, incluindo o pacote `material.dart` para construir interfaces de usuário e o arquivo `login\_page.dart` que contém a definição da tela de login.

void main() {

runApp(MyApp());

}

```

- Esta função `main` é o ponto de entrada do aplicativo Flutter. Ela chama a função `runApp` para iniciar a execução do aplicativo, passando uma instância de `MyApp` como argumento.

class MyApp extends StatelessWidget {

- Esta é a definição da classe `MyApp`, que é um StatelessWidget. Isso significa que o estado deste widget é imutável, ou seja, não pode ser alterado uma vez que é construído.

@override

Widget build(BuildContext context) {

- Este é o método `build` da classe `MyApp`, que é chamado para construir a interface de usuário do widget.

return MaterialApp(

- Aqui, um `MaterialApp` é retornado. Este widget é usado para configurar o aplicativo Flutter com configurações específicas do Material Design, como temas, localizações e roteamento.

home: Scaffold(

- A propriedade `home` do `MaterialApp` é definida como um `Scaffold`, que fornece uma estrutura básica para a tela inicial do aplicativo.

appBar: AppBar(

title: Text('Exemplo de Botão'),

),

- Aqui, uma `AppBar` é definida como a barra de aplicativos da tela inicial, com um título "Exemplo de Botão".

body: Builder(

- A propriedade `body` do `Scaffold` é definida como um `Builder`. O `Builder` é usado para garantir que o contexto fornecido inclua um `Navigator`.

// Use Builder para garantir que o contexto fornecido inclua um Navigator

builder: (context) => Center(

child: ElevatedButton(

onPressed: () {

- Aqui, é criado um `Center` widget que centraliza seu filho horizontal e verticalmente. Dentro do `Center`, há um `ElevatedButton` que será usado para navegar para a tela de login quando pressionado.

// Navegar para a tela de login quando o botão é pressionado

Navigator.push(

context,

MaterialPageRoute(

builder: (context) => LoginPages(), // Use LoginPages aqui

),

);

},

child: Text('Ir para a tela de login'),

),

),

),

),

);

}

}

- Aqui é o final do método `build` e da classe `MyApp`. Fecha-se todas as chaves e parênteses.

* class LoginPages extends StatefulWidget {: Define a classe LoginPages como um widget com estado.
* const LoginPages({Key? key}) : super(key: key);: Construtor da classe LoginPages.
* @override: Indica que o método createState está sobrescrevendo o método da classe base.
* State<LoginPages> createState() => \_LoginPagesState();: Retorna um novo estado para a classe LoginPages.

**Classe** \_LoginPagesState**:**

* class \_LoginPagesState extends State<LoginPages> {: Define a classe \_LoginPagesState como o estado da classe LoginPages.
* TextEditingController emailController = TextEditingController();: Cria um controlador para o campo de entrada de e-mail.
* TextEditingController passwordController = TextEditingController();: Cria um controlador para o campo de entrada de senha.
* @override: Indica que o método build está sobrescrevendo o método da classe base.
* Widget build(BuildContext context) {: Constrói o widget da tela de login.

**Construindo a tela de login:**

* return Material(: Retorna um widget Material.
* child: SizedBox(: Define o filho do widget Material como um widget SizedBox.
* width: double.infinity,: Define a largura do widget SizedBox como infinito.
* height: double.infinity,: Define a altura do widget SizedBox como infinito.
* child: Padding(: Define o filho do widget SizedBox como um widget Padding.
* padding: const EdgeInsets.all(20.0),: Define o espaçamento do widget Padding como 20 em todas as direções.
* child: Column(: Define o filho do widget Padding como um widget Column.
* mainAxisAlignment: MainAxisAlignment.center,: Centraliza os widgets filhos do widget Column na vertical.
* children: [: Define os widgets filhos do widget Column.

**Campos de entrada:**

* // Campo de entrada de e-mail: Comentário que indica o início do campo de entrada de e-mail.
* TextField(: Constrói um widget TextField.
* controller: emailController,: Define o controlador do widget TextField como emailController.
* decoration: InputDecoration(: Define a decoração do widget TextField.
* hintText: 'E-mail',: Define a dica de texto do widget TextField.
* labelText: 'E-mail',: Define o rótulo do widget TextField.
* border: OutlineInputBorder(),: Define a borda do widget TextField como uma borda com contorno.
* ),: Fecha o widget TextField.
* SizedBox(height: 20),: Adiciona um espaço de 20 pixels entre os campos de entrada.
* // Campo de entrada de senha: Comentário que indica o início do campo de entrada de senha.
* TextField(: Constrói um widget TextField.
* controller: passwordController,: Define o controlador do widget TextField como passwordController.
* obscureText: true,: Oculta a senha digitada no campo.
* decoration: InputDecoration(: Define a decoração do widget TextField.
* hintText: 'Senha',: Define a dica de texto do widget TextField.
* labelText: 'Senha',: Define o rótulo do widget TextField.
* border: OutlineInputBorder(),: Define a borda do widget TextField como uma borda com contorno.
* ),: Fecha o widget TextField.

**Botão de login:**

* SizedBox(height: 20),: Adiciona um espaço de 20 pixels entre os campos de entrada e o botão.
* // Botão de login: Comentário que indica o início do botão de login.
* ElevatedButton(: Constrói um widget ElevatedButton.
* onPressed: () {: Define a ação a ser executada quando o botão é pressionado.

**Validação de login:**

* // Valores fornecidos: Comentário que indica o início da definição dos valores fornecidos.
* final String usuarioFornecido = 'italo';: Define a variável usuarioFornecido com o valor "italo".
* final String senhaFornecida = 'Teste@1234';: Define a variável senhaFornecida com o valor "Teste@1234".