Cross-Plateforme

Flutter 3

Nous allons créer une application simple de gestion de tâches où l'utilisateur peut ajouter et supprimer des tâches.

Étape 1 : Configuration du projet Flutter avec GetX

a. Créer un nouveau projet Flutter :

```
flutter create task_management
cd task_management
```

b. Ajouter GetX dans le fichier `pubspec.yaml` :

Ouvre le fichier `pubspec.yaml` et ajoute GetX dans les dépendances :

```
dependencies:
  flutter:
    sdk: flutter
  get: ^4.6.5 # Dernière version stable de GetX
```

Ensuite, exécute la commande suivante pour installer les packages :

```
flutter pub get
```

Étape 2 : Créer un modèle de tâche (Task Model)

Commençons par créer un modèle simple pour représenter une tâche.

a. Créer un fichier `task_model.dart` :

```
class Task {
  String title;
  bool isDone;

  Task({required this.title, this.isDone = false});
}
```

Ce modèle représente une tâche avec un titre et un état booléen `isDone` pour indiquer si la tâche est terminée ou non.

Étape 3 : Créer un contrôleur pour gérer l'état (GetX Controller)

Nous allons maintenant créer un contrôleur qui gérera la logique de l'application, comme l'ajout et la suppression des tâches, ainsi que la gestion de leur état (fait ou non).

a. Créer un fichier `task_controller.dart` :

```
import 'package:get/get.dart';
import 'task_model.dart';

class TaskController extends GetxController {
  // Liste observable de tâches
  var tasks = <Task>[].obs;

  // Ajouter une nouvelle tâche
  void addTask(String title) {
    tasks.add(Task(title: title));
  }

  // Marquer une tâche comme terminée
  void toggleTaskStatus(int index) {
    tasks[index].isDone = !tasks[index].isDone;
    tasks.refresh();
  }

  // Supprimer une tâche
  void deleteTask(int index) {
    tasks.removeAt(index);
  }
}
```

Le contrôleur contient une liste observable de tâches (`tasks`), ainsi que des méthodes pour ajouter une tâche, changer l'état d'une tâche, et supprimer une tâche. La méthode `.refresh()` est appelée après avoir modifié l'état d'une tâche pour informer GetX que l'interface doit être mise à jour.

Étape 4 : Créer l'interface utilisateur

Maintenant que nous avons notre modèle et notre contrôleur, nous allons construire l'interface utilisateur de l'application.

a. Créer un fichier `home_page.dart` :

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

class HomePage extends StatelessWidget {
  final TaskController taskController = Get.put(TaskController());

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Task Manager'),
      ),
```

```
      body: Column(
        children: [
          Expanded(
            child: Obx(() => ListView.builder(
              itemCount: taskController.tasks.length,
              itemBuilder: (context, index) {
                final task = taskController.tasks[index];
                return ListTile(
                  title: Text(
                    task.title,
                    style: TextStyle(
                      decoration: task.isDone ? TextDecoration.lineThrough
: null,
                    ),
                  ),
                  trailing: IconButton(
                    icon: Icon(Icons.delete),
                    onPressed: () {
                      taskController.deleteTask(index);
                    },
                  ),
                  leading: Checkbox(
                    value: task.isDone,
                    onChanged: (value) {
                      taskController.toggleTaskStatus(index);
                    },
                  ),
                );
              },
            )),
          ),
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: Row(
              children: [
                Expanded(
                  child: TextField(
                    controller: taskController.taskInputController,
                    decoration: InputDecoration(
                      labelText: 'Ajouter une tâche',
                      border: OutlineInputBorder(),
                    ),
                  ),
                ),
                SizedBox(width: 10),
                ElevatedButton(
                  onPressed: () {

taskController.addTask(taskController.taskInputController.text);
                    taskController.taskInputController.clear();
                  },
                  child: Text('Ajouter'),
                ),
              ],
            ),
          ),
        ],
      ),
    );
  }
}
```

b. Explication :

1. **Obx** : Cette méthode est utilisée pour mettre à jour l'interface utilisateur chaque fois que la liste des tâches change.
2. **ListView.builder** : Il affiche la liste des tâches.
3. **TextField** : Il permet à l'utilisateur d'entrer le titre d'une tâche.
4. **Checkbox** : Il permet de marquer une tâche comme terminée ou non.
5. **IconButton (delete)** : Pour supprimer une tâche de la liste.

Étape 5 : Lancer l'application

Modifie le fichier `main.dart` pour afficher la page d'accueil :

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      debugShowCheckedModeBanner: false,
      home: HomePage(),
    );
  }
}
```

Maintenant, exécute l'application :

```
flutter run
```

Conclusion

Dans cet atelier, tu as appris à :

- Utiliser **GetX** pour gérer l'état (state management).
- Créer un modèle de tâche.
- Créer un contrôleur GetX pour gérer les tâches.
- Construire une interface utilisateur qui permet d'ajouter, de marquer comme fait, et de supprimer des tâches.

GetX facilite la gestion de l'état en rendant les données observables et en les reliant directement à l'interface utilisateur. Tu peux maintenant continuer à développer cette application en y ajoutant plus de fonctionnalités ou à explorer d'autres fonctionnalités de GetX.