

TP6 –Une application avec différents écrans

Objectif du TP

L'objectif de ce TP est l'utilisation des formulaires ainsi que l'utilisation de différentes pages pour l'application.

Durée de réalisation de ce TP : 3h

Enoncé :

1. Créez un nouveau projet sous Android Studio (File→ New→New Flutter Project) que vous allez appeler **tp6**.
2. Supprimez ensuite le répertoire appelé test (on n'a pas besoin de ce répertoire).
3. Notre application consiste en une application de magazine comportant 4 pages (voir Figure 1) :

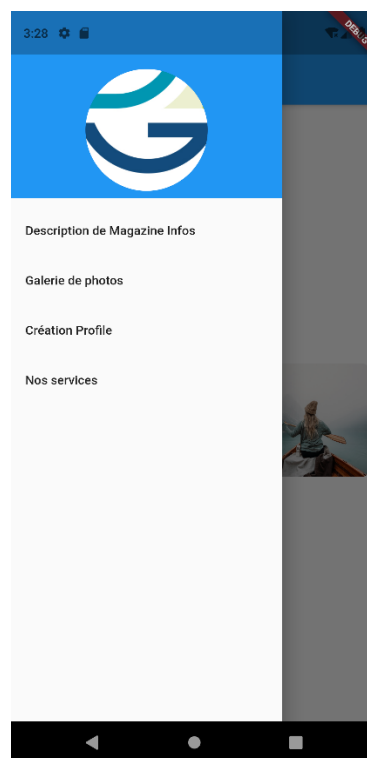


Figure 1. Aperçu général de l'application

- **La première page** : est la page d'accueil qui permet de présenter une description du magazine infos (voir Figure 2).

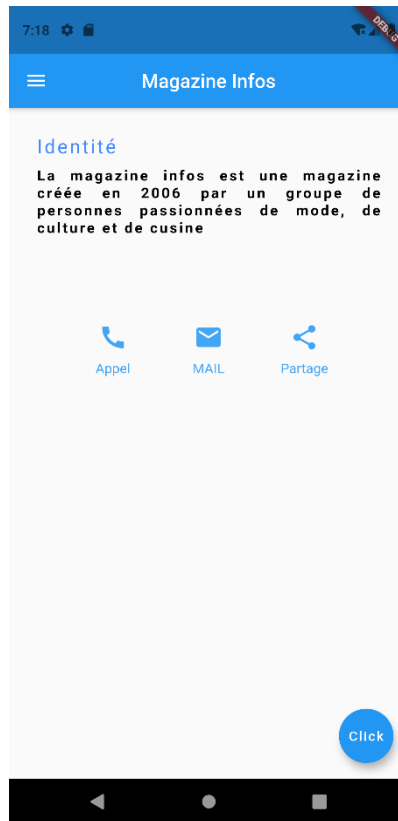


Figure 2. Page d'accueil de l'application de Magazine Infos

- **La deuxième page :** présente la galerie de photos enregistrées dans le magazine sous la forme d'une galerie (voir Figure 3).

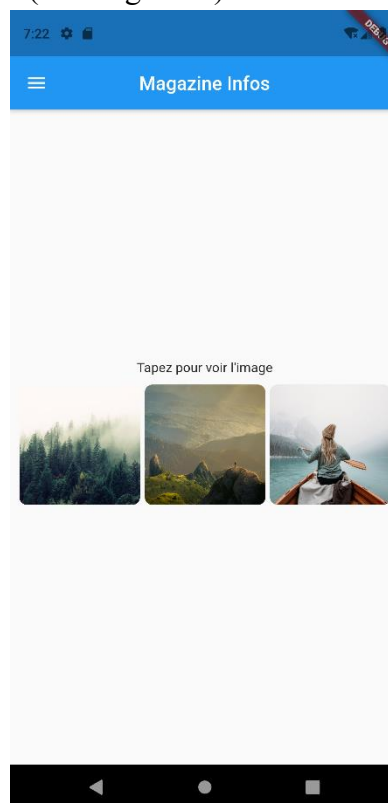


Figure 3. Galerie photos de Magazine Infos

- **La troisième page** : elle concerne le formulaire d'inscription au magazine Infos (voir Figure 4).

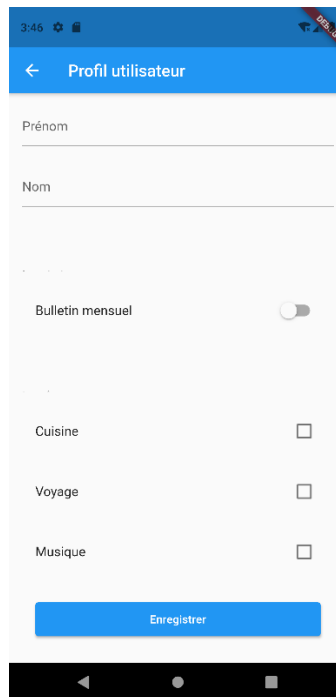


Figure 4. Formulaire d'inscription

- **La quatrième page** : elle concerne les services offerts par le magazine (voir Figure 5).

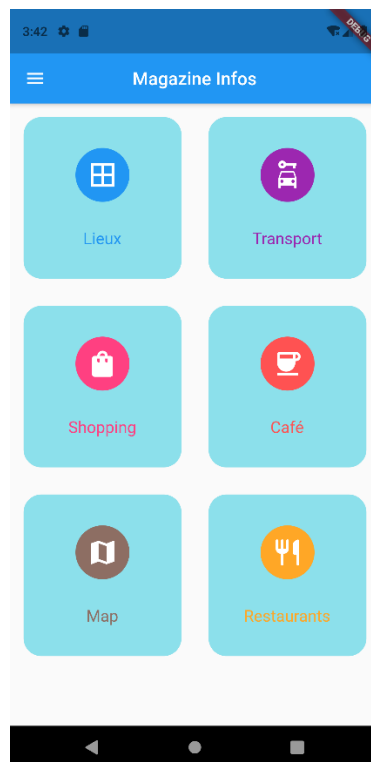


Figure 5. Services de Magazine Infos

4. Indications pour la classe Main :

La classe main est censée retourner un material App Widget dans lequel on précise les différentes pages et comment y accéder.

Remarques : la classe MaterialApp possède une propriété routes de type Map<String, WidgetBuilder>. C'est ici que nous allons définir les routes que nous allons utiliser dans l'application. En effet, dans flutter, les écrans sont appelés **routes**.

```
return MaterialApp(title: 'Magzine Infos', routes: {  
  "/": (context) => const MaPremierePage(),  
  "/deuxieme": (context) => const MaDeuxiemePage(),  
  "/troisieme": (context) => MatroisiemePage(),  
  "/quatrieme": (context) => MaquatriemePage(),  
})
```

5. Pour créer les différentes pages créez un répertoire sous le lib appelé pages. C'est sous ce répertoire que vous allez mettre toutes les classes des différentes pages.

6. Indications pour la page 1 :

- Créez un fichier dart appelé **page1** sous le répertoire pages.
- Dans la page1 créez la classe maPremierePage qui étend le comportement d'un **StatelessWidget**. Cette classe est censée renvoyer un scaffold contenant grâce à la propriété drawer un **Widget Drawer**. Ce dernier permet d'obtenir le menu de la Figure 1 et comporte un ensemble de Widgets comme illustré dans la Figure 6.
- Le Widget Drawer comporte un child qui est une **ListView** comportant à son tour la propriété children pour mettre les différents widgets (DrawerHeader, ListTile, etc.).

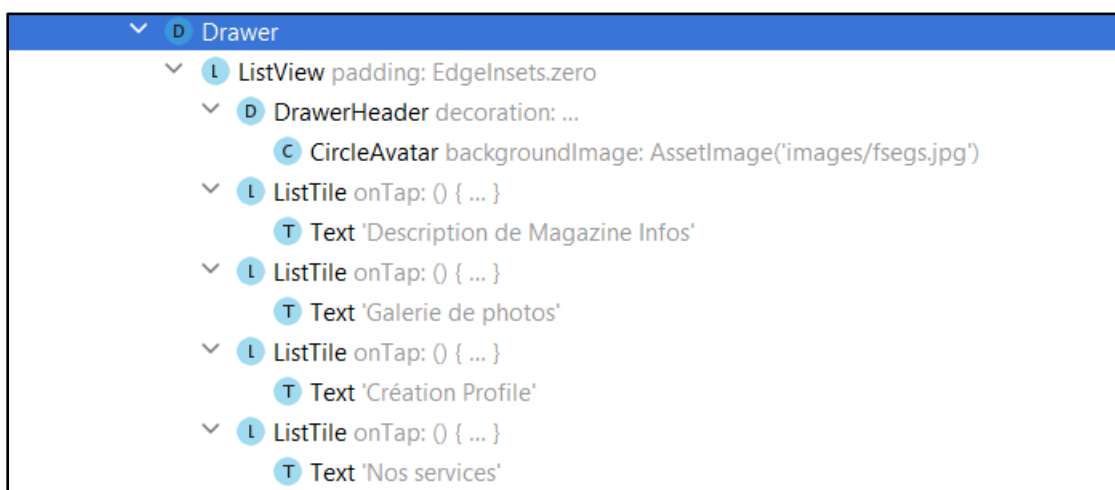


Figure 6. Arborescence de Widgets de maPremierePage

- Chaque élément du menu doit comporter un lien vers la page concernée.

Remarque : Dans flutter, il existe deux façons de naviguer vers un nouvel itinéraire (un nouvel écran) :

1. Utilisation de `Navigator.push ()`
 2. Utilisation de `Navigator.pushNamed ()`
- Si vous avez un itinéraire limité (comme un ou deux), vous pouvez utiliser la méthode `Navigator.push ()`.
 - Si vous avez plusieurs routes (le cas de notre application), il faut utiliser `Navigator.pushNamed ()`. Ainsi, nous devons suivre deux étapes :
 - Déclarer la **routes** propriété dans le `MaterialApp` constructeur (déjà fait dans la question 4)
 - Appeler la méthode `Navigator.pushNamed ()` si nécessaire.

Par exemple :

```
ListTile(
  title: const Text('Galerie de photos'),
  onTap: () {
    Navigator.pushNamed(context, '/deuxieme');
  },
),
```

- Outre le `Drawer`, **maPremierePage** comporte l'arborescence de Widgets suivante (voir Figure 7) pour réaliser le rendu de la Figure 1.



Figure 7. Suite de l'arborescence de la classe maPremierePage

- Il est à noter que la classe **_Buttons** permet de créer les différents boutons en se basant sur une classe personnalisée (**CustomButton**) qui attribue à chaque bouton un nom et une icône (voir Figure 8).

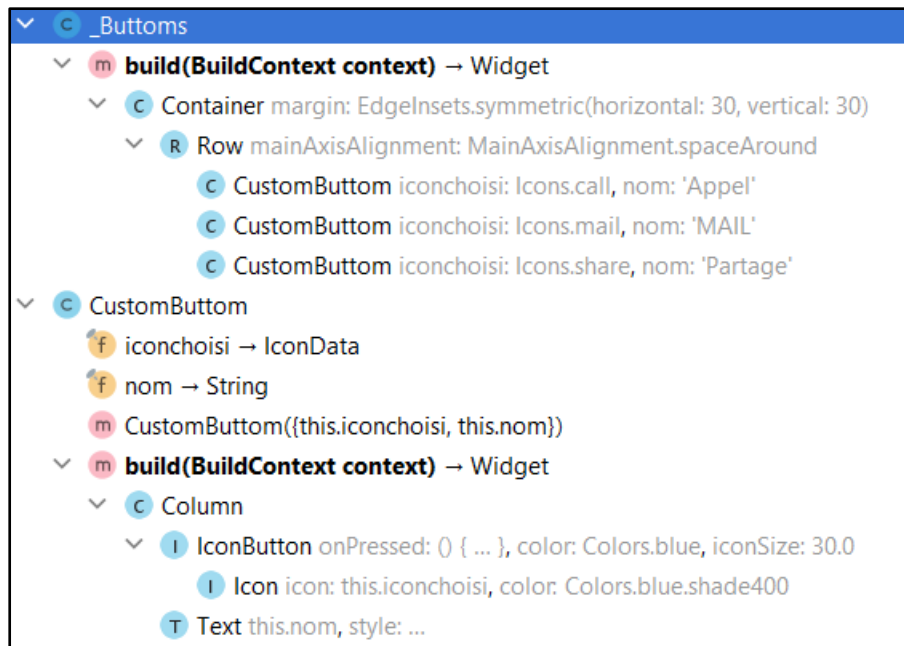


Figure 8. Arborescence des deux classes `_Buttons` et `CustomButton`

- Pour des raisons de simplification, nous n'allons pas programmer ces boutons.
- Le Floating Action Button nous ramène vers la deuxième page.

7. Indications pour la page 2 :

- Cette page comporte le Widget Drawer pour pouvoir naviguer tout comme la page 1.
- Pour réaliser cette page nous allons nous baser sur `galleryimage 1.1.0` un package (plugin) développé par la communauté flutter et mis en ligne sur cette adresse : <https://pub.dev/packages/galleryimage>. Ce plugin liste les images à partir d'URLs et vous permet d'afficher et de zoomer plusieurs images sur iOS et Android.
- En cliquant sur l'image on pourra l'agrandir (voir)

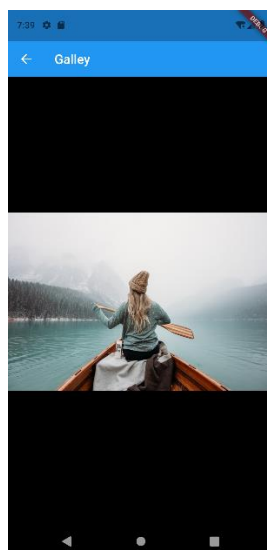


Figure 9. Un zoom sur une image

- Pour utiliser ce plugin vous devriez l'importer dans votre classe et aussi le déclarez dans le fichier de configuration pubspec.yaml sous la section *dependencies* :

```
dependencies:
  galleryimage: ^1.1.0
  flutter:
    sdk: flutter
```

- Référez-vous à la documentation de ce plugin : <https://pub.dev/packages/galleryimage> afin de récupérer le code. Pensez à changer la liste des URL en se basant sur le site unsplash.com

- Au final, cette page respectera l'arborescence suivante (voir Figure 10) :

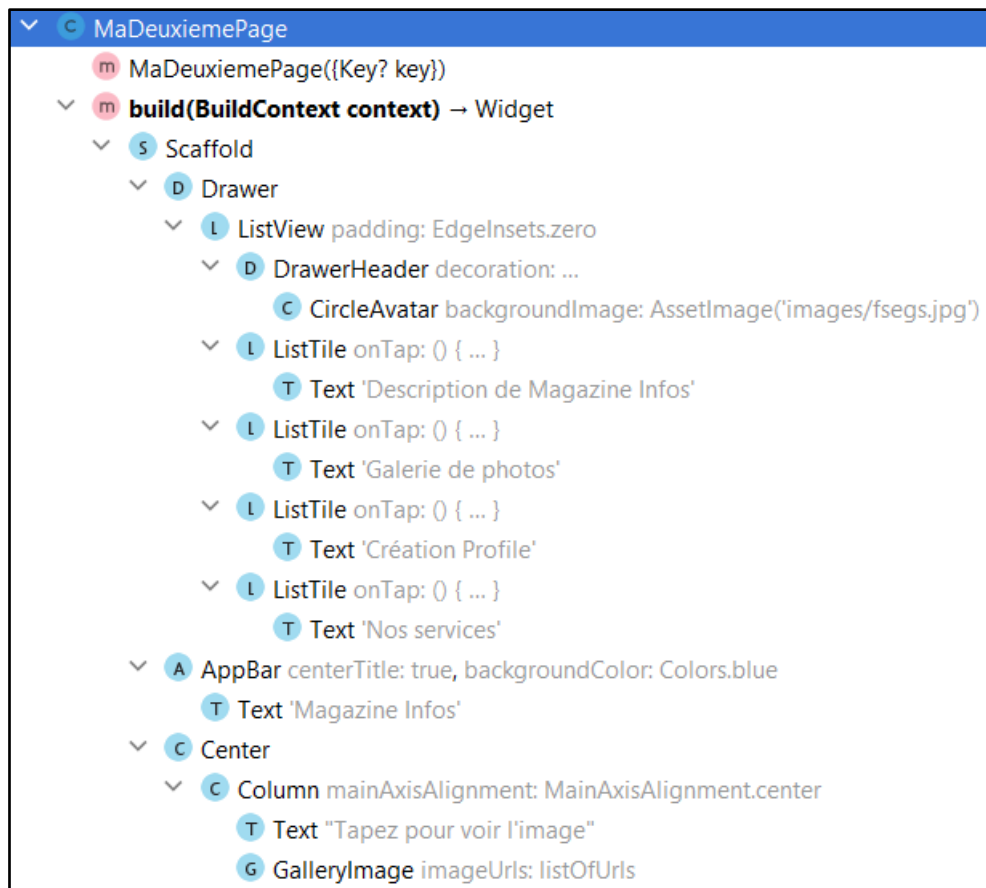


Figure 10. Arborescence de MaDeuxiemePage

8. Indications pour la page 3 :

- Cette classe étend le comportement d'un StatefulWidget et comporte un formulaire d'inscription.
- Flutter fournit un Widget **Form** pour créer un formulaire. Le Widget **Form** agit comme un conteneur, qui nous permet de regrouper et de valider les multiples champs du formulaire. Lorsque vous créez un formulaire, il est nécessaire de fournir la **GlobalKey**. Cette clé identifie de manière unique le formulaire et vous permet d'effectuer toute validation dans les champs du formulaire.
- Le Widget de formulaire utilise le widget enfant **TextFormField** pour permettre aux utilisateurs de saisir le champ de texte. Ce Widget rend un champ de texte de Material Design et nous permet également d'afficher les erreurs de validation lorsqu'elles se produisent.

- Un exemple d'utilisation du *validator* d'un TextFormField :

```
TextFormField(
  child: TextFormField(
    decoration:
      InputDecoration(hintText: 'Prénom'),
    validator: (value) {
      if (value!.isEmpty) {
        return 'Merci de saisir votre prénom';
      }
    },
  ),
```

- Afin d'enregistrer les valeurs des différents champs saisis par l'utilisateur nous allons :
 - Créez un répertoire *models* sous le *lib* du projet Flutter.
 - Créez sous ce répertoire une classe dart appelée *user*. Cette classe comportera les différents attributs comme le nom, le prénom, une map des passions et le bulletin.
 - Il faut rajouter aux différents champs les deux méthodes : *on saved* (pour enregistrer la valeur du champ en question), *onChanged* (pour capturer les changements des valeurs) et *setState()*.

A titre d'exemple :

```
child: CheckboxListTile(
  title: const Text('Cuisine'),
  value: _user.passions[User.PassionCuisine],
  onChanged: (value) {
    setState(() =>
      _user.passions[User.PassionCuisine] = value!);
  }),
```

- Le bouton **enregistrer** est censé :
 - Enregistrer le form, enregistrer l'utilisateur et afficher un *SnackBar* (voir Figure 11).

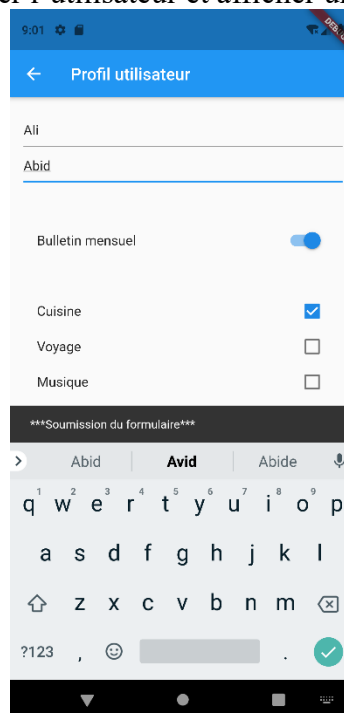


Figure 11. Rendu du *SnackBar*

9. Indications pour la page 4 :

- La classe implémentant la page 4 devrait s'appeler *MaquatriemePage*. Cette page comporte le *Drawer* pour naviguer entre les différentes pages. Elle inclut également un *TableRow* qui est un groupe horizontal de cellules dans un tableau. Chaque ligne du tableau doit avoir le même nombre d'enfants. Vous pouvez consulter la documentation :
- Outre le *Drawer*, cette page respecte l'arborescence suivante :

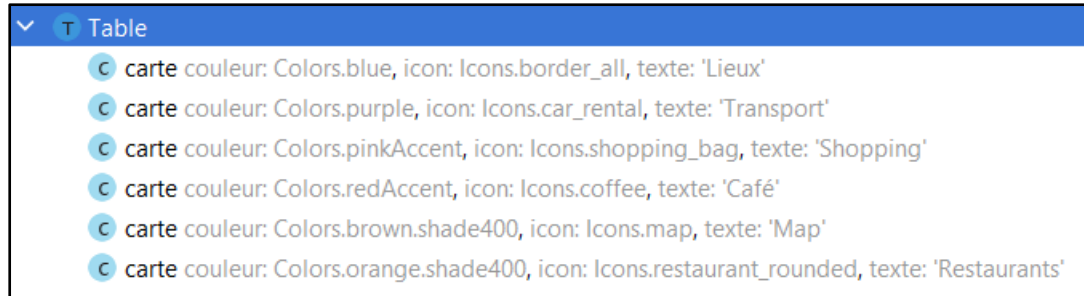


Figure 12. Arborescence de TableRow

- Cette classe inclut une classe *Carte* qui permet de dresser la couleur de l'icône, le type de l'icône et le texte associé. A son tour, la classe *Carte* utilise la classe *_Cartearriere*.

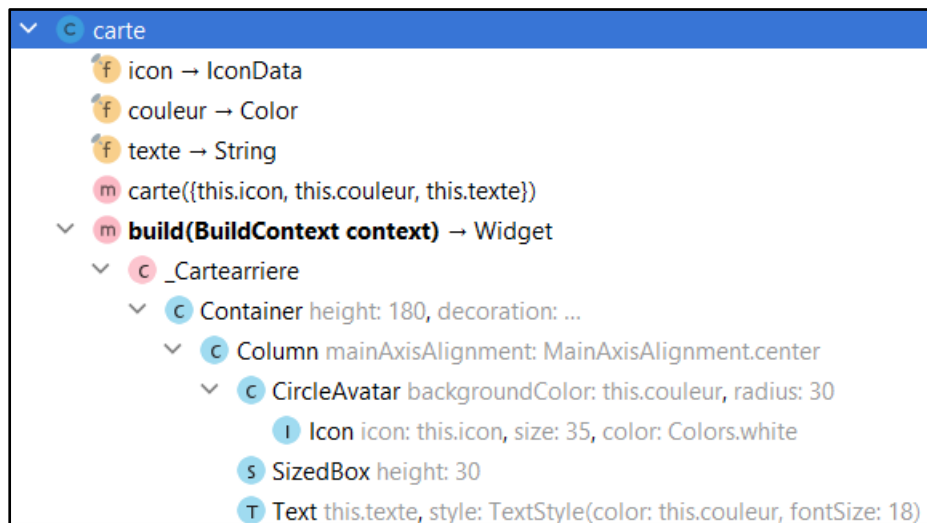


Figure 13. Arborescence de la classe carte

- La classe *_Cartearriere* permet de mettre un arrière-plan aux différentes cartes. Voici le code source commenté de cette classe :

```
class _Cartearriere extends StatelessWidget {
  final Widget child;
  const _Cartearriere({required this.child}); // elle s'applique
  sur le child courant

  @override
  Widget build(BuildContext context) => Container(
    margin: EdgeInsets.all(15),
    child: ClipRRect(// pour arrondir l'arrière plan du conteneur
```

```

borderRadius: BorderRadius.circular(20),
child: BackdropFilter(// pour appliquer un filtre
  filter: ImageFilter.blur(sigmaX: 5, sigmaY: 5),
  child: this.child, //le filtre est appliqué sur le child
en question
  ),
),
);
}

```