
UNITAU
Universidade de Taubaté

# Internacionalização

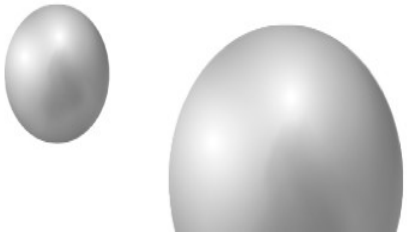
# Internacionalização

Existem várias alternativas para fazer a internacionalização das aplicações em Flutter.

Em geral, os plugins para internacionalização utilizam arquivos Json de algum formato padronizado para armazenar as traduções dos textos nas diferentes linguagens suportadas pela aplicação.

O plugin deve prover uma forma de gerenciar a seleção da localização e a obtenção dos textos na linguagem selecionada.

O MaterialApp tem propriedades para suporte a internacionalização.



# Internacionalização

Crie uma nova aplicação Flutter com o nome **internationalization**.

Altere o arquivo **pubspec.yaml** para incluir as dependências de **shared_preferences**, **devicelocale** e **flutter_localizations**:

```
dependencies:
  shared_preferences: ^0.5.6+3
  devicelocale: ^0.2.3
  flutter_localizations:
    sdk: flutter
  flutter:
    sdk: flutter
```

Acrescente a referência ao subdiretório onde serão armazenadas as traduções:

```
  assets:
    - assets/i18n/
```

Instale os pacotes adicionados nas dependências.

# Internacionalização

Crie o subdiretório **assets/i18n**.

Crie o arquivo **i18n_en.json** no subdiretório **assets/i18n**.

```
{
    "app_title": "Application Title",
    "main_title": "Hello World",
    "button_en": "English",
    "button_pt": "Portuguese",
    "button_tooltip": "Increment",
    "body_text": "You have pushed the button $1 times",
    "body_text_once": "You have pushed the button once"
}
```

# Internacionalização

Crie o arquivo i18n_pt_BR.json no subdiretório assets/i18n.

```
{
    "app_title": "Titulo da Aplicacao",
    "main_title": "Alo Mundo",
    "button_en": "Ingles",
    "button_pt": "Portugues",
    "button_tooltip": "Incremento",
    "body_text": "Voce pressionou o botao $1 vezes",
    "body_text_once": "Voce pressionou o botao uma vez"
}
```

# Internacionalização

## Altere a função main no arquivo main.dart:

```
import 'package:flutter/material.dart';
import
'package:flutter_localizations/flutter_localizations.dart';
import 'pages/home.dart';
import 'utils/globaltranslations.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  // Initializes the translation module
  await translations.init();
  // then start the application
  runApp( MyApp(),);
}

class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
}
```

# Internacionalização

```
class _MyAppState extends State<MyApp> {
  @override
  void initState(){
    super.initState();
    // Initializes a callback should something need
    // to be done when the language is changed
    translations.onLocaleChangedCallback = _onLocaleChanged;
  }

  // If there is anything special to do when the user
changes the language
  _onLocaleChanged() async {
    // do anything you need to do if the language changes
    print('Language has been changed to: $
{translations.currentLanguage}');
  }
```

# Internacionalização

```
  // Main initialization
  @override
  Widget build(BuildContext context){
    return MaterialApp(
      localizationsDelegates: [
        GlobalMaterialLocalizations.delegate,
        GlobalWidgetsLocalizations.delegate,
      ],
      // Tells the system which are the supported languages
      supportedLocales: translations.supportedLocales(),
      debugShowCheckedModeBanner: false,
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: Home(),
    );
  }
}
```

# Internacionalização

Crie o subdiretório **lib/pages**. Crie o arquivo **home.dart** no subdiretório **lib/pages**:

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

class Home extends StatefulWidget {
  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<Home> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }
```

# Internacionalização

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title:
Text(translations.text('main_title'))),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              translations.text(
                (_counter==1)?'body_text_once':'body_text',
                ['$_counter']
              ),
              style: TextStyle(fontSize: 20.0)
            ),
          ],
        ),
      ),
```

# Internacionalização

```
      bottomNavigationBar: ButtonBar(
        alignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          RaisedButton(
            child: Text(translations.text('button_en')),
            onPressed: () async {
              await translations.setNewLanguage('en',true);
              setState((){});
            },
          ),
          RaisedButton(
            child: Text(translations.text('button_pt')),
            onPressed: () async {
              await
translations.setNewLanguage('pt_BR',true);
              setState((){});
            },
          ),
        ],
      ),
```

# Internacionalização

```
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: translations.text('button_tooltip'),
        child: Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer
for build methods.
    );
  }
}
```

# Internacionalização

Crie o subdiretório **lib/utils**. Crie o arquivo **globaltranslations.dart** no subdiretório **lib/utils**:

```
import 'package:flutter/services.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:devicelocale/devicelocale.dart';
import 'dart:async';
import 'dart:convert';
import 'dart:ui';

//
// Preferences related
//
const String _storageKey = "MyApplication_";
const List<String> _supportedLanguages = ['en','pt_BR'];
Future<SharedPreferences> _prefs =
SharedPreferences.getInstance();
```

# Internacionalização

```
class GlobalTranslations {
  static final GlobalTranslations _translations =
GlobalTranslations._internal();
  factory GlobalTranslations() => _translations;
  GlobalTranslations._internal();

  Locale _locale;
  Map<dynamic, dynamic> _localizedValues;
  VoidCallback _onLocaleChangedCallback;

  // Returns the list of supported Locales
  Iterable<Locale> supportedLocales() =>
_supportedLanguages.map<Locale>((lang) => Locale(lang, ''));
```

# Internacionalização

```
  // Returns the translation that corresponds to the [key]
  String text(String key, [List<String> args]) {
    // Return the requested string
    String ret = (_localizedValues == null ||
_localizedValues[key] == null)?
      '** ${key} not found'
    : _localizedValues[key];
    if (args != null) {
      for( int i=0; i<args.length; i++ ) {
        String placeHolder = "\$${i+1}";
        ret = ret.replaceAll(placeHolder,args[i]);
      };
    }
    return ret;
  }

  // Returns the current language code
  get currentLanguage => _locale == null ? '' :
_locale.languageCode;

  // Returns the current Locale
  get locale => _locale;
```

# Internacionalização

```
// One-time initialization
Future<Null> init([String language]) async {
  String deviceLocale = await Devicelocale.currentLocale;
  print("Device locale: ${deviceLocale}");
  if (_locale == null){
    if (language == null) {
      language = await getPreferredLanguage();
      if (language == "")
        language = deviceLocale;
    }
    await setNewLanguage(language);
  }
  return null;
}
```

# Internacionalização

```
  // Restores the preferred language
  getPreferredLanguage() async {
    return _getApplicationSavedInformation('language');
  }

  // Save the preferred language
  setPreferredLanguage(String lang) async {
    return _setApplicationSavedInformation('language',
lang);
  }
```

# Internacionalização

```
  // Change the language
  Future<Null> setNewLanguage([String newLanguage, bool
saveInPrefs = false]) async {
    String language = newLanguage;
    if (language == null)
      language = await getPreferredLanguage();
    if (language == "")
      language = _supportedLanguages.first;
    if (!_supportedLanguages.contains(language))
      language = _supportedLanguages.first;
    _locale = Locale(language, "");
    // Load the language strings
    String jsonContent = await
rootBundle.loadString("assets/i18n/i18n_$
{_locale.languageCode}.json");
    _localizedValues = json.decode(jsonContent);
    if (saveInPrefs){await setPreferredLanguage(language);}
    // Notify that a language has changed
    if (_onLocaleChangedCallback != null){
      _onLocaleChangedCallback(); }
    return null;
  }
```

# Internacionalização

```
  // Callback to be invoked when the user changes the
language
  set onLocaleChangedCallback(VoidCallback callback){
    _onLocaleChangedCallback = callback;
  }

  // Fetch an application preference
  Future<String> _getApplicationSavedInformation(String
name) async {
    final SharedPreferences prefs = await _prefs;
    return prefs.getString(_storageKey + name) ?? '';
  }

  // Saves an application preference
  Future<bool> _setApplicationSavedInformation(String name,
String value) async {
    final SharedPreferences prefs = await _prefs;
    return prefs.setString(_storageKey + name, value);
  }
}

GlobalTranslations translations = GlobalTranslations();
```

# Internacionalização