

Découvrir les bases de développement multiplateforme

50 heures

SOMMAIRE



INTRODUIRE LE DÉVELOPPEMENT MULTIPLATEFORME

01 Partie 1
INTRODUIRE FLUTTER

- Activité 1 : Installation de l'environnement Flutter

02 Partie 2
DÉCOUVRIR LE LANGAGE DE PROGRAMMATION DART

- Activité 1 : Programmer orientée objet avec Dart
- Activité 2 : Utilisation des widgets de base

03 Partie 3
3. MANIPULER LES DONNÉES

- Activité 1 : Convert et parser du JSON
- Activité 2 : Interactions avec la base de données

04 Partie 4
4. MAÎTRISER LA NAVIGATION ET LE ROUTING

- Activité 1 : Création facile d'une nouvelle route
- Activité 2 : Passage des données d'un écran à un autre

MODALITÉS PÉDAGOGIQUES



1

LE GUIDE DE SOUTIEN

Il contient le résumé théorique et le manuel des travaux pratiques



2

LA VERSION PDF

Une version PDF est mise en ligne sur l'espace apprenant et formateur de la plateforme WebForce Life



3

DES CONTENUS TÉLÉCHARGEABLE

Les fichiers des résumés ou des exercices sont téléchargeables sur WebForce Life



4

DU CONTENU INTERACTIF

Vous disposez de contenus interactifs sous forme d'exercices et de cours à utiliser sur WebForce Life

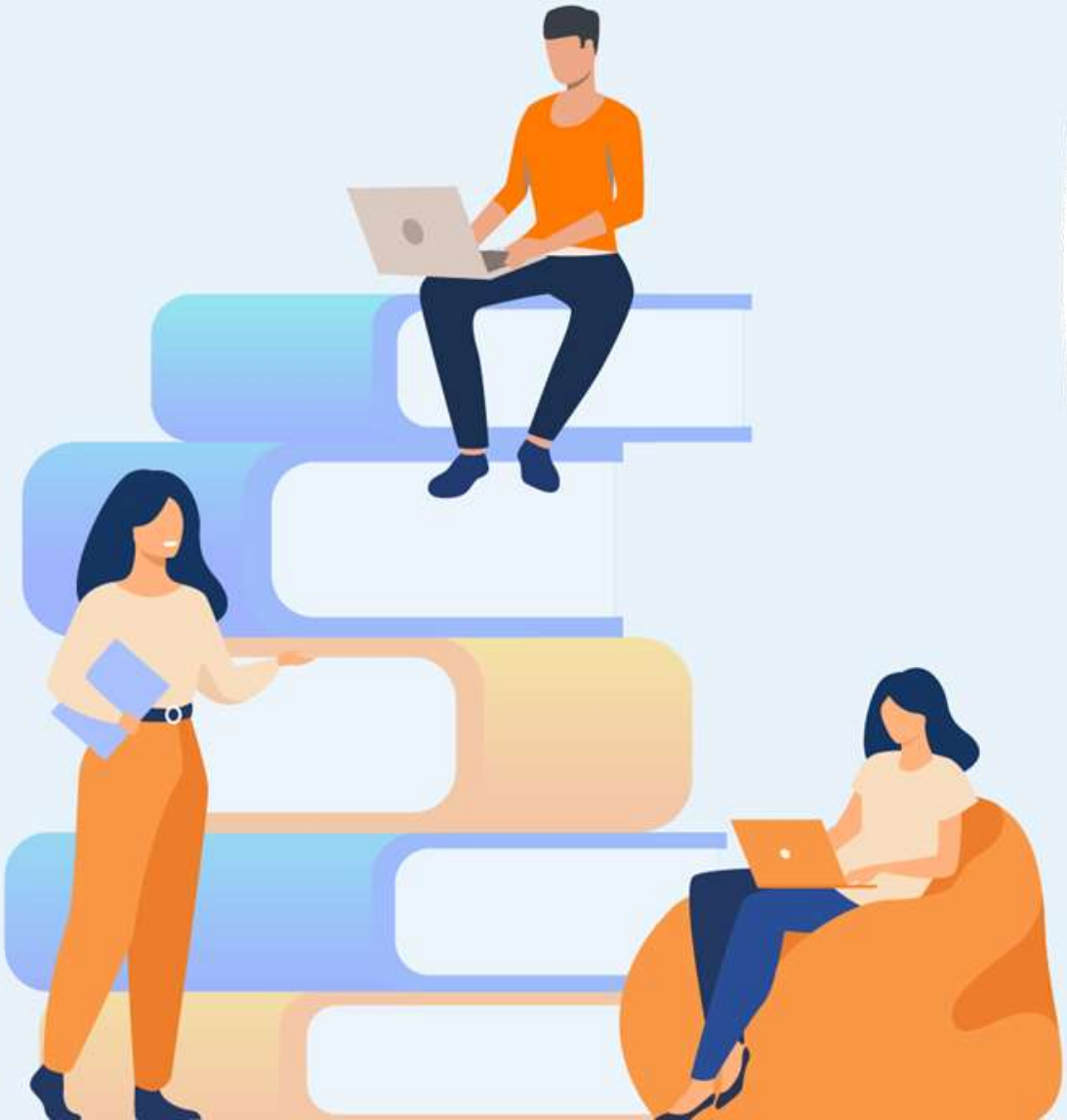


5

DES RESSOURCES EN LIGNES

Les ressources sont consultables en synchrone et en asynchrone pour s'adapter au rythme de l'apprentissage

Copyright - Tout droit réservé - OFPPT



PARTIE 1

Introduire Flutter

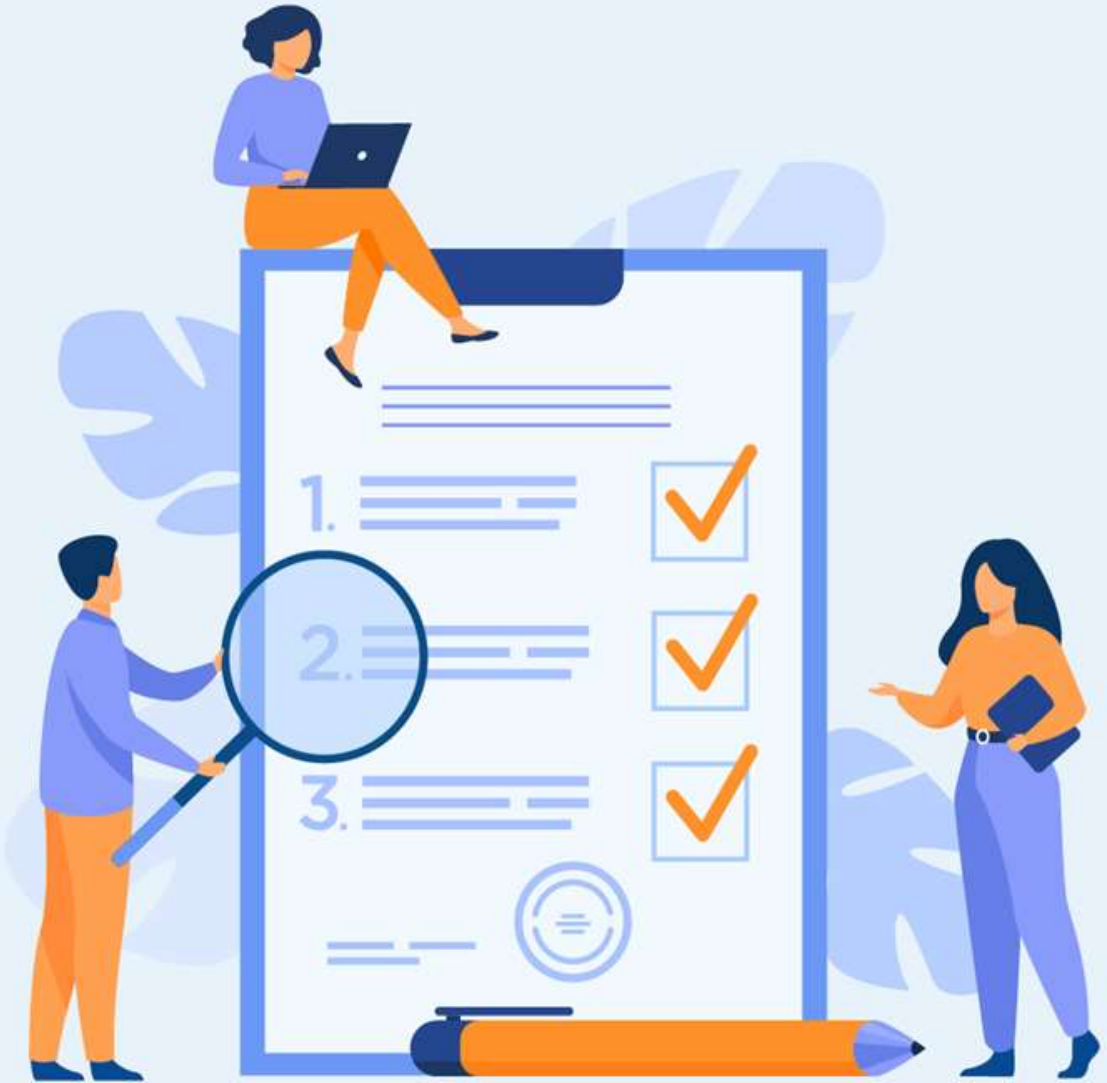
Dans ce module, vous allez :

- Découvrir l'IDE Android Studio
- Manipuler les plugins de base de Flutter

10 heures

Activité 1

Installation de l’environnement Flutter



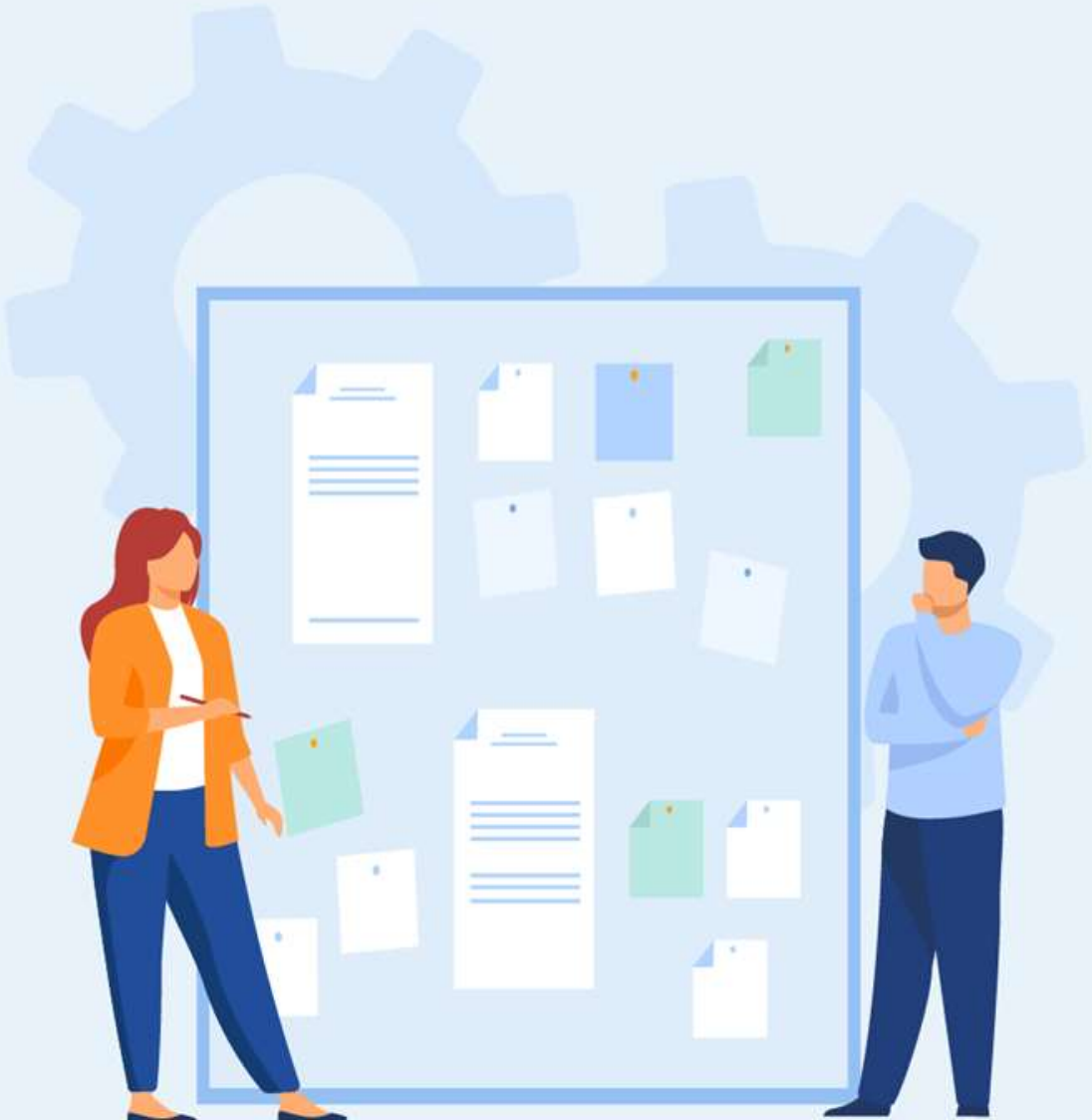
Compétences visées :

- Installer Android Studio
- Installer les extensions Dart/Flutter

Recommandations clés :

- Suivre les instructions du TP
- Préparer la machine pour réaliser le TP
- Exécuter les codes sur l’IDE

05 heures



CONSIGNES

Pour le formateur

- Demander aux apprenants de suivre les étapes de l'énnoncé et exécuter le code fourni
- Demander aux apprenants de réaliser le travail sur leurs machines

Pour l'apprenant

- Installer Android Studio et ajouter les extensions de Flutter/Dart
- Créer un projet du travail

Conditions de réalisation

- Support de résumé théorique
- DartPad ou Android Studio installé

Critère de réussite

- Le stagiaire est-il capable de :
  - Démarrer l'IDE pour coder en Dart/Flutter

Android Studio

1- Télécharger Android Studio depuis le site officiel : https://developer.android.com/studio



Le nom de la version d'Android Studio

Remarques

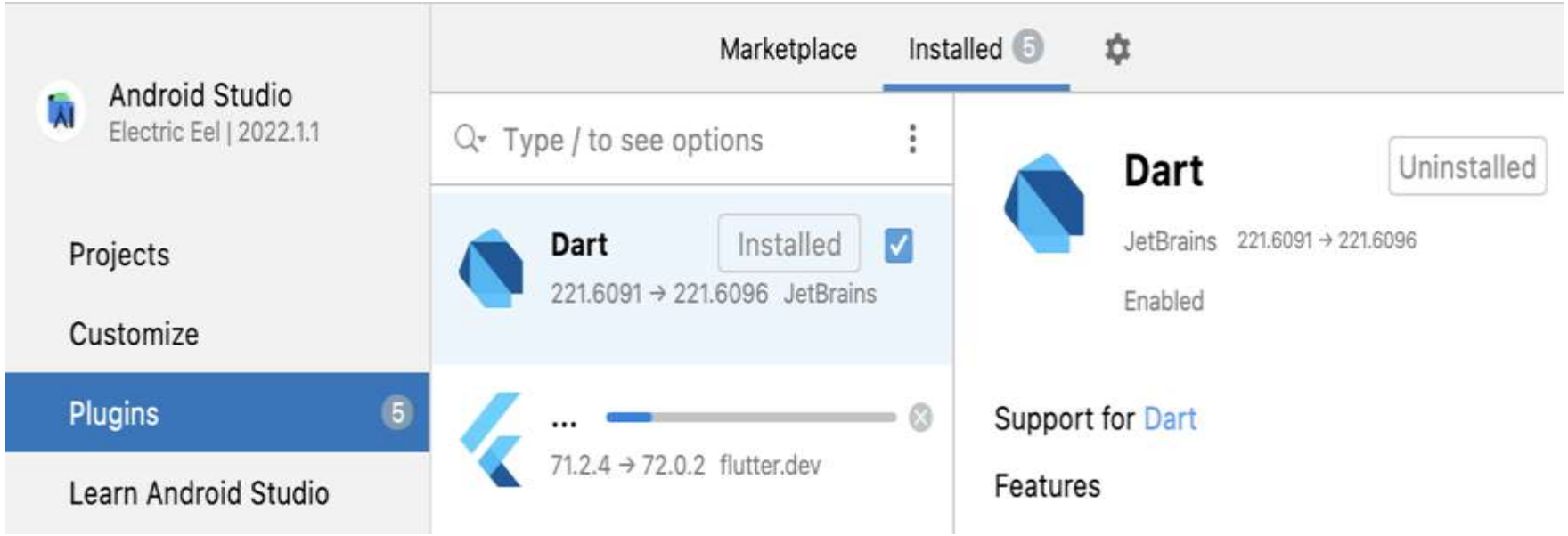
- Le nom d'Android Studio change par version.

PARTIE 1

Copyright - Tout droit réservé - OFPPT

Ajouter les extensions à Android Studio

• Installer les extensions « **Dart** » et « **Flutter** » pour préparer Android Studio à utiliser les fonctionnalités de Dart et FLutter.



Les plugins Dart et Flutter

Remarques

- Gardez toujours les plugins Dart et Flutter à jour.

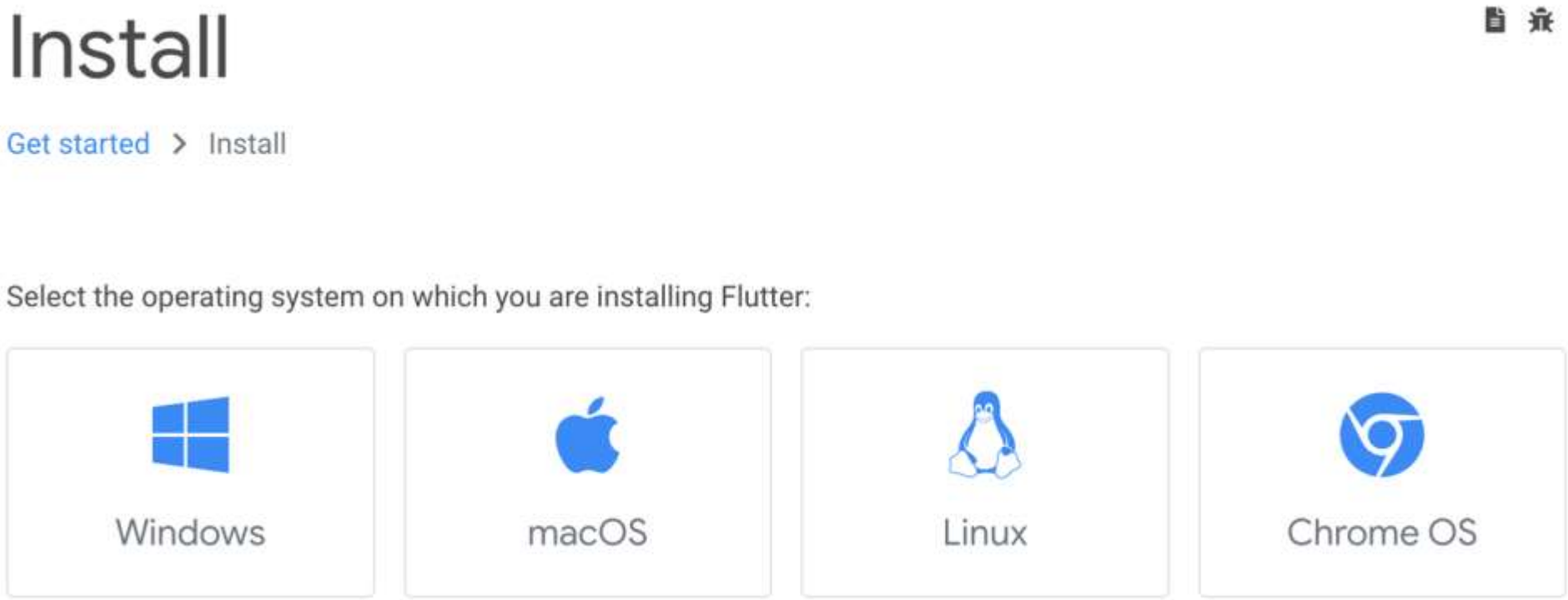
PARTIE 1

Copyright - Tout droit réservé - OFPPT

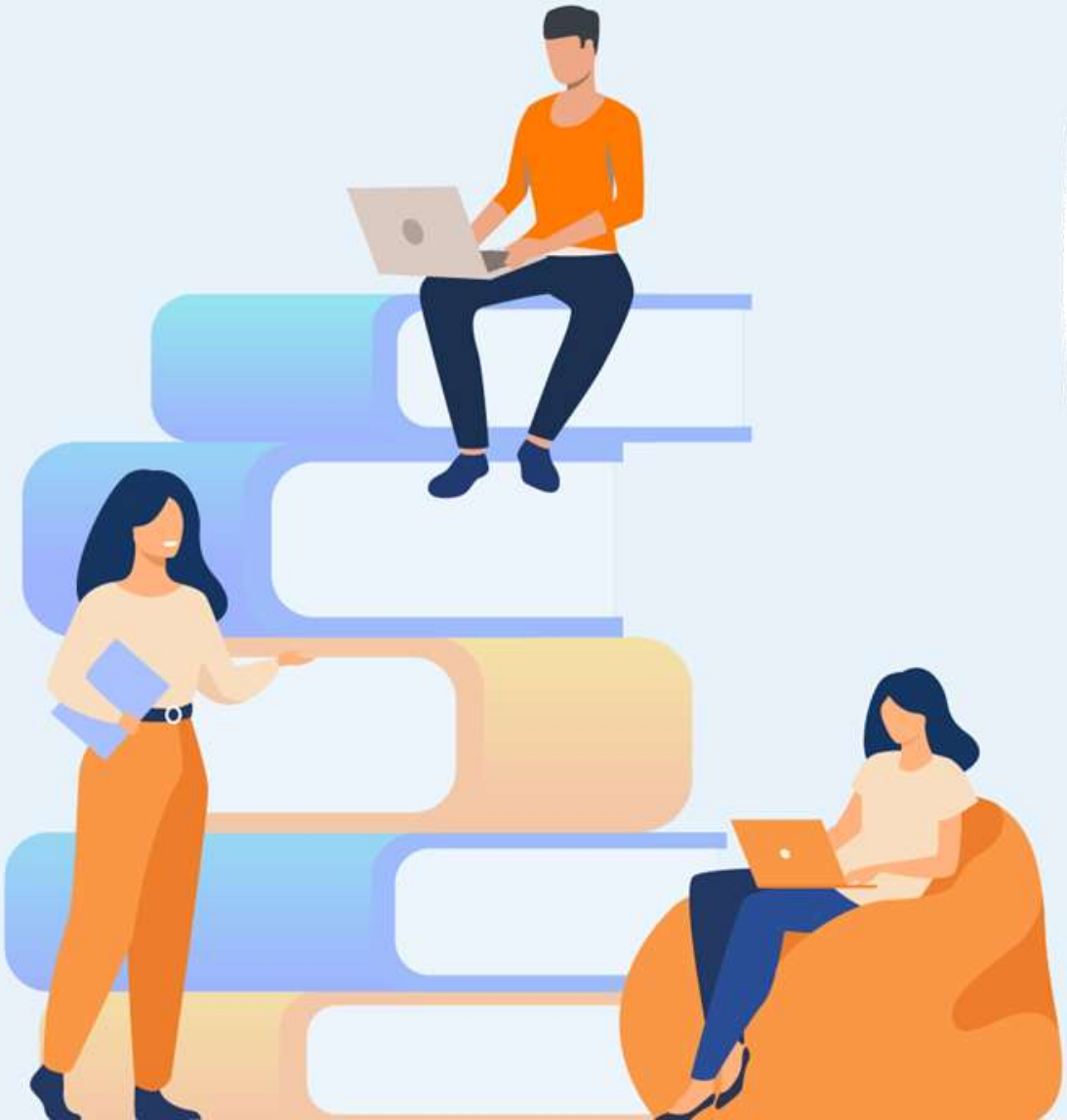
PARTIE 1

Flutter SDK

1- Télécharger Flutter SDK depuis le site officiel : https://docs.flutter.dev/get-started/install



Copyright - Tout droit réservé - OFPPT



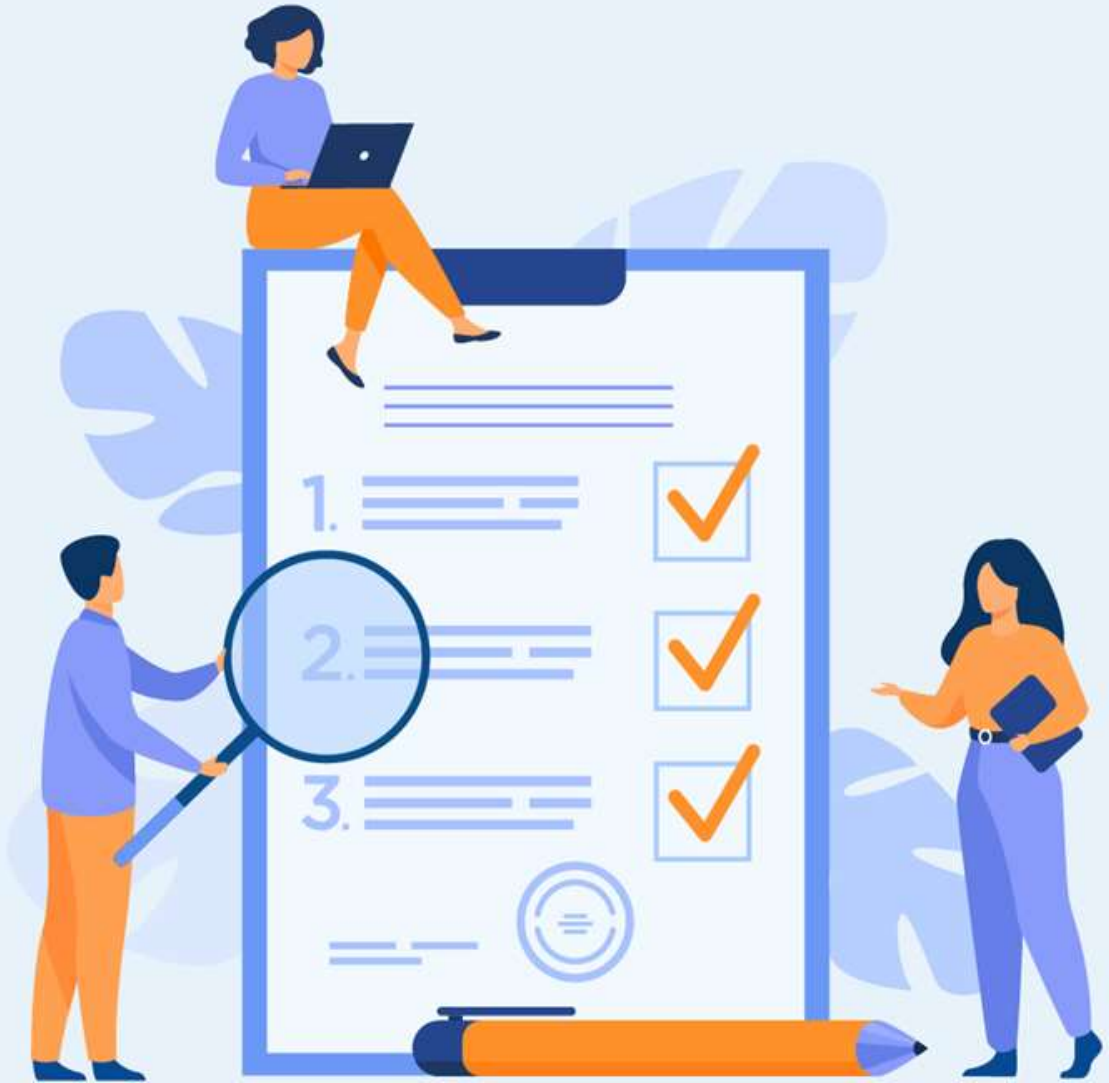
PARTIE 2

Découvrir Le Langage De Programmation Dart

Dans ce module, vous allez :

- Manipuler les bases de langage Dart
- Concevoir un code en langage Dart

10 heures



Activité 1

Programmer orientée objet avec Dart

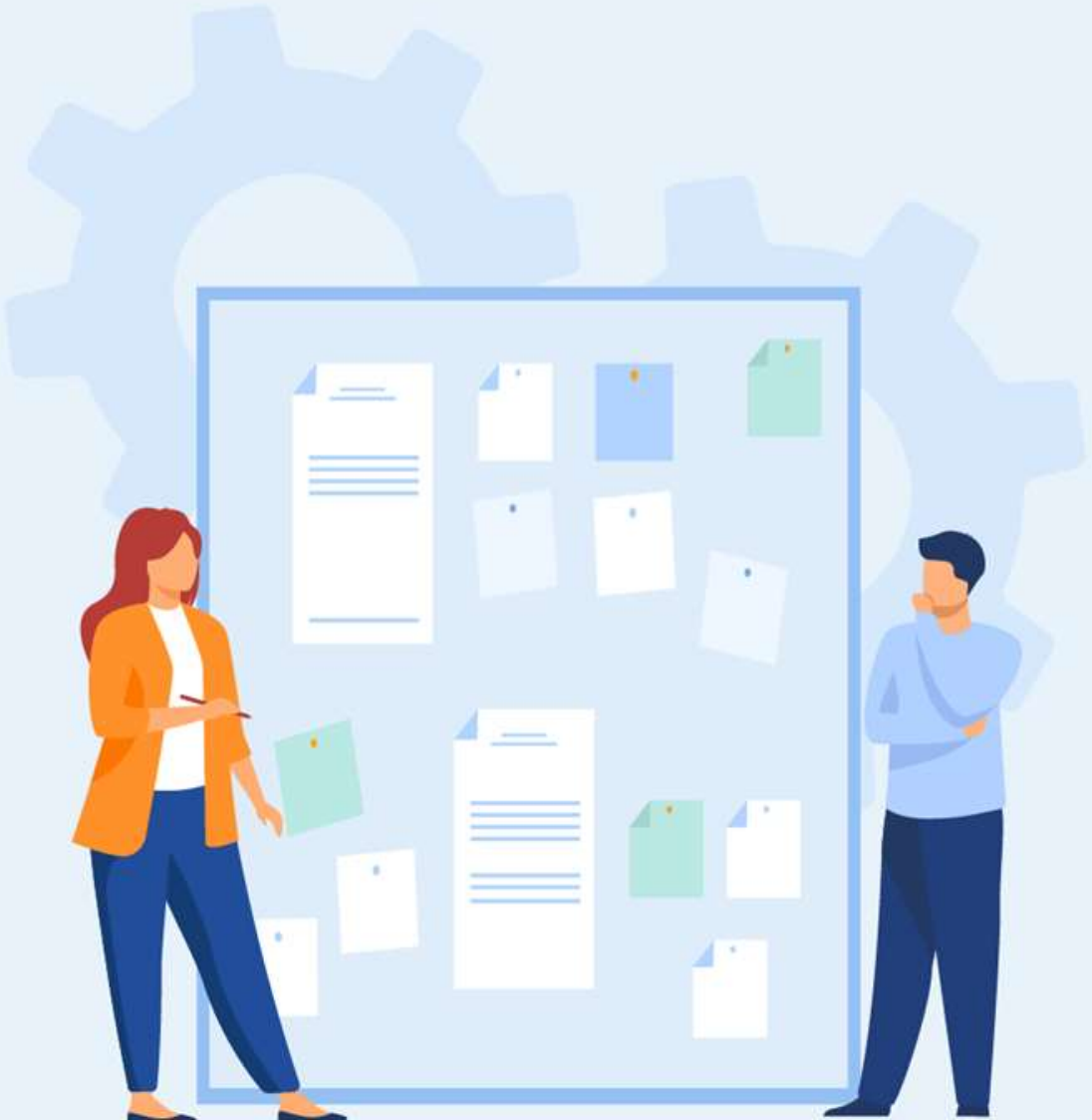
Compétences visées :

- Comprendre la POO
- Ecrire un code en langage Dart
- Exécuter le code dans DartPad/Android Studio

Recommandations clés :

- Suivre les instructions du TP
- Créer un nouveau projet pour exécuter les codes
- Utiliser le résumé théorique pour comprendre le code

10 heures



CONSIGNES

1. Pour le formateur

- Demander aux apprenants de suivre les étapes de l'énnoncé et exécuter le code fourni
- Demander aux apprenants de réaliser le travail sur leurs machines

2. Pour l'apprenant

- Mettre en place la POO en Dart
- Développer des fonctions en Dart

3. Conditions de réalisation :

- Suivre les indications présentées sur le document
- Plugin Dart installé sur Android Studio
- Emulateur installé

4. Critères de réussite :

- Le stagiaire est-il capable de :
  - Coder en Dart sur DartPad/Android Studio
  - Exécuter son code sur l'émulateur

POO Dart

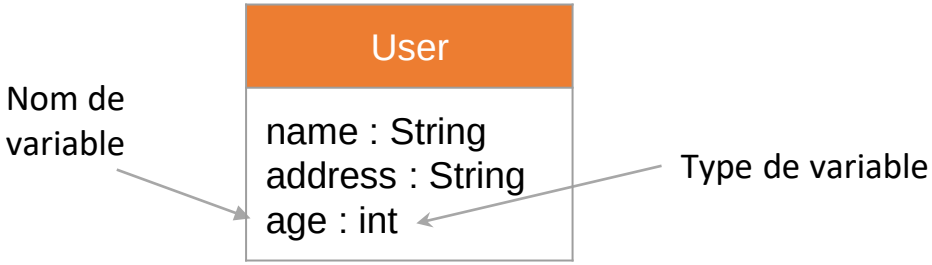
```
class User {
  String name = "";
  String address = "";
  int age = 0;
  User(String name, String address,
  int age) {
    this.name = name;
    this.address = address;
    this.age = age;
                              }
  void display()
{     print("name  =  ${name}  address
=${address} age =${age}");
 }
}
```

La classe User

```
void main() {

var user = User("test", "addresse", 35);
 user.display();
}
```

La fonction Main



La classe Point

```
name = test address = addresse age = 35
```

Exécution sur **DartPad**

Remarque

- Un éditeur de Dart en ligne prenant en charge les applications console, Web et Flutter
  Url : https://dartpad.dev/?

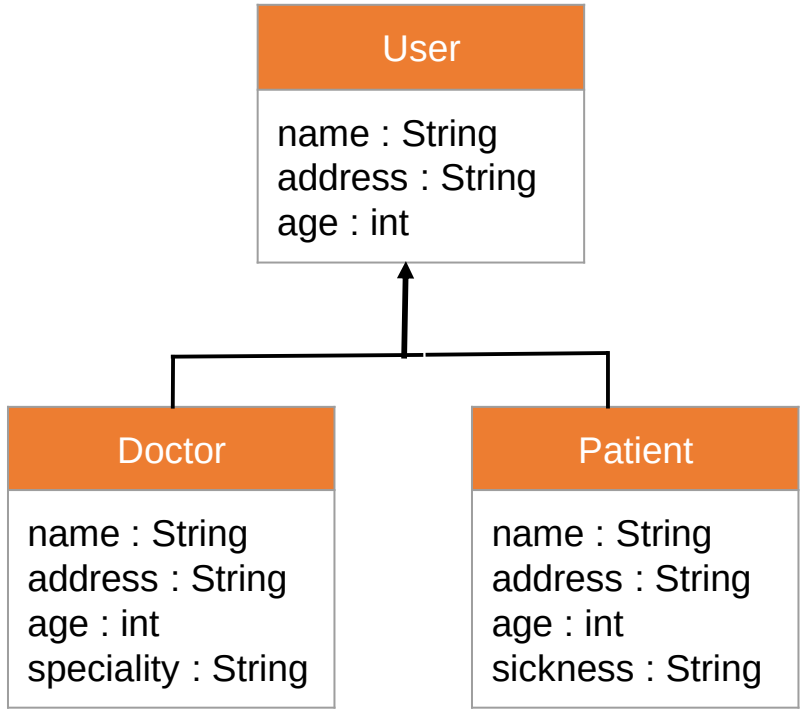
PARTIE 2

Copyright - Tout droit réservé - OFPPT

PARTIE 2

Héritage

```
class Doctor  extends User {
String speciality;
Doctor(String name, String address, int age,
this.speciality)
: super(name, address, age);
}
class Patient extends User
{
 String sickness;
 Patient(String name, String address, int age,
this.sickness)
: super(name, address, age);
}
void main() {
 Doctor doc = new Doctor("Doc", "Rabat", 36,"Ortho");
 Patient pat = new Patient("Pat", "Casa", 54, "headache");
 doc.display();
 pat.display();
}
```



Exécution en **DartPad**

```
name = Doc address = Rabat age = 36
name = Pat address = Casa age = 54
```

Copyright - Tout droit réservé - OFPPT

Gestion des erreurs

```dart
void main() {
  int a = 1, b = 0;
  try {
    int result = a ~/ b;
  } catch (ex) {
    print('cannot divide by 0');
  }
}
```

Exception

```dart
void main() {
 int a = 1, b = 0;
  try {
    int result = a ~/ b;
  } catch (ex) {
    print('cannot divide by 0');
  } finally {
    print('finally block executed');
  }
}
```

Exception avec le bloc **finally**

```
cannot divide by 0
```

```
cannot divide by 0
finally block executed
```

Exécution en DartPad

Remarque

- Un bloc **finally** est en général utilisé pour effectuer des nettoyages (fermer des fichiers, libérer des ressources...).

Copyright - Tout droit réservé - OFPPT



Activité 2

Utilisation des widgets de base

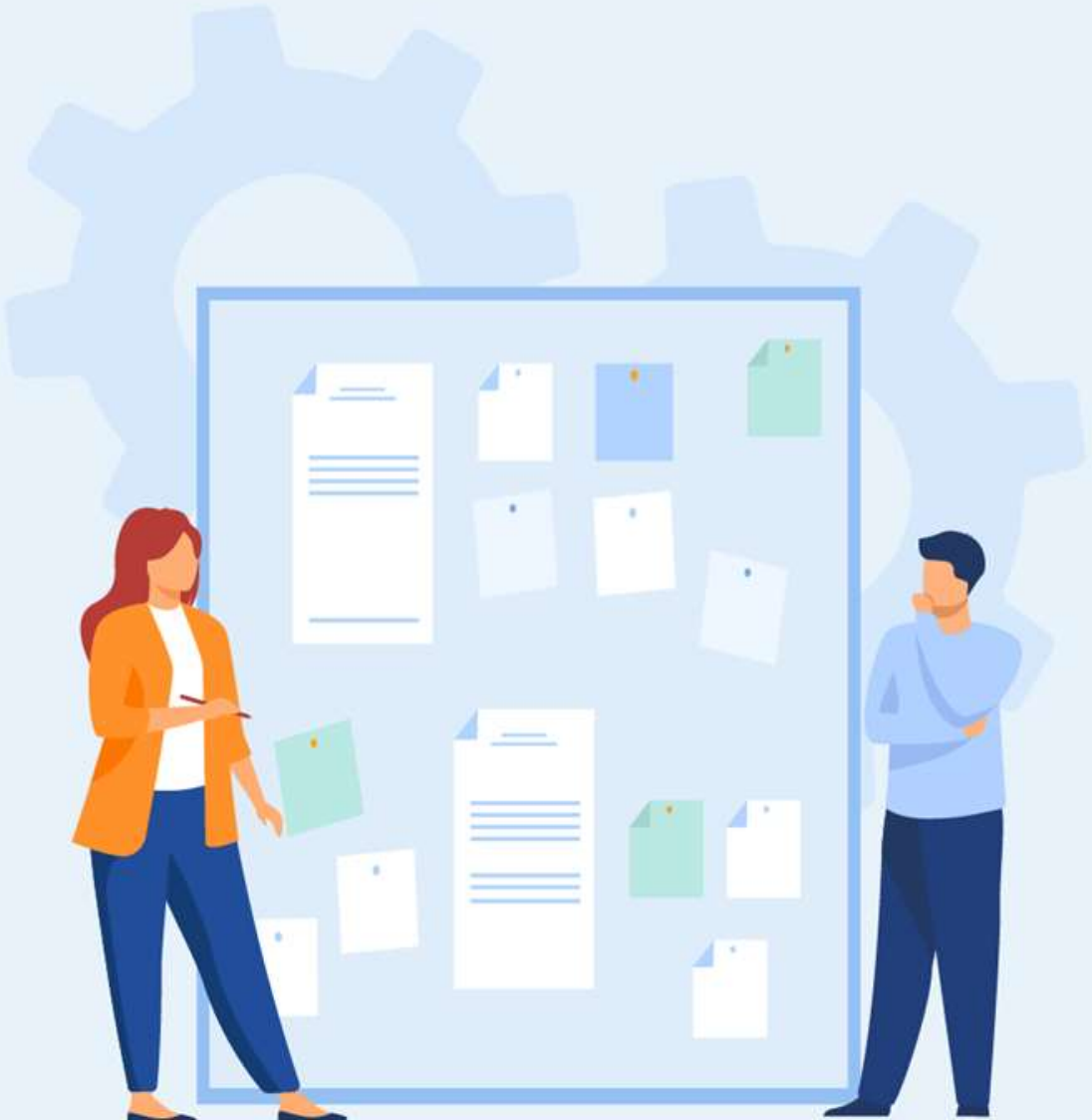
Compétences visées :

- Implémenter les widgets de base
- Comprendre le cycle de vie des widgets Stateless/Statefull
- Exécuter le code dans l'IDE Android Studio

Recommandations clés :

- Suivre l'énoncé du TP pour mettre en place le projet
- Utiliser le résumé théorique pour compléter le projet

15 heures



CONSIGNES

1. Pour le formateur

- Demander aux apprenants de suivre les étapes du résumé théorique du cours
- Demander aux apprenants d'exécuter les codes du résumé pratique du cours

2. Pour l'apprenant

- Mettre en place la widget Stateless/Statefull
- Concevoir le cycle de vie des widgets

3. Conditions de réalisation :

- Suivre les indications présentées sur le document
- Plugin Dart installé sur Android Studio
- Emulateur installé

4. Critères de réussite :

- Le stagiaire est-il capable de :
  - Implémenter une widget Stateless/Stateful
  - Exécuter son code sur l'émulateur

Activité 2
Utilisation des widgets de base

PARTIE 2

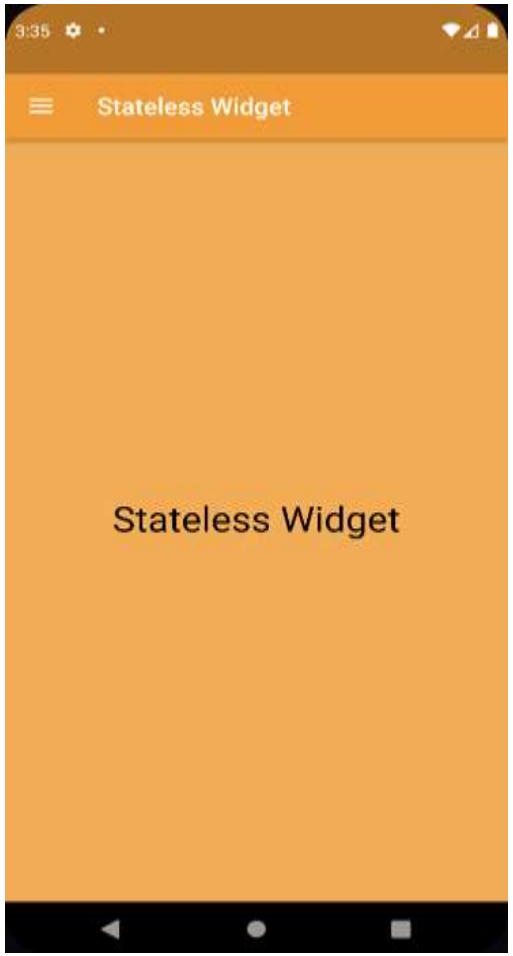
Stateless Widget

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

//This function triggers the build process
void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
 const MyApp({Key? key}) : super(key:
key);

@override
 Widget build(BuildContext context) {
  return MaterialApp(
    debugShowCheckedModeBanner:
false,
    home: Scaffold(
    backgroundColor: const Color.fromARGB(255,
230, 255, 201),
      appBar: AppBar(
       leading: const Icon(Icons.menu),
       backgroundColor: Colors.green,
       title: const Text(
        "Stateless Widget",
        textAlign: TextAlign.start,
       ),
      ), // AppBar
      body: const Center(
       child: Text(
        "Stateless Widget",
        style: TextStyle(color: Colors.black, fontSize:
30),
       ),
      ), // Container
    ), // Scaffold
  ); // MaterialApp
 }
}
```



Exécution

Copyright - Tout droit réservé - OFPPT

Activité 2
Utilisation des widgets de base

PARTIE 2

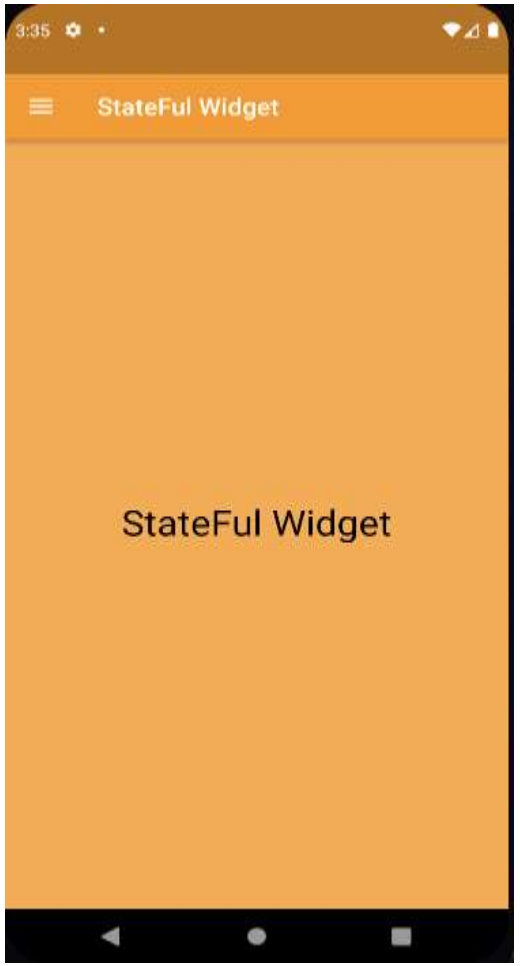
Stateful Widget

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

//This function triggers the build process
void main() => runApp(const MyApp());

// StatefulWidget
class MyApp extends StatefulWidget {
 const MyApp({Key? key}) : super(key: key);

 @override
// ignore: library_private_types_in_public_api
 _MyAppState createState() => _MyAppState();
}
class _MyAppState extends State<MyApp> {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     debugShowCheckedModeBanner: false,
     home: Scaffold(
       backgroundColor: Colors.orangeAccent,
       appBar: AppBar(
         leading: const Icon(Icons.menu),
         backgroundColor: Colors.orange,
         title: const Text(
           "StateFul Widget",
           textAlign: TextAlign.start,
         ),
       ), // AppBar
       body: const Center(
         child: Text(
           "StateFul Widget",
           style: TextStyle(color: Colors.black, fontSize: 30),
         ),
       ), // Container
     ), // Scaffold
   ); // MaterialApp
 }
}
```



Exécution

Copyright - Tout droit réservé - OFPPT

Activité 2
Utilisation des widgets de base

PARTIE 2

Bottom Navigation Bar Widget

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

  @override
  State<MyStatefulWidget> createState() =>
_MyStatefulWidgetState();
}
class _MyStatefulWidgetState extends
State<MyStatefulWidget> {
  int _selectedIndex = 0;
  static const TextStyle optionStyle =
      TextStyle(fontSize: 30, fontWeight:
FontWeight.bold);
  static const List<Widget> _widgetOptions =
<Widget>[
    Text(
      'Index 0: Home',
      style: optionStyle,
    ),
  Text(
      'Index 1: Business',
      style: optionStyle,),
    Text( 'Index 2: School',
      style: optionStyle,
    ),
  ];
void _onItemTapped(int index) {setState(() {
      _selectedIndex = index; });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('BottomNavigationBar Sample'),
      ),
      body: Center(
        child: _widgetOptions.elementAt(_selectedIndex),
      ),
      bottomNavigationBar: BottomNavigationBar(
        items: const
<BottomNavigationBarItem>[
          BottomNavigationBarItem(
            icon: Icon(Icons.home),
            label: 'Home',  ),
          BottomNavigationBarItem(
            icon: Icon(Icons.business),
            label: 'Business',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.school),
            label: 'School',  ),      ],
        currentIndex: _selectedIndex,
        selectedItemColor: Colors.amber[800],
        onTap: _onItemTapped,  ),);
    }
}
```



Exécution

Copyright - Tout droit réservé - OFPPT

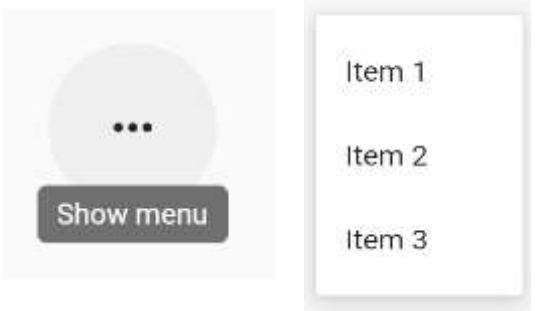
PARTIE 2

Popup Menu Button Widget

```
class PopupMenuExample extends
StatefulWidget {
  const PopupMenuExample({super.key});
 @override
  State<PopupMenuExample> createState() =>
_PopupMenuExampleState();
}

class _PopupMenuExampleState extends
State<PopupMenuExample> {
  SampleItem? selectedMenu;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const
Text('PopupMenuButton')),
      body: Center(
        child: PopupMenuButton<SampleItem>(
          initialValue: selectedMenu,
          // Callback that sets the selected popup
menu item.
          onSelected: (SampleItem item) {
            setState(() {
              selectedMenu = item;
            });
          },
          itemBuilder: (BuildContext context) =>
<PopupMenuEntry<SampleItem>>[
            const PopupMenuItem<SampleItem>(
              value: SampleItem.itemOne,
              child: Text('Item 1'),
            ),
            const PopupMenuItem<SampleItem>(
              value: SampleItem.itemTwo,
              child: Text('Item 2'),  ),
            const
PopupMenuItem<SampleItem>(
              value:
SampleItem.itemThree,
              child: Text('Item 3'),
            ),
          ],
        ),
      ),
    );
  }
}
```



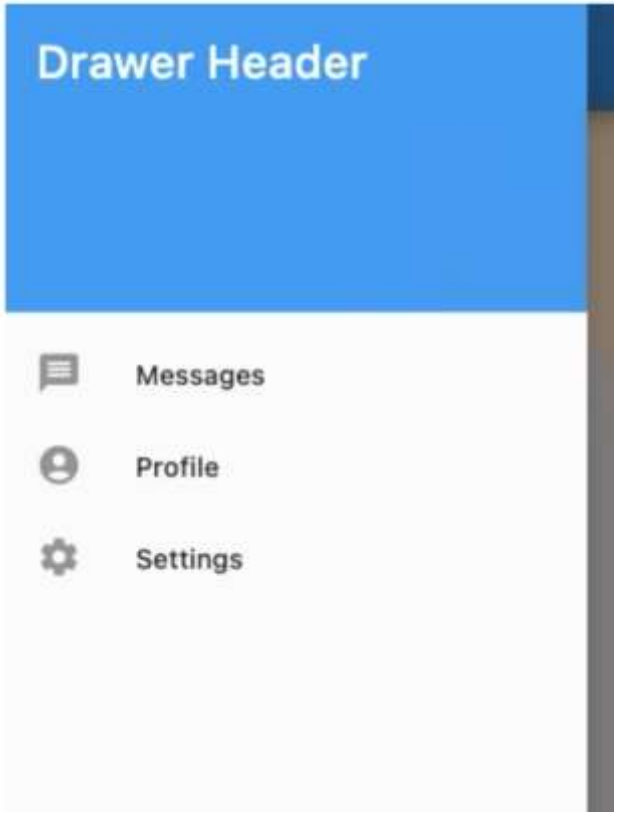
Exécution

Copyright - Tout droit réservé - OFPPT

Drawer Widget

```
Scaffold(
  appBar: AppBar(
    title: const Text('Drawer Demo'),
  ),
  drawer: Drawer(
    child: ListView(
      padding: EdgeInsets.zero,
      children: const <Widget>[
        DrawerHeader(
          decoration: BoxDecoration(
            color: Colors.blue,
          ),
          child: Text(
            'Drawer Header',
            style: TextStyle(
              color: Colors.white,
              fontSize: 24,
            ),),),),
```

```
ListTile(
          leading: Icon(Icons.message),
          title: Text('Messages'),
        ),
        ListTile(
          leading: Icon(Icons.account_circle),
          title: Text('Profile'),
        ),
        ListTile(
          leading: Icon(Icons.settings),
          title: Text('Settings'),
        ),
      ],
    ),
  ),
)
```



Exécution

Copyright - Tout droit réservé - OFPPT

PARTIE 2

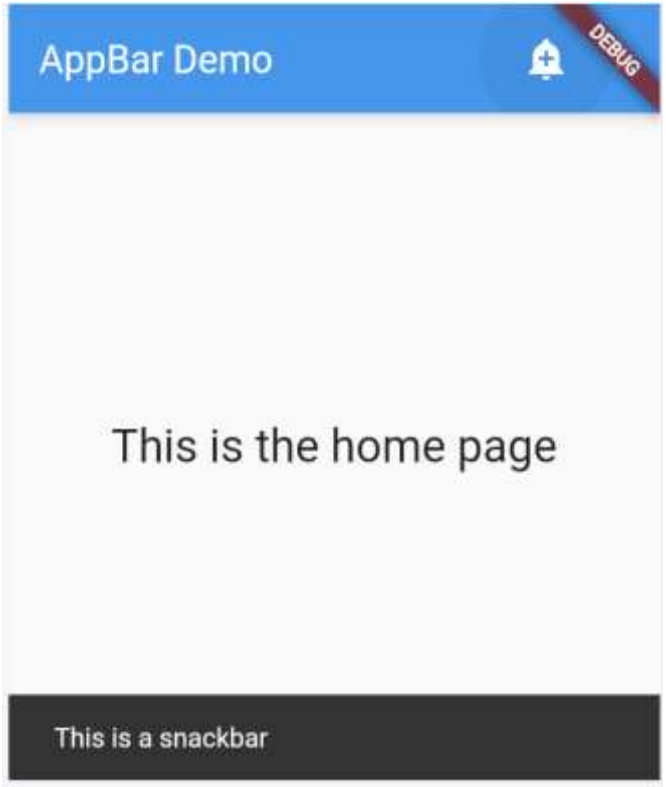
AppBar Widget

```
class AppBarExample extends
StatelessWidget {
  const AppBarExample({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('AppBar Demo'),
        actions: <Widget>[
          IconButton(
            icon: const Icon(Icons.add_alert),
            tooltip: 'Show Snackbar',
            onPressed: () {

ScaffoldMessenger.of(context).showSnack
Bar( const SnackBar(content: Text('This is
a snackbar')));
            },),
```

```
          IconButton(
            icon: const Icon(Icons.navigate_next),
            tooltip: 'Go to the next page',
            onPressed: () {
              Navigator.push(context,
MaterialPageRoute<void>(
                builder: (BuildContext context) {
                  return Scaffold(
                    appBar: AppBar(
                      title: const Text('Next page'),
                    ),
                    body: const Center(
                      child: Text(
                        'This is the next page',
                        style: TextStyle(fontSize: 24),
                      ), ), ); }, )); }, ),  ], ),
        body: const Center(
          child: Text(
            'This is the home page',
            style: TextStyle(fontSize: 24),
          ),),);}}
```



Exécution

Copyright - Tout droit réservé - OFPPT

Activité 2
Utilisation des widgets de base

radio button Widget

```
class _MyStatefulWidgetState extends
State<MyStatefulWidget> {
  SingingCharacter? _character =
SingingCharacter.lafayette;

  @override
  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        ListTile(
          title: const Text('Lafayette'),
          leading: Radio<SingingCharacter>(
            value: SingingCharacter.lafayette,
            groupValue: _character,
            onChanged: (SingingCharacter? value) {
              setState(() {
                _character = value;
              });
            },),),
```

```
 ListTile(
          title: const Text('Thomas
Jefferson'),
          leading: Radio<SingingCharacter>(
            value: SingingCharacter.jefferson,
            groupValue: _character,
            onChanged: (SingingCharacter?
value) {
              setState(() {
                _character = value;
              });
            },
          ),
        ),
      ],
    );
  }
}
```



Exécution

PARTIE 2

Copyright - Tout droit réservé - OFPPT

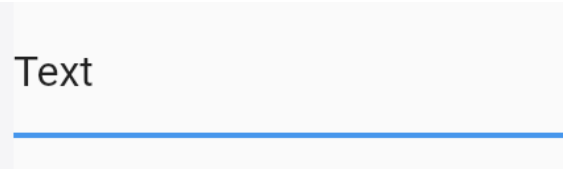
PARTIE 2

TextField Widget

```
class _MyStatefulWidgetState extends
State<MyStatefulWidget> {
  late TextEditingController _controller;

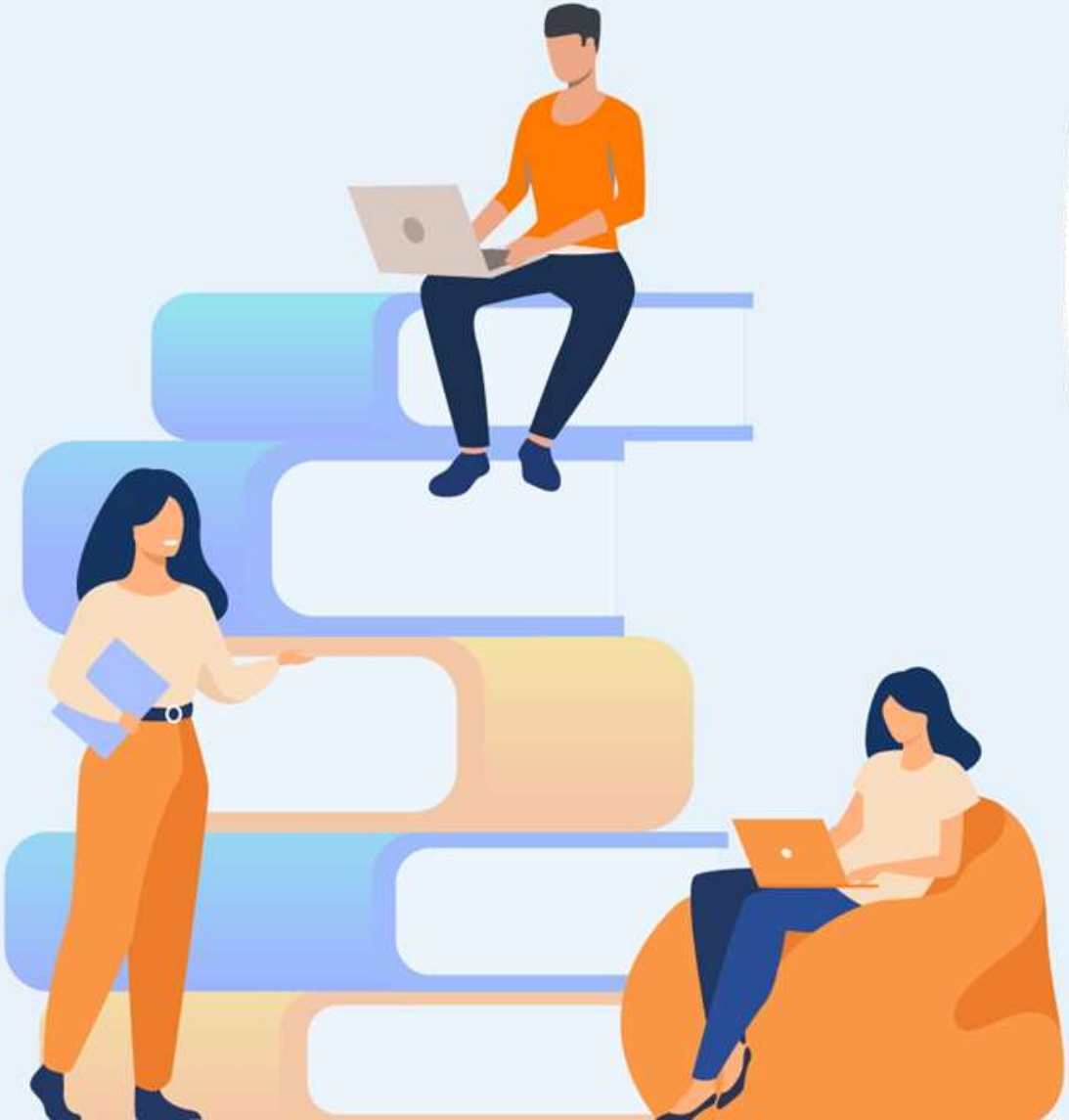
  @override
  void initState() {
    super.initState();
    _controller = TextEditingController();
  }

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }
 @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: TextField(
          controller: _controller,
          onSubmitted: (String value) async {
            await showDialog<void>(
              context: context,
              builder: (BuildContext context) {
                return AlertDialog(
                  title: const Text('Thanks!'),
                  content: Text(
                 'You typed "$value", which has length
${value.characters.length}.'),
                  actions: <Widget>[
                    TextButton(
                      onPressed: () {
                      Navigator.pop(context);
                      },
                      child: const Text('OK'),
                    ),
                  ],
                );
              },
            );
          },),),); }}
```



Exécution

Copyright - Tout droit réservé - OFPPT



PARTIE 3

Manipuler les données

Dans ce module, vous allez :

- Apprendre à utiliser facilement la bibliothèque dart:convert
- Apprendre à manipuler aisément les opérations CRUD et les interactions avec la base de données

10 heures

Activité 1

Convert et parser du JSON



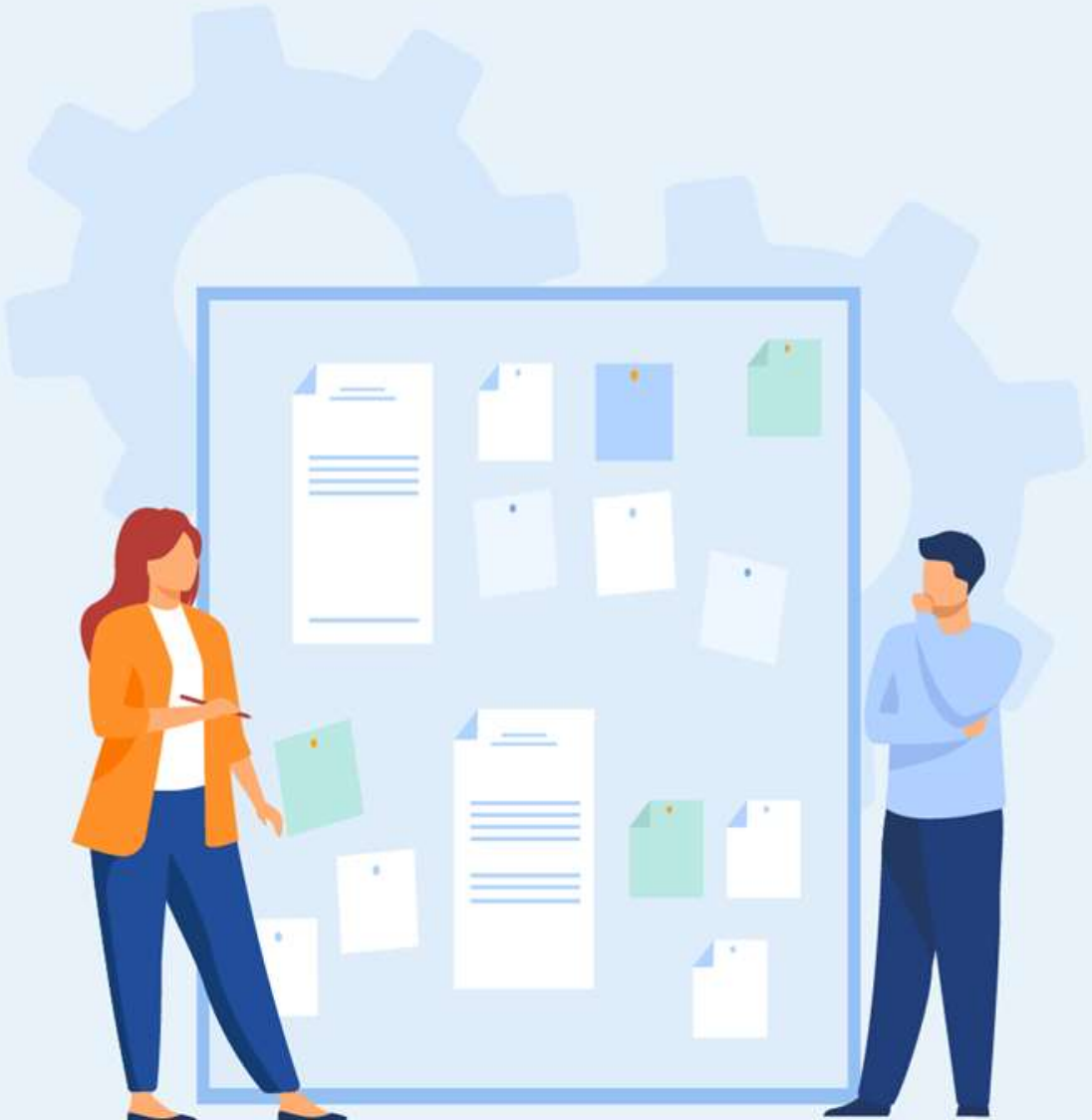
Compétences visées :

- Utilisation correcte du package Http
- Connexion à un serveur externe
- Parsing du JSON

Recommandations clés :

Révision générale du résumé théorique

05 heures



CONSIGNES

1. Pour le formateur

- Demander aux apprenants de suivre les étapes du résumé théorique du cours
- Demander aux apprenants d'exécuter le code sur leurs machines

2. Pour l’apprenant

- Mettre en place le Parsing JSON
- Affichage des données
- Définir l'opération du Parsing du JSON
- Mapping des données

3. Conditions de réalisation :

- Appel réseau : connexion à un serveur distant
- Flux stocké en local

4. Critères de réussite :

- Le stagiaire est-il capable de :
  - Parser un flux de JSON
  - Exécuter son code sur l’émulateur

Activité 1
Convert et Parser JSON

PARTIE 3

Parser du JSON

On veut parser le flux de JSON disponible sur ce

lien : https://jsonplaceholder.typicode.com/albums/1

```
{
  userId: 1,
  id: 1,
  title: "quidem molestiae enim"
}
```

Flux JSON

On commence par mettre en place la dépendance http package dans le fichier pubspec.yaml, ensuite on ajoute la permission d'Internet dans le fichier **AndroidManifest.xml**

<uses-permission android:name="android.permission.INTERNET" />.

Une fois que notre projet est prêt, on commence l'implémentation de la requête qui permet de parser JSON y compris les classes models (Album).

```
class Album {
  final int userId;
  final int id;
  final String title;

  const Album({
    required this.userId,
    required this.id,
    required this.title,
  });

  factory Album.fromJson(Map<String,
dynamic> json) {
    return Album(
      userId: json['userId'],
      id: json['id'],
      title: json['title'],
    );
  }
}
```

Classe Album

Copyright - Tout droit réservé - OFPPT

Activité 1
Convert et Parser JSON

Parser du JSON

```
import 'dart:async';
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

Future<Album> fetchAlbum() async {
  final response = await http

.get(Uri.parse('https://jsonplaceholder.typicode.com/albums/1'));
  if (response.statusCode == 200) {
    return
Album.fromJson(jsonDecode(response.body));
  } else {
    throw Exception('Failed to load album');
  }
}
```

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

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

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

StatefulWidget : le widget avec état

jsonDecode : la méthode qui permet de décoder le flux reçu.

PARTIE 3

Copyright - Tout droit réservé - OFPPT

Activité 1
Convert et Parser JSON

Parser du JSON

```
class _MyAppState extends State<MyApp> {
  late Future<Album> futureAlbum;

  @override
  void initState() {
    super.initState();
    futureAlbum = fetchAlbum();
  }
```

L'exécution

quidem molestiae enim

```
@override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Fetch Data Example',
     theme: ThemeData(
       primarySwatch: Colors.blue,
     ),
     home: Scaffold(
       appBar: AppBar(title: const Text('Fetch Data Example'),),
       body: Center(
         child: FutureBuilder<Album>(
           future: futureAlbum,
           builder: (context, snapshot) {
             return Text(snapshot.data!.title);
           },
         ),
       ),
     ),
   );
 }
}
```

PARTIE 3

Copyright - Tout droit réservé - OFPPT

Activité 2

Interactions avec la base de données



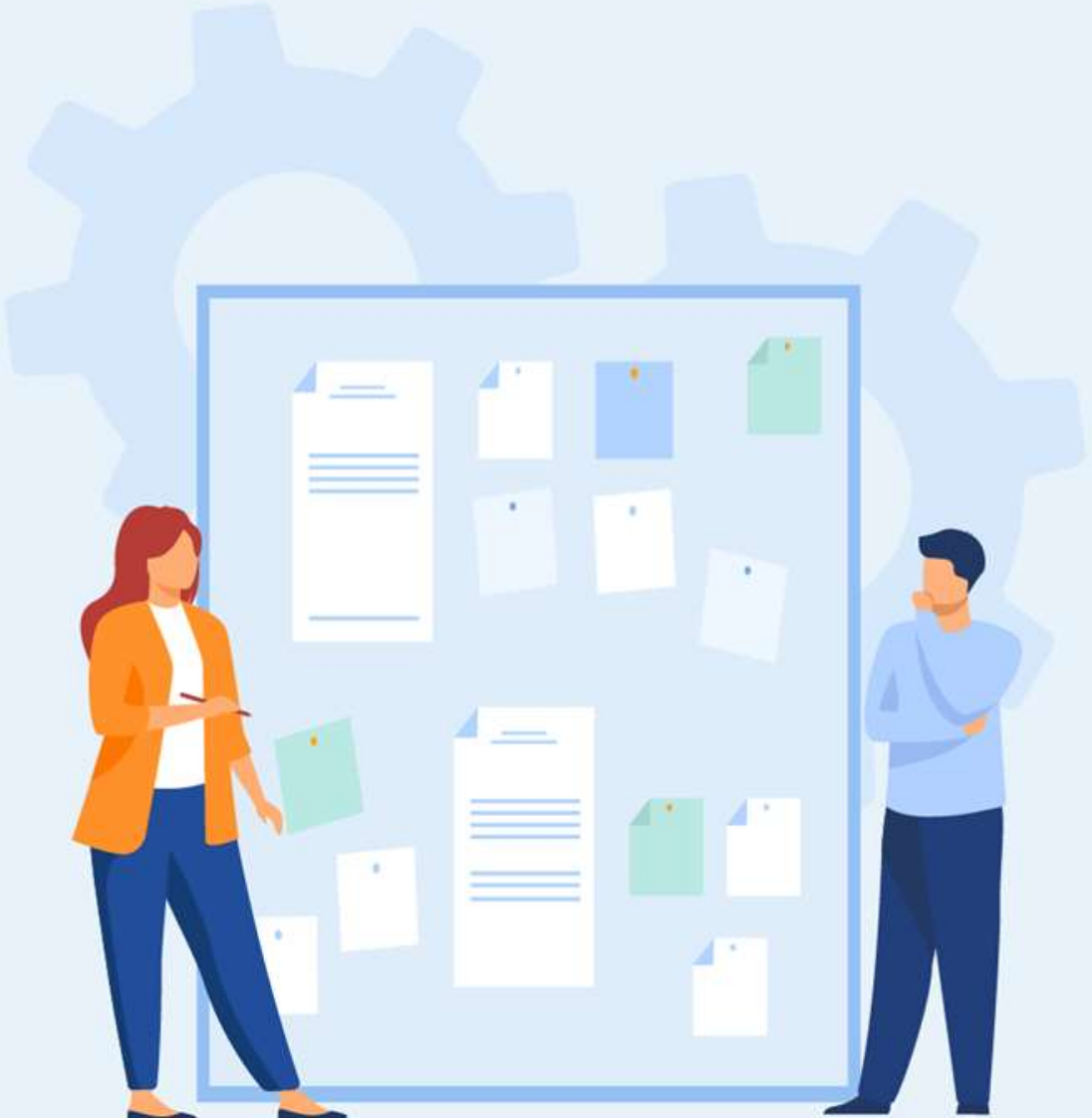
Compétences visées :

- Manipulation des opérations CRUD
- Interactions avec la base de données

Recommandations clés :

Révision générale du résumé théorique

10 heures



CONSIGNES

1. Pour le formateur

- Demander aux apprenants de définir les opérations du CRUD
- Exécuter le code pour interroger la base de données

2. Pour l'apprenant

- Mettre en place les opérations du CRUD

3. Conditions de réalisation :

- Appel réseau : connexion à un serveur distant
- Flux stocké en local
- Mettre en place la base de données

4. Critères de réussite :

- Le stagiaire est-il capable de :
  - Parser un flux de JSON
  - Stocker les données dans la BDD
  - Exécuter son code sur l'émulateur

PARTIE 3

CRUD : sqflite

Les applications Flutter peuvent utiliser les bases de données SQLite via le plugin **sqflite** disponible sur **pub.dev**. Cette recette montre les bases de l'utilisation **sqflite** pour insérer, lire, mettre à jour et supprimer des données.

1. Ajouter les dépendances

```
dependencies:
  flutter:
    sdk: flutter
  sqflite:
  path:
```

Importer les packages dans le fichier dans lequel vous allez travailler.

```
import 'dart:async';
import 'package:flutter/widgets.dart';
import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';
```

2. Définir le modèle de données User

```
class User {
  final int id;
  final String name;
  final int age;

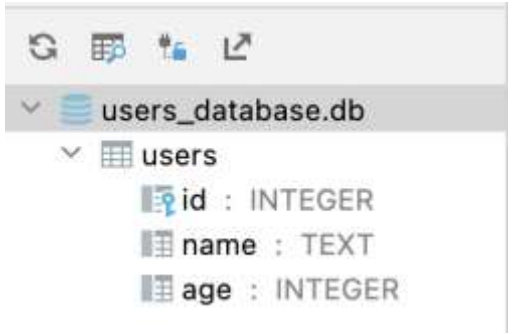
  const User({
    required this.id,
    required this.name,
    required this.age,
  });
}
```

3. Ouvrez la base de données

```
WidgetsFlutterBinding.ensureInitialized();

final database = openDatabase(

  join(await getDatabasesPath(),
'users_database.db'),
);
```



La création de la base de données

Remarque

- Pour utiliser le mot-clé **await**, le code doit être placé à l'intérieur d'une async fonction. Vous devez placer toutes les fonctions de table suivantes à l'intérieur de **void main() async {}**.

Copyright - Tout droit réservé - OFPPT

PARTIE 3

CRUD : sqflite

4. Créez le tableau Users

```
final database = openDatabase(

  join(await getDatabasesPath(), 'users_database.db'),

  onCreate: (db, version) {
    return db.execute(
      'CREATE TABLE users(id INTEGER PRIMARY KEY, name TEXT, age INTEGER)',
    );},
  version: 1,
);
```



L'exécution de la méthode **insertUser**()

5. Insérez un User dans la base de données

- Convertissez l'User en un Map

```
Map<String, dynamic> toMap() {
  return {
    'id': id,
    'name': name,
    'age': age,
  };
}

@override
String toString() {
  return 'User{id: $id, name: $name, age: $age}';
}
```

- Utilisez la méthode insert() pour stocker le Map dans la table Users.

```
Future<void> insertUser(User user) async {
  final db = await database;

  await db.insert(
    'users',
    user.toMap(),
    conflictAlgorithm: ConflictAlgorithm.replace,
  );
}
```

```
// Create a User and add it to the users table
var fido = const User(
  id: 0,
  name: 'Fido',
  age: 35,
);

await insertUser(fido);
```

Copyright - Tout droit réservé - OFPPT

CRUD : sqflite

6. Récupérer la liste des Users

```
Future<List<User>> users() async {
  final db = await database;
  final List<Map<String, dynamic>> maps = 
await db.query('users');

return List.generate(maps.length, (i) {

    return User(
      id: maps[i]['id'],
      name: maps[i]['name'],
      age: maps[i]['age'],
    );
  });
}
```

```
// Now, use the method above to retrieve all the Users.
print(await Users()); // Prints a list that include Fido.
```

7. Mettre à jour un User dans la base de données.

```
Future<void> updateUser(User user) async {
  final db = await database;

  await db.update(
    'users',
    user.toMap(),
    where: 'id = ?',
    whereArgs: [user.id],
  );}
updateUser(User (
 id: 1,
 name: "fido",
 age: 37
));
```



L'exécution de la méthode **UpdateUser()**

Remarque

- Utilisez toujours `whereArgs` pour passer des arguments à une `where` instruction. Cela permet de se prémunir contre les attaques par injection SQL.
- N'utilisez pas d'interpolation de chaîne, telle que `where: "id = ${user.id}"`!

PARTIE 3

Copyright - Tout droit réservé - OFPPT

CRUD : sqflite

8. Supprimer un User de la base de données

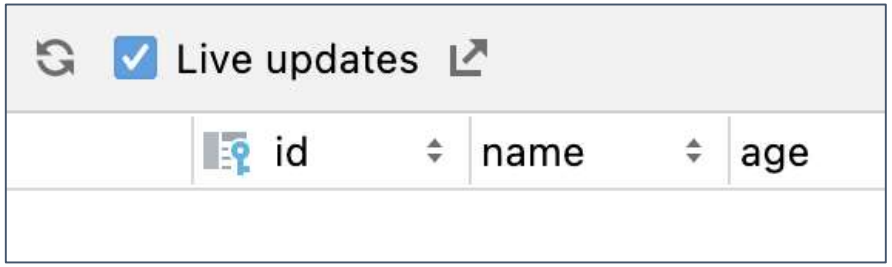
```
Future<void> deleteUser(int id) async {
  // Get a reference to the database.
  final db = await database;

  // Remove the User from the database.
  await db.delete(
    'users',

    where: 'id = ?',

    whereArgs: [id],
  );
}
```

La méthode qui permet de supprimer un **élément** de la base de données



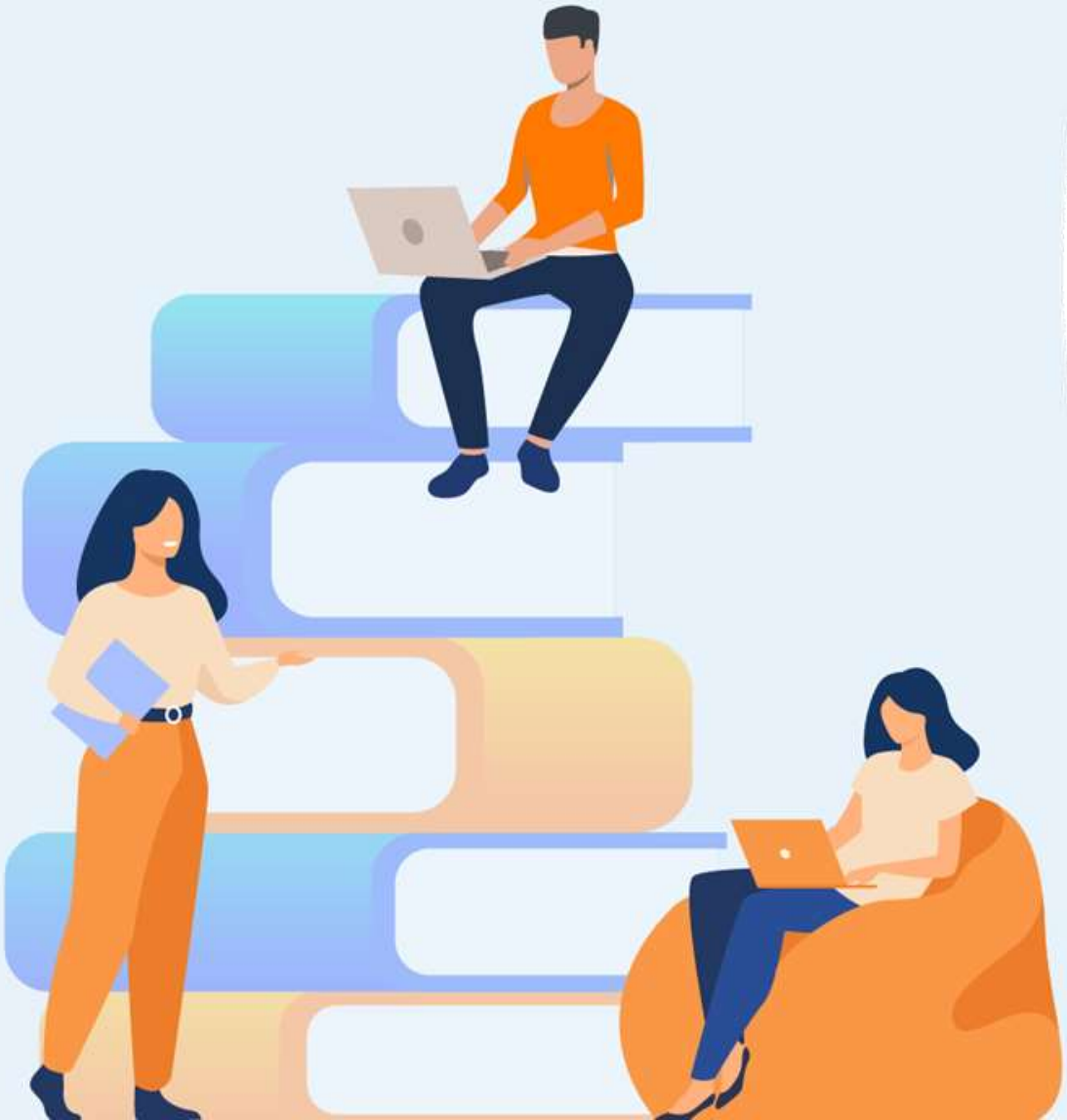
L'exécution de la méthode **deleteUser()**

Remarque

- Utilisez toujours un message d'alerte avant la suppression d'un élément pour demander la confirmation de l'utilisateur.

PARTIE 3

Copyright - Tout droit réservé - OFPPT



PARTIE 4

Maîtriser La Navigation Et Le Routing

Dans ce module, vous allez :

- Apprendre à créer facilement une nouvelle route
- Apprendre à passer correctement les données d’un écran à un autre
- Apprendre à naviguer de manière simple entre les routes

06 heures



Activité 1

Création facile d’une nouvelle route

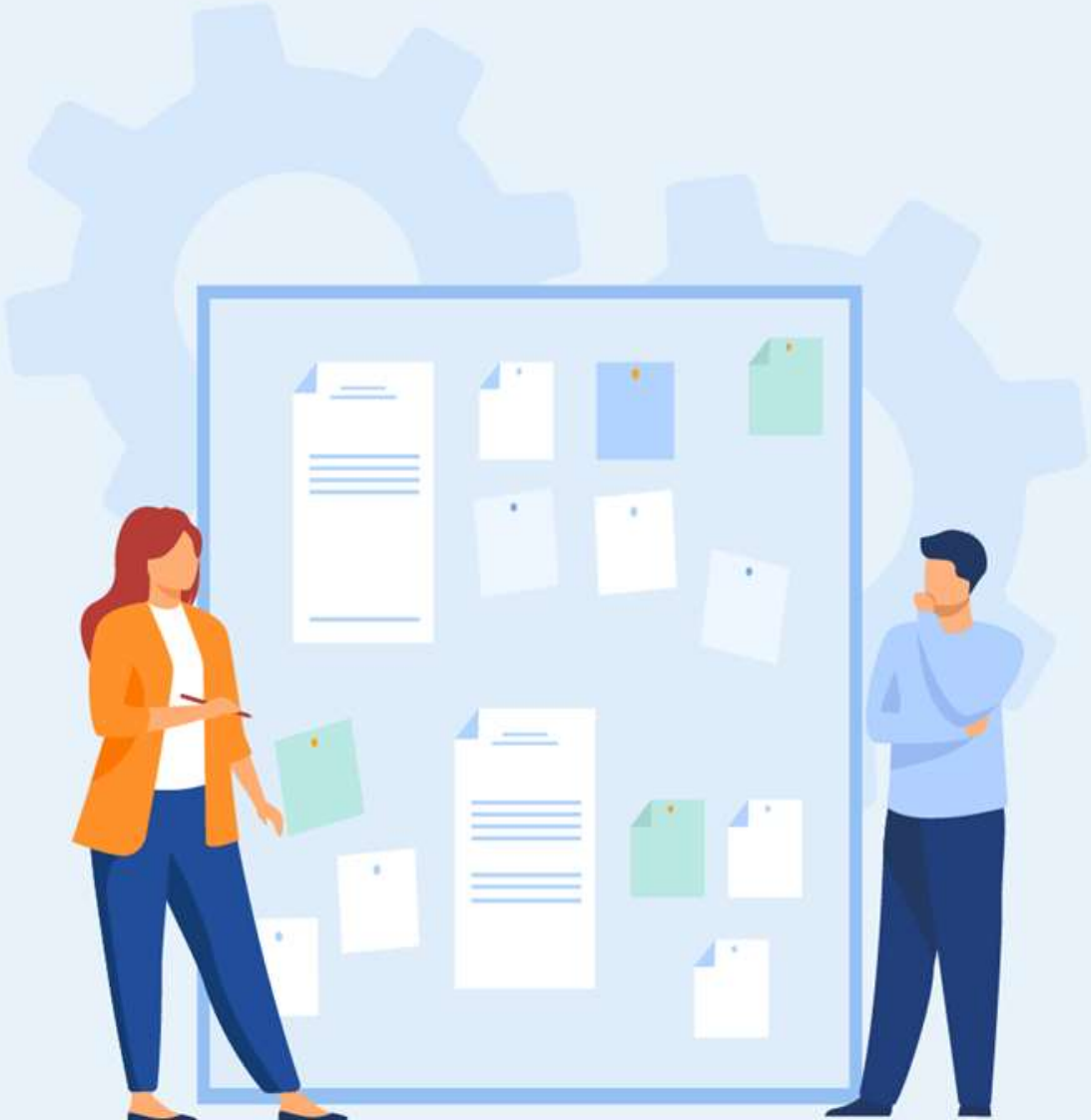
Compétences visées :

- Créer une route qui lie les 2 écrans
- Assurer le retour en arrière

Recommandations clés :

- Lire attentivement les consignes du résumé théorique

05 heures



CONSIGNES

1. **Pour le formateur**

- Rappeler la méthode qui permet de créer une nouvelle route
- Demander aux apprenants de s’assurer que les 2 écrans sont créés

2. **Pour l’apprenant**

- Créer les 2 écrans de départ et d’arrivée
- Créer la route qui fait la liaison entre les 2 écrans

3. **Conditions de réalisation :**

- Support de résumé théorique accompagnant
- Installation d’extension de Flutter/Dart

4. **Critères de réussite :**

- Le stagiaire est-il capable de :
  - Mettre en place des écrans
  - Créer une nouvelle route

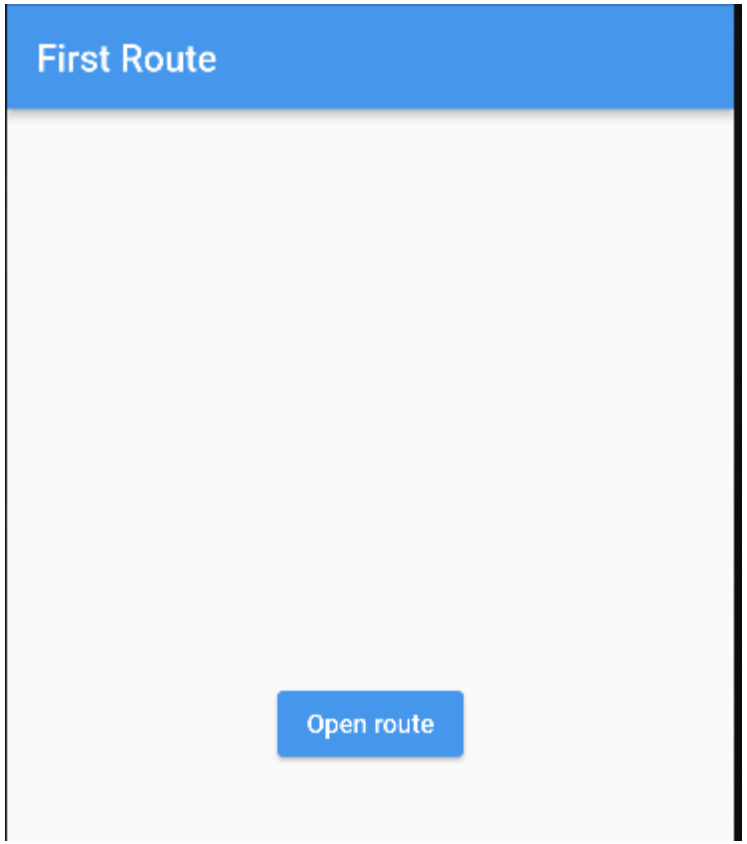
Activité 1
Création facile d'une nouvelle route

Créer deux écrans avec retour en arrière

Sous Android, une route équivaut à une activité. Dans iOS à un ViewController et dans Flutter, une route n'est qu'un **widget**.

```
class FirstRoute extends StatelessWidget {
  const FirstRoute({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Route'), ),
      body: Center(
        child: ElevatedButton(
          child: const Text('Open route'),
          onPressed: () {  // Navigate to second route when tapped.},
        ),
      ),
    );
  }
}
```



La 1ère route

Remarques

- Dans Flutter, les écrans et les pages sont appelés routes .

PARTIE 4

Copyright - Tout droit réservé - OFPPT

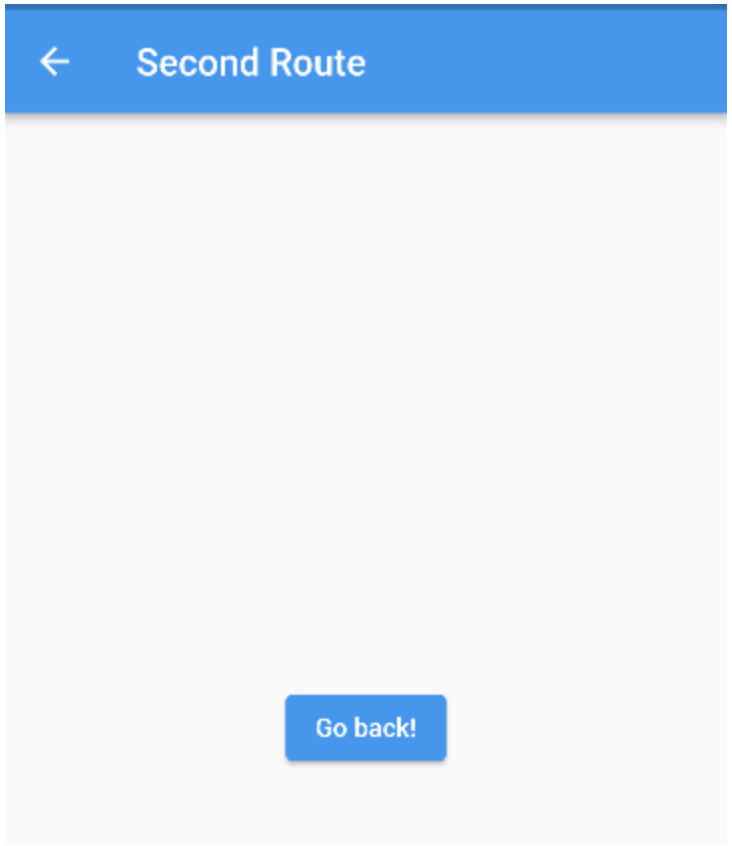
Activité 1
Création facile d'une nouvelle route

Créer deux écrans avec retour en arrière

Le 2ème écran avec la possibilité du retour en arrière.

```
class SecondRoute extends StatelessWidget {
  const SecondRoute({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Second Route'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Navigate back to first route when tapped.
          },
          child: const Text('Go back!'),
        ),
      ),
    );
  }
}
```



La 2ème route

PARTIE 4

Copyright - Tout droit réservé - OFPPT

Activité 1

Création facile d'une nouvelle route

Accédez à la deuxième route à l'aide de Navigator.push()

```
// Within the `FirstRoute` widget
onPressed: () {
  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => const SecondRoute()),
  );
}
```



La flèche de retour en arrière

Revenez au premier écran en utilisant Navigator.pop()

```
// Within the SecondRoute widget
onPressed: () {
  Navigator.pop(context);
}
```

PARTIE 4

Copyright - Tout droit réservé - OFPPT



Activité 2

Passage des données d’un écran à un autre

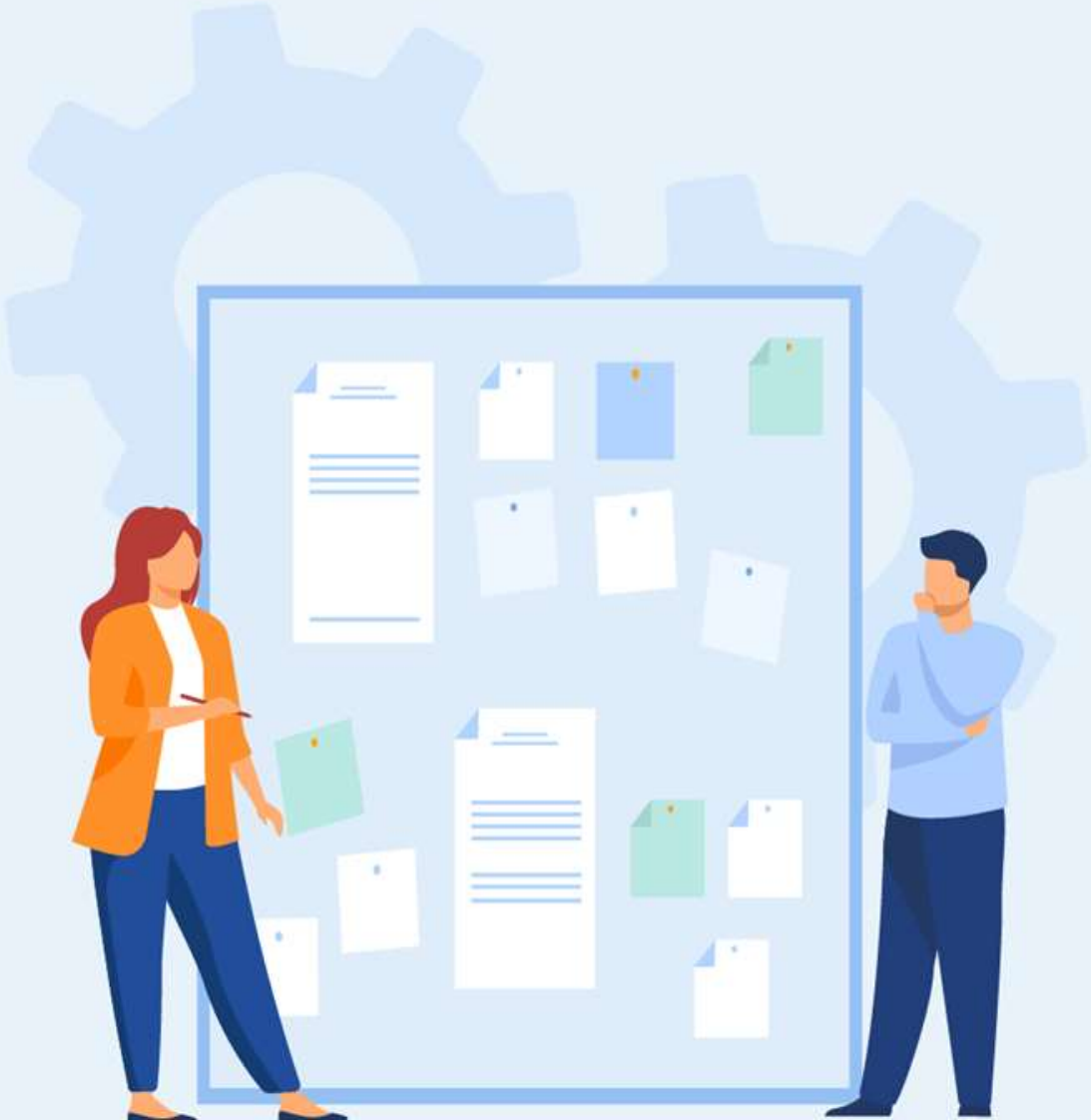
Compétences visées :

- Partager les données entre les 2 écrans
- Assurer le retour en arrière

Recommandations clés :

- Lire attentivement les consignes du résumé théorique

05 heures



CONSIGNES

1. Pour le formateur

- Rappeler la méthode qui permet de créer une nouvelle route
- Demander aux apprenants de partager les données entre les 2 écrans

2. Pour l’apprenant

- Créer les 2 écrans de départ et d’arrivée
- Échanger les données entre les 2 écrans
- Créer la route qui fait la liaison entre les 2 écrans

3. Conditions de réalisation :

- Support de résumé théorique accompagnant
- Installation d’extension de Flutter/Dart

4. Critères de réussite :

- Le stagiaire est-il capable de :
  - Mettre en place des écrans
  - Créer une nouvelle route
  - Partager les données entre les écrans

Activité 2

Passage des données d'un écran à un autre

Envoyer des données vers un nouvel écran

1. Définir toutes les classes

```
class Todo {

final String title;

final String description;

  const Todo(this.title,
this.description);
}
```

2. Créer une liste de tâches

Générer la liste des tâches

```
final todos = List.generate(
  20,
  (i) => Todo(
    'Todo $i',
    'A description of what needs to be done for Todo $i',
  ),
);
```

Afficher la liste des tâches à l'aide d'un ListView

```
ListView.builder(
  itemCount: todos.length,
  itemBuilder: (context, index) {
    return ListTile(
      title: Text(todos[index].title),
    );
  },),
```

3. Créer un écran Todo pour afficher la liste

```
class TodosScreen extends StatelessWidget {
  // Requiring the list of todos.
  const TodosScreen({super.key, required this.todos});

  final List<Todo> todos;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Todos'),
      ),
      body: ListView.builder(
        itemCount: todos.length,
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(todos[index].title),
          );
        },
      ),
    );}}
```

PARTIE 4

Copyright - Tout droit réservé - OFPPT

Activité 2

Passage des données d'un écran à un autre

PARTIE 4

Envoyer des données vers un nouvel écran

4. Créer un écran de détail pour afficher des informations sur une tâche

```
class DetailScreen extends StatelessWidget {
  // In the constructor, require a Todo.
  const DetailScreen({super.key, required this.todo});

  // Declare a field that holds the Todo.
  final Todo todo;

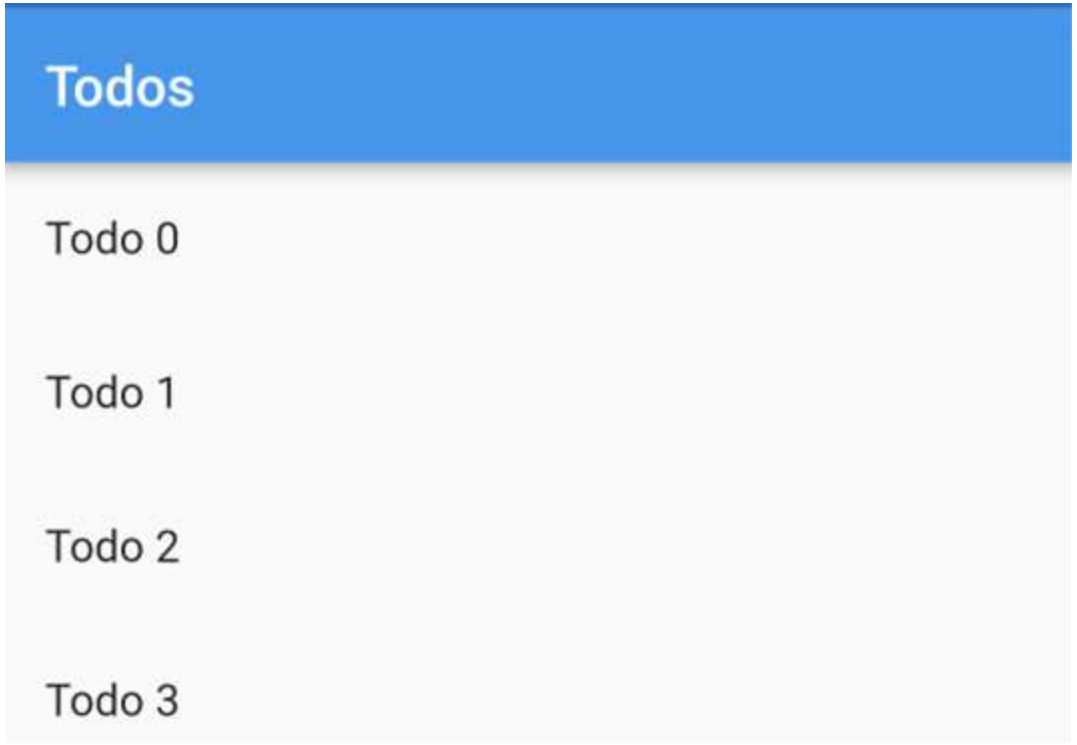
  @override
  Widget build(BuildContext context) {
    // Use the Todo to create the UI.
    return Scaffold(
      appBar: AppBar(
        title: Text(todo.title),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Text(todo.description), ),);
  }
}
```

5. Naviguer et transmettre les données à l'écran de détail

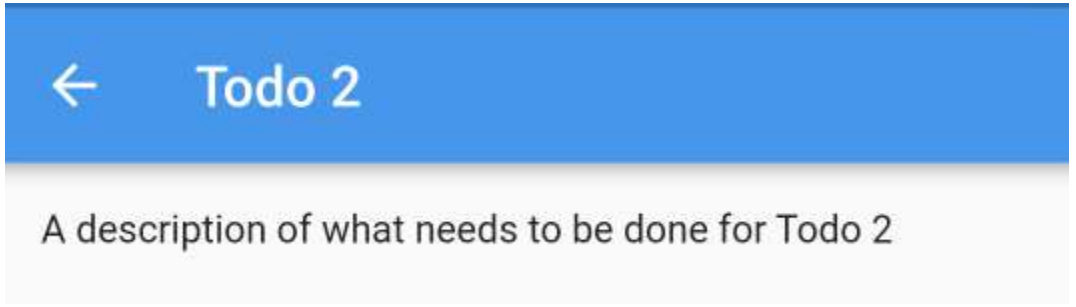
```
body: ListView.builder(
  itemCount: todos.length,
  itemBuilder: (context, index) {
    return ListTile(
      title: Text(todos[index].title),
      // When a user taps the ListTile, navigate to the DetailScreen.
      // Notice that you're not only creating a DetailScreen, you're
      // also passing the current todo through to it.
      onTap: () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => DetailScreen(todo: todos[index]),
          ),
        );
      },
    );
  },
),
```

Copyright - Tout droit réservé - OFPPT

Envoyer des données vers un nouvel écran



La liste des éléments



L'élément sélectionné de la liste

Copyright - Tout droit réservé - OFPPT