|  |  |
| --- | --- |
| Flutter |  |
| **Fiche 3 : Formulaires & État partagé** | |

Table des matières

[1 Objectifs à valider 2](#_Toc129183542)

[2 Concepts 2](#_Toc129183543)

[2.1 Widgets avec un état partagé 2](#_Toc129183544)

[2.2 D’un widget monolithique… 2](#_Toc129183545)

[2.3 …à des widgets réutilisables 4](#_Toc129183546)

[2.4 Gestion d’un formulaire 6](#_Toc129183547)

[2.5 Formulaire simple 6](#_Toc129183548)

[3 Exercice 8](#_Toc129183549)

[3.1 Introduction 8](#_Toc129183550)

[3.2 Liste de notes 8](#_Toc129183551)

[3.3 Formulaire 9](#_Toc129183552)

[3.4 État partagé 9](#_Toc129183553)

# Objectifs à valider

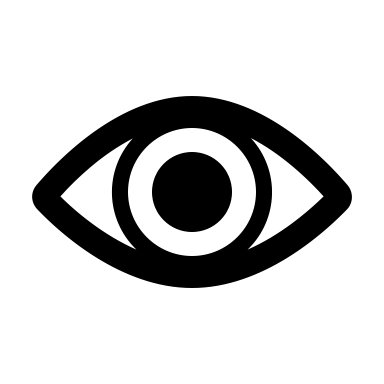
|  |  |  |
| --- | --- | --- |
| **ID** | **Objectifs** | **Indicateurs** |
| F04 | Gestion d’un formulaire | * Les messages de commit F04.1, F04.2 et F04.3 sont visibles. * Vous montrez et expliquez comment vous validez les champs de votre formulaire. * Vous montrez et expliquez comment vous accédez aux valeurs du formulaire lors de la création d’une note. |
| F05 | État partagé | * Le message de commit F05.1 est visible. * Vous savez expliquer quels widgets sont stateful et stateless et la raison de ce choix. * Vous savez expliquer comment les différents widgets partagent un état. |

# Concepts

Pour commencer, créez un nouveau projet flutter nommé ***ftuto3*** dans votre repository de cours.

## 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](https://docs.flutter.dev/development/ui/interactive#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.

## D’un widget monolithique…

Pour donner un exemple de ce phénomène, nous allons créer une application affichant un carré avec une couleur définie par l’utilisateur. Copiez le code suivant dans un fichier *home\_screen.dart*.

*class* HomeScreen *extends* StatefulWidget {  
 *const* HomeScreen({Key? key}) : *super*(key: key);  
  
 *@override* State<HomeScreen> createState() => \_HomeScreenState();  
}  
  
*class* \_HomeScreenState *extends* State<HomeScreen> {  
 *final* colors = {  
 "red": Colors.red,  
 "green": Colors.green,  
 "blue": Colors.blue,  
 "yellow": Colors.yellow,  
 "black": Colors.black,  
 "white": Colors.white,  
 };  
  
 *var* color = "blue";  
 *var* switchValue = *false*;  
  
 *@override* Widget build(BuildContext context) {  
 *return* Scaffold(  
 appBar: AppBar(title: *const* Text("Tutoriel 3")),  
 body: Padding(  
 padding: *const* EdgeInsets.all(32.0),  
 child: Column(  
 children: [  
 Expanded(  
 child: Center(  
 child: Container(  
 width: 100,  
 height: 100,  
 color: colors[color] ?? Colors.grey,  
 ),  
 ),  
 ),  
 Row(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: [  
 *const* Text("red"),  
 Switch(  
 value: color == "green",  
 activeColor: Colors.green,  
 inactiveThumbColor: Colors.red,  
 inactiveTrackColor: Colors.red.withOpacity(0.4),  
 onChanged: (value) =>  
 setState(() => color = value ? "green" : "red"),  
 ),  
 *const* Text("green"),  
 ],  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
}

Analysez ce code pour vous assurer de bien le comprendre. Faites appel à ce widget dans votre application, et lancez la pour le tester.

**[commit avec message : FT03.1 Widget monolithique]**

## …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({  
 Key? key,  
 *required this*.color,  
 *required this*.setColor,  
 }) : *super*(key: key);  
  
 *@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](https://api.flutter.dev/flutter/dart-html/VoidCallback.html) pour une fonction qui ne prend pas d’argument et ne renvoie rien, ou [ValueChanged<String>](https://api.flutter.dev/flutter/foundation/ValueChanged.html) 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*.

*class* MySquare *extends* StatelessWidget {  
 *final* colors = {  
 "red": Colors.red,  
 "green": Colors.green,  
 "blue": Colors.blue,  
 "yellow": Colors.yellow,  
 "black": Colors.black,  
 "white": Colors.white,  
 };  
 *final* String color;  
  
 MySquare({Key? key, *required this*.color}) : *super*(key: key);  
  
 *@override* Widget build(BuildContext context) {  
 *return* Expanded(  
 child: Center(  
 child: Container(  
 width: 100,  
 height: 100,  
 color: colors[color] ?? Colors.grey,  
 ),  
 ),  
 );  
 }  
}

Ce widget affiche le composant dédié à l’affichage du carré, en prenant sa couleur actuelle en argument.

Il ne reste plus qu’à modifier le fichier *home\_screen.dart* pour utiliser ces widgets. Remplacez la classe *\_HomeScreenState* par le code suivant.

*class* \_HomeScreenState *extends* State<HomeScreen> {  
 *var* color = "red";  
  
 *@override* Widget build(BuildContext context) {  
 *return* Scaffold(  
 appBar: AppBar(title: *const* Text("Tutoriel 3")),  
 body: Padding(  
 padding: *const* EdgeInsets.all(32.0),  
 child: Column(  
 children: [  
 MySquare(color: color),  
 MySwitch(  
 color: color,  
 setColor: (value) => setState(() => color = value),  
 ),   
 ],  
 ),  
 ),  
 );  
 }  
}

Nous pouvons retrouver l’appel à nos différents widgets, en leur 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.

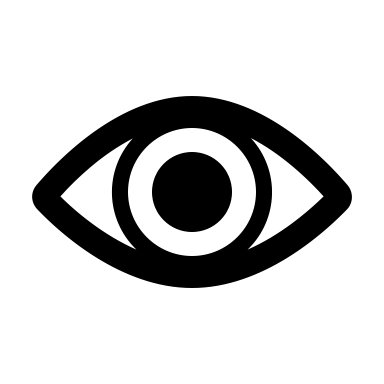
**[commit avec message : FT03.2 Widgets réutilisables]**

## 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](https://docs.flutter.dev/cookbook#forms).

Nous vous recommandons de lire :

* tout ce qui concerne la validation d’un formulaire : [Build a form with validation](https://docs.flutter.dev/cookbook/forms/validation).
* pour suivre les changement des champs textes : [Handle changes to a text field (option 1 ou option 2)](https://docs.flutter.dev/cookbook/forms/text-field-changes).
* 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](https://docs.flutter.dev/cookbook/forms/retrieve-input)

****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**

## 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, {Key? key}) : *super*(key: key);  
  
 *@override* State<MyForm> createState() => \_MyFormState();  
}  
  
*class* \_MyFormState *extends* State<MyForm> {  
 *final* controller = TextEditingController();  
 *final* key = GlobalKey<FormState>();  
  
 *@override  
 void* dispose() {  
 controller.dispose();  
 *super*.dispose();  
 }  
  
 *@override* Widget build(BuildContext context) {  
 *return* Form(  
 key: key,  
 child: Row(  
 crossAxisAlignment: CrossAxisAlignment.end,  
 children: [  
 Expanded(  
 child: TextFormField(  
 controller: controller,  
 decoration: *const* InputDecoration(labelText: "Enter color"),  
 validator: (value) =>  
 (value == *null* || value.isEmpty) ? "Color can't be empty" : *null*,  
 ),  
 ),  
 *const* SizedBox(width: 32.0),  
 ElevatedButton(  
 child: *const* Text("Change color"),  
 onPressed: () {  
 *if* (key.currentState!.validate()) {  
 widget.setColor(controller.text);  
 controller.text = "";  
 }  
 },  
 )  
 ],  
 ),  
 );  
 }  
}

Ce widget affiche un formulaire, utilisant un *Form* et une *key* 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 le widget parent.

Nous pouvons maintenant faire appel à ce formulaire dans le fichier *home\_screen.dart*.

children: [  
 MySquare(color: color),  
 MySwitch(  
 color: color,  
 setColor: (value) => setState(() => color = value),  
 ),  
 MyForm((value) => setState(() => color = value)),  
],

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

**[commit avec message : FT03.3 Formulaire]**

# Exercice

## 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 texte. 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]** ⚡

## 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](https://api.flutter.dev/flutter/material/IconButton-class.html).

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]** ⚡

## 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 texte 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.

**Une image contenant texte, capture d’écran, moniteur

Description générée automatiquement**

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

## 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]** ⚡