Flutter

Fiche 6 : Future builder & déploiement

Table des matières

1 Objectifs à valider

ID	Objectifs
F10	Affichage de différents widgets via un FutureBuilder
F11	Déploiement d'une app

2 Concepts

2.1 Introduction

Pour commencer le tutoriel, créez un nouveau projet (New Flutter Project) nommé *tuto6* dans votre repository de cours.

2.2 Future Builder & icône de chargement

Nous vous recommandons de regarder la vidéo donnée ici (1 minute) : FutureBuilder<T> class.

Pour le tutoriel, vous devez installer la librairie **http**. Veuillez donc ouvrir un terminal et taper : **flutter pub add http**

Nous allons maintenant développer une application qui affiche des « posts » offert par une API. Lors du chargement des données, nous souhaitons afficher un indicateur de progrès.

Veuillez ajouter le code de **MyHomeScreen** dans **main.dart** :

```
class MyHomeScreen extends StatefulWidget {
  const MyHomeScreen({super.key});

  @override
  _MyHomeScreenState createState() => _MyHomeScreenState();
}

class _MyHomeScreenState extends State<MyHomeScreen> {
  Future<List<dynamic>> _fetchData() async {
    final response =
        await get(Uri.parse('https://jsonplaceholder.typicode.com/posts'));
    if (response.statusCode == 200) {
      return json.decode(response.body);
    } else {
      throw Exception('Failed to load data');
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('FutureBuilder Example'),
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
      ),
      body: FutureBuilder<List<dynamic>>(
```

```
        future: _fetchData(),
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            return ListView.builder(
              itemCount: snapshot.data?.length,
              itemBuilder: (context, index) {
                final post = snapshot.data?[index];
                return ListTile(
                  title: Text(post['title']),
                  subtitle: Text(post['body']),
                );
              },
            );
          }
          if (snapshot.hasError) {
            return Center(child: Text('${snapshot.error}'));
          }

          return const Center(child: CircularProgressIndicator());
        },
      ),
    );
  }
}
```

Dans cet exemple, **_fetchData()** est une méthode asynchrone qui utilise le package **http** pour récupérer des données JSON d'une API. La méthode renvoie une liste de **dynamic** qui sera affichée dans la **ListView**.

Le **FutureBuilder** est utilisé pour créer une interface utilisateur réactive qui affichera les données une fois qu'elles seront disponibles. Le **builder** prend un **context** et un **snapshot** en entrée. Le **snapshot** contient les données renvoyées par la **Future** ainsi que des informations sur l'état actuel de la **future**.

Dans le **builder**, on utilise les propriétés **hasData**, **hasError** et **connectionState** du snapshot pour déterminer l'état actuel de la **Future**. Si les données ont été récupérées avec succès, la **ListView** sera construite avec les données récupérées. Si une erreur est survenue, un message d'erreur sera affiché. Si les données ne sont pas encore disponibles, une icône de chargement sera affichée à la place.

Veuillez tester votre application en intégrant **MyHomeScreen** au sein de votre **MaterialApp.**

[commit avec message : T06.1 FutureBuilder]

Si la connexion réseau est acceptable, nous n'avons pas vraiment le temps de voir l'icône de chargement. Nous allons donc rajouter un délai de 3 secondes au sein du **FutureBuilder**, en modifiant la valeur du paramètre **future** :

```
future: Future.delayed(const Duration(seconds: 3), () => _fetchData()),
```

Dans le code ci-dessus, la méthode **_fetchData()** est appelée à l'intérieur de la méthode **Future.delayed()** qui introduit un délai de 3 secondes avant de renvoyer la **Future**. Le widget **FutureBuilder** est alimenté par cette **Future** retournée par **Future.delayed()**, de sorte que le délai de 3 secondes est respecté avant que les données ne soient affichées.

[commit avec message : T06.2 Loading icon & delay]

3 Exercices

3.1 Introduction

Veuillez créer un nouveau projet (New Flutter Project) nommé *ex6* dans votre repository de cours.

Nous souhaitons créer une application permettant de gérer des photos qui sont offertes par l'API https://unreal-api.azurewebsites.net.

Pour notre application, seuls les données associées à nos photos (URLs, titre, id…) seront traitées par https://unreal-api.azurewebsites.net. Le stockage des fichiers « photos » référencés par l'API n'est pas considéré dans notre application.

⚡ Dès l'initialisation de votre exercice, vous devez prévoir de **développer** selon **l'architecture MVVM** vue précédemment, en utilisant un **état partagé** pour afficher vos données : l'état de l'application sera une Future d'une liste de photos.

3.2 Affichage d'un loader & de photos avec un FutureBuilder

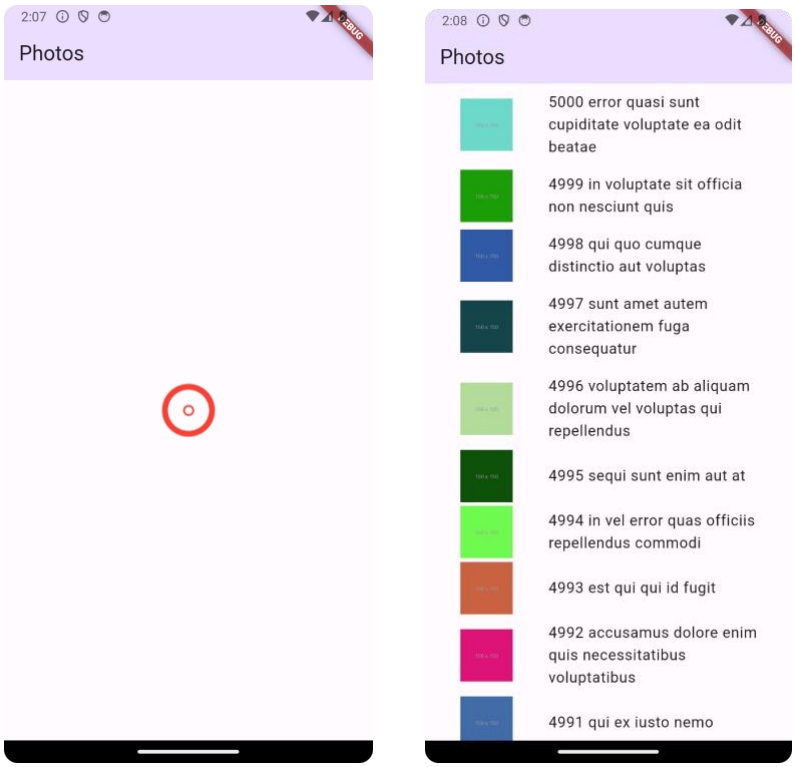
En utilisant des requêtes sur les ressources de type /photos, veuillez afficher à la demande les photos (**thumbnailUrl**, **id** & **title**) sous forme de galerie, dans l'ordre décroissant des propriétés **id**.

Si vous avez des soucis pour afficher les photos sur le Web, lancez votre application à partir d'un terminal : **flutter run -d chrome --web-renderer html**
Dans ce cas-ci, vous pouvez bénéficier d'un **hot reload** en tapant sur **r** au sein du terminal.

Au chargement de l'application, veuillez ajouter un délai de 4 secondes avant d'afficher la galerie de photos. Pendant ce délai, veuillez afficher une belle animation de chargement en utilisant un package offert par https://pub.dev/ ; voici les mots clés utiles pour votre recherche d'un package : **loading animation**.

⚡ **[commit avec message : F10.1 FutureBuilder & loader package]** ⚡

Voilà à quoi pourrait ressembler votre application :



3.3 Ajout d'une photo

Veuillez créer un formulaire permettant l'ajout d'une photo en faisant une requête de type **POST** sur /photos. Ce formulaire doit être atteignable via un **floating button**. Voici la représentation d'une nouvelle ressource de type « photos » que vous pourriez utiliser lors d'une requête POST via votre app :

```
{
  "title": "accusamus beatae ad facilis cum similique qui sunt",
  "thumbnailUrl": "https ://via.placeholder.com/150/92c952"
}
```

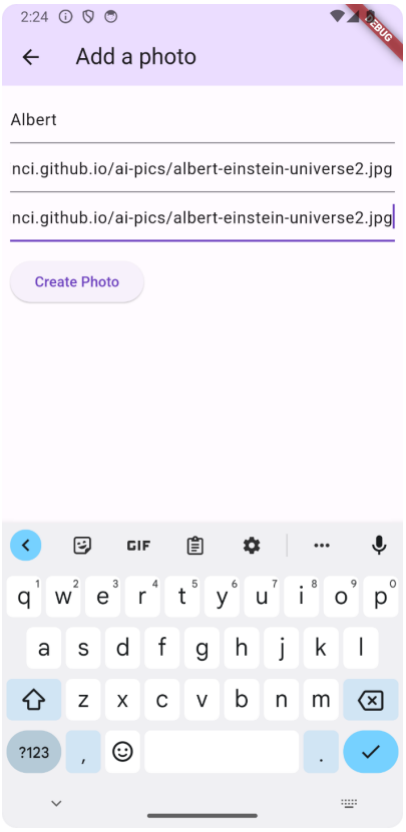
Notons qu'il n'est pas nécessaire d'ajouter tous les champs d'une ressource de type « photos » lors d'une requête POST (**albumId**, **url** ne sont pas obligatoires, **id** est généré automatiquement par l'API).

Pour créer une requête POST, dans le **body** de votre requête (voir la documentation d'http), vous pouvez utiliser la fonction jsonEncode en lui passant une **Map** pour créer une représentation JSON.

Une fois la photo ajoutée, vous devez rediriger l'utilisateur vers l'affichage de la galerie et assurer que la galerie est à nouveau chargée afin d'afficher la nouvelle photo sans action de l'utilisateur.

⚡ **[commit avec message : F10.2 Ajout de photos & reloading]** ⚡

Voilà à quoi pourrait ressembler votre application :



3.4 Déploiement d'une app

Et oui, même si ce cours est axé mobile, il ne l'est pas uniquement. On s'intéresse ici au développement multiplateforme !

Du coup, veuillez réaliser au choix un de ces déploiements :

- Déployer la version web de votre application sur le cloud (Github Pages par exemple)
- Déployer une version Windows de votre application
- Déployer une version Mac de votre application
- Déployer une version Linux de votre application

NB : Nous ne vous proposons pas de déployer sur l'Android App Store ni sur l'iOS App Store car cela est payant.

Si vous souhaitez déployer une version desktop de votre application, vous devrez d'abord rajouter la plateforme à votre projet. Vous devrez ensuite faire un build de votre projet pour cette plateforme pour générer l'application prête à être déployée (fichier **.exe** sur Windows, **.app** sur macOS, …).

🔍 Attention, nous vous demandons de réaliser les recherches nécessaires en autonomie : préparez vos outils de détective !

[Commit seulement si vous avez dû faire des changements pour déployer. Message éventuel du commit : "F11 Déploiement d'une app"]