Flutter Pizzeria - TD 6

# Important

**Faites une sauvegarde de votre projet car vous allez tout « casser » 😊.**

# Le panier

Nous avons tout mis en place pour pouvoir commander des pizzas. Nous allons donc faire le panier.

## Le provider

Voir <https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple>

Dans le TD 4, nous avions vu différentes stratégies pour gérer les états. Nous allons mettre en place l’utilisation des providers.

Le package Provider permet d’implémenter le Design Pattern Observable. Il se base donc sur un objet pouvant notifier une liste d’objets d’un changement de son état.

Dans notre cas, la classe Panier serait chargée de notifier tout changement aux autres classes de type UI.

Remarque : “*Le Provider n’est pas un système de gestion d'état en soi, mais peut être considéré comme une partie d’un système de gestion d'état.*” (extrait de <https://www.didierboelens.com/fr/2019/07/provider-points-dint%C3%A9r%C3%AAt-points-dattention/>).

## Les éléments nécessaires

### Le package

Pour utiliser les provider, il faut logiquement intégrer le package dans pubspec.yaml (extrait) :

dependencies:

flutter:

sdk: flutter

provider: ^6.0.0

### Les classes

Les classes minimales sont :

* ChangeNotifierProvider : Le fournisseur de l'élément “à surveiller”
* ChangeNotifier : permet de gérer les listeners et donc d’émettre une notification de changement
* Consumer : permet d’écouter les notifications émises

# Prodiver - Mise en place

## Le fournisseur

Dans notre cas, nous voulons que tout changement dans le panier soit reflété dans l’application. Nous n'avons donc qu'une seule classe. Dans ce cas nous utiliserons la classe ChangeNotifierProvider.

Si nous avions plusieurs classes “à surveiller”, il faudrait utiliser la classe MultiProvider.

## Mise en place

Le provider doit être créé au lancement de l'application comme ceci (fichier main.dart) :

void main() {

runApp(

ChangeNotifierProvider(

create: (context) => Cart(),

child: MyApp(),

),

);

}

La classe ChangeNotifierProvider ajoute la classe Cart (à surveiller), elle est liée au contexte.

## Emission de notification lors de changement

La classe Cart doit « prévenir » les autres classes de tout changement.

La classe Cart doit utiliser la classe ChangeNotifier et “prévenir” du changement avec la méthode notifyListeners.

Le panier est modifié uniquement lors de l’ajout et la suppression d'éléments, le code sera alors le suivant (extrait de la classe) :

class Cart extends ChangeNotifier {

*/\* Le reste du code \*/*

void addProduct(Pizza pizza) {

*// Recherche du produit*

int index = \_findCartItemIndex(pizza);

if (index == -1) {

*// Ajout*

\_items.add(CartItem(pizza));

} else {

*// Incrémente la quantité*

CartItem item = \_items[index];

item.quantity++;

}

***// Emission d'une notification de changement***

**notifyListeners();**

}

void removeProduct(Pizza pizza) {

*// Recherche du produit*

int index = \_findCartItemIndex(pizza);

if (index != -1) {

CartItem item = \_items[index];

*// Décrémente la quantité*

if (--item.quantity == 0) {

*// Suppression*

\_items.removeAt(index);

}

***// Emission d'une notification de changement***

**notifyListeners();**

}

}

}

## Réception des notifications

Différentes manières de recevoir une notification :

* Provider.of<T>(context, [listen = true])
* context.watch<T>()
* Consumer Widget

Les deux premières façons s'utilisent dans la méthode build car elles nécessitent d'avoir le contexte.

Le widget Consumer peut s’utiliser n’importe où dans la hiérarchie des widgets.

Nous allons voir comment les utiliser dans les widgets.

# 

# Le widget BuyButtonWidget

Ce widget permet d’ajouter un élément dans le panier.

Le widget doit accéder à l’objet cart du provider mais il ne doit pas recevoir les notifications car le bouton ne change pas lors d’ajout/suppression d’un élément.

Le code sera le suivant :

class BuyButtonWidget extends StatelessWidget {

final Pizza \_pizza;

const BuyButtonWidget(this.\_pizza, {Key? key}) : super(key: key);

@override

Widget build(BuildContext context) {

***// Obtention de l'objet cart depuis le provider***

***// Nous n'écoutons pas les notifications émises***

**var cart = Provider.*of*<Cart>(context, listen: false);**

*// La suite du code…*

}

}

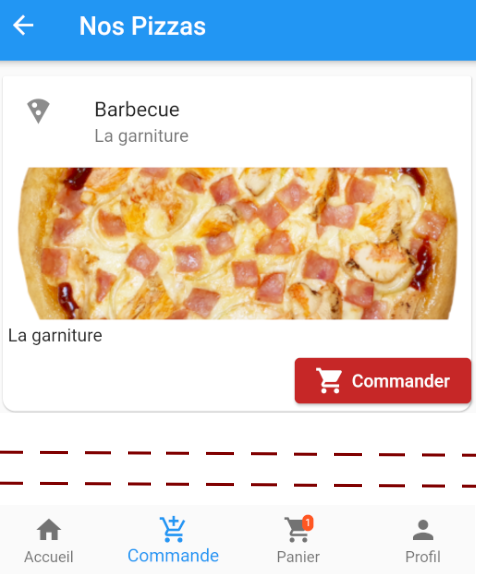
Explication : pour obtenir l’objet cart il faut utiliser soit “Provider.of<T>(context, [listen = true])” soit “context.watch<T>()”. Or la méthode watch implique de recevoir les notifications.

Nous utilisons “Provider.*of*<Cart>(context, listen: false);” en mettant “listen” à false pour ne pas recevoir de notifications.

# 

# Le widget BottomNavigationBarWidget

Dans cette version de l’application Pizzeria, nous utilisons un [BottomNavigationBar](https://api.flutter.dev/flutter/material/BottomNavigationBar-class.html) afin d’avoir l’UI suivante :



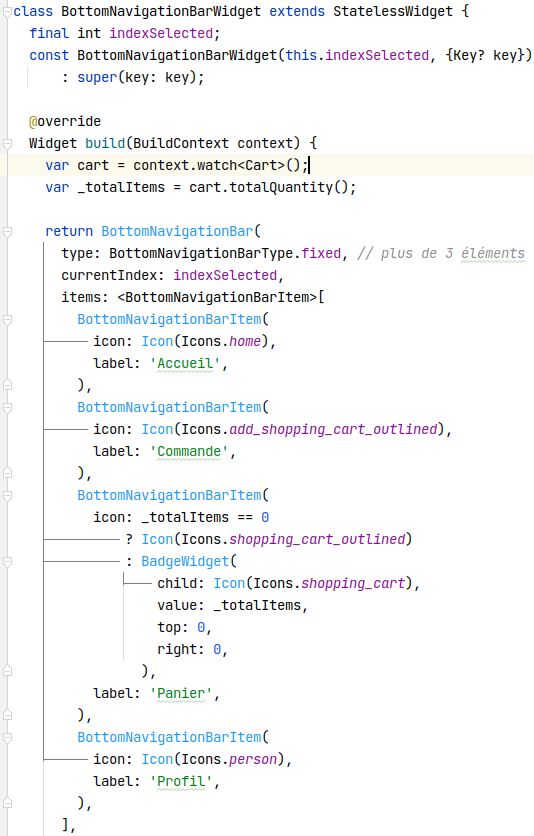
Ce widget affiche le nombre d’éléments présents dans le panier.

Il doit donc être rafraîchi à chaque changement du panier (Cart).

Nous allons utiliser la version “context.watch<T>()” pour être notifié des changements.

Le widget BadgeWidget est défini plus loin.

Le code complet sera le suivant :





Vous noterez que nous utilisons ici [la navigation nommée](https://docs.flutter.dev/cookbook/navigation/named-routes).

## La navigation nommée

Pour naviguer vers une page, vous avez 2 choix :

Soit donner le nom du widget, ex :

Navigator.*of*(context)  
 .push(MaterialPageRoute(  
 builder: (context) => PizzaList())  
);

Soit donner le nom de la route (exemple ci-dessus) mais dans ce cas vous devez déclarer les routes.

### Déclaration des routes

Nous déclarons les routes au lancement du programme (fichier main.dart). Vous devez modifier le code comme ceci (extrait) :

class MyApp extends StatelessWidget {  
 *// This widget is the root of your application.* @override  
 Widget build(BuildContext context) {  
 return MaterialApp( *// Le code précédent …*  
 home: HomePage(title: 'Ma Pizzeria'),  
 *// Start the app with the "/" named route. In this case, the app starts  
 // on the FirstScreen widget.  
 /\*initialRoute: '/',\*/* routes: {  
 '/profil': (context) => Profil(),  
 '/panier': (context) => Panier(),  
 },  
 );  
 }  
}

## Le widget BadgeWidget

Ce widget doit afficher :

* Une icône panier
* Un cercle ayant une couleur de fond
* Un nombre indiquant le nombre total de produits achetés

Pour ce faire, nous allons utiliser un widget [Stack](https://api.flutter.dev/flutter/widgets/Stack-class.html) et un widget [Positioned](https://api.flutter.dev/flutter/widgets/Positioned-class.html).

Le code sera le suivant :





# Le widget Panier

Ce widget permet d’afficher la liste des éléments du panier.

Pour rappel, le panier affiche :

* la liste des éléments
* le total

Ces 2 parties doivent être rafraîchies à chaque changement du panier.

Dans le TD 4, vous avez écrit du code qui vous permet :

* D’afficher la liste des pizzas commandées : déplacez ce code vers le widget \_CartList (vous avez un extrait de code plus loin)
* D’afficher le total de la commande en utilisant les widgets Table et TableRow : déplacez ce code vers le widget \_CartTotal (vous avez un extrait de code plus loin)

Le code de départ du panier est le suivant :



Les widgets \_CartList et \_CartTotal doivent être rafraîchis.

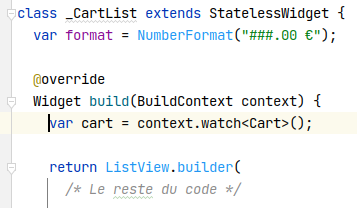
Nous allons utiliser 2 manières différentes juste pour montrer ces manières. Nous pourrions utiliser une seule manière.

## Le widget \_CartList

Vous pouvez

* Soit définir le widget en « private » avec le « underscore » devant le nom de la classe et dans ce cas vous devez mettre le widget dans le fichier « panier.dart »
* Soit définir le widget en « public » et dans ce cas vous pouvez créer un fichier cart\_list.dart pour ce widget.

Nous allons utiliser la méthode “context.watch<T>()” comme ceci (extrait) :



## Le widget \_CartTotal

Vous pouvez

* Soit définir le widget en « private » avec le « underscore » devant le nom de la classe et dans ce cas vous devez mettre le widget dans le fichier « panier.dart »
* Soit définir le widget en « public » et dans ce cas vous pouvez créer un fichier cart\_total.dart pour ce widget.

Nous allons utiliser la méthode “Consumer”. Ce widget permet d’être intégré dans la hiérarchie des widgets comme ceci (extrait) :



# Résumé

Vous devez maitriser les éléments suivants :

* Utilisation de [Providers](https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple)
* Utilisation de la [navigation nommée](https://docs.flutter.dev/cookbook/navigation/named-routes)