Flutter

Fiche 3 : Formulaires & État partagé

Table des matières

1 Objectifs de la fiche

ID	Objectifs
F04	Gestion d'un formulaire
F05	État partagé

2 Concepts

Pour commencer le tutoriel, créez un projet flutter nommé *tuto3* dans votre repository de cours.

2.1 Widgets avec un état partagé

Dans la fiche précédente, nous avons découvert les **Stateful Widgets** qui ont des attributs définissant un état intégré au sein du widget. Mais souvent, l'état doit être partagé par plusieurs widgets. Comment faire dans ce cas-ci, qui va gérer l'état ? Il existe plusieurs approches pour gérer l'état, veuillez lire l'article Managing state.

OBSERVATIONS & QUESTIONS

- Il est important de comprendre comment l'information circule dans un framework déclaratif. Comment faire descendre de l'info d'un widget parent vers un widget enfant ? Et l'inverse ?
- Nous avons vu que l'information circule facilement d'un widget parent vers un widget enfant. Il suffit de passer des paramètres aux constructeurs des enfants.
- Pour passer de l'information d'un enfant vers un parent, il faut notifier le parent. Pour ce faire, la façon la plus classique est d'appeler une callback (une fonction) lors d'un événement bien spécifique (un clic par exemple). Cette callback sera offerte par le parent qui exposera ainsi son état dans la callback.

2.2 D'un widget monolithique…

Pour donner un exemple de ce phénomène, nous allons créer une application permettant de changer son thème dynamiquement. Un interrupteur permet de passer d'un thème rouge à un thème vert et vice-versa. Un carré permet également de visualiser la couleur du thème de l'application facilement. Modifiez le contenu du fichier **main.dart** avec le code suivant.

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

const colors = {
  "red": Colors.red,
  "pink": Colors.pink,
  "purple": Colors.purple,
  "deep purple": Colors.deepPurple,
  "indigo": Colors.indigo,
  "blue": Colors.blue,
  "light blue": Colors.lightBlue,
  "cyan": Colors.cyan,
  "teal": Colors.teal,
  "green": Colors.green,
  "light green": Colors.lightGreen,
  "lime": Colors.lime,
  "yellow": Colors.yellow,
  "amber": Colors.amber,
  "orange": Colors.orange,
  "deep orange": Colors.deepOrange,
  "brown": Colors.brown,
  "blue grey": Colors.blueGrey,
  "grey": Colors.grey,
};

MaterialColor getColorValue(String color) => colors[color] ?? Colors.grey;

void main() => runApp(const MyApp());

class MyApp extends StatefulWidget {
```

```
  const MyApp({super.key});

  @override
  State<MyApp> createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  var color = "red";

  void setColor(String value) => setState(() => color = value);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        useMaterial3: true,
        colorScheme:
        ColorScheme.fromSeed(seedColor: getColorValue(color)),
      ),
      home: Scaffold(
        appBar: AppBar(
          title: const Text("Tutoriel 3"),
          backgroundColor: getColorValue(color).shade200,
        ),
        body: Padding(
          padding: const EdgeInsets.all(32.0),
          child: Column(
            children: [
              Expanded(
                child: Center(
                  child: Container(
                    width: 100,
                    height: 100,
                    color: getColorValue(color),
                  ),
                ),
              ),
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  const Text(
                    "red",
                    style: TextStyle(
                      color: Colors.red,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                  Switch(
                    value: color == "green",
                    activeColor: Colors.green,
                    inactiveThumbColor: Colors.red,
                    inactiveTrackColor: Colors.red.withOpacity(0.4),
                    onChanged: (value) => setColor(value ? "green" :
"red"),
                  ),
                  const Text(
                    "green",
                    style: TextStyle(
                      color: Colors.green,
                      fontWeight: FontWeight.bold,
```

```
                    ),
                  ),
                ],
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Analysez ce code pour vous assurer de bien le comprendre, et lancez l'application pour le tester.

[commit avec message : T03.1 Widget monolithique]

2.3 …à des widgets réutilisables

Nous souhaiterions maintenant définir des widgets réutilisables pour les différents composants, de sorte à pouvoir les réutiliser facilement dans d'autres contextes. Nous allons pour ça devoir utiliser les techniques de partage d'état abordées ci-dessus.

Copiez le code suivant dans un fichier **my_switch.dart**.

```
class MySwitch extends StatelessWidget {
  final String color;
  final void Function(String) setColor;

  const MySwitch({
    super.key,
    required this.color,
    required this.setColor,
  });

  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const Text(
          "red",
          style: TextStyle(color: Colors.red, fontWeight: FontWeight.bold),
        ),
        Switch(
          value: color == "green",
          activeColor: Colors.green,
          inactiveThumbColor: Colors.red,
          inactiveTrackColor: Colors.red.withOpacity(0.4),
          onChanged: (value) => setColor(value ? "green" : "red"),
        ),
        const Text(
          "green",
          style: TextStyle(color: Colors.green, fontWeight:
FontWeight.bold),
        ),
      ],
    );
  }
}
```

Ce widget affiche le composant dédié à l'interrupteur entre la couleur **red** et la couleur **green**. Il prend la couleur actuelle en argument, et une callback pour définir la nouvelle couleur suivant l'action de l'interrupteur.

Le type d'une fonction callback est écrit de la façon suivante : **TypeRetour Function(TypeArgument, TypeArgument, …)**. Alternativement, il est possible d'utiliser des types redéfinis comme VoidCallback pour une fonction qui ne prend pas d'argument et ne renvoie rien, ou ValueChanged<String> pour une fonction **void** qui prend une valeur en argument comme dans ce cas-ci.

Copiez ensuite le code suivant dans un fichier **my_square.dart**. Comme toujours, effectuez les imports nécessaires y compris dans ce cas la fonction **getColorValue** qui vient du fichier **main.dart**.

```
class MySquare extends StatelessWidget {
  final String color;

  const MySquare({super.key, required this.color});

  @override
  Widget build(BuildContext context) {
    return Expanded(
      child: Center(
        child: Container(
          width: 100,
          height: 100,
          color: getColorValue(color),
        ),
      ),
    );
  }
}
```

Ce widget affiche le composant dédié à la visualisation de la couleur du thème actuelle au sein d'un carré, en prenant cette couleur actuelle en argument.

Copiez ensuite le code suivant dans un fichier **home_screen.dart**.

```
class HomeScreen extends StatelessWidget {
  final String color;
  final void Function(String) setColor;

  const HomeScreen({super.key, required this.color, required
this.setColor});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Tutoriel 3"),
        backgroundColor: getColorValue(color).shade200,
      ),
      body: Padding(
        padding: const EdgeInsets.all(32.0),
        child: Column(
          children: [
            MySquare(color: color),
            MySwitch(color: color, setColor: setColor),
          ],
```

```
        ),
      ),
    );
  }
}
```

Nous pouvons y retrouver l'appel aux différents widgets que nous venons de créer. Mais ce n'est pas encore ce widget qui est responsable de l'application. Il prend également en argument la couleur et la callback, pour pouvoir les donner à ses enfants.

Il ne reste plus qu'à modifier le fichier **main.dart** comme suit pour utiliser notre écran.

```
class _MyAppState extends State<MyApp> {
  var color = "red";

  void setColor(String value) => setState(() => color = value);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        useMaterial3: true,
        colorScheme:
            ColorScheme.fromSeed(seedColor: getColorValue(color)),
      ),
      home: HomeScreen(color: color, setColor: setColor),
    );
  }
}
```

Nous pouvons retrouver l'appel à notre écran, en lui donnant la valeur de la variable d'état **color** et avec la méthode callback **setColor** permettant de la modifier. Lancez l'application et vérifiez qu'elle fonctionne toujours comme attendu.

OBSERVATIONS & QUESTIONS

- L'état de l'application, la variable **color**, est enregistré dans le widget **HomeScreen**. Pour y accéder, les autres widgets **MySquare** et **MySwitch** accèdent à cet état par leurs arguments.
- Pour afficher la valeur d'une variable d'état partagée dans un widget enfant, nous passons simplement cette valeur en argument, comme le paramètre **color** du widget **MySquare**.
- Pour modifier la valeur d'une variable d'état partagée depuis un widget enfant, nous passons une fonction callback qui permet faire le changement et reconstruire les widgets suivant le nouvel état, comme le paramètre **setColor** du widget **MySwitch**.

[commit avec message : T03.2 Widgets réutilisables]

2.4 Gestion d'un formulaire

Nous allons maintenant découvrir comment afficher un formulaire et récupérer les inputs de l'utilisateur. Les équipes de flutter ont défini de nombreuses « recettes de cuisine » pour apprendre à écrire des formulaires. Toutes les recettes de cuisine pour gérer un formulaire sont données sur Flutter : Forms.

Nous vous recommandons de lire :

- tout ce qui concerne la validation d'un formulaire : Build a form with validation.
- pour suivre les changement des champs textes : Handle changes to a text field (option 1 ou option 2).
- pour accéder à la valeur d'un champs texte, vous pouvez :
    - soit gérer un état qui est mis à jour par un champs texte lorsque vous suivez ses changements ;
    - soit récupérer la valeur d'un champ texte via un **TextEditingController** : Retrieve the value of a text field

OBSERVATIONS & QUESTIONS

Voici un résumé des points importants :

- Il faut créer un formulaire au sein d'un stateful widget.
- Pour permettra la validation des champs d'un formulaire,
    - Création d'un **Form**, d'une **GlobalKey** et ajout d'un ou plusieurs **TextFormField**
      NB : pour la création de la clé : **final formKey = GlobalKey<FormState>();**
    - Ajout de la logique de validation à chaque **TextFormField** via le passage d'une fonction à l'argument **validator**
    - Lorsqu'un bouton associé au formulaire est pressé, spécifiez l'action en fonction de l'état de validation du formulaire via **formKey.currentState!.validate()**
- Si on n'a pas besoin de validation des champs d'un formulaire, on peut ne pas créer de **Form** et juste ajouter des **TextField** comme champs textes.
- Pour suivre les mises à jour d'un champs texte, il existe deux options :
    - soit via une callback qui doit être passée à l'argument **onChanged** d'un **TextFormField** ou d'un **TextField**. Dans ce cas-là, la callback est appelée à chaque changement.
    - soit via un **TextEditingController** que l'on passe à l'argument **controller** d'un **TextFormField** ou d'un **TextField**. Dans ce cas-là, il faut écouter les changement en ajoutant un écouteur d'événements à un **TextEditingController** via la méthode **addListener** qui reçoit en argument une callback indiquant l'action à réaliser à chaque changement.
- Pour récupérer les valeurs d'un champ texte :
    - Soit vous utilisez une variable d'état et vous la mettez à jour en suivant les mises à jour de ce champs texte (comme décrit au bullet précédent) ;
    - Soit vous utiliser directement un **TextEditingController** et sa propriété **text**, comme par exemple : **myController.text**

2.5 Formulaire simple

Nous souhaitons rajouter à notre application un formulaire permettant de changer la couleur du carré par une couleur indiquée par l'utilisateur.

Copiez le code suivant dans un fichier **my_form.dart**.

```
class MyForm extends StatefulWidget {
  final void Function(String) setColor;

  const MyForm(this.setColor, {super.key});

  @override
  State<MyForm> createState() => _MyFormState();
}

class _MyFormState extends State<MyForm> {
  final controller = TextEditingController();
  final formKey = GlobalKey<FormState>();

  @override
  void dispose() {
    controller.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Form(
      key: formKey,
      child: Row(
        crossAxisAlignment: CrossAxisAlignment.end,
        children: [
          Expanded(
            child: TextFormField(
              controller: controller,
              decoration: const InputDecoration(labelText: "Enter color"),
              validator: (value) {
                return (value == null || value.isEmpty)
                    ? "Color can't be empty"
                    : null;
              },
            ),
          ),
          const SizedBox(width: 32.0),
          ElevatedButton(
            child: const Text("Change color"),
            onPressed: () {
              if (formKey.currentState!.validate()) {
                widget.setColor(controller.text);
                controller.text = "";
              }
            },
          )
        ],
      ),
    );
  }
}
```

Ce widget affiche un formulaire, utilisant un **Form,** une **formKey** pour effectuer une validation du champ, et un **controller** pour accéder à la valeur introduite par l'utilisateur. Il prend également une fonction callback en paramètre pour partager cette valeur avec les widgets parents.

Nous pouvons maintenant faire appel à ce formulaire dans le fichier **home_screen.dart**.

```
Column(
  children: [
    MySquare(color: color),
    MySwitch(color: color, setColor: setColor),
    MyForm(setColor),
  ],
)
```

Lancez l'application et vérifiez que vous pouvez utiliser le formulaire.

[commit avec message : T03.3 Formulaire]

3 Exercice

3.1 Introduction

Veuillez créer un nouveau projet (New Flutter Project) nommé *ex3* dans votre repository de cours.

L'objectif de cet exercice est de créer une application pour enregistrer une liste de notes. Les notes sont caractérisées par un titre et un contenu. Commencez par définir un objet de données correspondant.

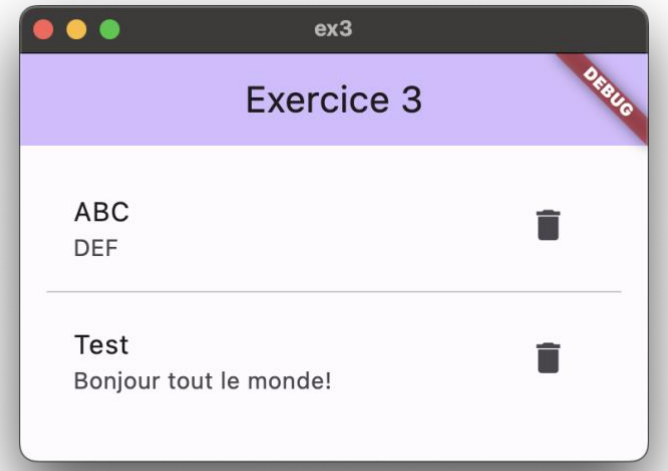
Afin de pouvoir manipuler ces objets au sein de listes, il est nécessaire de redéfinir l'opérateur == (l'équivalent de la fonction **equals** en Java) et la fonction **hashCode**. Android est capable de les générer pour vous en effectuant un clic droit, et en choisissant « ==() and hashCode » dans le menu **Generate**. Deux notes sont considérées comme équivalentes si elles ont le même titre.

⚡ **[commit avec message : F04.1 Représentation des données]** ⚡

3.2 Liste de notes

Créez un widget **HomeScreen** qui contient une liste de notes. Chaque note doit afficher son titre et son texte. Un bouton à la fin de chaque ligne doit permettre de supprimer la note de la liste. Vous pouvez utiliser le widget **IconButton** pour afficher ce bouton. Vous pouvez retrouver la documentation de ce widget à ce lien : IconButton.

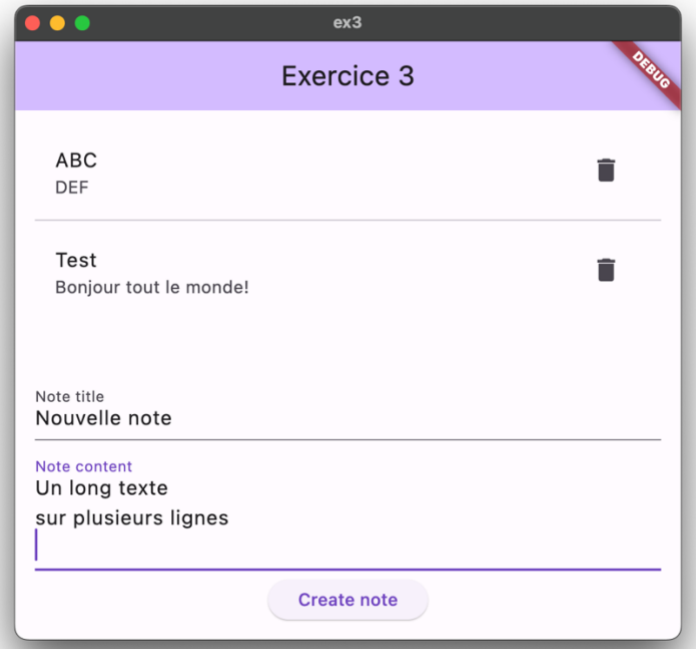
Testez votre application avec une liste de notes par défaut. À ce stade, votre application devrait ressembler à la capture suivante.



⚡ **[commit avec message : F04.2 Liste de notes]** ⚡

3.3 Formulaire

Créez en dessous de la liste au sein du widget **HomeScreen** un formulaire permettant de créer une note. Ce formulaire doit afficher deux champs de texte, le premier pour le titre et le deuxième pour le contenu de la note. En appuyant sur un bouton de soumission du formulaire, la note est créée et rajoutée à la liste. Si un des champs est vide, le formulaire doit afficher une erreur. À ce stade, votre application devrait ressembler à la capture suivante.



⚡ **[commit avec message : F04.3 Formulaire]** ⚡

3.4 État partagé

Modifiez maintenant votre application pour créer des widgets de composants réutilisables pour la liste de notes ainsi que pour le formulaire de création de note. Après avoir fait cette modification, votre application devrait rester fonctionnellement identique.

⚡ **[commit avec message : F05.1 État partagé]** ⚡

3.5 Challenges optionnels

Si vous souhaitez aller plus loin, vous pouvez modifier l'application pour avoir ces comportements :

- Dans le formulaire de création d'une note, la note est créée comme si l'utilisateur avait cliqué sur le bouton de soumission lorsque l'utilisateur appuie sur la touche Enter en modifiant le champ « Note content ». Ce champ ne peut pas être multiligne pour que cette extension soit possible.
- Lorsque une note est créée, La liste de notes défile automatiquement jusqu'à la dernière note pour afficher celle que l'on vient de créer.