MODUL PEMROGRAMAN MOBILE

CIM 430

MODUL PERTEMUAN 13

Flutter – Internationalization

DISUSUN OLEH TIM DOSEN :

7174 – SAWALI WAHYU, S.KOM, M.KOM

8126 – JEFRY SUNUPURWA ASRI, S.KOM, M.KOM

7176 – IKSAN RAMADHAN, S.KOM, M.KOM

UNIVERSITAS ESA UNGGUL

FAKULTAS ILMU KOMPUTER

TAHUN 2021

Flutter – Internationalization

A. Kemampuan Akhir Yang Diharapkan

Setelah mempelajari modul ini, diharapkan mahasiswa mampu :

1. Mahasiswa mampu memahami konsep CustomLocalizations Pada Aplikasi Mobile.
2. Mahasiswa mampu membuat model aplikasi untuk membuat CustomLocalizations
3. Mahasiswa mampu membuat aplikasi dengan kompleksitas Aplikasi Mobile yang dapat memiliki output yang relevan.

Flutter - Internationalization

Saat ini, aplikasi seluler digunakan oleh pelanggan dari berbagai negara dan akibatnya, aplikasi dituntut untuk menampilkan konten dalam berbagai bahasa. Mengaktifkan aplikasi untuk bekerja dalam berbagai bahasa disebut Internasionalisasi aplikasi.

Agar aplikasi dapat bekerja dalam bahasa yang berbeda, pertama-tama harus menemukan lokal saat ini dari sistem tempat aplikasi berjalan dan kemudian perlu menampilkan isinya di lokal tersebut, dan proses ini disebut Lokalisasi.

Flutter framework menyediakan tiga kelas dasar untuk pelokalan dan kelas utilitas ekstensif yang diturunkan dari kelas dasar untuk melokalkan aplikasi.

Kelas dasarnya adalah sebagai berikut -

- *Lokal* - Lokal adalah kelas yang digunakan untuk mengidentifikasi bahasa pengguna. Misalnya, en-us mengidentifikasi bahasa Inggris Amerika dan dapat dibuat sebagai.

```
Locale en_locale = Locale('en', 'US')
```

Di sini, argumen pertama adalah kode bahasa dan argumen kedua adalah kode negara. Contoh lain untuk membuat lokal *Spanyol (es-ar) Argentina* adalah sebagai berikut -

```
Locale es_locale = Locale('es', 'AR')
```

- *Pelokalan* - Pelokalan adalah widget umum yang digunakan untuk menyetel Lokal dan sumber daya turunannya yang dilokalkan.

```
class CustomLocalizations {
   CustomLocalizations(this.locale);
   final Locale locale;
   static CustomLocalizations of(BuildContext context) {
      return Localizations.of<CustomLocalizations>(context,
CustomLocalizations);
   }
   static Map<String, Map<String, String>> _resources = {
      'en': {
         'title': 'Demo',
         'message': 'Hello World'
      },
      'es': {
         'title': 'Manifestación',
         'message': 'Hola Mundo',
      },
   };
   String get title {
      return _resources[locale.languageCode]['title'];
   }
   String get message {
      return _resources[locale.languageCode]['message'];
   }
```

- Di sini, CustomLocalizations adalah kelas kustom baru yang dibuat khusus untuk mendapatkan konten lokal tertentu (judul dan pesan) untuk widget. metode menggunakan kelas Lokalisasi untuk mengembalikan kelas CustomLocalizations baru.
- LocalizationsDelegate <T> - LocalizationsDelegate <T> adalah kelas pabrik tempat widget Lokalisasi dimuat. Ini memiliki tiga metode over-ridable -
    - isSupported - Menerima lokal dan mengembalikan apakah lokal yang ditentukan didukung atau tidak.

```
@override
bool isSupported(Locale locale) =>
['en',
'es'].contains(locale.languageCode);
```

Di sini, delegasi hanya bekerja untuk en dan es locale.

- load - Menerima lokal dan mulai memuat sumber daya untuk lokal yang ditentukan.

```
@override
Future<CustomLocalizations> load(Locale locale)
   { return
SynchronousFuture<CustomLocalizations>(CustomLocalizations(local
e
));
```

```
}
```

Di sini, metode muat mengembalikan CustomLocalizations. CustomLocalizations yang dikembalikan bisa digunakan untuk mendapatkan nilai judul dan pesan dalam bahasa Inggris dan Spanyol

- shouldReload - Menentukan apakah reload CustomLocalizations diperlukan ketika widget Lokalisasinya dibangun kembali.

```
@override
bool shouldReload(CustomLocalizationsDelegate old) => false;
```

- Kode lengkap CustomLocalizationDelegate adalah sebagai berikut -

```
class CustomLocalizationsDelegate extends
LocalizationsDelegate<CustomLocalizations> {
   const CustomLocalizationsDelegate();
   @override
   bool isSupported(Locale locale) => ['en',
'es'].contains(locale.languageCode);
   @override
   Future<CustomLocalizations> load(Locale locale) {
      return
SynchronousFuture<CustomLocalizations>(CustomLocalizations(locale
));
   }
   @override bool shouldReload(CustomLocalizationsDelegate old)
=> false;
}
```

Secara umum, aplikasi Flutter didasarkan pada dua widget level root, MaterialApp atau WidgetsApp. Flutter menyediakan pelokalan siap pakai untuk kedua widget dan keduanya adalah MaterialLocalizations dan WidgetsLocaliations. Lebih lanjut, Flutter juga menyediakan delegasi untuk memuat MaterialLocalizations dan WidgetsLocaliations dan mereka masing-masing adalah GlobalMaterialLocalizations.delegate dan GlobalWidgetsLocalizations.delegate.

Mari kita buat aplikasi berkemampuan internasionalisasi sederhana untuk menguji dan memahami konsep tersebut.

- Buat aplikasi flutter baru, flutter_localization_app.
- Flutter mendukung internasionalisasi menggunakan paket flutter eksklusif, flutter_localizations. Idenya adalah untuk memisahkan konten yang dilokalkan dari SDK utama. Buka pubspec.yaml dan tambahkan kode di bawah ini untuk mengaktifkan paket internasionalisasi -

```
dependencies
   :
   flutter:
      sdk: flutter
   flutter_localizations
   :
      sdk: flutter
```

- Android studio akan menampilkan peringatan berikut bahwa pubspec.yaml telah diperbarui.

- Klik Dapatkan opsi ketergantungan. Android studio akan mendapatkan paket dari Internet dan mengkonfigurasinya dengan benar untuk aplikasi.
- Impor paket flutter_localizations di main.dart sebagai berikut -

```
import
'package:flutter_localizations/flutter_localizations.dart';
import 'package:flutter/foundation.dart' show
SynchronousFuture;
```

- Di sini, tujuan SynchronousFuture adalah memuat lokalisasi kustom secara sinkron.
- Buat pelokalan kustom dan delegasi yang sesuai seperti yang ditentukan di bawah ini -

```
class CustomLocalizations {
   CustomLocalizations(this.locale);
   final Locale locale;
   static CustomLocalizations of(BuildContext context) {
      return Localizations.of<CustomLocalizations>(context,
CustomLocalizations);
   }
   static Map<String, Map<String, String>> _resources = {
      'en': {
         'title': 'Demo',
         'message': 'Hello World'
      },
      'es': {
         'title': 'Manifestación',
```

```
            'message': 'Hola Mundo',
        },
    };
    String get title {
        return _resources[locale.languageCode]['title'];
    }
    String get message {
        return _resources[locale.languageCode]['message'];
    }
}
    class CustomLocalizationsDelegate
    extends
    LocalizationsDelegate<CustomLocali
    zations> {
        const CustomLocalizationsDelegate();

        @override
        bool isSupported(Locale locale)
    => ['en',
    'es'].contains(locale.languageCode
    );

        @override
        Future<CustomLocalizations> load(Locale
            locale) { return
    SynchronousFuture<CustomLocalizations>(CustomLocalizatio
    ns(locale
    ));
}
 @override bool shouldReload(CustomLocalizationsDelegate old)
=> false;
}
```

- Di sini, CustomLocalizations dibuat untuk mendukung pelokalan untuk judul dan pesan dalam aplikasi dan CustomLocalizationsDelegate digunakan untuk memuat CustomLocalizations.
- Tambahkan delegasi untuk MaterialApp, WidgetsApp dan CustomLocalization menggunakan properti MaterialApp, localizationsDelegates, dan supportLocales seperti yang ditentukan di bawah -

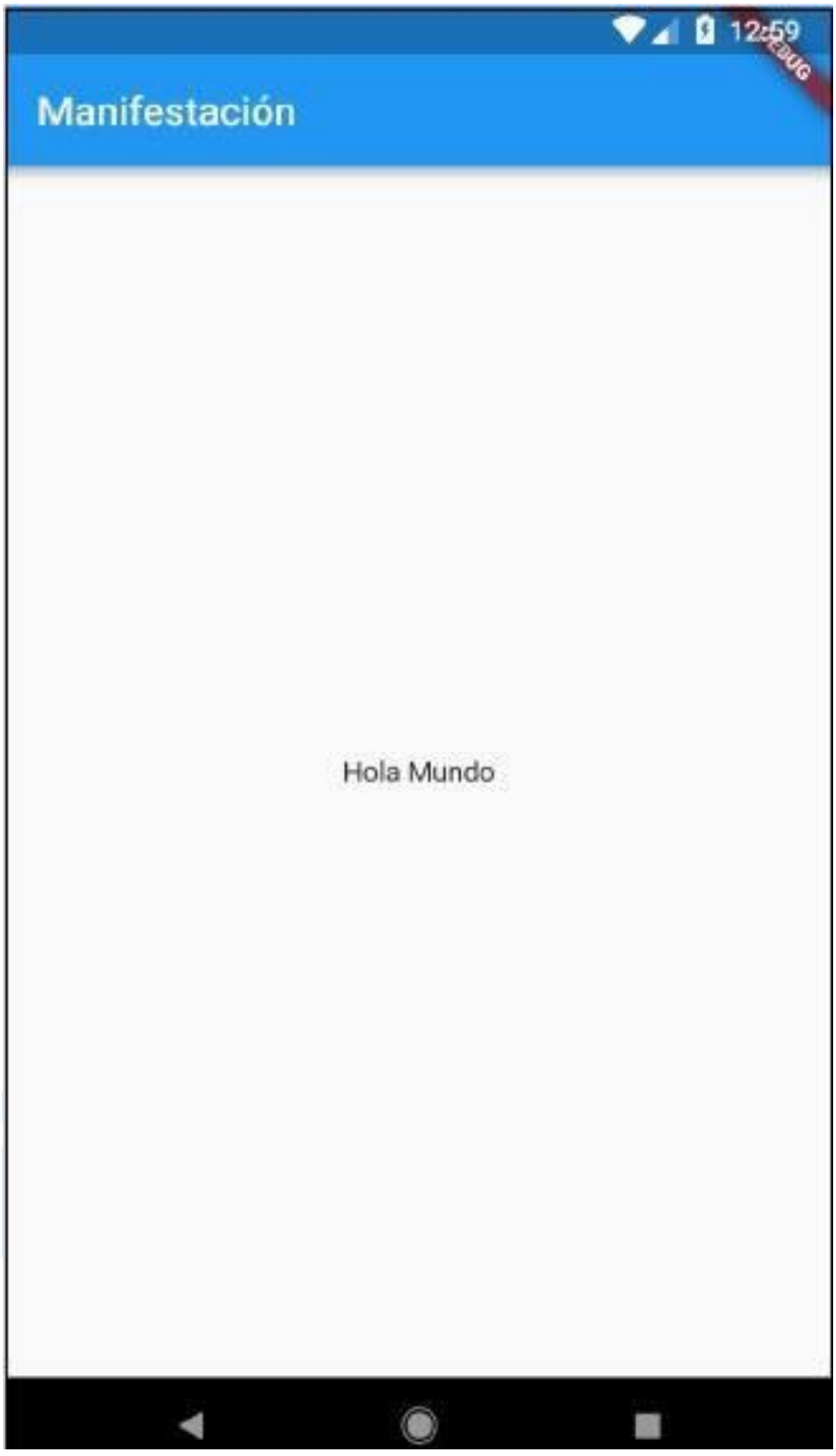
```
localizationsDelegates: [
   const CustomLocalizationsDelegate(),
   GlobalMaterialLocalizations.delegate,
   GlobalWidgetsLocalizations.delegate,
],
supportedLocales: [
   const Locale('en', ''),
   const Locale('es', ''),
],
```

- Gunakan metode CustomLocalizations, untuk mendapatkan nilai lokal dari judul dan pesan dan menggunakannya di tempat yang tepat seperti yang ditentukan di bawah ini -

```
class MyHomePage extends StatelessWidget {
   MyHomePage({Key key, this.title}) : super(key: key);
   final String title;
```

```
   @override
   Widget build(BuildContext context) {
      return Scaffold(
         appBar: AppBar(title: Text(CustomLocalizations
.of(context) .title), ),
         body: Center(
            child: Column(
               mainAxisAlignment: MainAxisAlignment.center,
               children: <Widget>[
                  Text( CustomLocalizations .of(context)
.message, ),
               ],
            ),
         ),
      );
   }
}
```

- Di sini, kami telah memodifikasi kelas MyHomePage dari StatefulWidget menjadi StatelessWidget untuk alasan kesederhanaan dan menggunakan CustomLocalizations untuk mendapatkan judul dan pesan.
- Kompilasi dan jalankan aplikasi. Aplikasi akan menampilkan isinya dalam bahasa Inggris.
- Tutup aplikasinya. Buka **Pengaturan → Sistem → Bahasa dan Input → Bahasa *** .
- Klik Tambahkan opsi bahasa dan pilih Spanyol. Ini akan menginstal bahasa Spanyol dan kemudian mencantumkannya sebagai salah satu opsi.
- Pilih bahasa Spanyol dan pindahkan ke atas bahasa Inggris. Ini akan ditetapkan sebagai bahasa Spanyol sebagai bahasa pertama dan semuanya akan diubah ke teks bahasa Spanyol.
- Sekarang luncurkan kembali aplikasi internasionalisasi dan Anda akan melihat judul dan pesan dalam bahasa Spanyol.
- Kami dapat mengembalikan bahasa ke bahasa Inggris dengan memindahkan opsi bahasa Inggris di atas opsi Spanyol di pengaturan.
- Hasil aplikasi (dalam bahasa Spanyol) ditunjukkan pada gambar di bawah -


12:59
DEBUG
Manifestación
Hola Mundo

Menggunakan Paket intl

Flutter menyediakan paket intl untuk lebih menyederhanakan pengembangan aplikasi seluler yang dilokalkan. Paket intl menyediakan metode dan alat khusus untuk menghasilkan pesan khusus bahasa semi-otomatis.

Mari kita buat aplikasi lokal baru dengan menggunakan paket intl dan memahami konsepnya.

- Buat aplikasi flutter baru, flutter_intl_app.

Buka pubspec.yaml dan tambahkan detail paket.

```
dependencies:
   flutter:
      sdk: flutter
   flutter_localizations:
      sdk: flutter
   intl: ^0.15.7
   intl_translation: ^0.17.3
```

- Android studio akan menampilkan peringatan seperti yang ditunjukkan di bawah ini yang menginformasikan bahwa pubspec.yaml diperbarui.

- Klik Dapatkan opsi ketergantungan. Android studio akan mendapatkan paket dari Internet dan mengkonfigurasinya dengan benar untuk aplikasi.
- Salin main.dart dari contoh sebelumnya, flutter_internationalization_app.
- Impor pacakge intl seperti yang ditunjukkan di bawah ini -

```
import 'package:intl/intl.dart';
```

- Perbarui kelas CustomLocalization seperti yang ditunjukkan pada kode yang diberikan di bawah ini -

```
class CustomLocalizations {
   static Future<CustomLocalizations> load(Locale
      locale) { final String name =
      locale.countryCode.isEmpty ?
locale.languageCode : locale.toString();
      final String localeName = Intl.canonicalizedLocale(name);

      return initializeMessages(localeName).then((_)
         { Intl.defaultLocale = localeName;
         return CustomLocalizations();
      });
   }
   static CustomLocalizations of(BuildContext context) {
      return
      Localizations.of<CustomLocalizations>(context,
CustomLocalizations);
   }
   String get title {
      return
      Intl.message(
         'Demo',
         name: 'title',
         desc: 'Title for the Demo application',
      );
   }
   String get message{
      return
      Intl.message(
         'Hello World',
         name:
         'message',
         desc: 'Message for the Demo application',
      );
   }
```

```
}
class CustomLocalizationsDelegate extends
LocalizationsDelegate<CustomLocalizations> {
   const CustomLocalizationsDelegate();

   @override
   bool isSupported(Locale locale) => ['en',
'es'].contains(locale.languageCode);
   @override
   Future<CustomLocalizations> load(Locale locale) {
      return CustomLocalizations.load(locale);
   }
   @override
   bool shouldReload(CustomLocalizationsDelegate old) => false;
}
```

- Di sini, kami telah menggunakan tiga metode dari paket intl alih-alih metode kustom. Kalau tidak, konsepnya sama.
    - Intl.canonicalizedLocale - Digunakan untuk mendapatkan nama lokal yang benar.
    - Intl.defaultLocale - Digunakan untuk mengatur lokal saat ini
    - Intl.message - Digunakan untuk menetapkan pesan baru.
- impor file **l10n / messages_all.dart** . Kami akan segera membuat file ini

```
import 'l10n/messages_all.dart';
```

- Sekarang, buat folder, lib / l10n
- Buka prompt perintah dan buka direktori root aplikasi (di mana pubspec.yaml tersedia) dan jalankan perintah berikut -

```
flutter packages pub run intl_translation:extract_to_arb
-- output-
   dir=lib/l10n lib/main.dart
```

- Di sini, perintah akan menghasilkan file intl_message.arb, template untuk membuat pesan di lokasi yang berbeda. Isi file tersebut adalah sebagai berikut -

```
{
   "@@last_modified": "2019-04-19T02:04:09.627551",
   "title": "Demo",
   "@title": {
      "description": "Title for the Demo application",
      "type": "text",
      "placeholders": {}
   },
   "message": "Hello World",
   "@message": {
      "description": "Message for the Demo
      application",
      "type": "text",
      "placeholders": {}
```

```
    }
}
```

- Salin intl_message.arb dan buat file baru, intl_en.arb.
- Salin intl_message.arb dan buat file baru, intl_es.arb dan ubah konten ke bahasa Spanyol seperti yang ditunjukkan di bawah ini -

```
{
   "@@last_modified": "2019-04-19T02:04:09.627551",
   "title": "Manifestación",
   "@title": {
      "description": "Title for the Demo application",
      "type": "text",
      "placeholders": {}
   },
   "message": "Hola Mundo",
   "@message": {
      "description": "Message for the Demo application",
      "type": "text",
      "placeholders": {}
   }
}
```

- Sekarang, jalankan perintah berikut untuk membuat file pesan terakhir, messages_all.dart.

```
flutter packages pub run intl_translation:generate_from_arb
--output-dir=lib\l10n --no-use-deferred-loading
lib\main.dart lib\l10n\intl_en.arb
lib\l10n\intl_es.arb
```

- Kompilasi dan jalankan aplikasi. Ini akan bekerja mirip dengan aplikasi di atas, flutter_localization_app.