

RÉSUMÉ THÉORIQUE – FILIÈRE DÉVELOPPEMENT DIGITAL

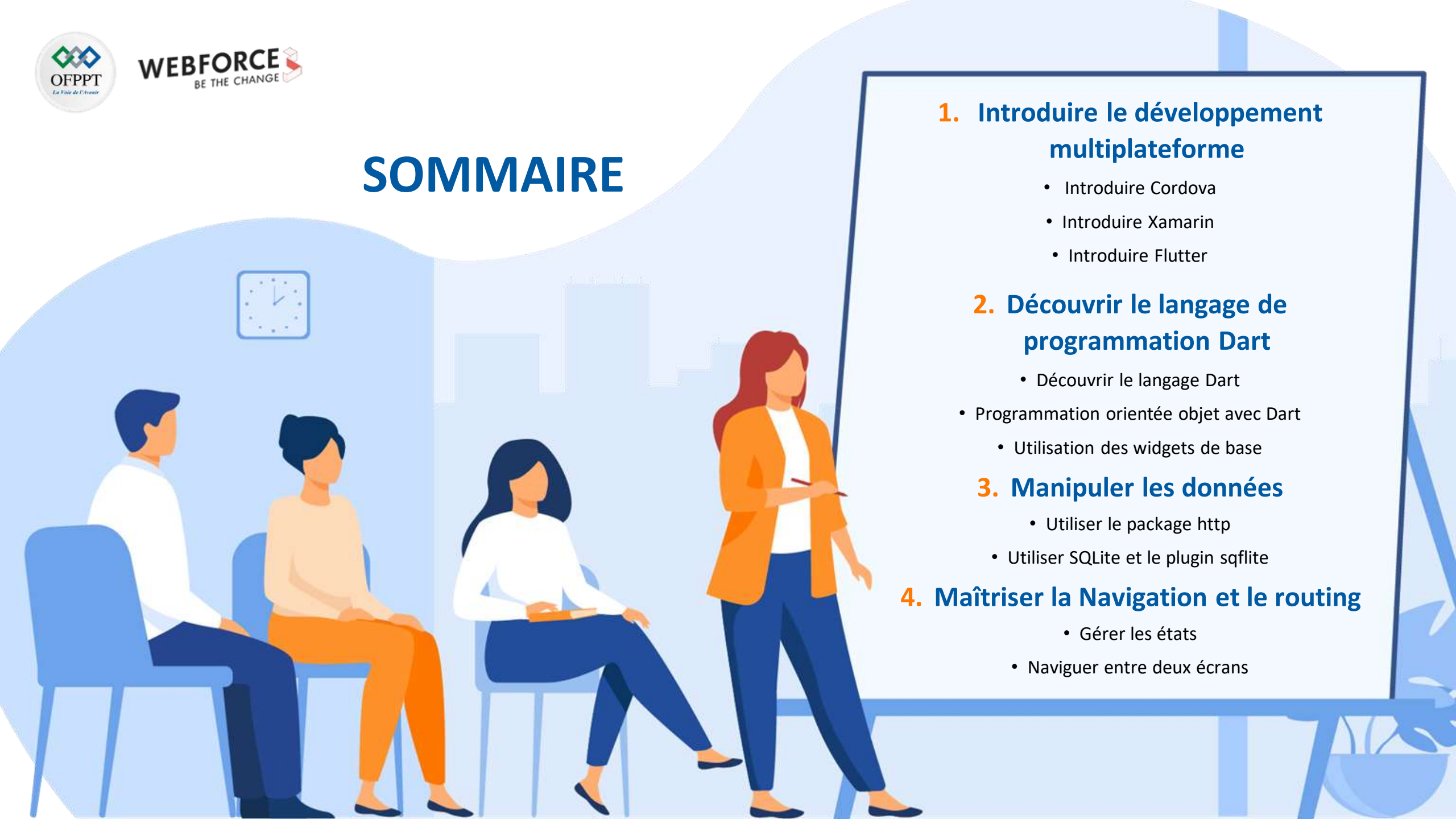
Option – Application mobiles

M208 - Développement multiplateforme

40 heures

SOMMAIRE

1. **Introduire le développement multiplateforme**
   - Introduire Cordova
   - Introduire Xamarin
   - Introduire Flutter
2. **Découvrir le langage de programmation Dart**
   - Découvrir le langage Dart
   - Programmation orientée objet avec Dart
   - Utilisation des widgets de base
3. **Manipuler les données**
   - Utiliser le package http
   - Utiliser SQLite et le plugin sqflite
4. **Maîtriser la Navigation et le routing**
   - Gérer les états
   - Naviguer entre deux écrans



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 TÉLÉCHARGEABLES

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

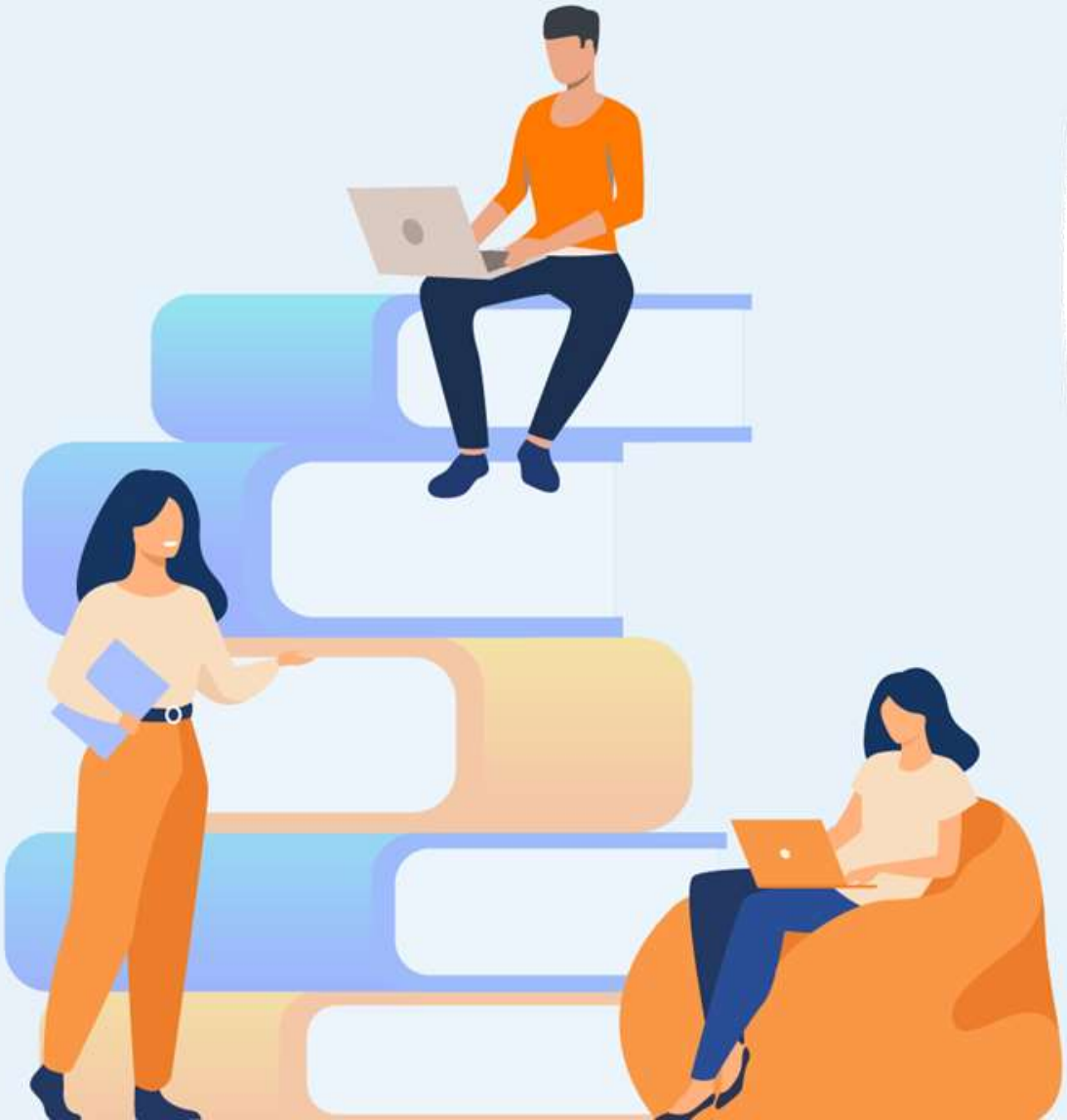


5

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



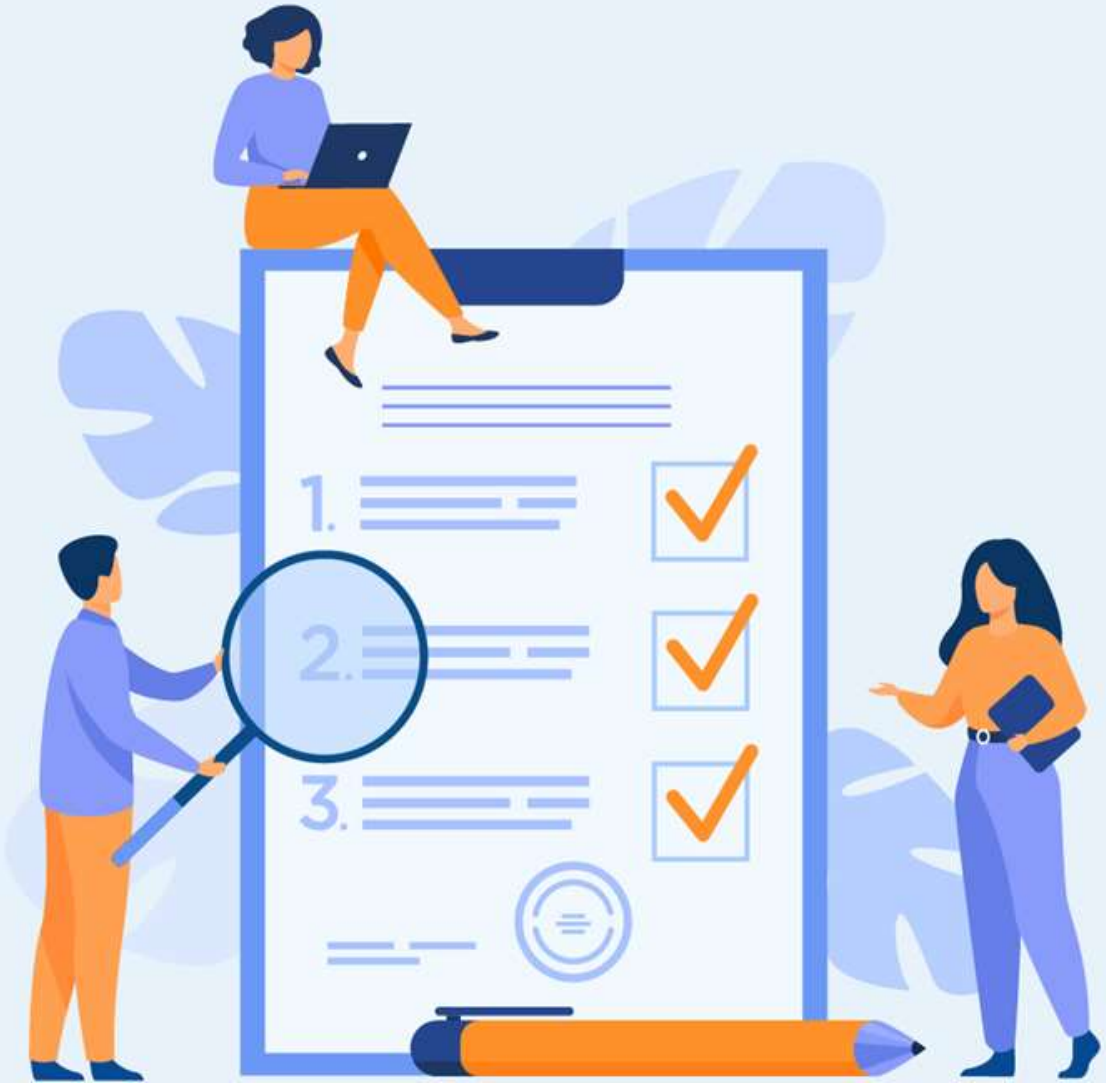
PARTIE 1

Introduire le développement multiplateforme

Dans ce module, vous allez :

- Acquérir une bonne connaissance des différentes plateformes de développement cross plateforme (Cordova, Xamarin)
- Acquérir une bonne connaissance de la plateforme Flutter

10 heures



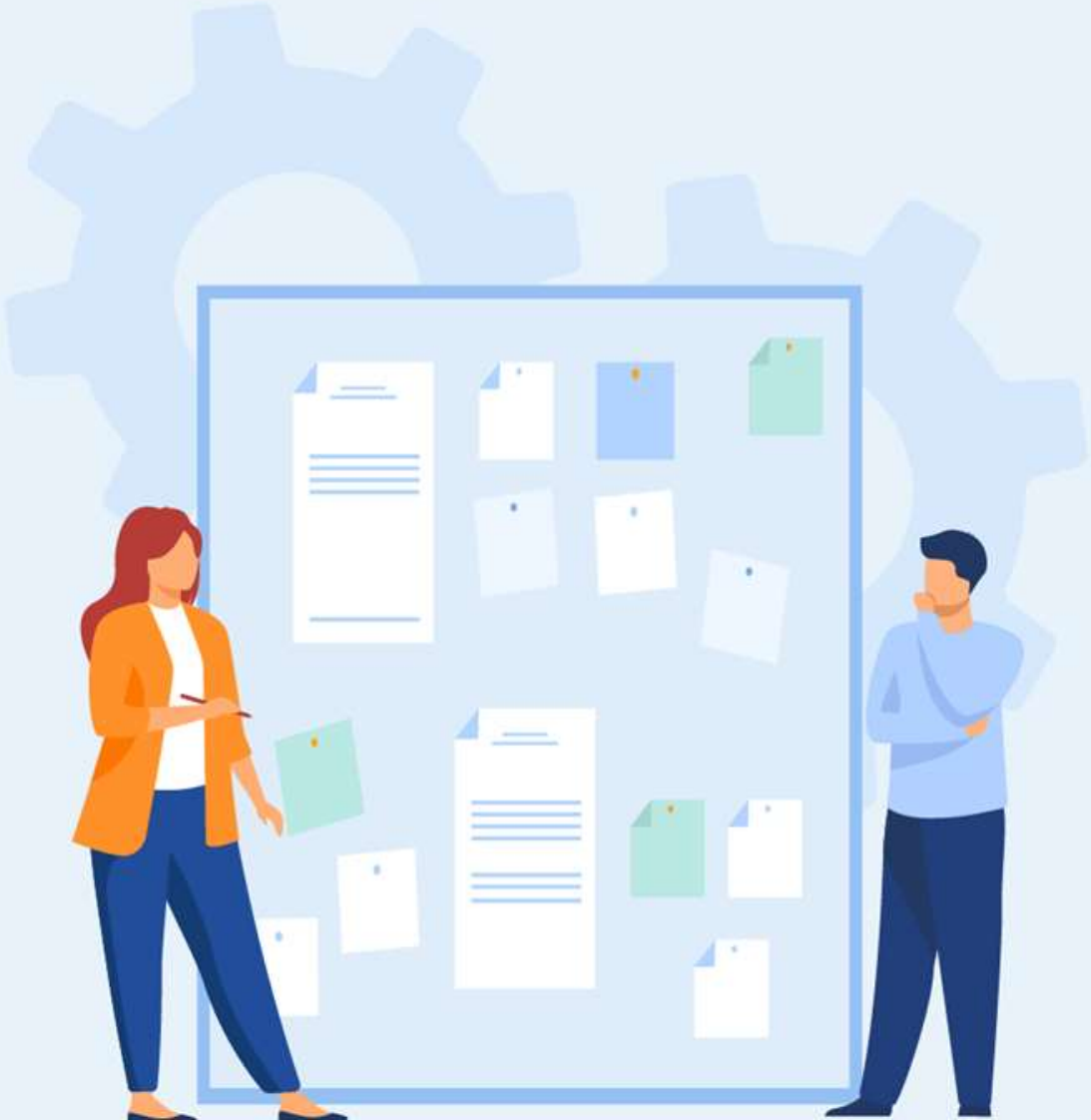
CHAPITRE 1

Introduire Cordova/ionic

Ce que vous allez apprendre dans ce chapitre :

- les avantages et les limites de Cordova
- Le fonctionnement technique de Cordova
- Les outils Cordova

03 heures



CHAPITRE 1

Introduire Cordova/ionic

1. **Avantages et limites de Cordova**
2. Fonctionnement technique de Cordova
3. Outils Cordova

Avantages et limites de Cordova

Avantages

- Un outil open-source.
- Il utilise les standards du web.
- Il agit comme une surcouche.
- Evolutive.
- Maintienbilité.
- Rapidité.

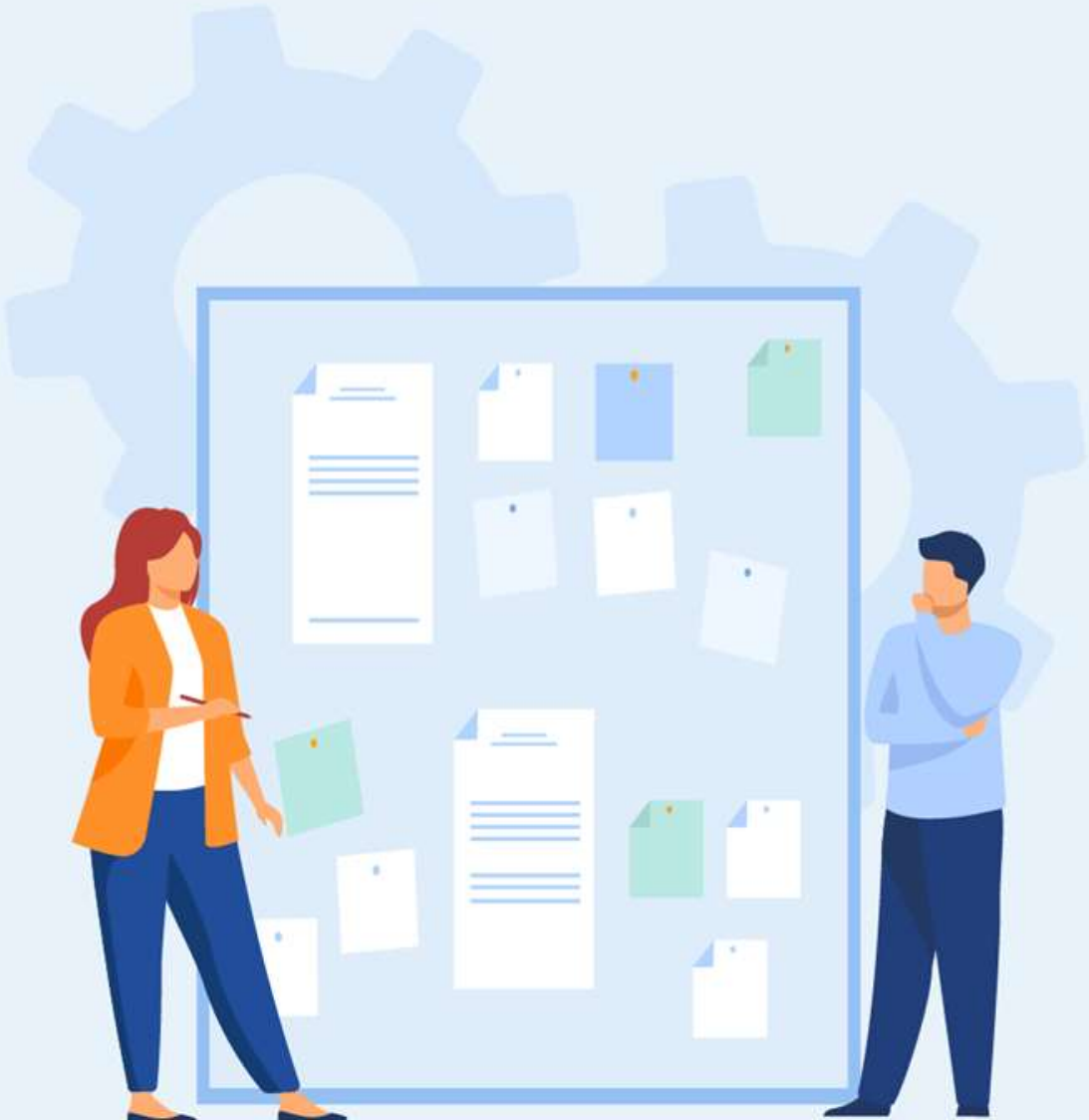
Limites

- Support incomplet des fonctionnalités et capteurs. spécifiques.
- Applications lourdes.
- Performances amoindries.
- Problèmes de compatibilité.
- Modification des plugins requise.

Standards du web

- Html
- CSS
- JavaScript

Copyright - Tout droit réservé - OFPPT



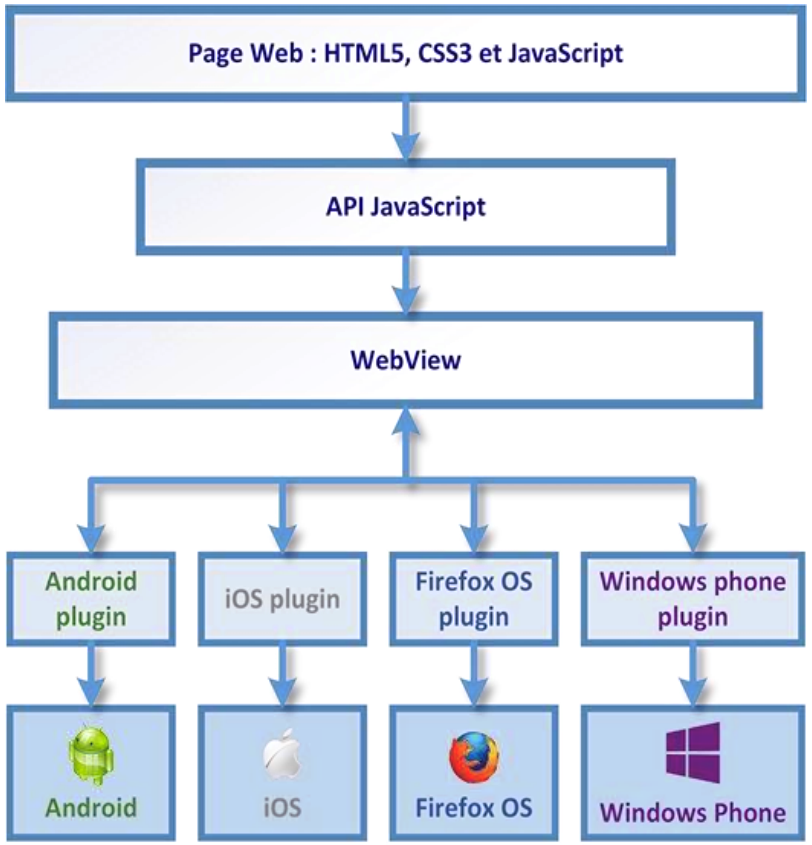
CHAPITRE 1

Introduire Cordova/ionic

1. Avantages et limites de Cordova
2. **Fonctionnement technique de Cordova**
3. Outils Cordova

Principe de fonctionnement

- L'application est développée via technologie du WEB (HTML, CSS et JavaScript).
- Cordova est une API javascript, elle rend le code cohérent pour la plateforme ciblée.
- L'application est intégrée via une webview.
- Les bonnes classes appellent les OS ciblés.



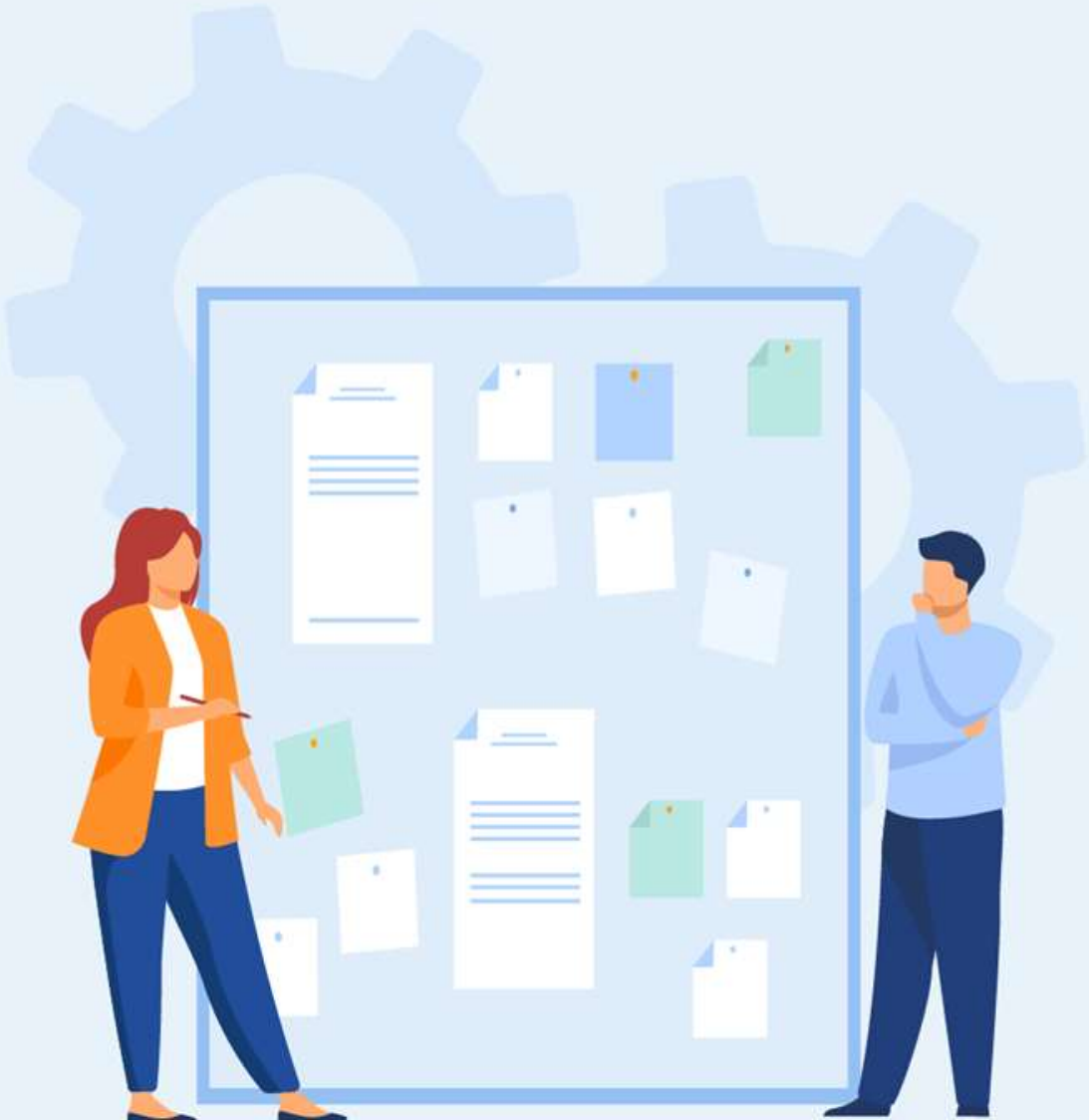
Source: développez.com

WebView

- Un composant système alimenté par Chrome qui permet aux applications Android d'accéder aux pages Web.

PARTIE 1

Copyright - Tout droit réservé - OFPPT



CHAPITRE 1

Introduire Cordova/ionic

1. Avantages et limites de Cordova
2. Fonctionnement technique de Cordova
3. **Outils Cordova**

Les outils WEB

1. Installer NodeJS

Plateforme événementielle en JavaScript orientée vers les applications réseau https://nodejs.org/fr/

2. Installer Cordova

Cordova s'installe avec le Node Package Manager de NodeJS

via la ligne de commande > ***npm install –g cordova@latest***

Après l'installation, vérifier avec > ***cordova -v***

NodeJS

- Est une plateforme logicielle libre en JavaScript, orientée vers les applications réseau évènementielles hautement concurrentes qui doivent pouvoir monter en charge.

PARTIE 1

Copyright - Tout droit réservé - OFPPT

Les outils Android

1. Installer JDK (1.8 recommandé) kit de développement en Java

https://www.oracle.com/fr/java/technologies/javase/javase-jdk8-downloads.html

Créer variable d'environnement **JAVA_HOME** pointant vers la racine du JDK.

2. Installer SDK Android

Kit de développement logiciel Android contenant tous les outils (SDK Manager, AVD manager, etc.)

https://developer.android.com/studio#downloads

Ajouter les dossiers tools et platform-tools dans le PATH des variables d'environnement.

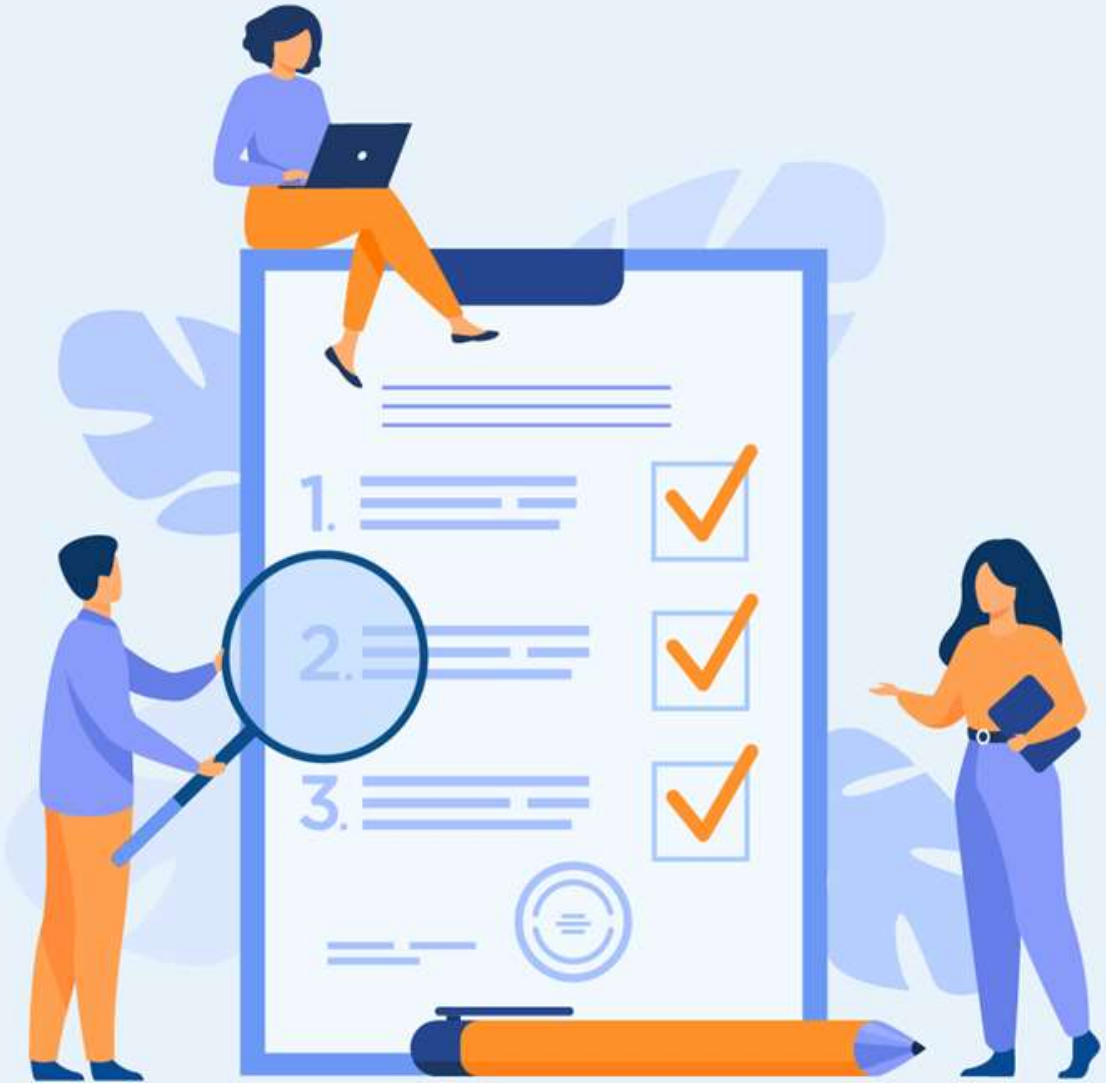
3. Installer Gradle

Pour la compilation des projets https://gradle.org/install/ Ajouter le dossier bin de Gradle au PATH du système.

Gradle

- Un moteur de production fonctionnant sur la plateforme Java. Il permet de construire des projets en Java, Scala, Groovy voire C++.

Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

Introduire Xamarin

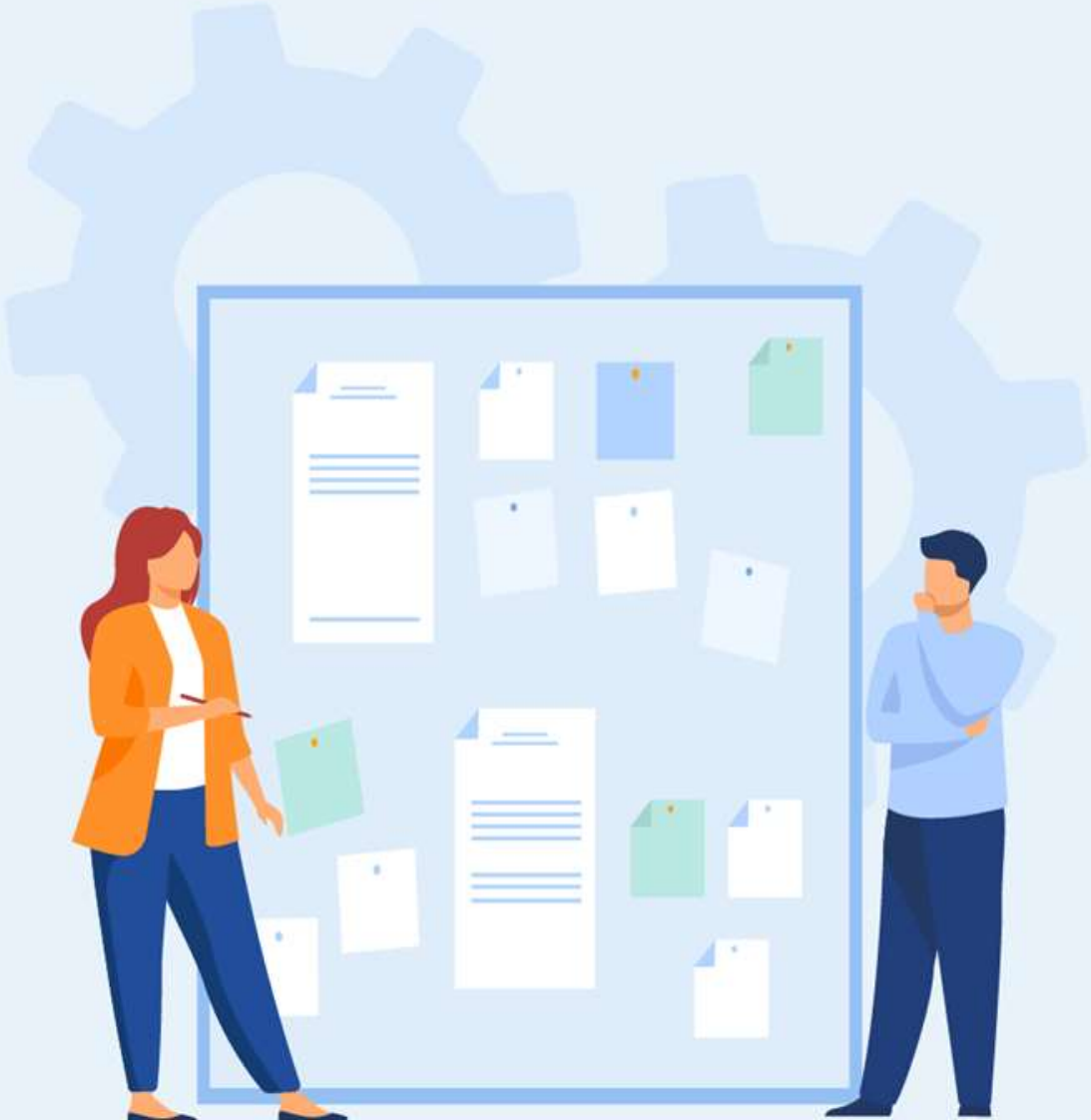
Ce que vous allez apprendre dans ce chapitre :

- Avantages et limites de Xamarin
- Fonctionnement technique de Xamarin
- Outils Xamarin

03 heures

CHAPITRE 2

Introduire Xamarin



1. **Avantages et limites de Xamarin**
2. Fonctionnement technique de Xamarin
3. Outils Xamarin

Avantages et limites de Xamarin

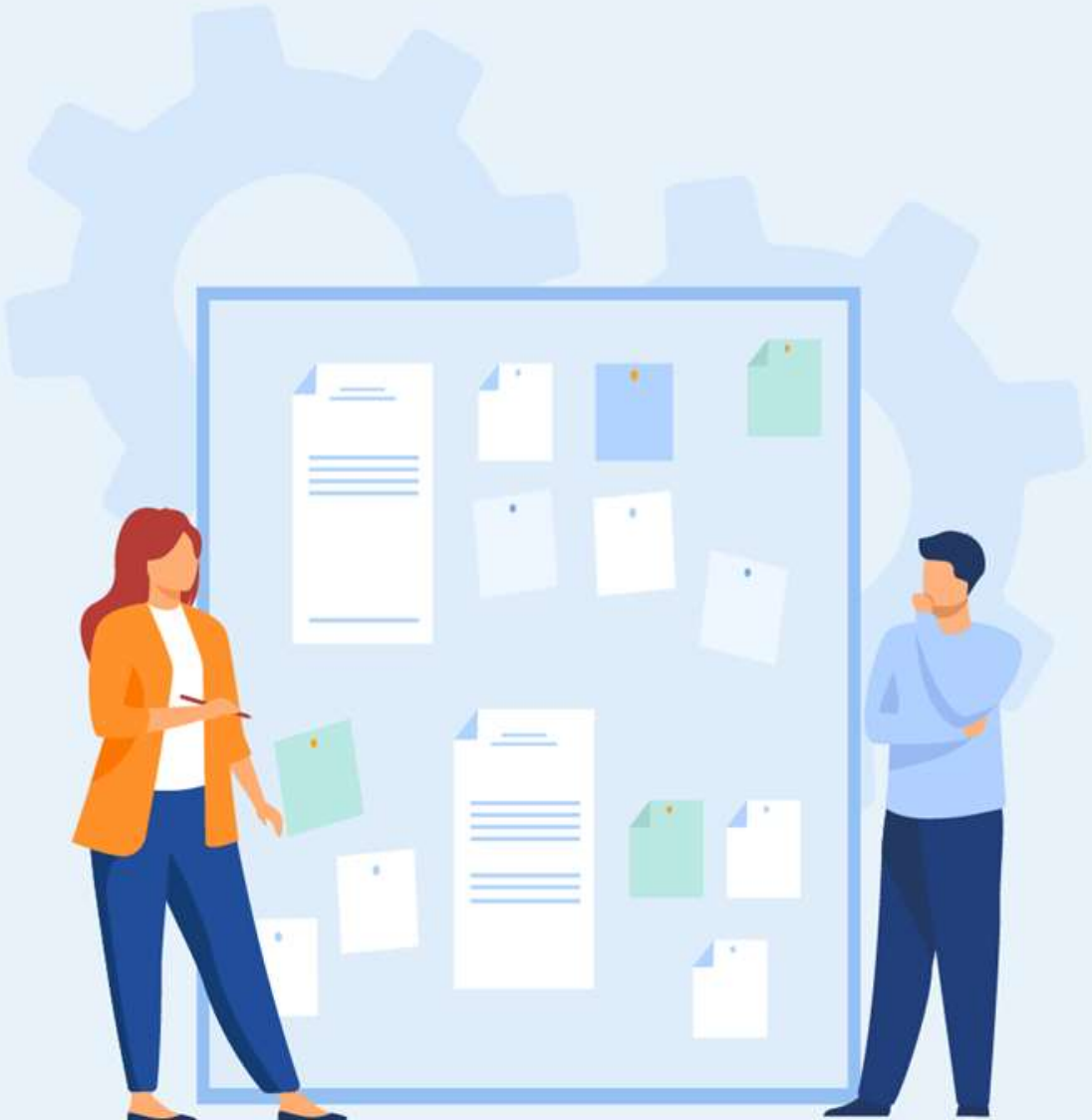
Avantages

- Framework équipé de Visual Studio.
- Logique d'application partagée.
- Interface utilisateur.
- Nuage de test Xamarin.
- Un incroyable magasin de composants.

Limites

- Frais généraux de l'application.
- Manque de soutien communautaire.
- Impossible de partager des codes en dehors de Xamarin.
- Certains manuels de codage sont requis.

Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

Introduire Xamarin

1. Avantages et limites de Xamarin
2. **Fonctionnement technique de Xamarin**
3. Outils Xamarin

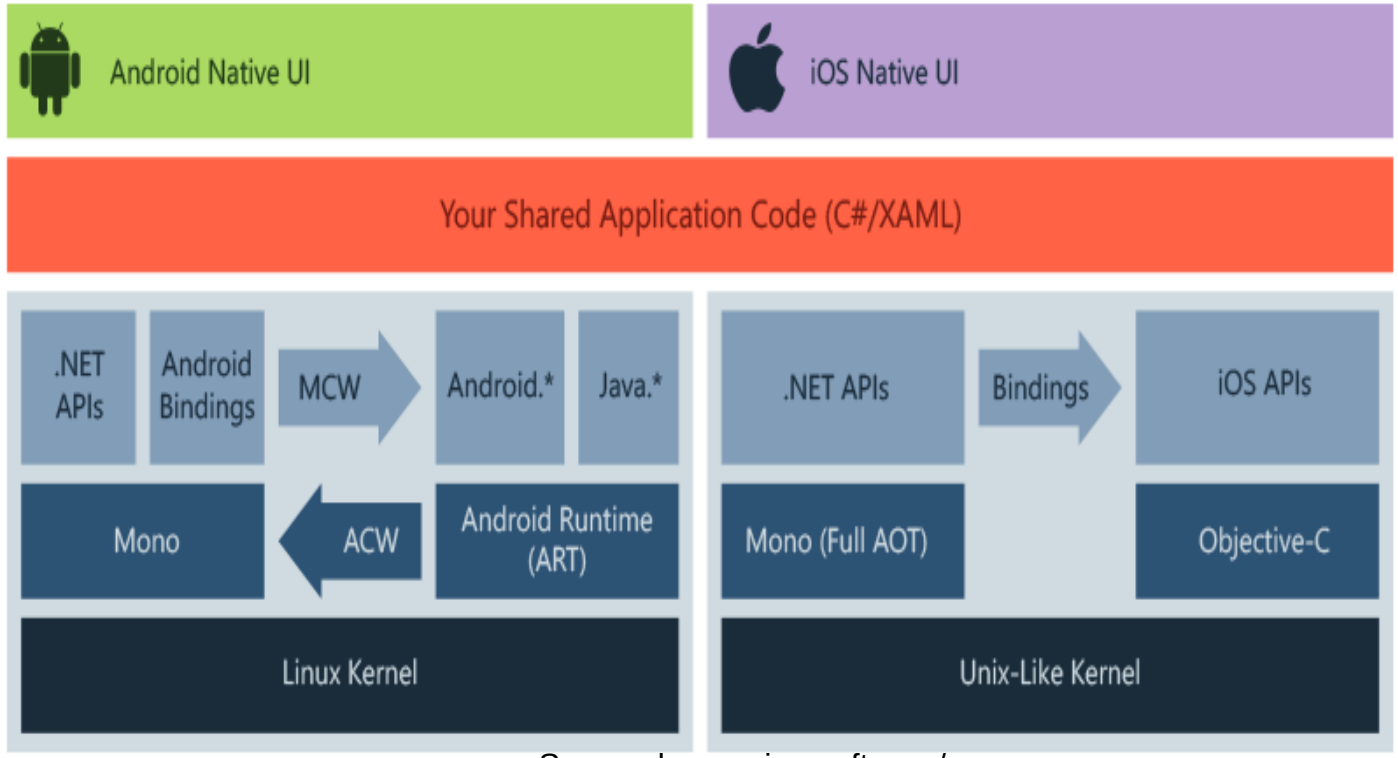
Fonctionnement de Xamarin

Le diagramme montre l'architecture globale d'une application Xamarin multiplateforme.

- Xamarin vous permet de créer une IU native sur chaque plateforme et d'écrire une logique métier en C#, qui est partagée entre les plateformes.

Dans la plupart des cas, 80 % du code d'application peut être partagé à l'aide de Xamarin.

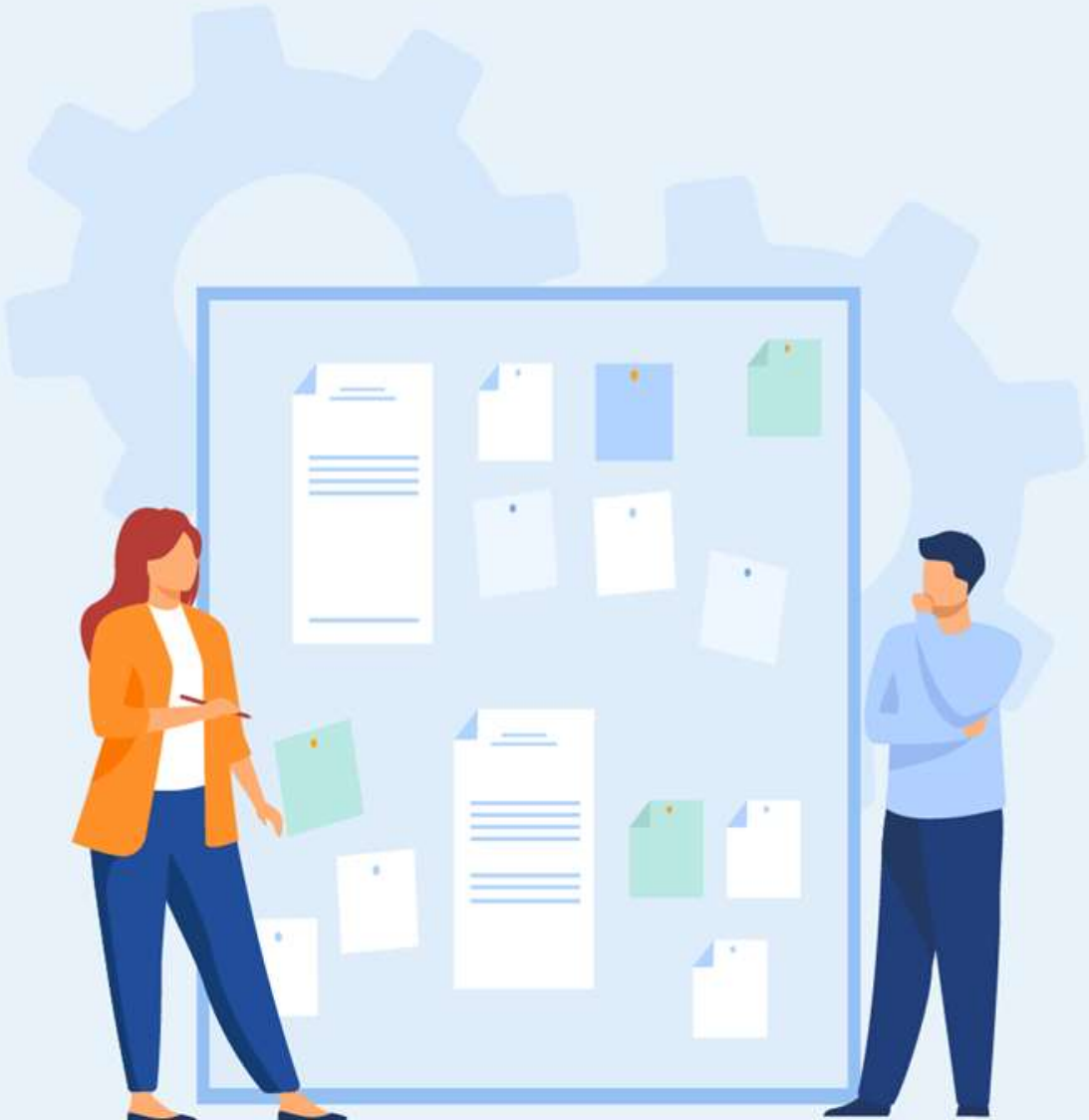
- Xamarin est basé sur **.NET**, qui gère automatiquement les tâches telles que l'allocation de mémoire, le garbage collection et l'interopérabilité avec les plateformes sous-jacentes.



Source: learn.microsoft.com/

PARTIE 1

Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

Introduire Xamarin

1. Avantages et limites de Xamarin
2. Fonctionnement technique de Xamarin
3. **Outils Xamarin**

Outils Xamarin

Les outils conçus pour améliorer votre expérience de développement Xamarin.





- **iOS Simulator pour Windows**

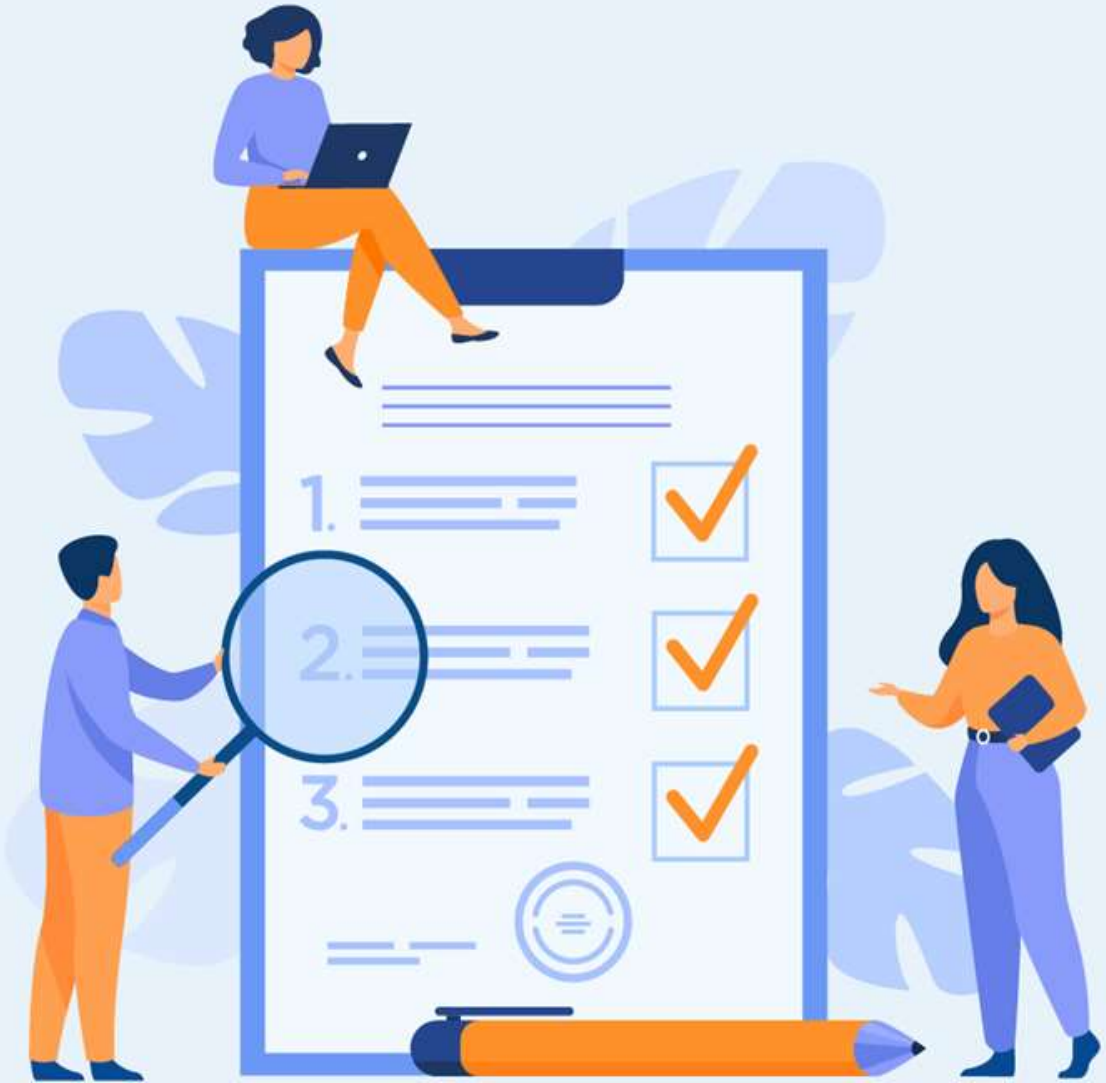
Interagissez avec iOS Simulator sur Windows à l'aide de Visual Studio.

- **Xamarin Profiler**

Identifiez et résolvez les problèmes de performances.

PARTIE 1

Copyright - Tout droit réservé - OFPPT



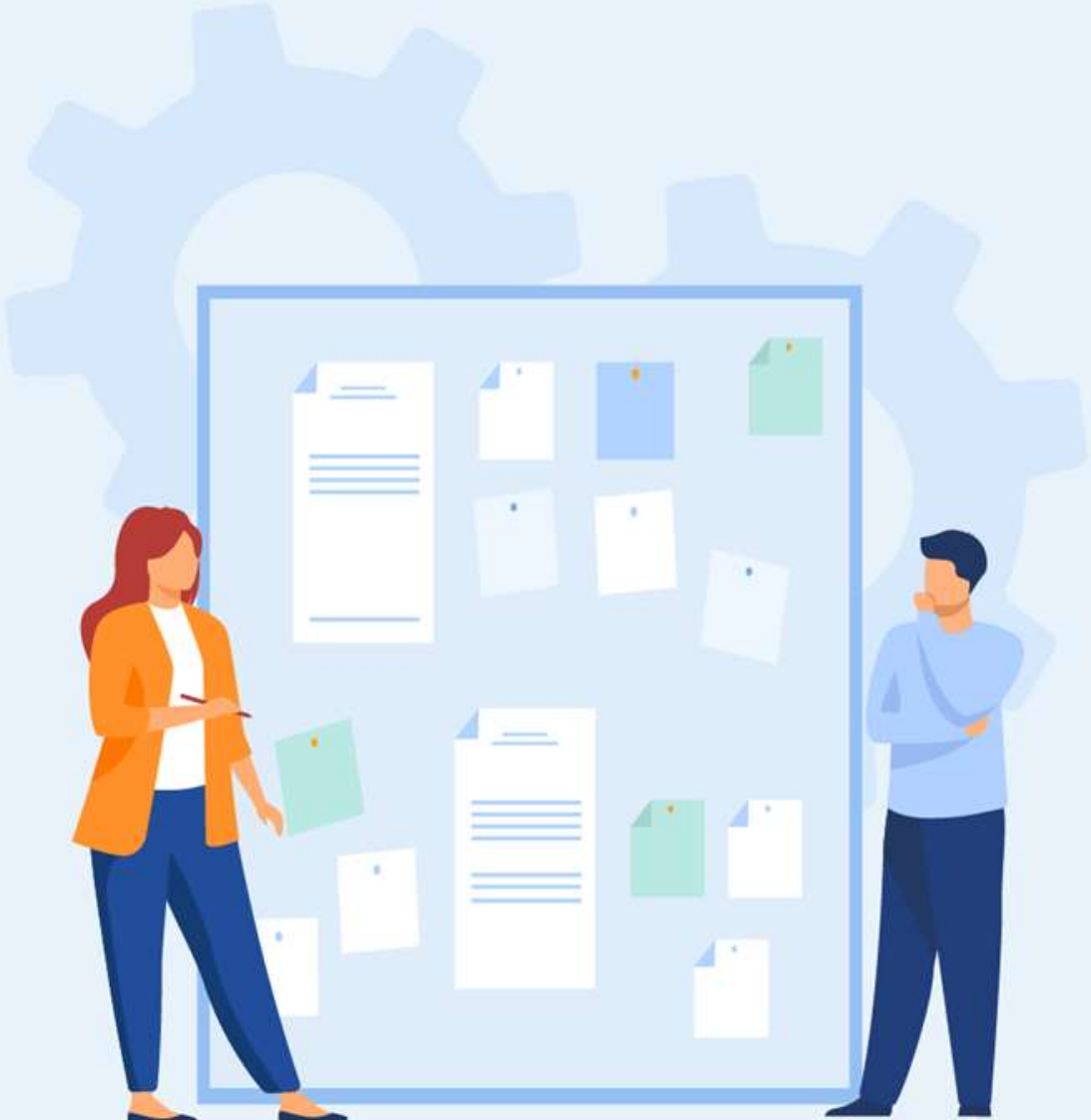
CHAPITRE 3

Introduire Flutter

Ce que vous allez apprendre dans ce chapitre :

- Les avantages et limites de Flutter
- Le fonctionnement technique de Flutter
- Les outils Flutter

04 heures



CHAPITRE 3

Introduire Flutter

1. **Avantages et limites de Flutter**
2. Fonctionnement technique de Flutter
3. Outils Flutter

Avantages et limites de Flutter

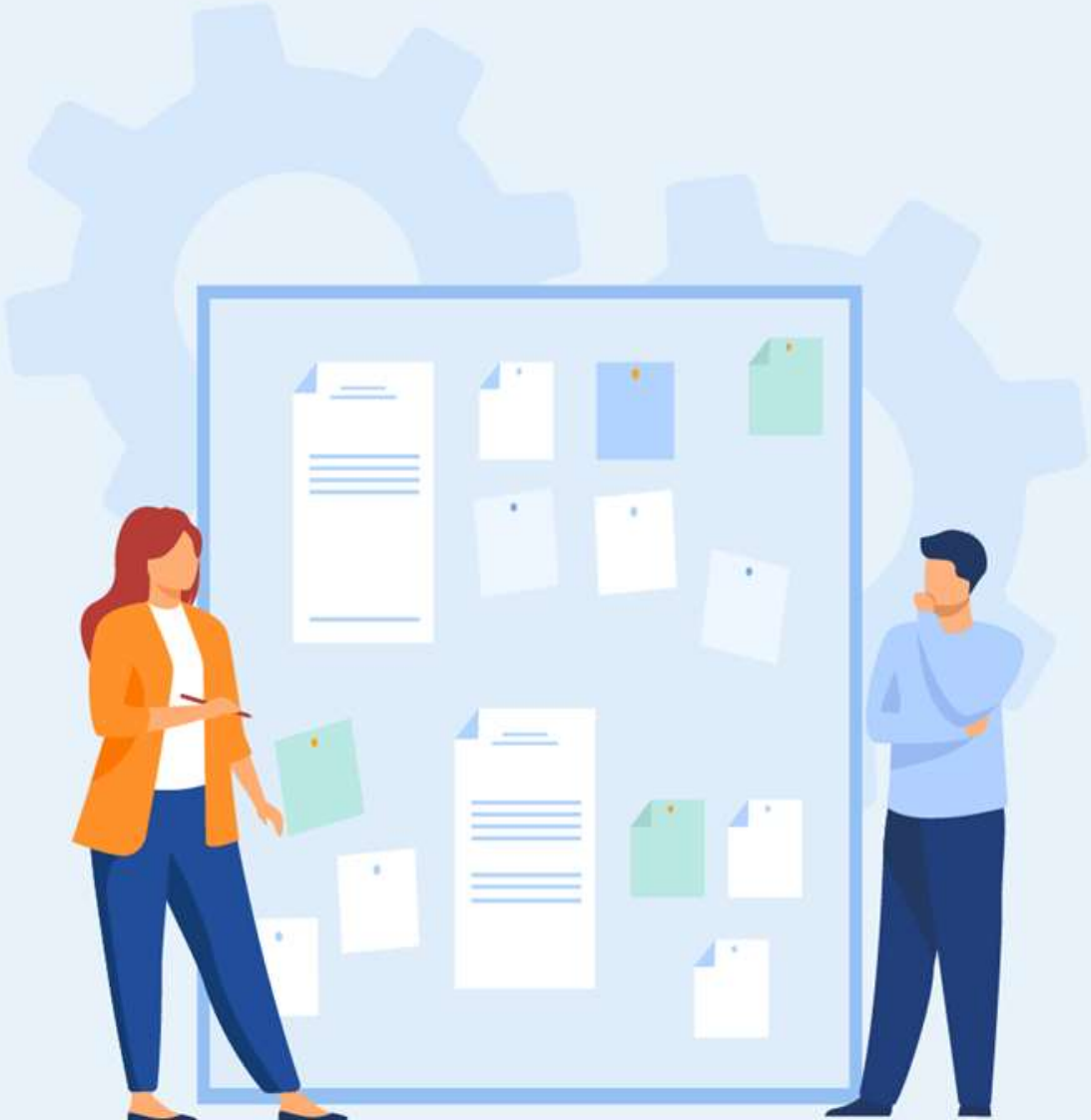
Avantages

- La qualité de l'environnement de développement.
- Les plugins Android Studio et VSCode.
- La documentation officielle.
- Une bonne rétrocompatibilité.
- Une grande communauté.

Limites

- Devoir apprendre Dart.
- La parité des plateformes.
- La propriété de Google.
- La performance est limitée face à une application native.

Copyright - Tout droit réservé - OFPPT



CHAPITRE 3

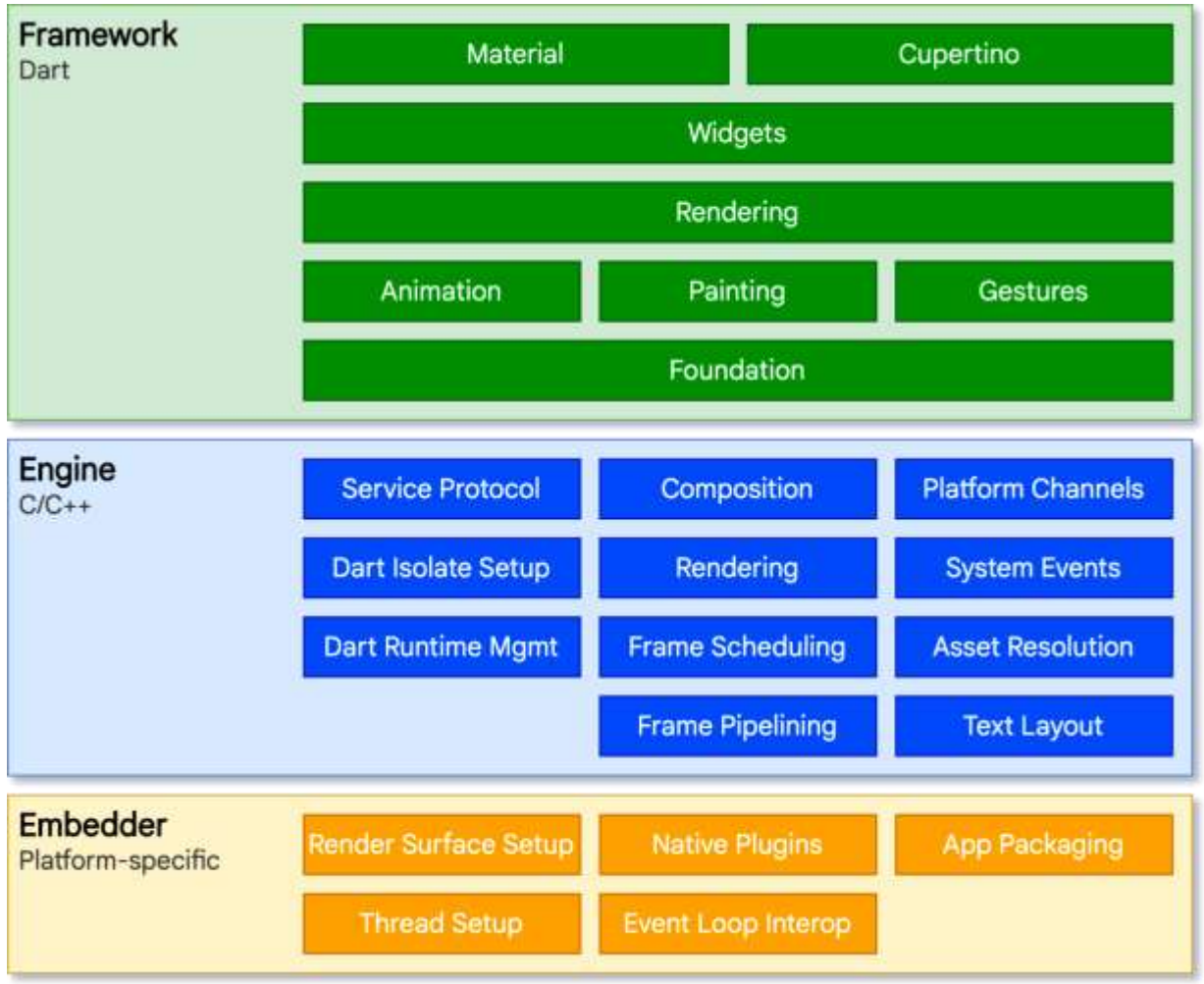
Introduire Flutter

1. Avantages et limites de Flutter
2. **Fonctionnement technique de Flutter**
3. Outils Flutter

Fonctionnement technique de Flutter

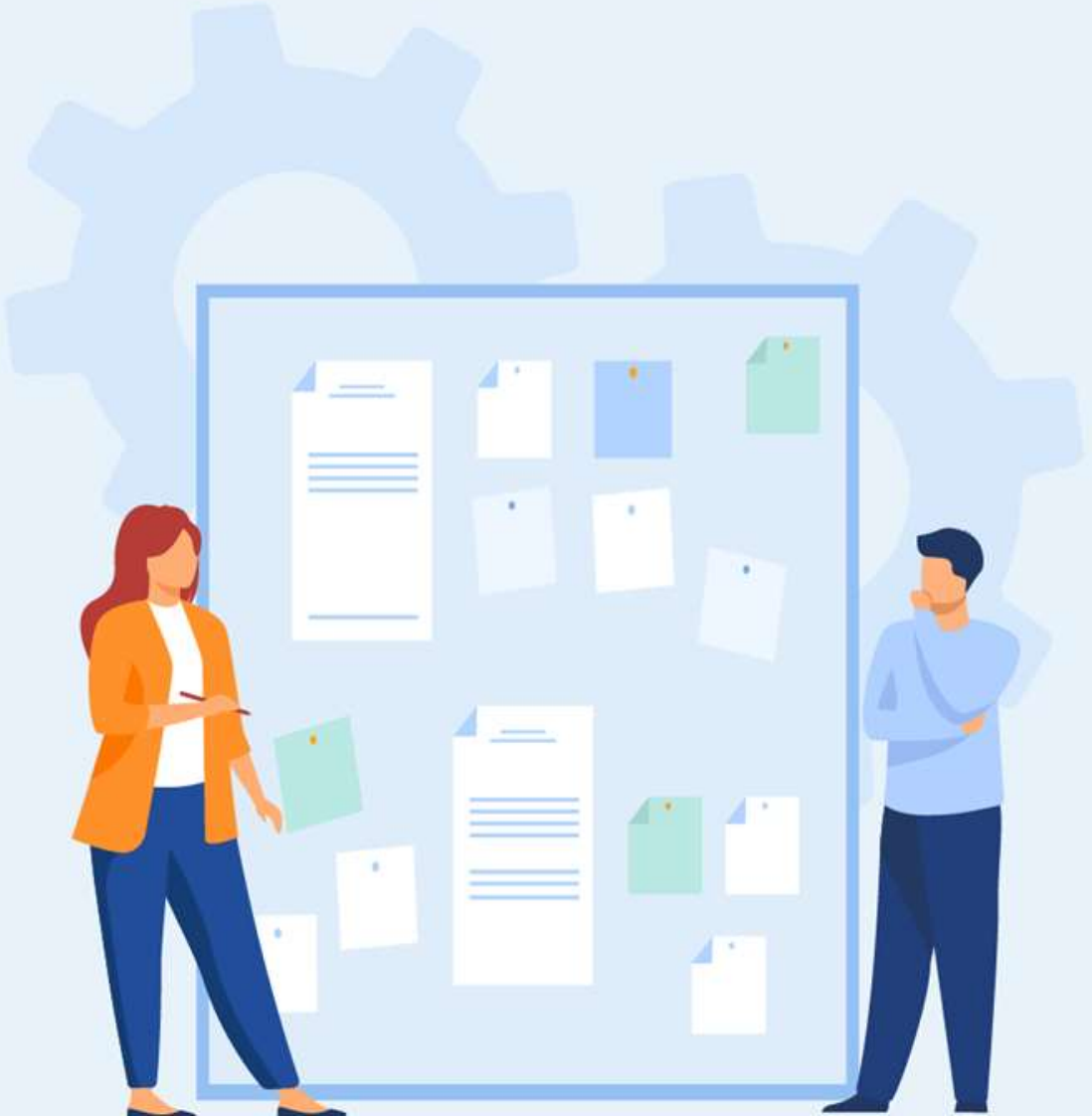
Flutter est composé de 3 couches :

1- Le framework, qui s'appuie sur Dart,
2- Le moteur, qui utilise du C++,
3- La couche de communication avec les fonctionnalités natives.



source : kaliop.com

Copyright - Tout droit réservé - OFPPT



CHAPITRE 3

Introduire Flutter

1. Avantages et limites de Flutter
2. Fonctionnement technique de Flutter
3. **Outils Flutter**

Outils Flutter

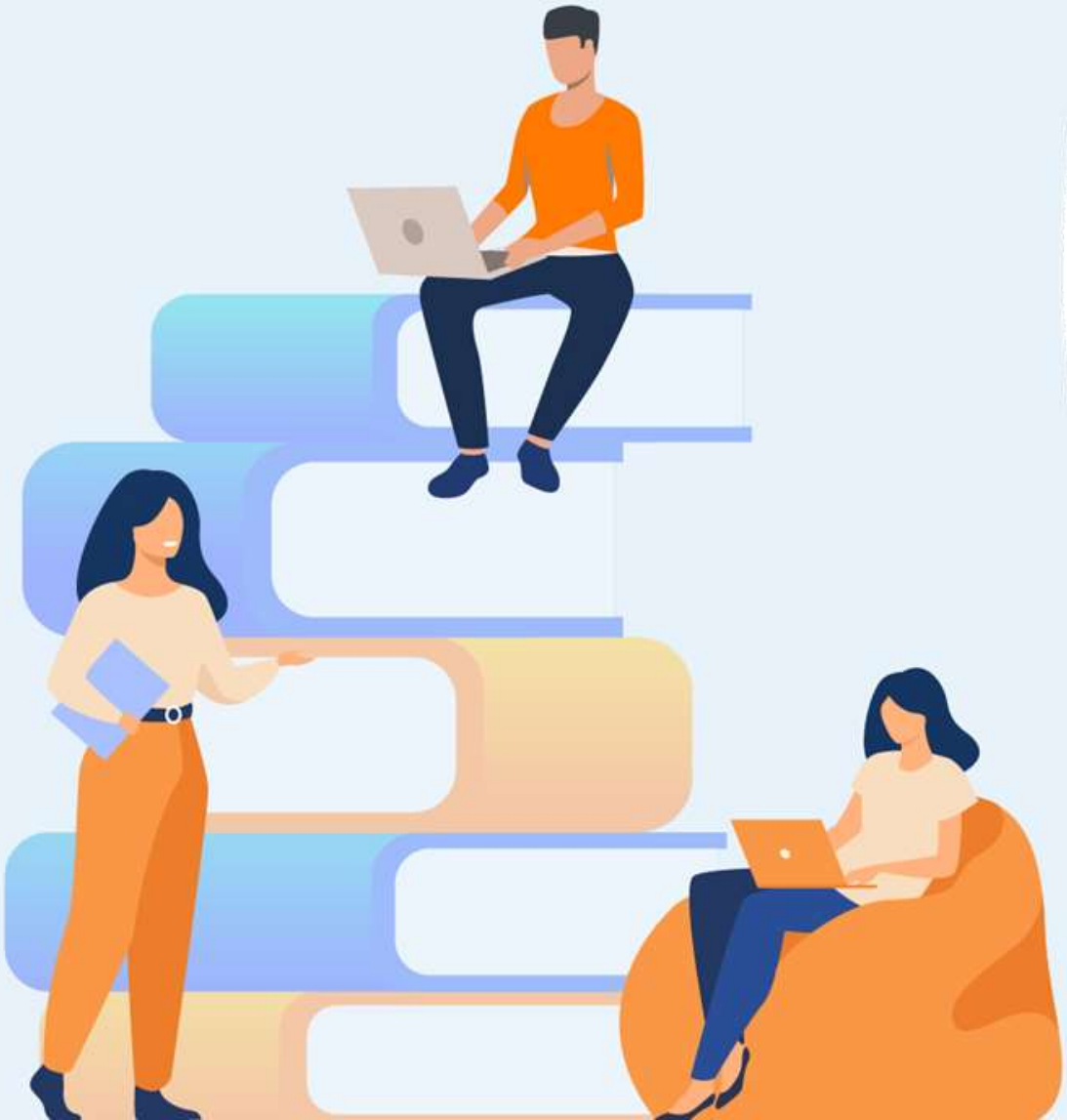
SDK Flutter : contient les packages et les outils de ligne de commande dont vous avez besoin pour développer des applications Flutter sur plusieurs plates-formes.

Url : https://docs.flutter.dev/get-started/install

Les éléments suivants sont disponibles via le SDK Flutter :

- Kit de développement de Dart
- Moteur de rendu 2D optimisé pour les mobiles avec une excellente prise en charge du texte
- Cadre moderne de style réactif
- Riche ensemble de widgets mettant en œuvre les styles Material Design et iOS
- API pour les tests unitaires et d'intégration
- API d'interopérabilité et de plug-in pour se connecter au système et aux SDK tiers
- Coureur de test sans tête pour exécuter des tests sur Windows, Linux et Mac
- Dart DevTools pour tester, déboguer et profiler votre application
- Flutter et Dart des outils de ligne de commande pour créer, construire, tester et compiler vos applications

Copyright - Tout droit réservé - OFPPT



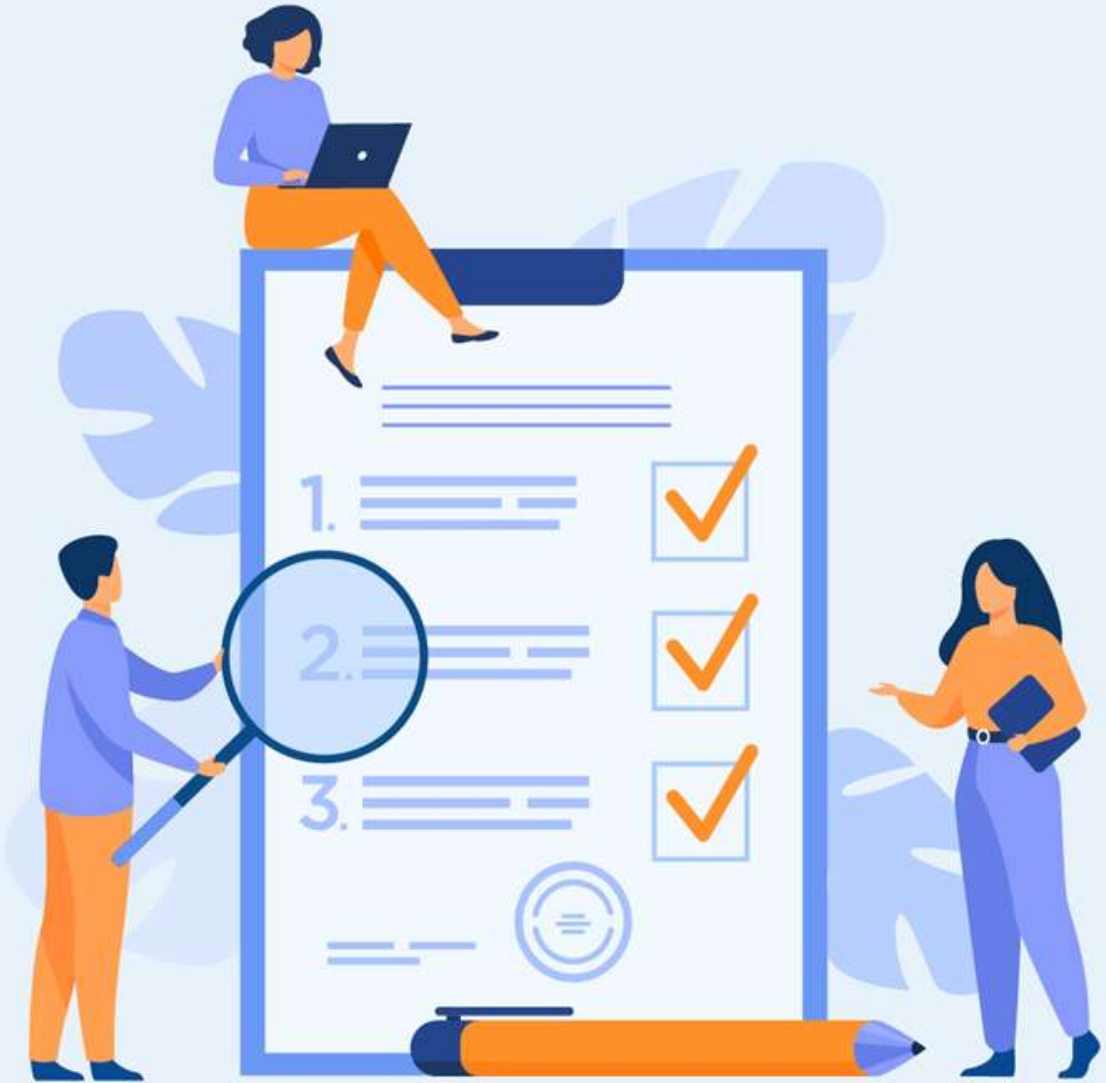
PARTIE 2

Découvrir le langage de programmation Dart

Dans ce module, vous allez :

- Acquérir une bonne maîtrise de la logique orientée widget
- Avoir une configuration facile de l'environnement Flutter
- Maîtriser la programmation orientée objet avec Dart

15 heures



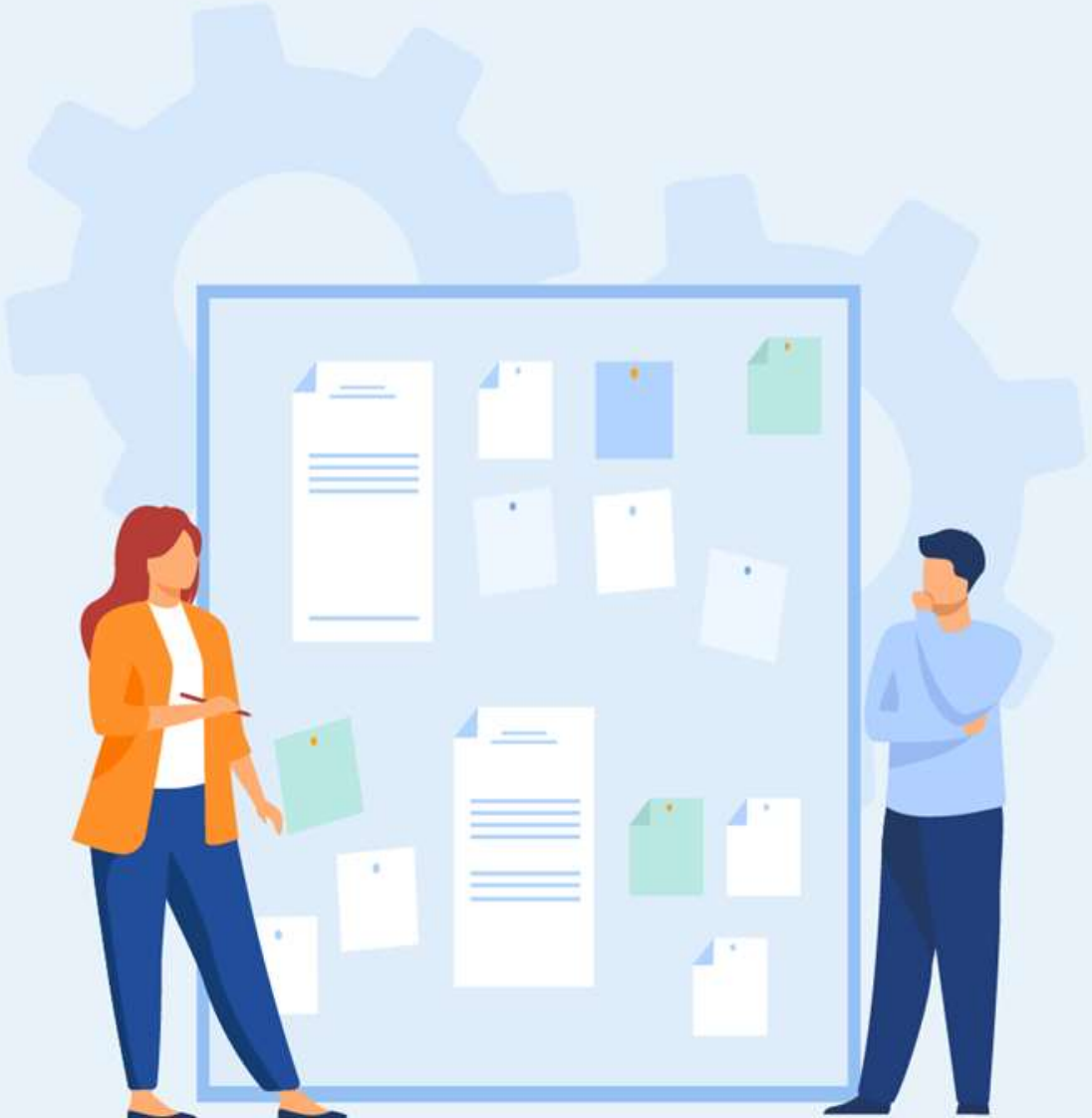
Chapitre 1

Découvrir le langage de programmation Dart

Ce que vous allez apprendre dans ce chapitre :

- Les principes et Les concepts fondamentaux
- Les compilations JIT (Just In Time) et AOT (Ahead Of Time)
- Les conventions de codage, noms et ordonnancement

05 heures



Chapitre 1

Découvrir le langage de programmation Dart

1. **Principes et concepts fondamentaux**
2. Compilations JIT (Just In Time) et AOT (Ahead Of Time)
3. Conventions de codage, noms et ordonnancement

Dart

Dart est un langage de programmation optimisé pour les applications sur plusieurs plate-formes.

Il est développé par Google et utilisé pour créer des applications mobiles, de bureau et du web.

Dart est un langage orienté objet, basé sur la classe. Il existe plusieurs environnements permettant de programmer dans le langage Dart tels que l'invite de commande, l'environnement web et un terminal smartphone.



source : dart.dev

Copyright - Tout droit réservé - OFPPT

Principes et concepts fondamentaux

Dart contient comme tous les langages de programmation :

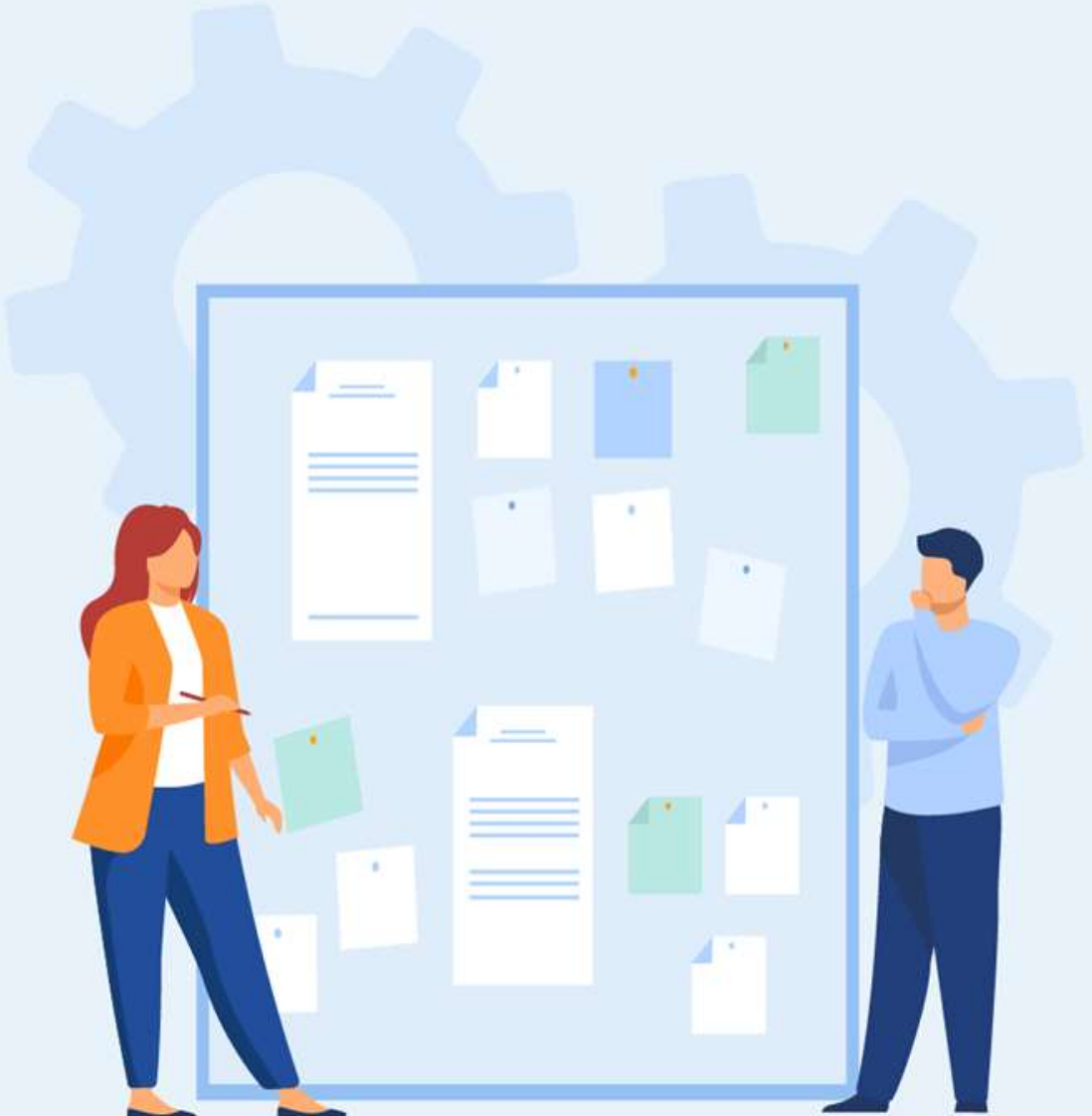
- Les Variables
- Les nombres
- Les chaînes de caractères
- Les booléens

```
int maVariable = 100;
num realNumber = 1;
double simpleDc = 10.4;
bool dartIsFun = true;
var nombre = 124;
```

Il y a quatre façons d'exécuter le code Dart :

- Compilé en JavaScript.
- Autonome.
- Compilation anticipée.
- Native.

Copyright - Tout droit réservé - OFPPT



Chapitre 1

Découvrir le langage de programmation Dart

1. Principes et concepts fondamentaux
2. **Compilations JIT (Just In Time) et AOT (Ahead Of Time)**
3. Conventions de codage, noms et ordonnancement

La compilation

Flutter propose 2 types de compilations :

- **AOT** (Ahead Of Time) :

  Génère une application native pour chaque plateforme, c'est l'avantage de Flutter, par rapport à ses concurrents, puisque le code est optimisé directement pour l'architecture sur laquelle il fonctionnera.

- **JIT** (Just-In-Time) :

  Offre la fonctionnalité de **"Hot Reload"** lors des développements. Si vous avez déjà eu l'occasion de concevoir une application Android native, vous avez probablement été effrayés devant les temps nécessaires entre chaque build.

  L'idée du Hot Reload en Flutter est de corriger ce problème en ne mettant plus que quelques millisecondes (voire secondes dans le pire des cas) entre chaque modification.

  Le développement de son application est alors bien plus rapide.

Copyright - Tout droit réservé - OFPPT

Compilations JIT (Just In Time)

Un compilateur **JIT** convertit le code source du programme en code machine natif juste avant l'exécution du programme.

Les compilateurs sont généralement l'un des facteurs déterminants de la vitesse d'une application, tant en développement qu'en phase de production.

Les compilateurs **JIT** peuvent être utilisés pour améliorer la vitesse des performances et améliorer le temps d'exécution des applications.

Il est courant dans le langage de programmation JAVA et constitue également l'une des façons dont les codes sources du programme sont exécutés dans DART.

Les compilateurs **JIT** essaient de prédire quelles instructions seront exécutées ensuite afin de pouvoir compiler le code à l'avance.

Copyright - Tout droit réservé - OFPPT

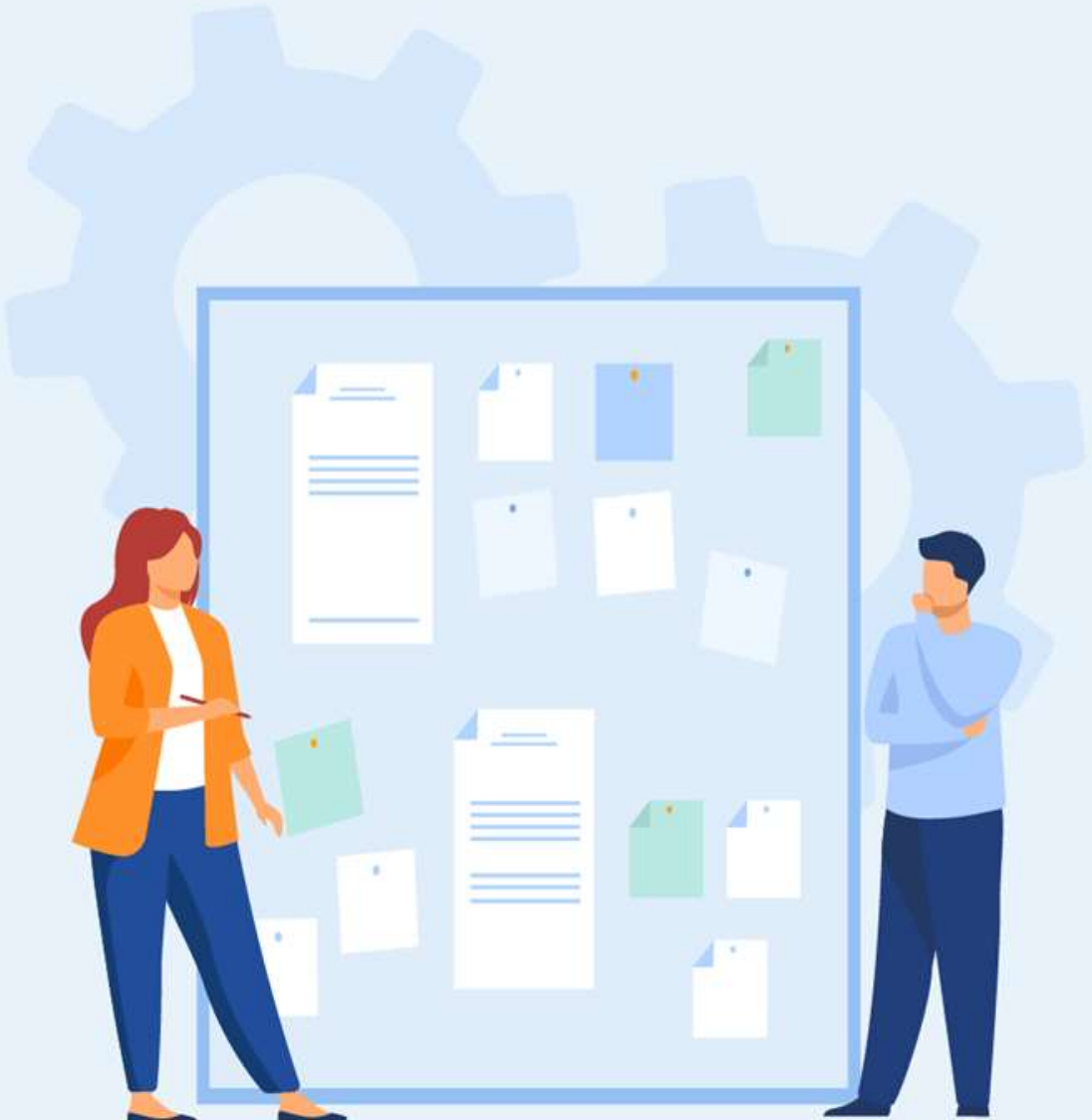
Compilations AOT (Ahead Of Time)

Le compilateur **AOT** fonctionne en compilant votre code avant qu'il ne soit **livré** à n'importe quel environnement d'exécution qui exécute le code.

Le compilateur **AOT** est généralement utilisé lorsque l'application est prête à être déployée sur une boutique d'applications ou sur un backend de production interne.

Le code compilé par **AOT** s'exécute dans un runtime Dart efficace qui applique le système de type Dart sonore et gère la mémoire à l'aide d'une allocation d'objet rapide et d'un ramasse-miettes générationnel, comme spécifié sur le site Web de documentation de Dart.

Copyright - Tout droit réservé - OFPPT



Chapitre 1

Découvrir le langage de programmation Dart

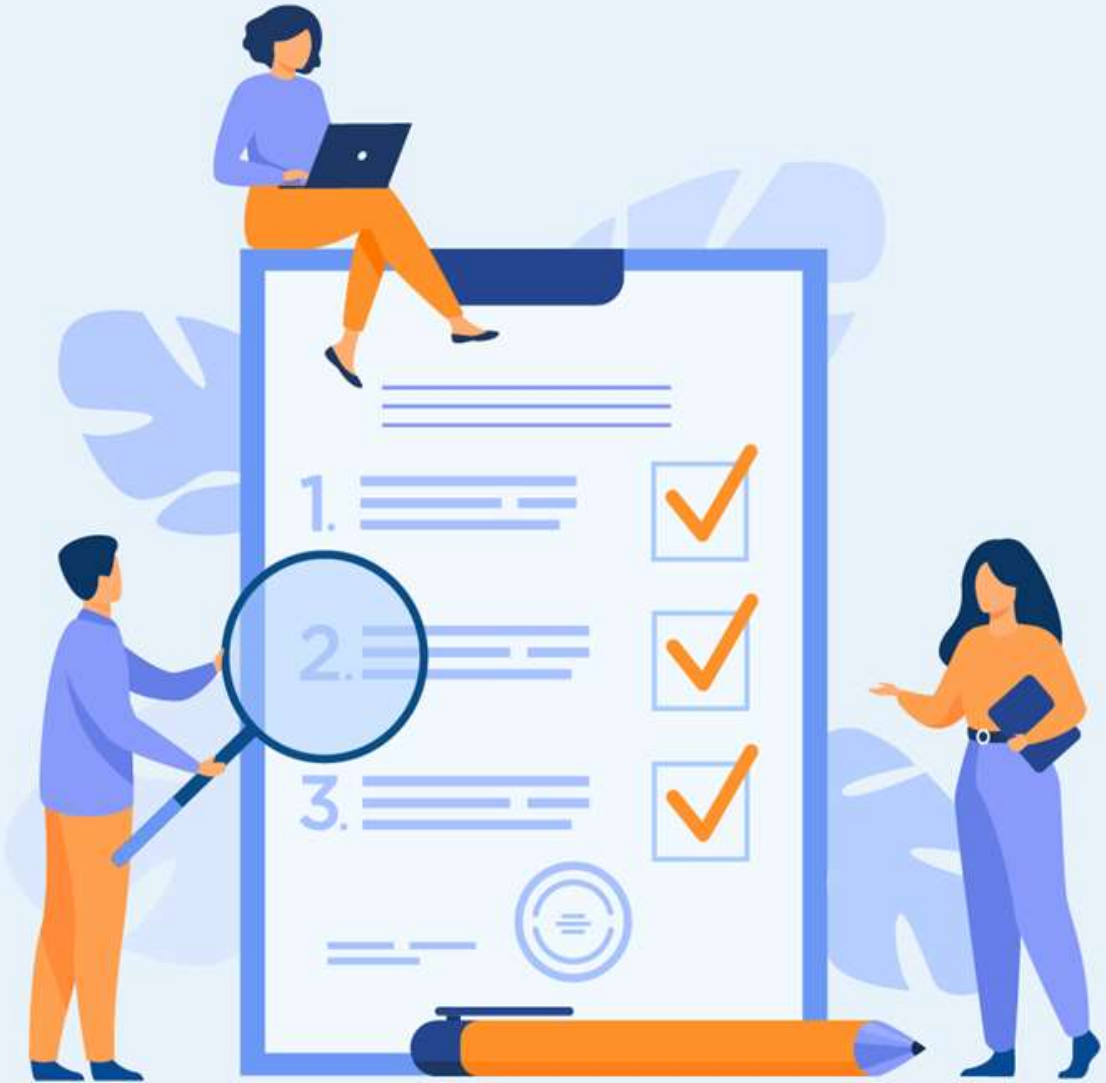
1. Principes et concepts fondamentaux
2. Compilations JIT (Just In Time) et AOT (Ahead Of Time)
3. **Conventions de codage, noms et ordonnancement**

Conventions de codage, noms et ordonnancement

Pour déclarer une variable et des méthodes dans la programmation de Dart, nous utilisons l'approche :

- **CamelCase :** Le premier mot de départ est mis en minuscule tandis que le mot suivant doit commencer par une majuscule (int testingNumbers = 1;)
- **PascalCase :** Tous les mots commencent par une majuscule (classe StudentsRecord {})
- **serpent_case :** On utilise des mots minuscules joints par une barre oblique _ (test_user_data .dart)
- **STREAMING_CASE :** Toutes les lettres sont en majuscules et les mots sont joints avec la barre oblique descendante _const (USER_ID = 1)

Copyright - Tout droit réservé - OFPPT



Chapitre 2

Programmation orientée objet avec Dart

Ce que vous allez apprendre dans ce chapitre :

- Classes
- Constructeurs
- Héritage
- Gestion des erreurs

05 heures

OFPPT
La Voie de l'Avenir

WEBFORCE
BE THE CHANGE



Chapitre 2

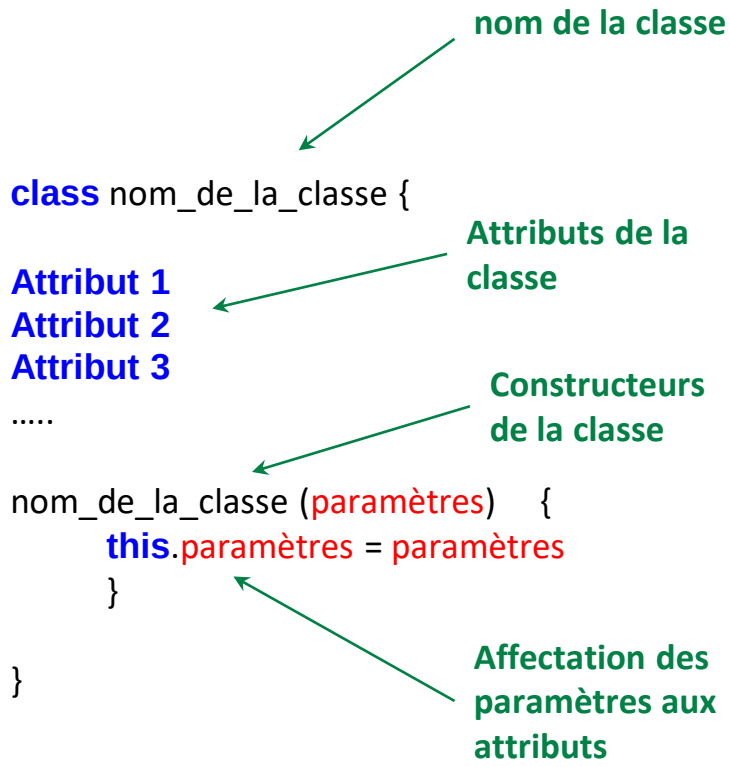
Programmation orientée objet avec Dart

1. **Classes**
2. Constructeurs
3. Héritage
4. Gestion des erreurs

Structure générale d'une classe Dart

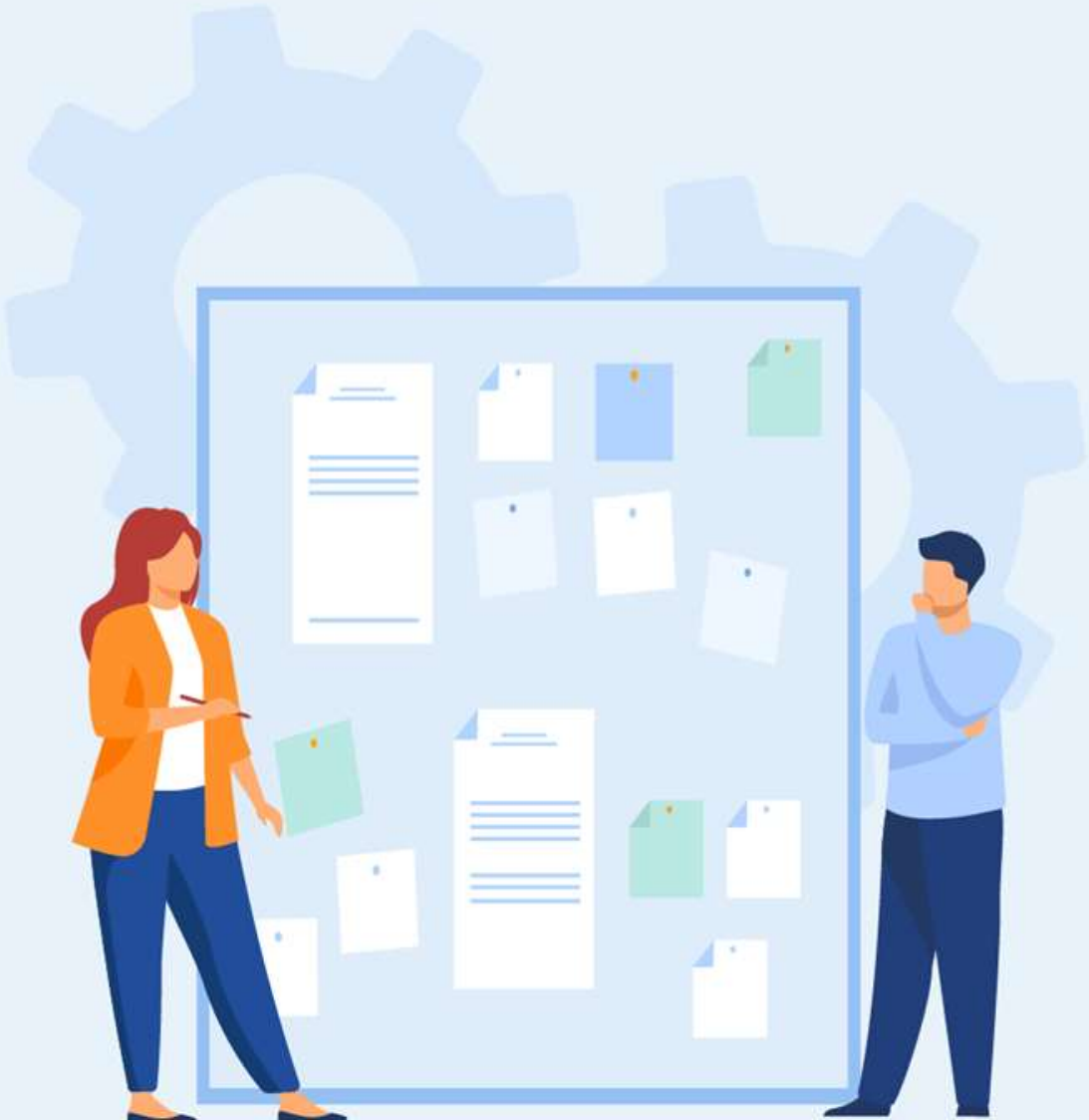
Une classe **Dart** possède exactement la même structure qu'une classe Java :

1. **Déclaration** à l'aide de l'instruction **class**
2. **Déclaration des attributs** à l'aide des mots clé: **String, int, var, final**...
3. **Déclaration** du **constructeur** sous le **même nom** de la **classe.**
4. **Affectation des paramètres aux attributs** via l'instruction **this.**
5. **Instanciation** via la commande **new**



PARTIE 2

Copyright - Tout droit réservé - OFPPT



Chapitre 2

Programmation orientée objet avec Dart

1. Classes
2. **Constructeurs**
3. Héritage
4. Gestion des erreurs

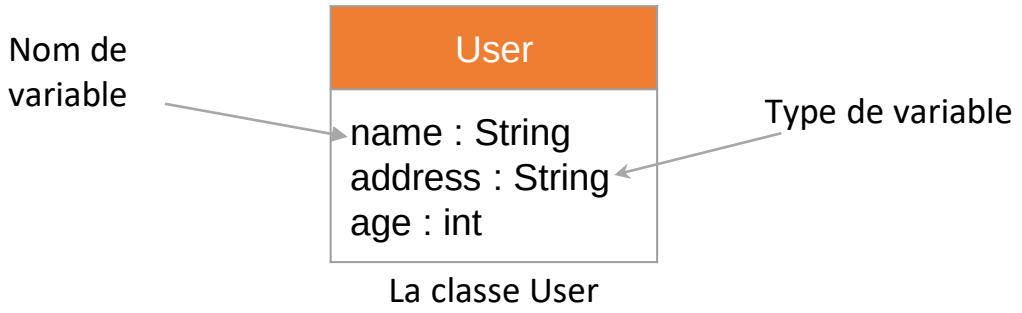
Constructeurs

Déclarez un constructeur en créant une fonction avec le même nom que sa classe (plus, évidemment, un identifiant supplémentaire).

La forme la plus courante de constructeur, le constructeur générateur qui crée une nouvelle instance d'une classe :

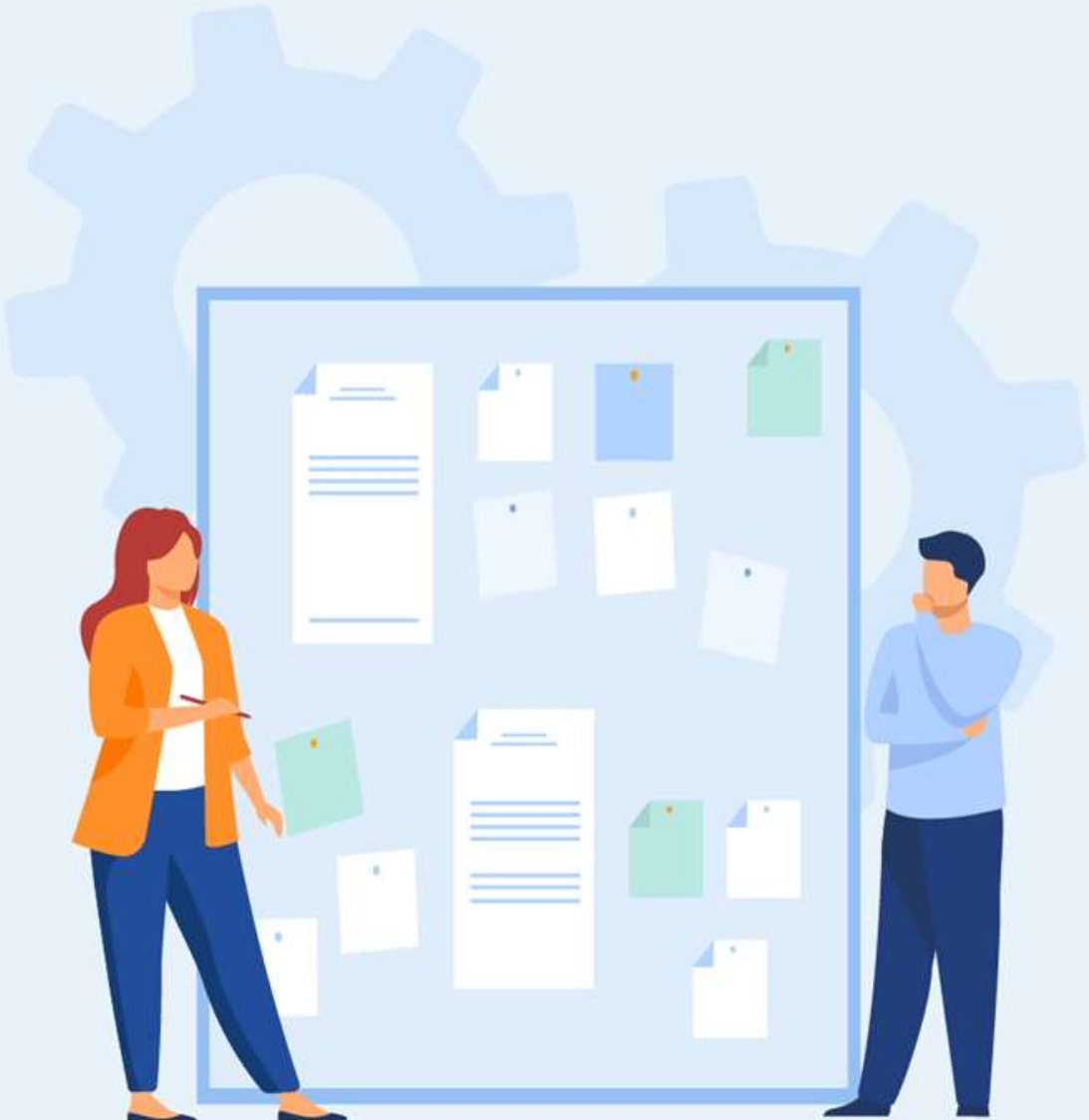
```
class User {
  String name;
  String address;
  int age;

  User(String name, String address,
  int age) {
    this.name = name;
    this.address = address;
    this.age = age;
                              }
     }
```



Le mot-clé `this` fait référence à l'instance actuelle.

Copyright - Tout droit réservé - OFPPT



Chapitre 2

Programmation orientée objet avec Dart

1. Classes
2. Constructeurs
3. **Héritage**
4. Gestion des erreurs

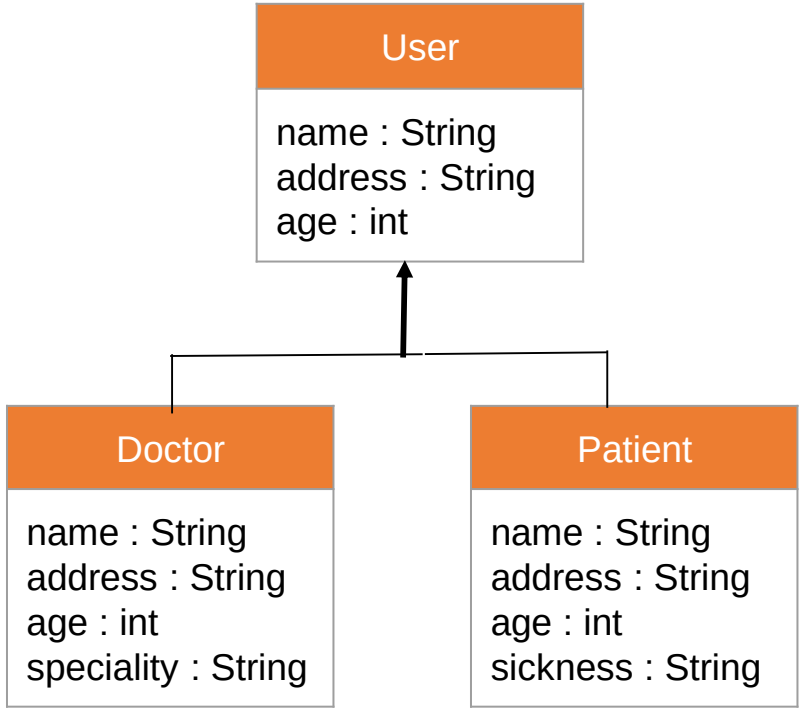
Héritage

Dart supporte l'héritage, il n'y a pas d'héritage multiple. Pour qu'une classe hérite d'une autre, on utilise le mot clé **extends**. Reprenant l'exemple de notre classe User, nous allons créer une classe **Doctor** pour ajouter "**speciality**" et la classe “Patient” pour ajouter “**sickness**”.

```
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);
}

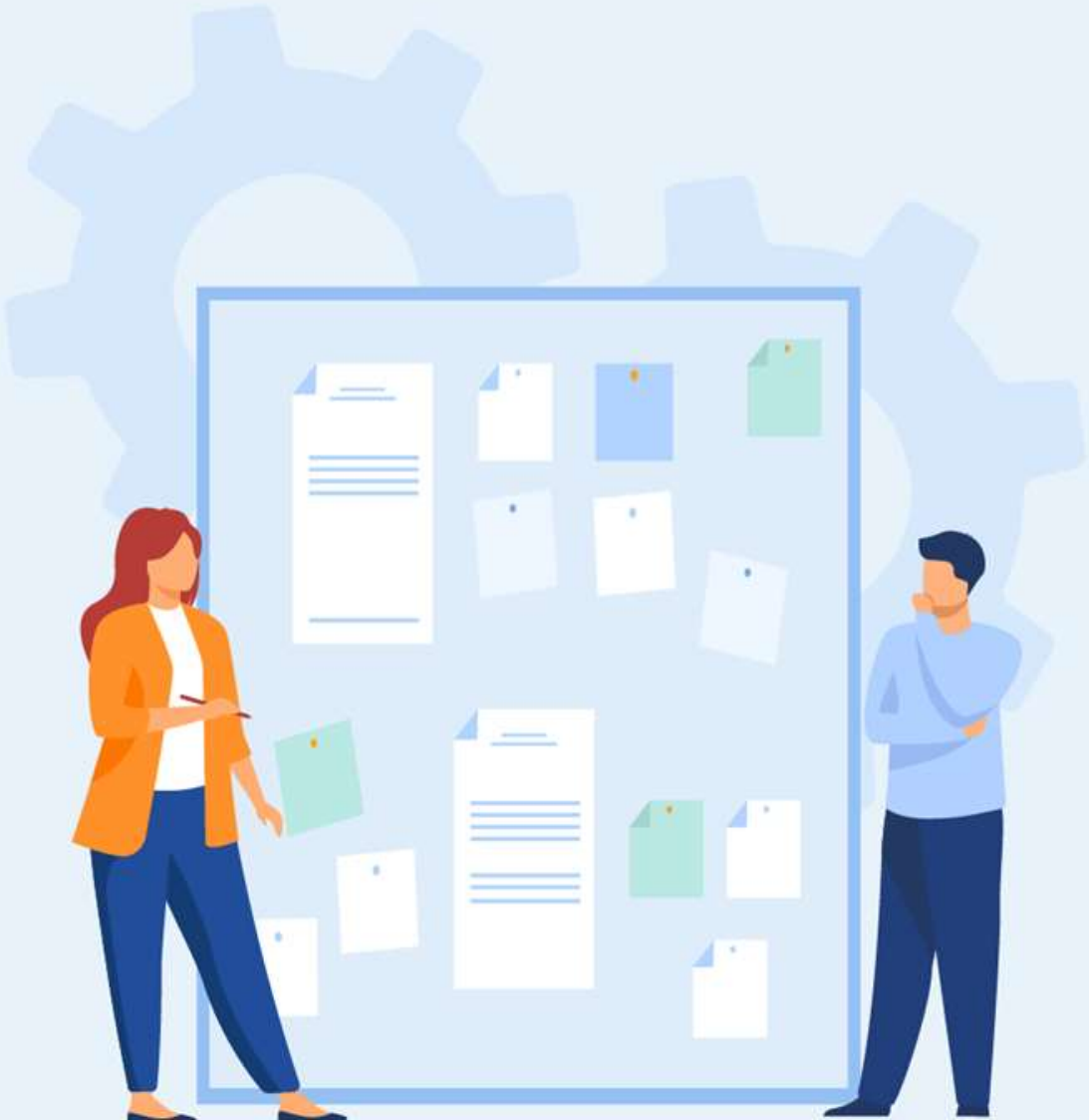
main() {
            Doctor doc = new Doctor(“Doc”, “Rabat”, 36, “Ortho”);
            Patient pat = new Patient(“Pat”, “Casa”, 54, “headache”);
}
```



Les classes Doctor et Patient héritent de la classe User

PARTIE 2

Copyright - Tout droit réservé - OFPPT



Chapitre 2

Programmation orientée objet avec Dart

1. Classes
2. Constructeurs
3. Héritage
4. **Gestion des erreurs**

Erreur vs assertions vs exception

Nous avons trois types d'erreurs dans Dart :

- **Les erreurs** sont utilisées pour faire savoir à un consommateur d'une API ou d'une bibliothèque qu'il l'utilise mal. Cela signifie que les erreurs ne doivent pas être gérées et doivent planter votre application.
- **Les assertions** sont similaires aux erreurs et sont utilisées pour signaler des états incorrects qui ne devraient jamais se produire.
- **Les exceptions** concernent un mauvais état attendu qui peut se produire lors de l'exécution. Étant donné que des exceptions sont attendues, vous devez les intercepter et les gérer de manière appropriée.

Copyright - Tout droit réservé - OFPPT

Gestion des erreurs

Attraper une erreur

- Pour attraper une erreur, nous devons développer

la méthode qui jette l'erreur dans un **Try** bloc.

Le **Try** bloc doit suivre un **catch** qui a un paramètre d'exception de type objet.

Exemple :

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

- Dart nous permet aussi d'exécuter un code appelé qu'une erreur se soit produite ou non. Pour cela, nous pouvons utiliser le **finally** bloc.

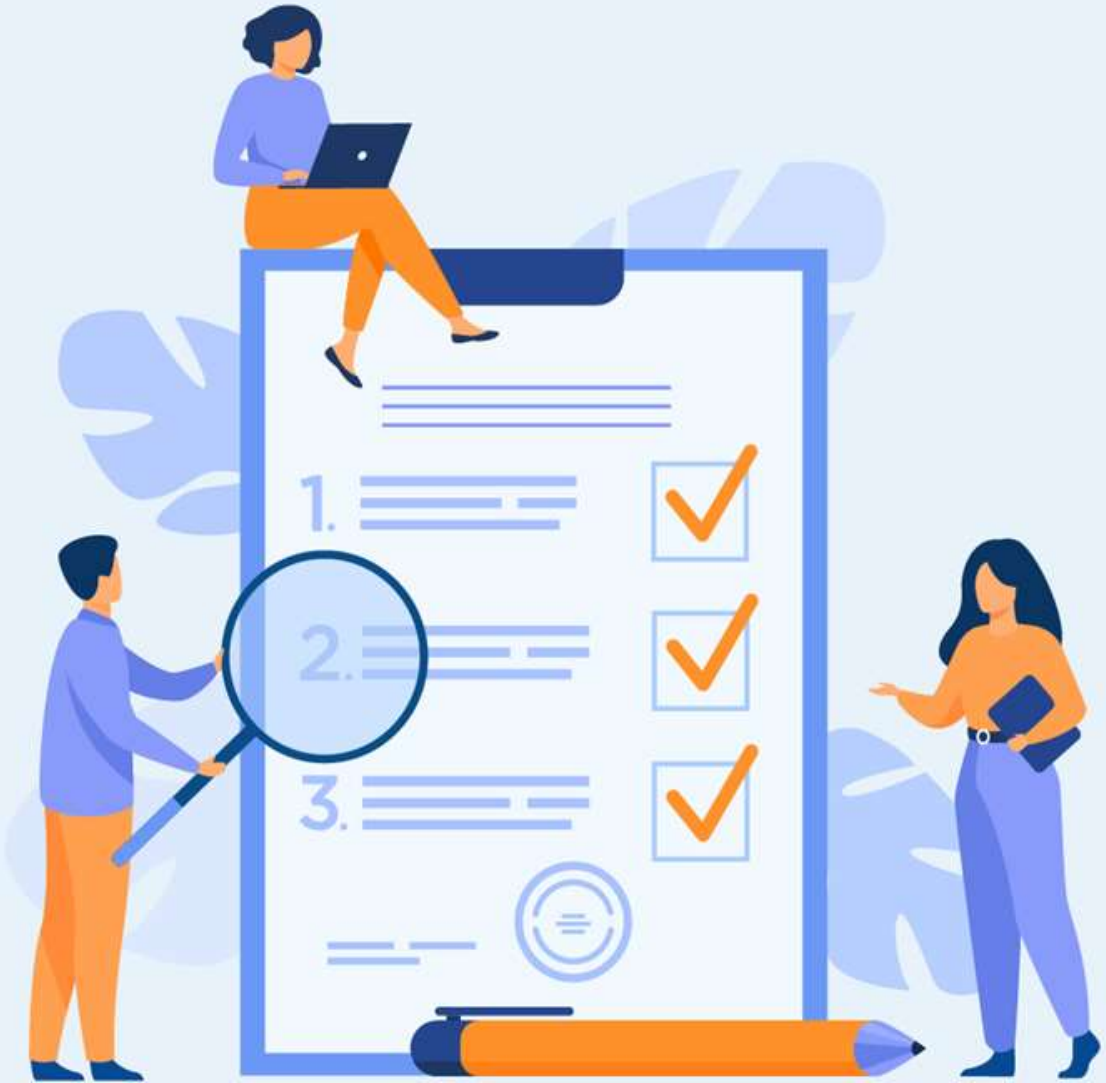
Le **finally** bloc est par exemple souvent utilisé pour arrêter une sorte d'animation de chargement qui s'affichait pendant l'appel.

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


Copyright - Tout droit réservé - OFPPT

OFPPT
La Voie de l'Avenir

WEBFORCE
BE THE CHANGE



Chapitre 3

Utilisation des widgets de base

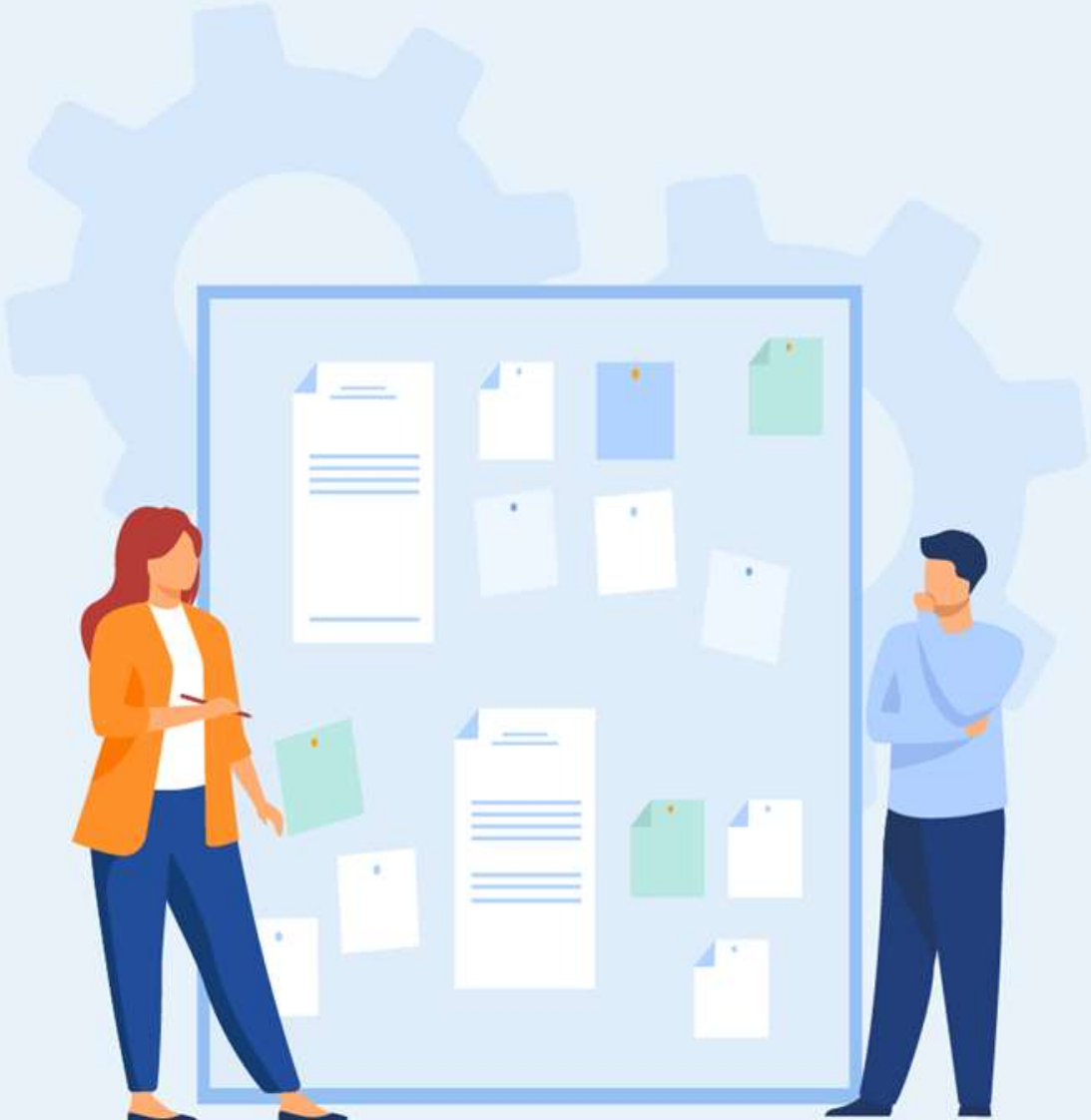
Ce que vous allez apprendre dans ce chapitre :

- Cycle de vie d'un widget
- Widgets de base : Text, Row, Column, Stack et Container
- Widgets Material Design
- Avantages du Hot Reload et Hot Restart

05 heures

OFPPT
La Voie de l'Avenir

WEBFORCE
BE THE CHANGE



Chapitre 3

Utilisation des widgets de base

1. **Cycle de vie d'un widget**
2. Widgets de base : Text, Row, Column, Stack et Container
3. Widgets Material Design
4. Avantages du Hot Reload et Hot Restart

Widget

Dans Flutter, tout ce qui apparaît dans l'interface utilisateur s'appelle widget et ils héritent tous de la classe **Widget**.

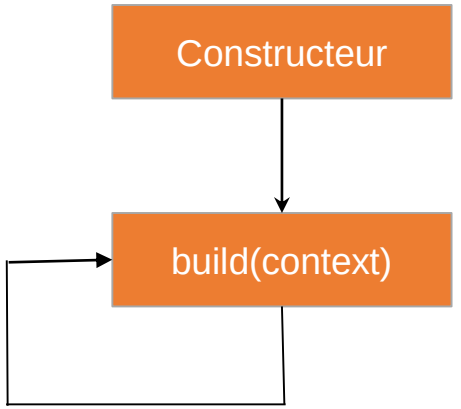
Lorsque vous créez une interface utilisateur dans Flutter, vous le faites en attachant chaque widget à l'écran de l'application afin qu'il s'adapte exactement à l'endroit où vous le souhaitez.

Les widgets Flutter sont organisés dans une arborescence de parents-enfants, pour former ce que vous voyez à l'écran. Cette arborescence est tous les widgets que vous utilisez pour construire l'application, c'est-à-dire que le code que vous écrivez créera cette structure.

Copyright - Tout droit réservé - OFPPT

Cycle de vie d'un Stateless widget

Si votre classe s'étend de Stateless Widget, cela signifie que cette page crée simplement des conceptions de widgets et ne les modifie pas.

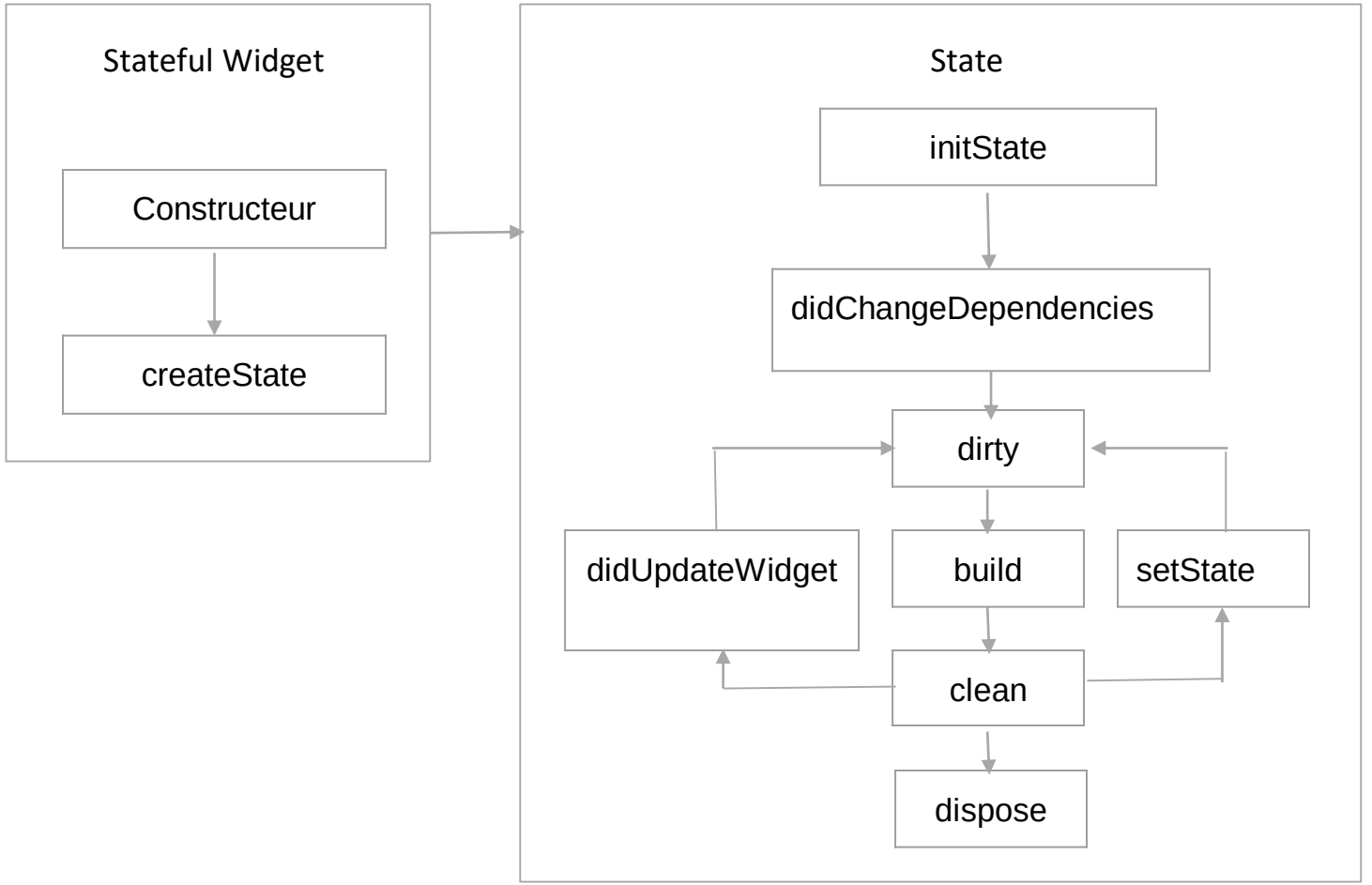


Cycle de vie de Stateless Widget

Copyright - Tout droit réservé - OFPPT

Cycle de vie d'un Stateful widget

Si votre classe s'étend de Stateful Widget, cela signifie que cette page crée des conceptions de widgets et ne les modifie pas.



Cycle de vie de Stateful Widget

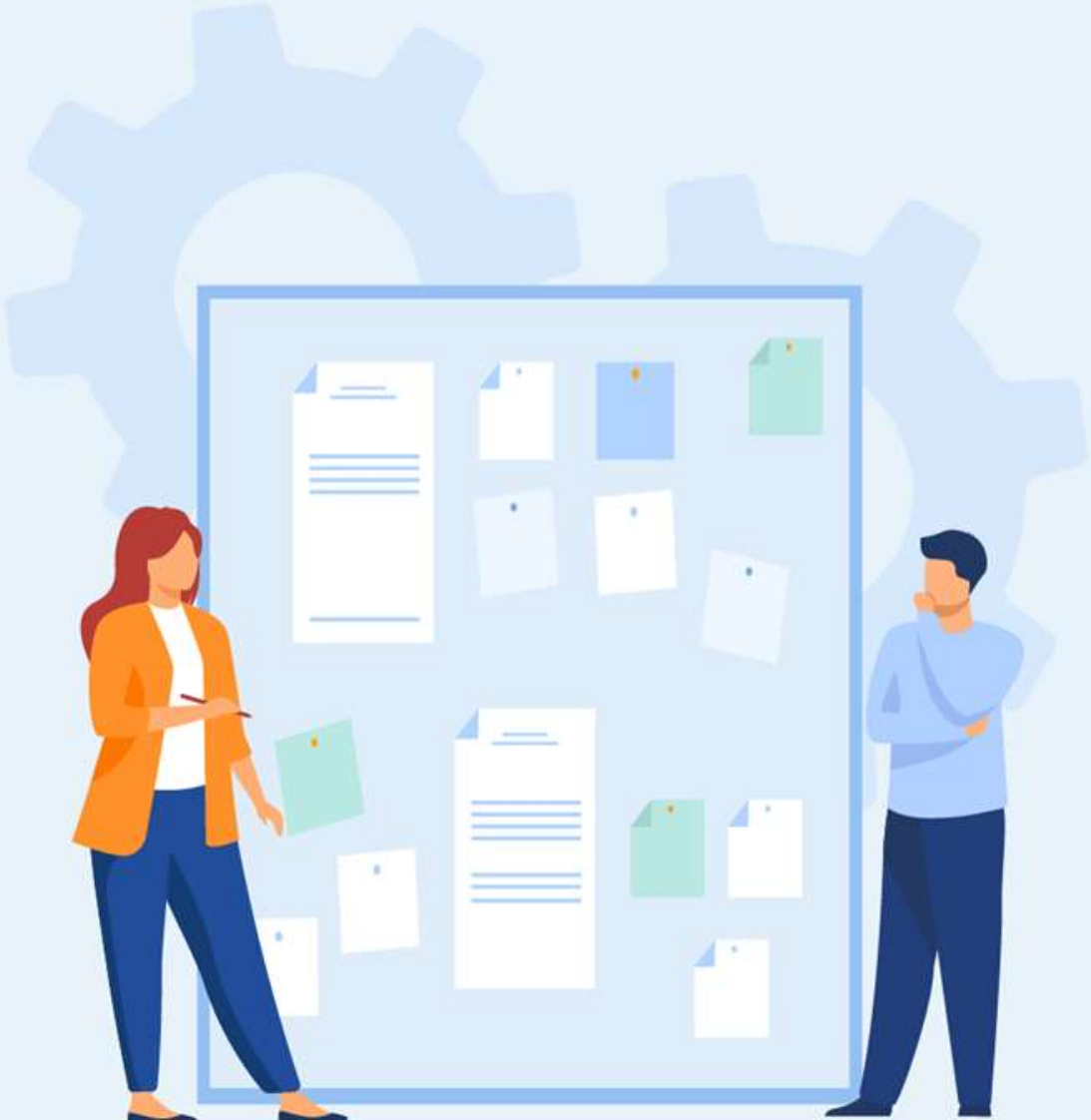
Copyright - Tout droit réservé - OFPPT

Cycle de vie d'un widget

Les widgets de Flutter sont associés aux méthodes de cycle de vie suivants :

1. **CreateState** : est appelée lorsque nous créons un autre widget avec état.
2. **InitState** : est appelée lorsque cet objet est inséré dans l'arbre. Ici, nous pouvons initialiser nos variables, objets, flux, AnimationController, etc.
3. **Build** : est appelée à chaque fois que le widget est reconstruit. Cela peut être après l'appel à initState(), didChangeDependencies(), didUpdateWidget() ou lorsque l'état est modifié via un appel à setState.
4. **DidChangeDependencies** : est appelée immédiatement après initState() et lorsque la dépendance de l'objet State est modifiée par InheritedWidget.
5. **DidUpdateWidget** : est appelée chaque fois que la configuration du widget change. Un cas typique est celui où un parent passe une variable au widget enfants via le constructeur.
6. **Deactivate** : est appelée lorsque l'objet est supprimé de l'arborescence.
7. **Dispose** : est appelée lorsque cet objet est définitivement supprimé de l'arborescence. Ici, nous devons libérer toutes les ressources retenues par cet objet comme l'arrêt de l'animation par exemple. Un exemple lorsque cette méthode est appelée, c'est en utilisant le **pushReplacement ()** du navigateur pour remplacer le widget actuel avec un nouveau.

Copyright - Tout droit réservé - OFPPT



Chapitre 3

Utilisation des widgets de base

1. Cycle de vie d’un widget
2. **Widgets de base : Text, Row, Column, Stack et Container**
3. Widgets Material Design
4. Avantages du Hot Reload et Hot Restart

Widgets de Base

- **Text**

Un widget vous permet de créer une série de textes stylés dans votre application.

- **Row, Column**

Ces widgets flexibles vous permettent de créer des mises en page flexibles dans les directions horizontale (Row) et verticale (Column). La conception de ces objets est basée sur le modèle de mise en page flexbox du Web.

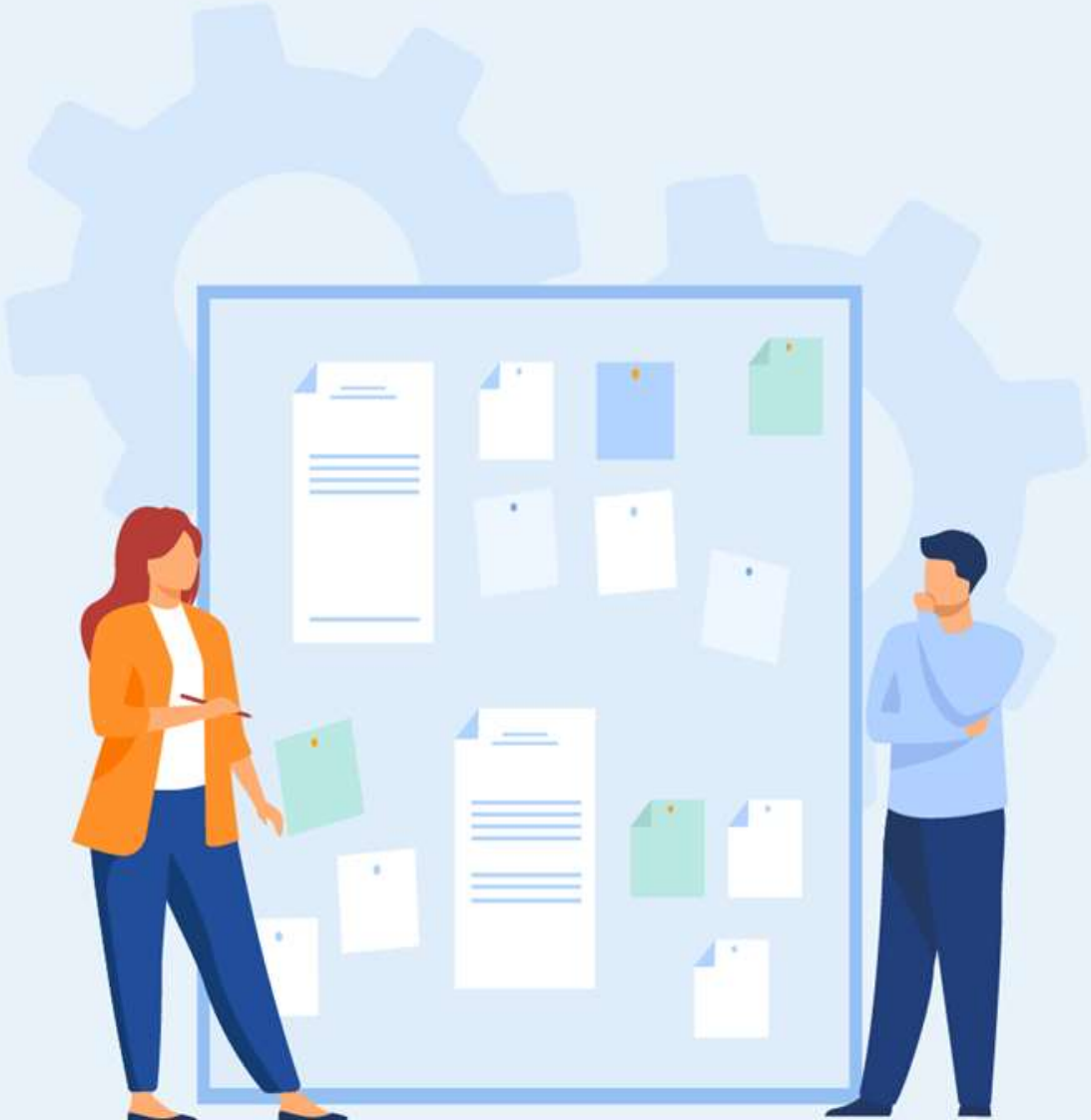
- **Stack**

Un Stack widget vous permet de placer des widgets les uns sur les autres dans l'ordre de la peinture. Vous pouvez ensuite utiliser le Positioned widget sur les enfants du Stack pour les positionner par rapport au bord supérieur, droit, inférieur ou gauche de la pile.
Les piles sont basées sur le modèle de disposition de positionnement absolu du Web.

- **Container**

Vous permet de créer un élément visuel rectangulaire, il peut être décoré avec un *BoxDecoration*, tel qu'un arrière-plan, une bordure ou une ombre. Un Container peut également avoir des marges, un rembourrage et des contraintes appliquées à sa taille. De plus, il peut être transformé dans un espace tridimensionnel à l'aide d'une matrice.

Copyright - Tout droit réservé - OFPPT



Chapitre 3

Utilisation des widgets de base

1. Cycle de vie d’un widget
2. Widgets de base : Text, Row, Column, Stack et Container
3. **Widgets Material Design**
4. Avantages du Hot Reload et Hot Restart

Material Design

Material Design est un système de conception open source conçu et pris en charge par les concepteurs et les développeurs de Google.

La dernière version, **Material 3**, permet des expériences personnelles, adaptatives et expressives, de la couleur dynamique et de l'accessibilité améliorée aux fondations pour les mises en page sur grand écran et les jetons de conception. Flutter est en cours de migration vers Material 3.



Remarques

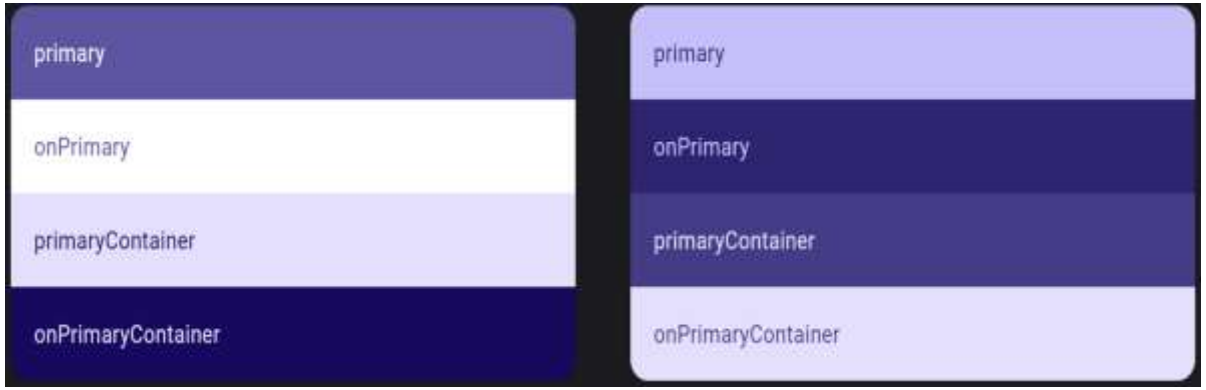
Quelques faits sur Material 3 (M3), la nouvelle génération de Material Design :

- La plupart des widgets Flutter ont déjà migrés vers M3.
- Vous pouvez vous inscrire au matériel 3 en utilisant le use Material3 flag.
- La grande majorité des widgets Flutter ont un comportement similaire dans M2 et M3. Cependant, quelques widgets ont un comportement sensiblement différent dans M3.
- Une fois la migration terminée, Material 3 deviendra l'apparence par défaut de la bibliothèque.
- La prise en charge du Material 2 sera éventuellement supprimée conformément à la politique de dépréciation de Flutter .

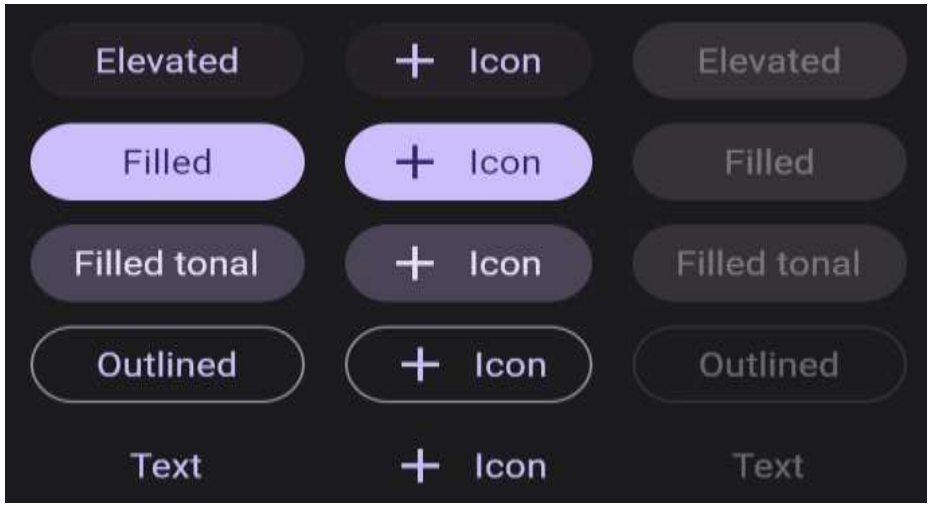
Copyright - Tout droit réservé - OFPPT

PARTIE 2

Material Design



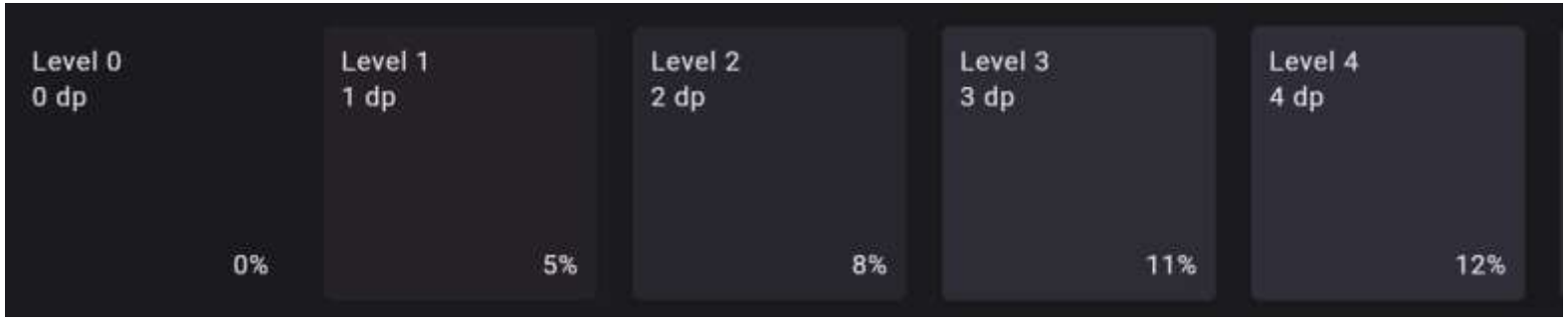
Couleurs



Boutons d'actions

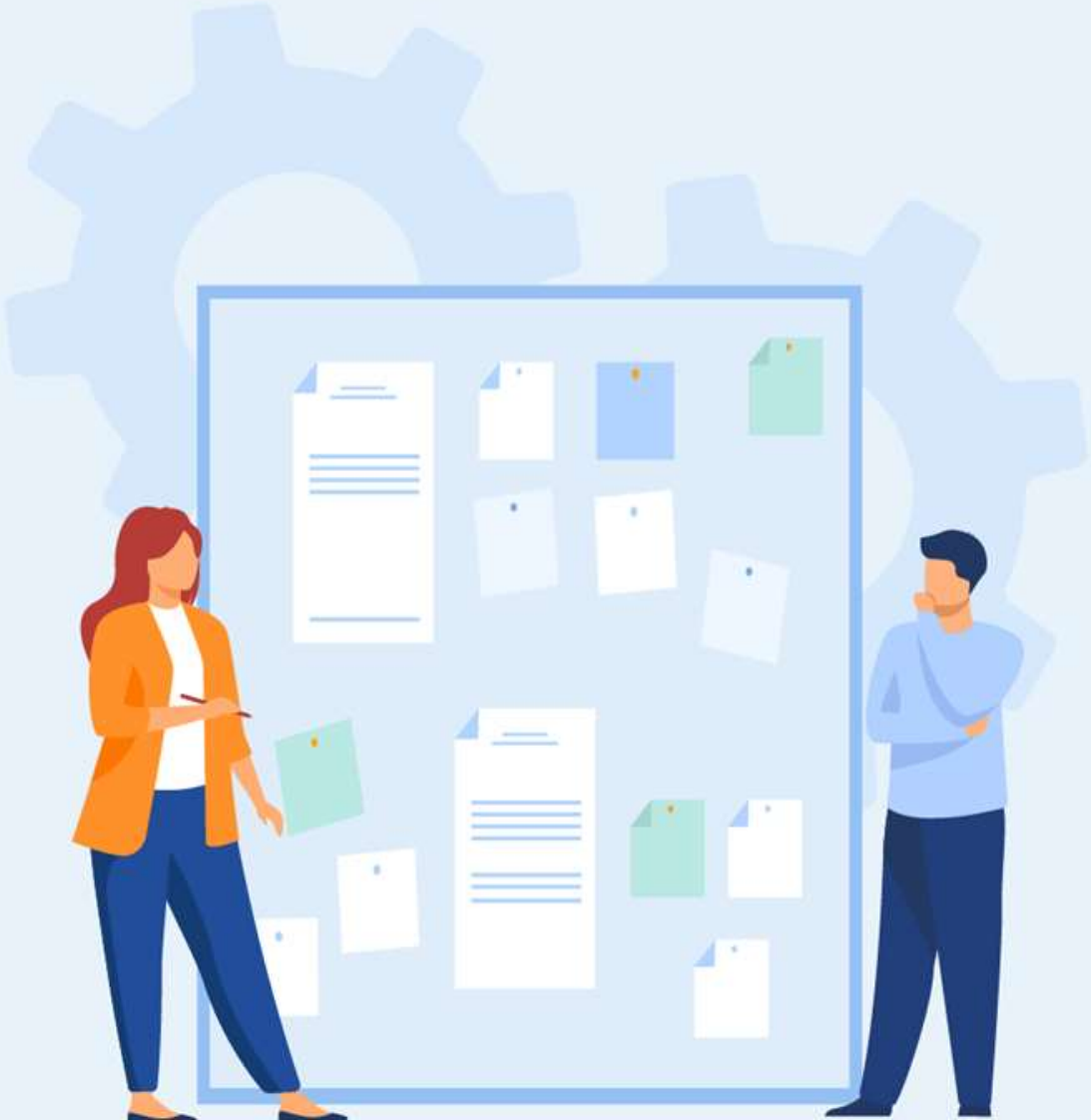


Typographie



Elevation

Copyright - Tout droit réservé - OFPPT



Chapitre 3

Utilisation des widgets de base

1. Cycle de vie d’un widget
2. Widgets de base : Text, Row, Column, Stack et Container
3. Widgets Material Design
4. **Avantages du Hot Reload et Hot Restart**

Avantages du Hot Reload

La fonction la plus simple et la plus rapide qui vous aide à :

- Appliquer des modifications.
- A corriger des bugs.
- A créer des interfaces utilisateur.
- A ajouter des fonctionnalités.

Il faut environ une seconde pour exécuter cette fonctionnalité, elle ne détruit pas l'état préservé.

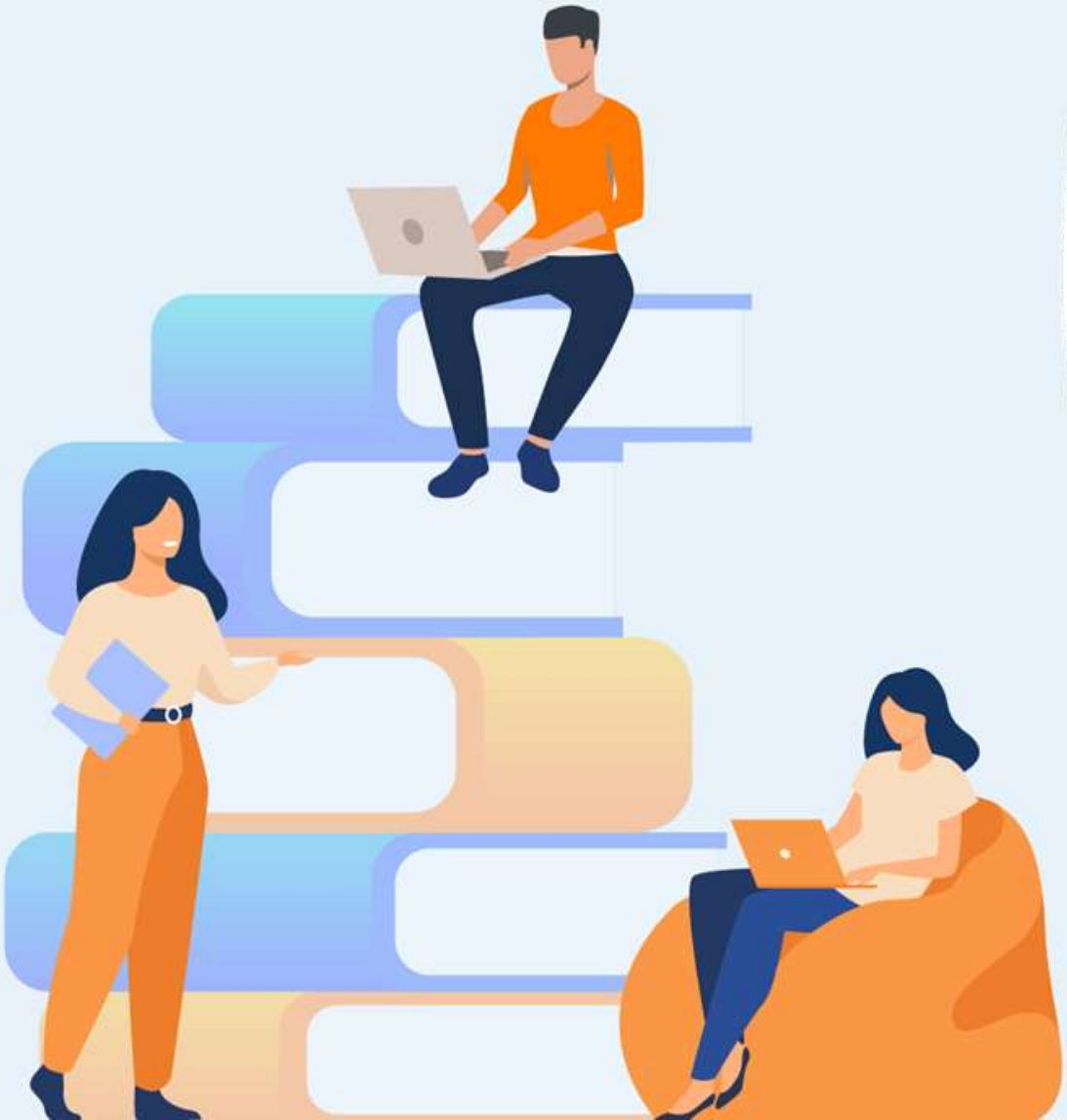
Copyright - Tout droit réservé - OFPPT

Avantages du Hot Restart

Une fonctionnalité légèrement différente par rapport à un Hot Reload.

- Il est plus rapide que la fonction de redémarrage complet.
- Il détruit les états préservés de notre application, et le code est à nouveau entièrement compilé et démarre à partir de l'état par défaut.
- Il prend plus de temps que le Hot Reload mais prend moins de temps que la fonction de redémarrage complet.

Copyright - Tout droit réservé - OFPPT



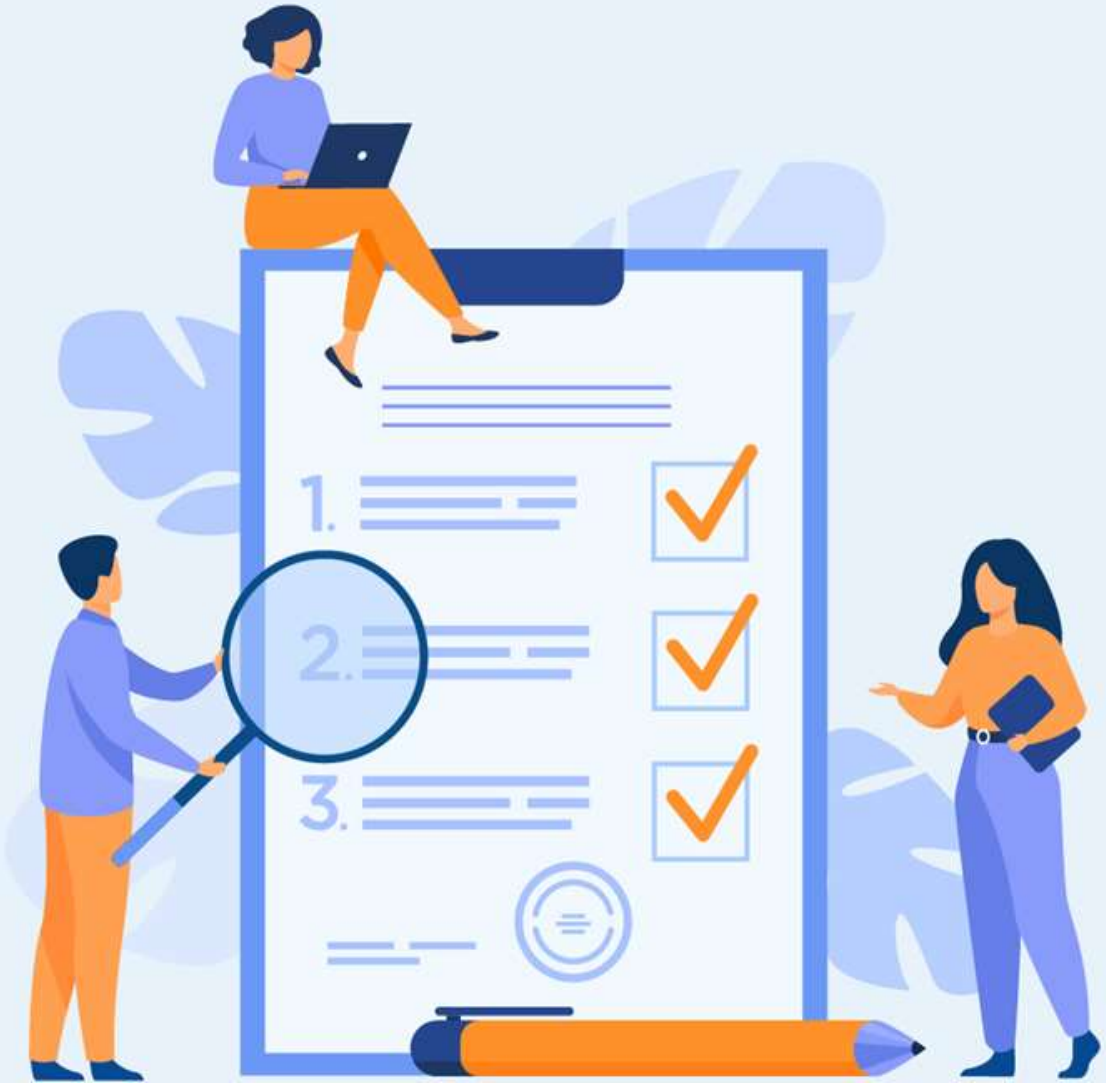
PARTIE 3

Manipuler les données

Dans ce module, vous allez :

- Apprendre à utiliser correctement le package http et la connexion à un serveur externe
- Apprendre à utilizer facile ment la bibliothèque dart:convert
- Apprendre à manipuler aisément des opérations CRUD et des interactions avec la base de données

10 heures



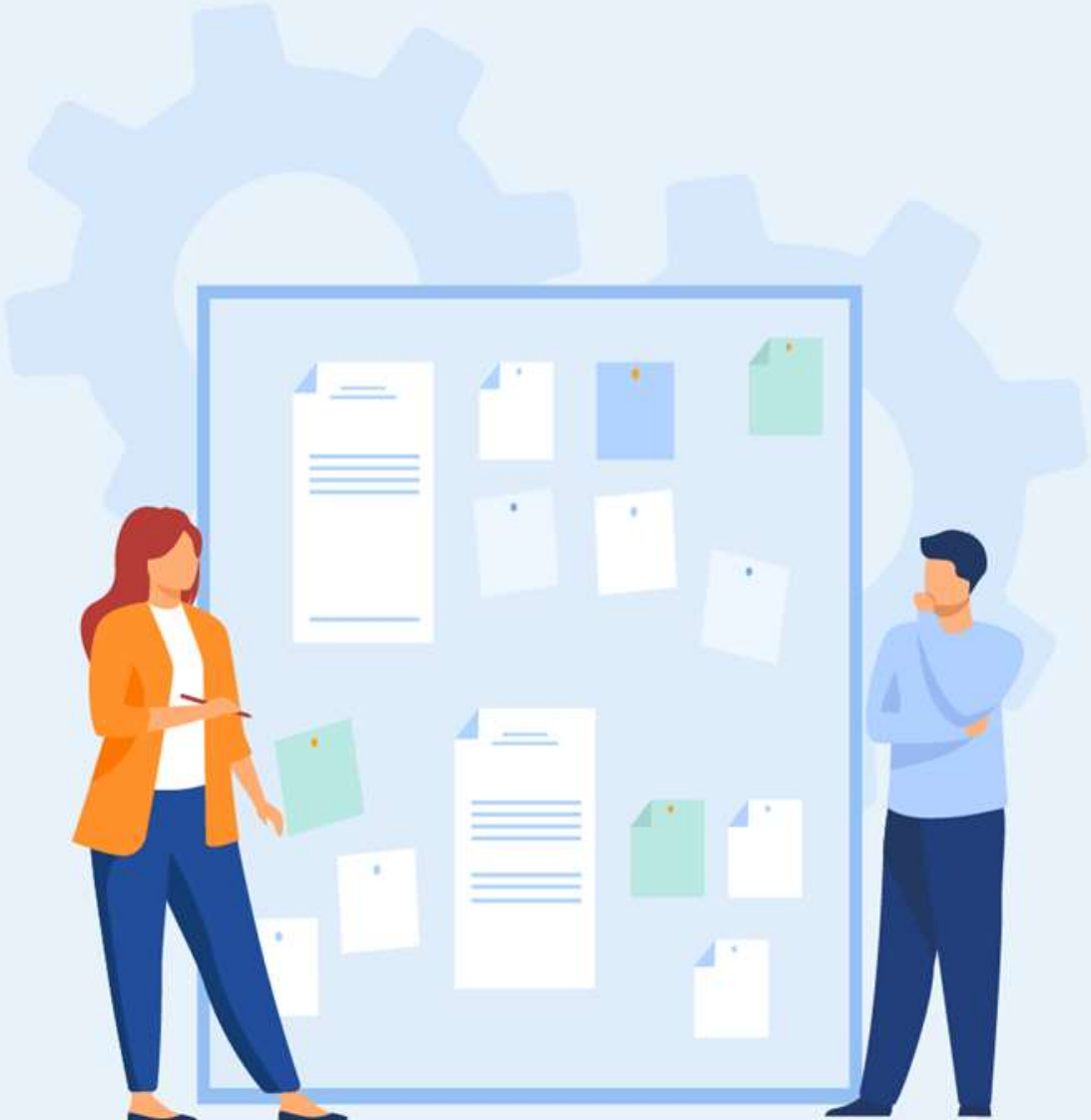
Chapitre 1

Utiliser le package http

Ce que vous allez apprendre dans ce chapitre :

- Bibliothèque dart:convert et parser du JSON
- API Request

05 heures



Chapitre 1

Utiliser le package http

1. **Bibliothèque dart:convert et parser du JSON**
2. API Request

Bibliothèque dart:convert

La bibliothèque **dart:convert** (Référence API) possède des convertisseurs pour JSON et UTF-8, ainsi que la prise en charge de la création des convertisseurs supplémentaires.

JSON est un format de texte simple pour représenter des objets structurés et des collections.

Url : https://pub.dev/packages/convert

UTF-8 est un codage commun à largeur variable qui peut représenter chaque caractère du jeu de caractère Unicode.

Pour utiliser cette bibliothèque, importez dart:convert.

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


Copyright - Tout droit réservé - OFPPT

Parser du JSON

Décode une chaîne encodée en JSON en un objet Dart avec jsonDecode():

```
import 'dart:convert';

void main()
{
var jsonString = '''
 [
   {"score": 40},
   {"score": 80}
 ]
''';

 var List = jsonDecode(jsonString);
 for(var item in List)
   print(item['score']);
}
```

jsonDecode() : permet de décoder une chaîne de caractères.

Copyright - Tout droit réservé - OFPPT

Parser du JSON

Encodez un objet Dart pris en charge dans une chaîne au format JSON avec jsonEncode() :

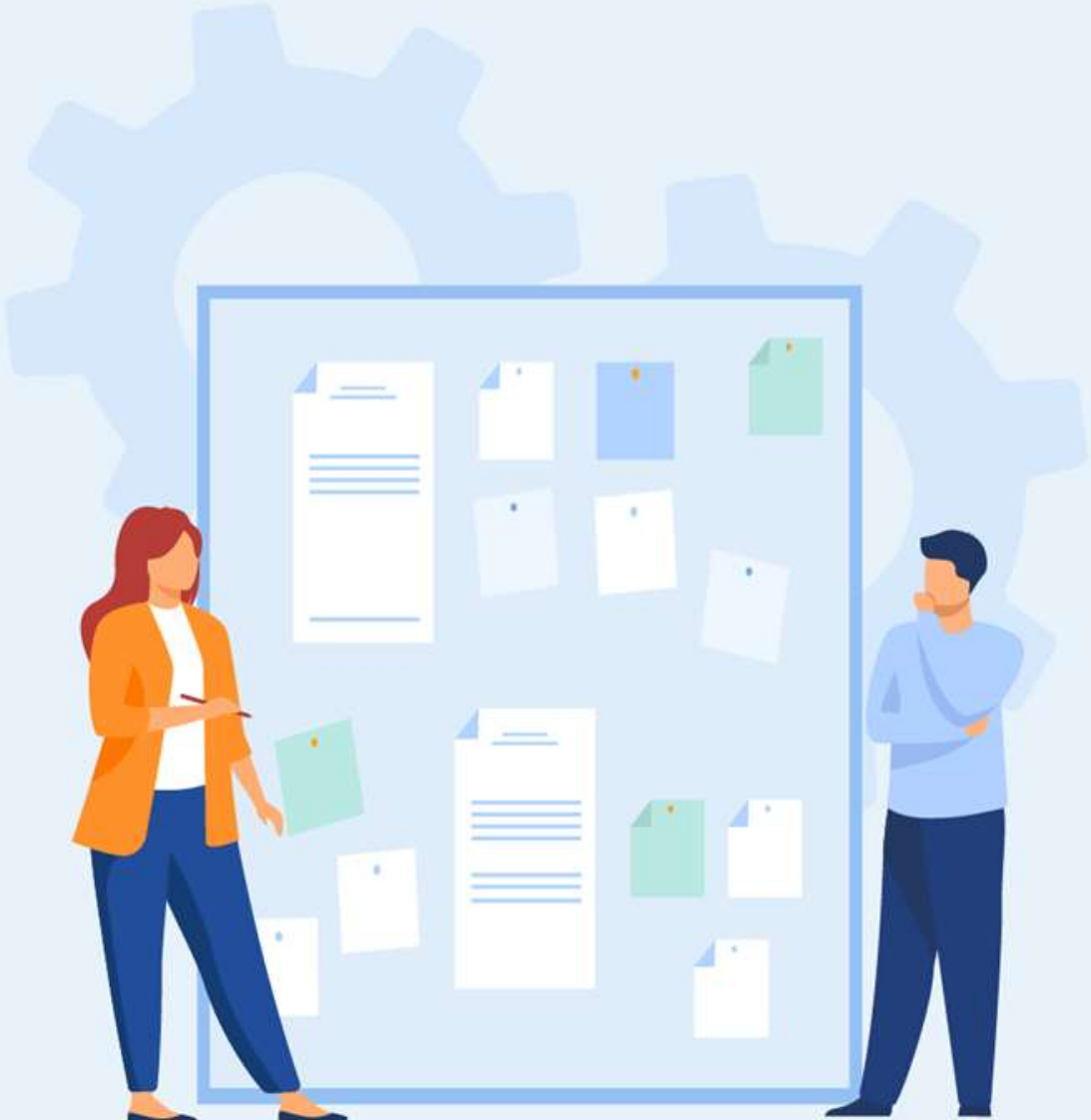
```
import 'dart:convert';

void main()
{
 var scores = [
   {'score': 40},
   {'score': 80},
   {'score': 100, 'overtime': true, 'special_guest': null}
 ];

 var jsonText = jsonEncode(scores);
 print(jsonText);
}
```

jsonEncode() : permet de coder une chaîne de caractères.

Copyright - Tout droit réservé - OFPPT



Chapitre 1

Utiliser le package http

1. Bibliothèque dart:convert et parser du JSON
2. **API Request**

01 - Utiliser le package http

API Request

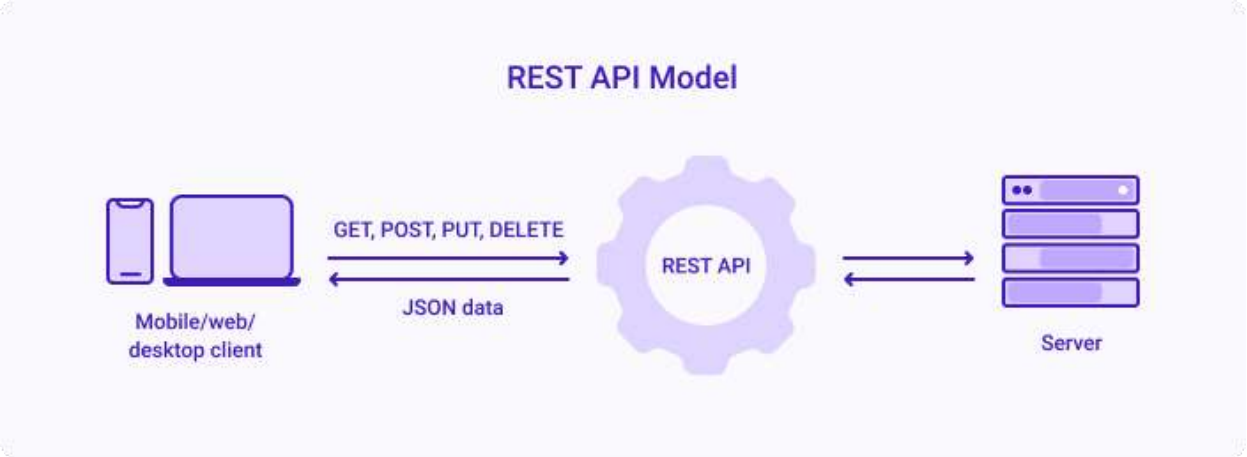
Récupérer les données sur Internet

La récupération des données sur Internet est nécessaire pour la plupart des applications.

Heureusement, Dart et Flutter fournissent des outils, tels que le package **http**, pour ce type de travail.

Cette recette utilise les étapes suivantes :

1. Ajoutez le package **http**.
2. Effectuez une requête réseau à l'aide du http package.
3. Convertissez la réponse en un objet Dart personnalisé.
4. Récupérez et affichez les données avec Flutter.



Source : rlogical.com

PARTIE 3

Copyright - Tout droit réservé - OFPPT

Requête HTTP

- **Ajoutez le http paquet**

Le package http fournit le moyen le plus simple de récupérer des données sur Internet.

Pour installer le http package, ajoutez-le à la section des dépendances du fichier pubspec.yaml. Vous pouvez trouver la dernière version du http package dans pub.dev.

```
dependencies:
  http: <latest_version>
```

- **Importez le package http.**

```
import 'package:http/http.dart' as http;
```

- **Ajoutez l'autorisation Internet, dans le fichier AndroidManifest.**

```
<!-- Required to fetch data from the internet. -->
<uses-permission android:name="android.permission.INTERNET" />
```


Copyright - Tout droit réservé - OFPPT

Manipuler le flux JSON

- **Faire une demande de réseau**

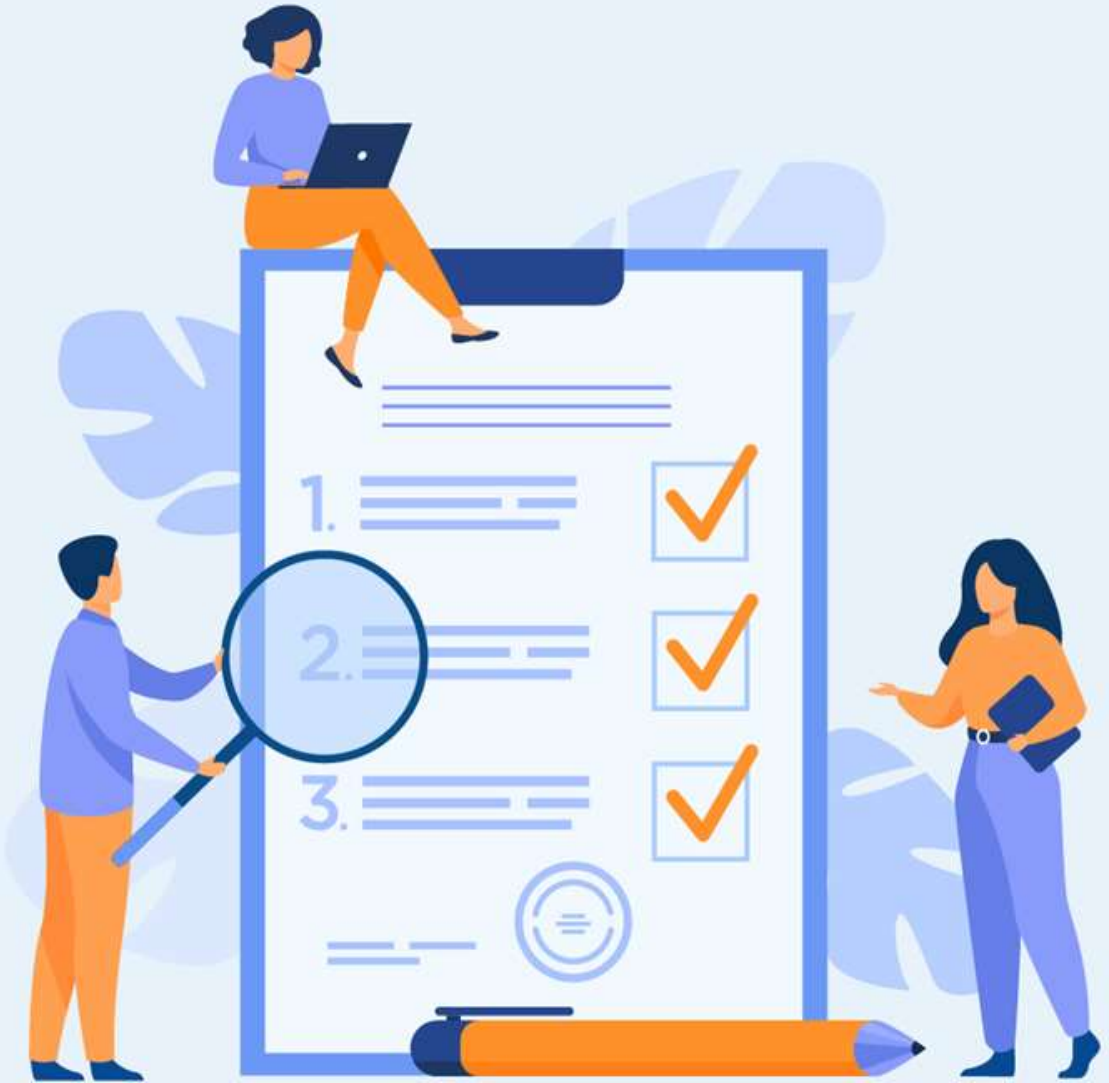
Cette recette explique comment récupérer un exemple d'album à partir du **JSONPlaceholder** à l'aide de la méthode **http.get()**.

```
Future<http.Response> fetchAlbum() {
  return http.get(Uri.parse('https://jsonplaceholder.typicode.com/albums/1'));
}
```

- **Convertissez la réponse en un objet Dart personnalisé**

Bien qu'il soit facile de faire une requête réseau, travailler avec un raw `Future<http.Response>` n'est pas très pratique. Pour vous faciliter la vie, convertissez le `http.Response` en un objet Dart.

Copyright - Tout droit réservé - OFPPT



Chapitre 2

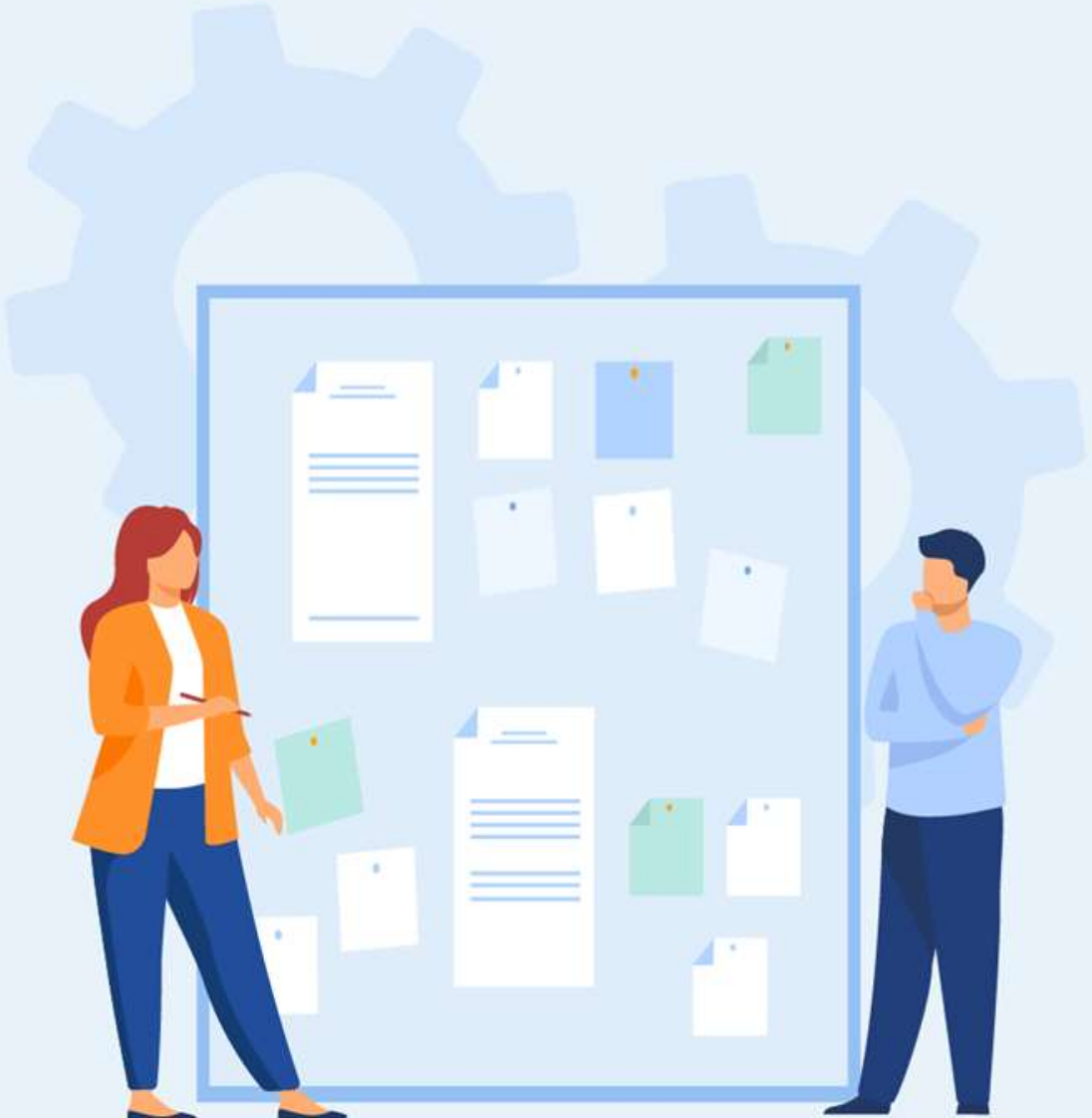
Utiliser SQLite et le plugin sqflite

Ce que vous allez apprendre dans ce chapitre :

- Les opérations CRUD et interactions avec la base de données
- SqfLite

- SQLite

05 heures



Chapitre 2

Utiliser SQLite et le plugin sqflite

1. **Opérations CRUD et interactions avec la base de données**
2. SqfLite
3. SQLite

Opérations CRUD

Si vous écrivez une application qui doit conserver et interroger de grandes quantités de données sur l'appareil local, envisagez d'utiliser une base de données au lieu d'un fichier local ou un magasin clé-valeur. En général, les bases de données fournissent des insertions, des mises à jour et des requêtes plus rapides par rapport aux autres solutions de persistance locales.

CRUD signifie créer, lire, mettre à jour et supprimer les quatre opérations essentielles de stockage persistant.

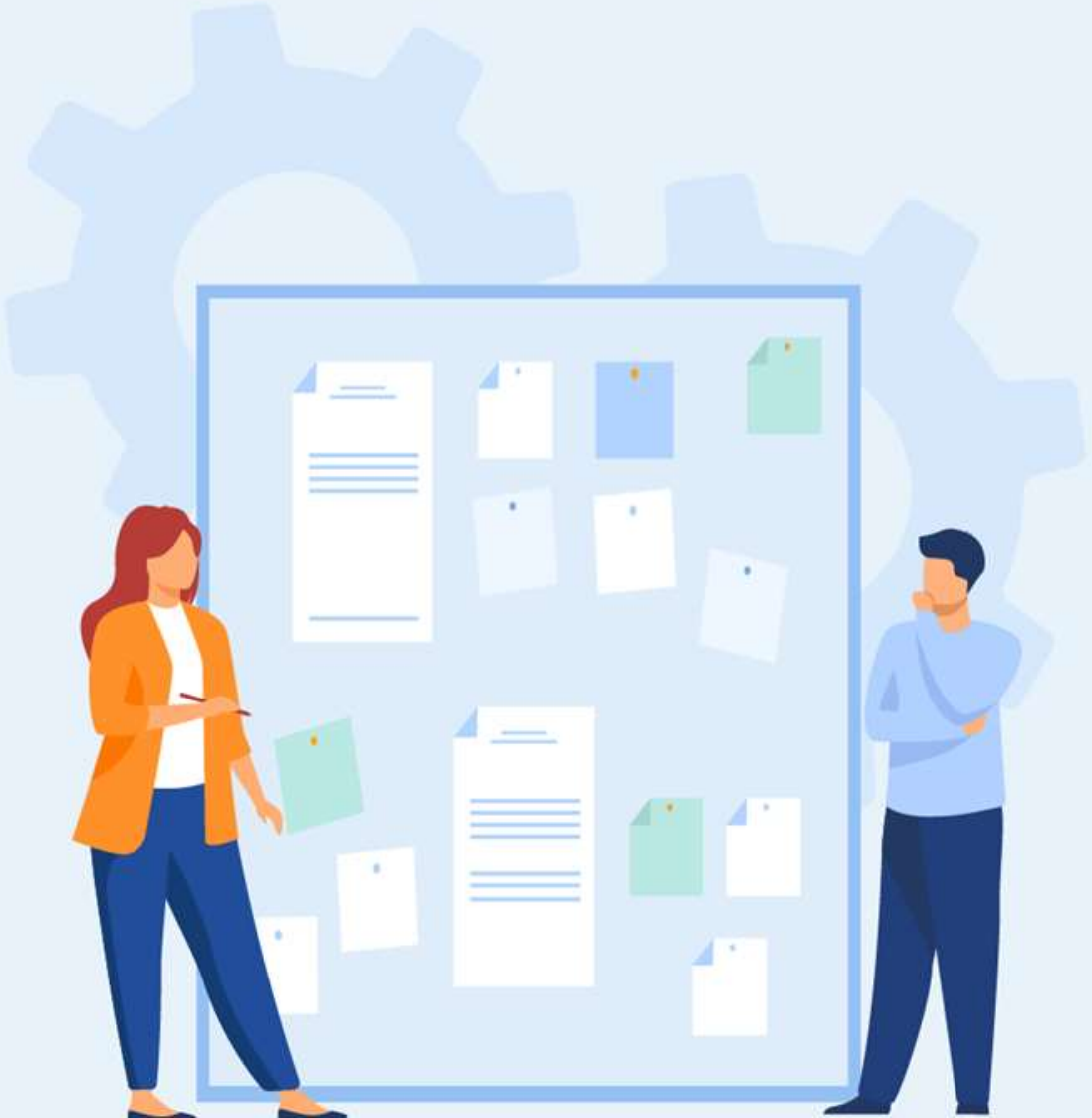
C → Create → POST

R → Read → GET

U → Update → PUT

D → Delete → DELETE

Copyright - Tout droit réservé - OFPPT



Chapitre 2

Utiliser SQLite et le plugin sqflite

1. Opérations CRUD et interactions avec la base de données
2. **SqfLite**
3. SQLite

SqfLite

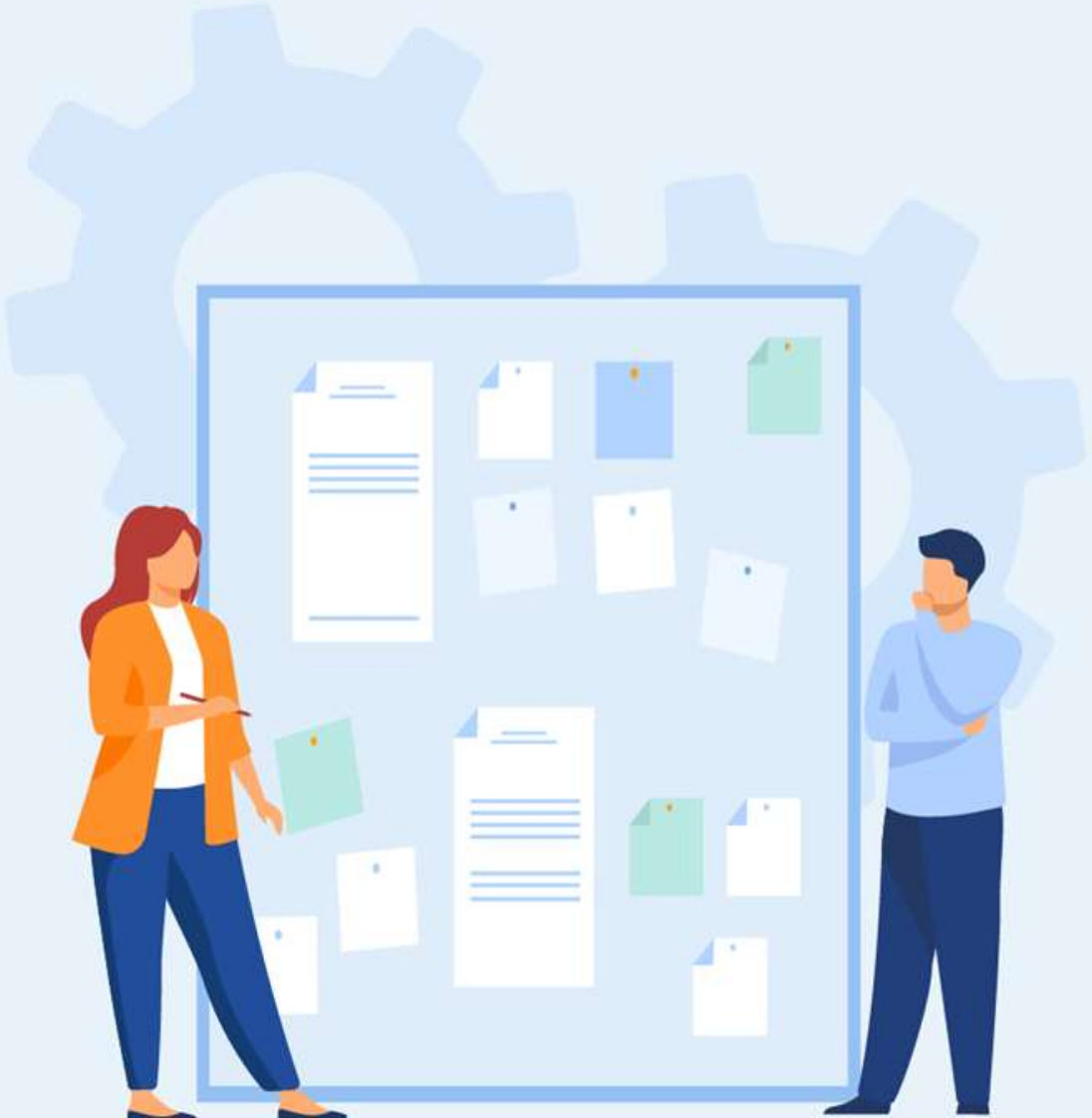
SqfLite : fournit des classes et des fonctions pour interagir avec une base de données SQLite.

Url : https://pub.dev/packages/sqflite

Pour intégrer sqflite à votre projet il faut :

- Ajouter les dépendances (sqflite, path_provider)
- créer le modèle de données
- Ajouter les importations
- Initialiser la base de données
- Insérer
- Lire
- Supprimer
- Mettre à jour

Copyright - Tout droit réservé - OFPPT



Chapitre 2

Utiliser SQLite et le plugin sqflite

1. Opérations CRUD et interactions avec la base de données
2. SqfLite

3. SQLite

SQLite

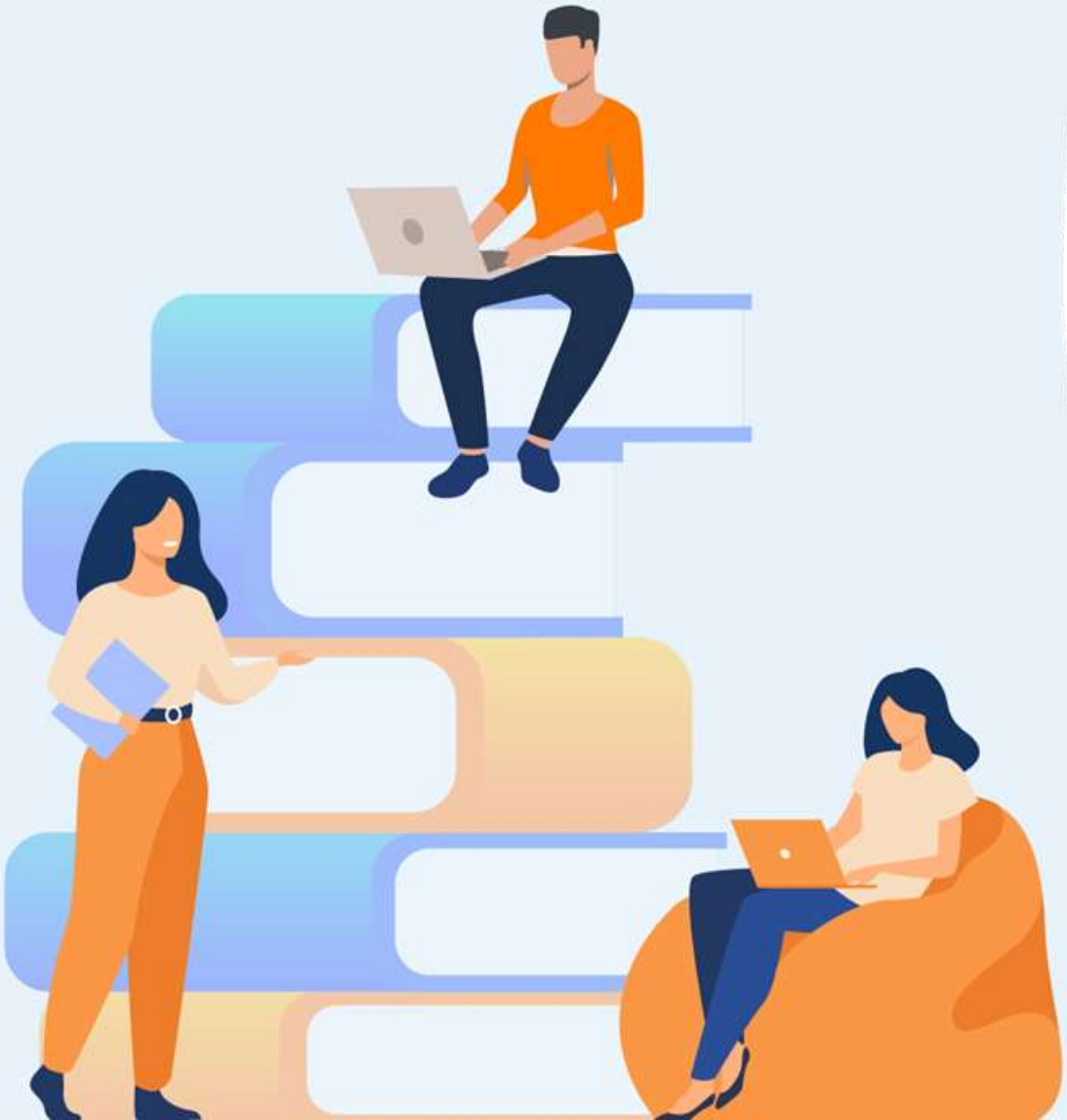
SQLite est une base de données relationnelle rapide qui peut être utilisée pour stocker des données hors ligne pour des applications mobiles.

Les applications Flutter peuvent utiliser les bases de données SQLite via le **sqfLite** plugin disponible sur **pub.dev**.



source : sqllite.com

Copyright - Tout droit réservé - OFPPT



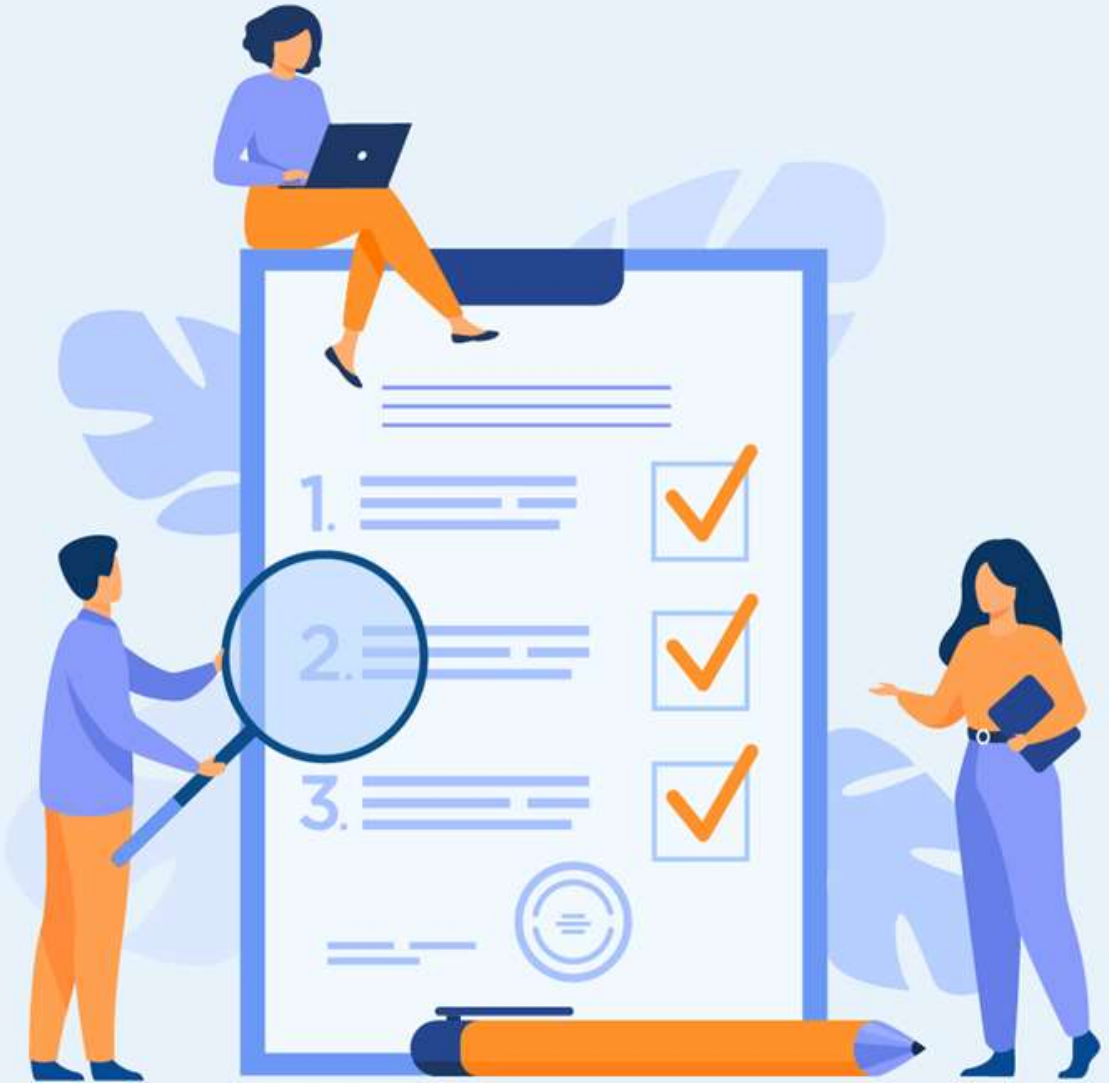
PARTIE 4

Maîtriser la Navigation et le routing

Dans ce module, vous allez apprendre :

- la création facile d’une nouvelle route
- Le passage correct des données d’un écran à un autre
- La navigation simple entre les routes

05 heures



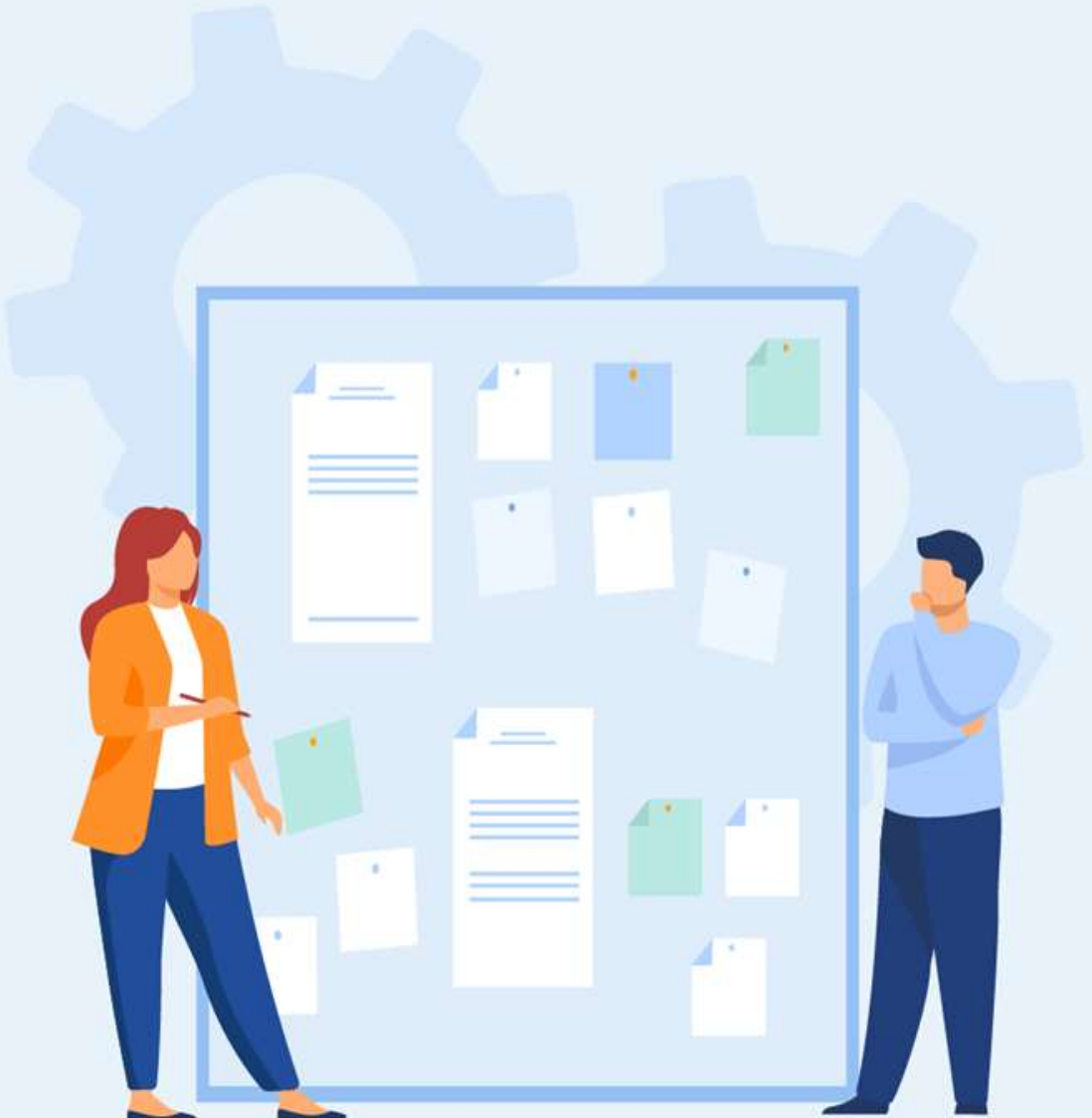
Chapitre 1

Gérer les états

Ce que vous allez apprendre dans ce chapitre :

- Stateful et stateless widgets
- Méthode setState()
- Widget interactif

03 heures



Chapitre 1

Gérer les états

1. **Stateful et Stateless widgets**
2. Méthode setState()
3. Widget interactif

Stateful widget

Un **widget avec état** est dynamique : par exemple, il peut changer d'apparence en réponse à des événements déclenchés par des interactions de l'utilisateur ou lorsqu'il reçoit des données. **Checkbox**, **Radio**, **Slider**, **InkWell**, **Form** et **TextField** sont des exemples de widgets avec état.



Widget

Copyright - Tout droit réservé - OFPPT

Stateless widget

Un **widget sans état** est statique, il ne stocke aucun état ou une valeur susceptible de changer.

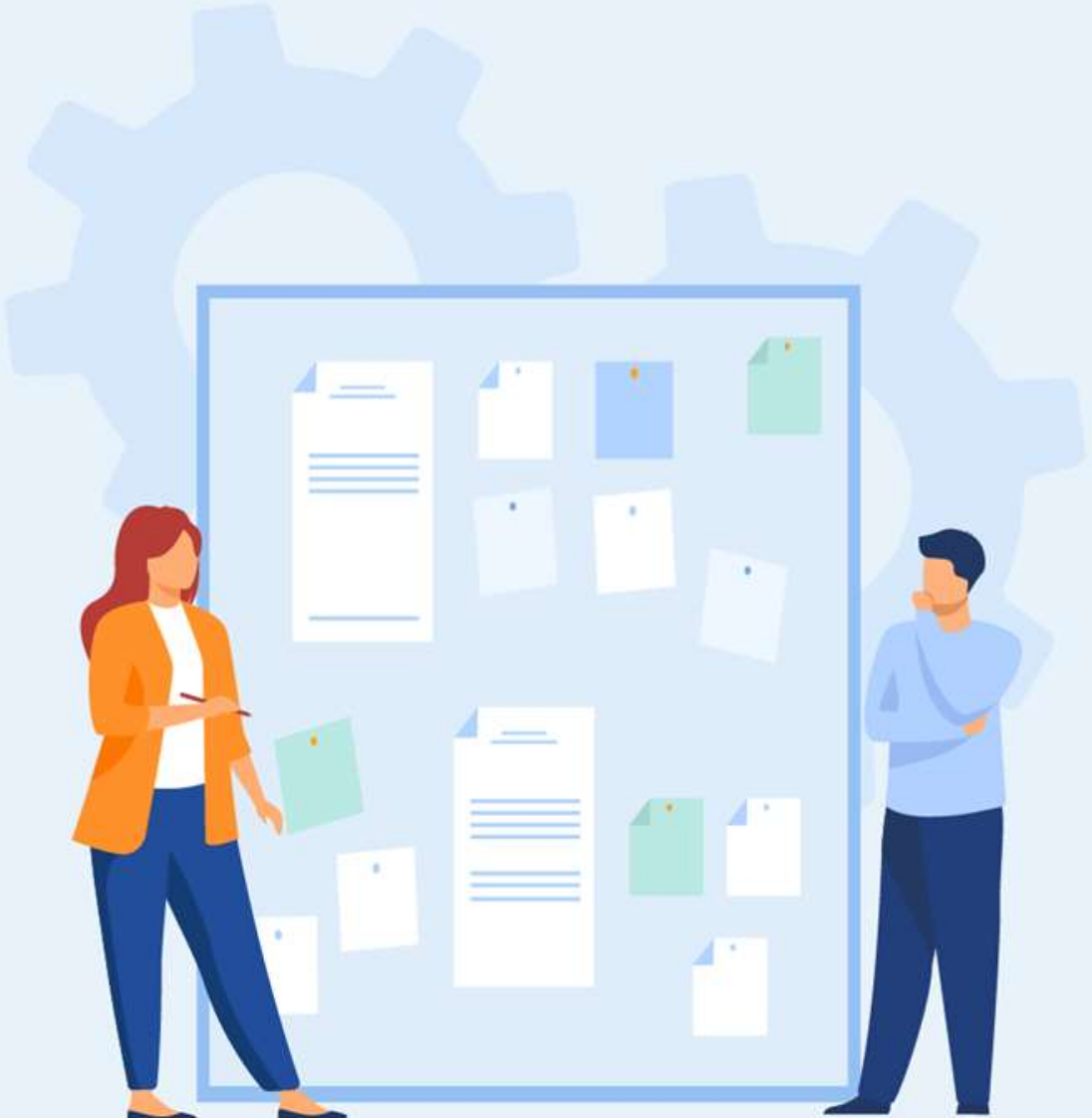
Par exemple, une **icône** est sans état ; vous définissez l'image de l'icône lorsque vous la créez, puis elle ne change plus.

Un widget **Texte** est également sans état.



Widget

Copyright - Tout droit réservé - OFPPT



Chapitre 1

Gérer les états

1. Stateful et stateless widgets
2. **Méthode setState()**
3. Widget interactif

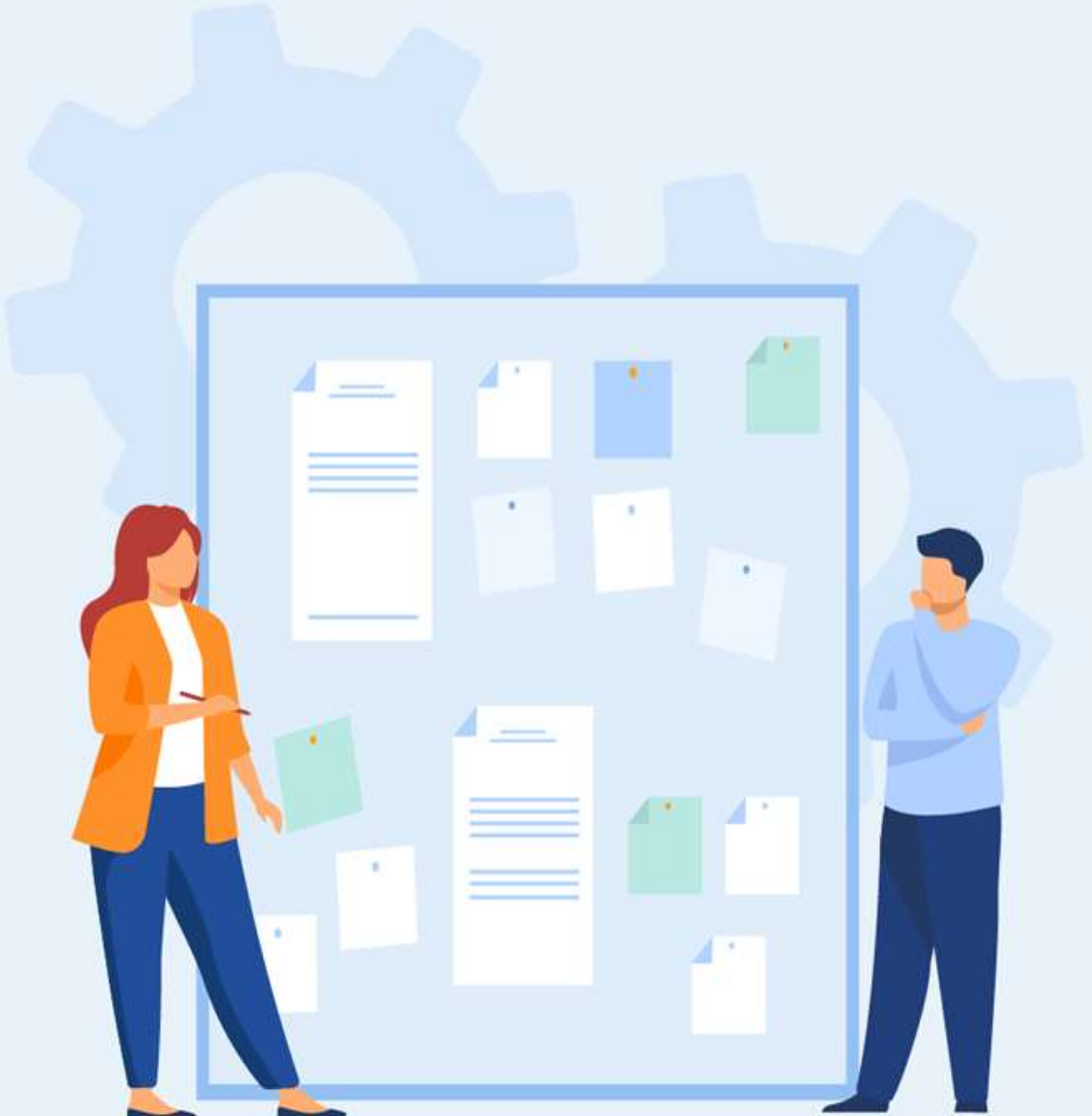
Méthode setState()

L'état d'un widget est stocké dans un objet **State**, séparant l'état du widget de son apparence.

L'état se compose de valeurs qui peuvent changer, comme la valeur actuelle d'un curseur ou si une case est cochée.

Lorsque l'état du widget change, l'objet d'état appelle **setState()**, indiquant au framework de redessiner le widget.

Copyright - Tout droit réservé - OFPPT



Chapitre 1

Gérer les états

1. Stateful et stateless widgets
2. Méthode setState()
3. **Widget interactif**

Widget interactif

Flutter propose une variété de boutons et de widgets interactifs similaires.

La plupart de ces widgets implémentent les directives de conception matérielle, qui définissent un ensemble de composants avec une interface utilisateur avisée.

Vous pouvez utiliser **GestureDetector :**

- Pour créer de l'interactivité dans n'importe quel widget personnalisé.

Vous pouvez trouver des exemples de GestureDetector dans l'état Gestion (state management).

Lorsque vous avez besoin d'interactivité, il est plus simple d'utiliser l'un des widgets préfabriqués. Voici une liste partielle :

Les Widgets standards

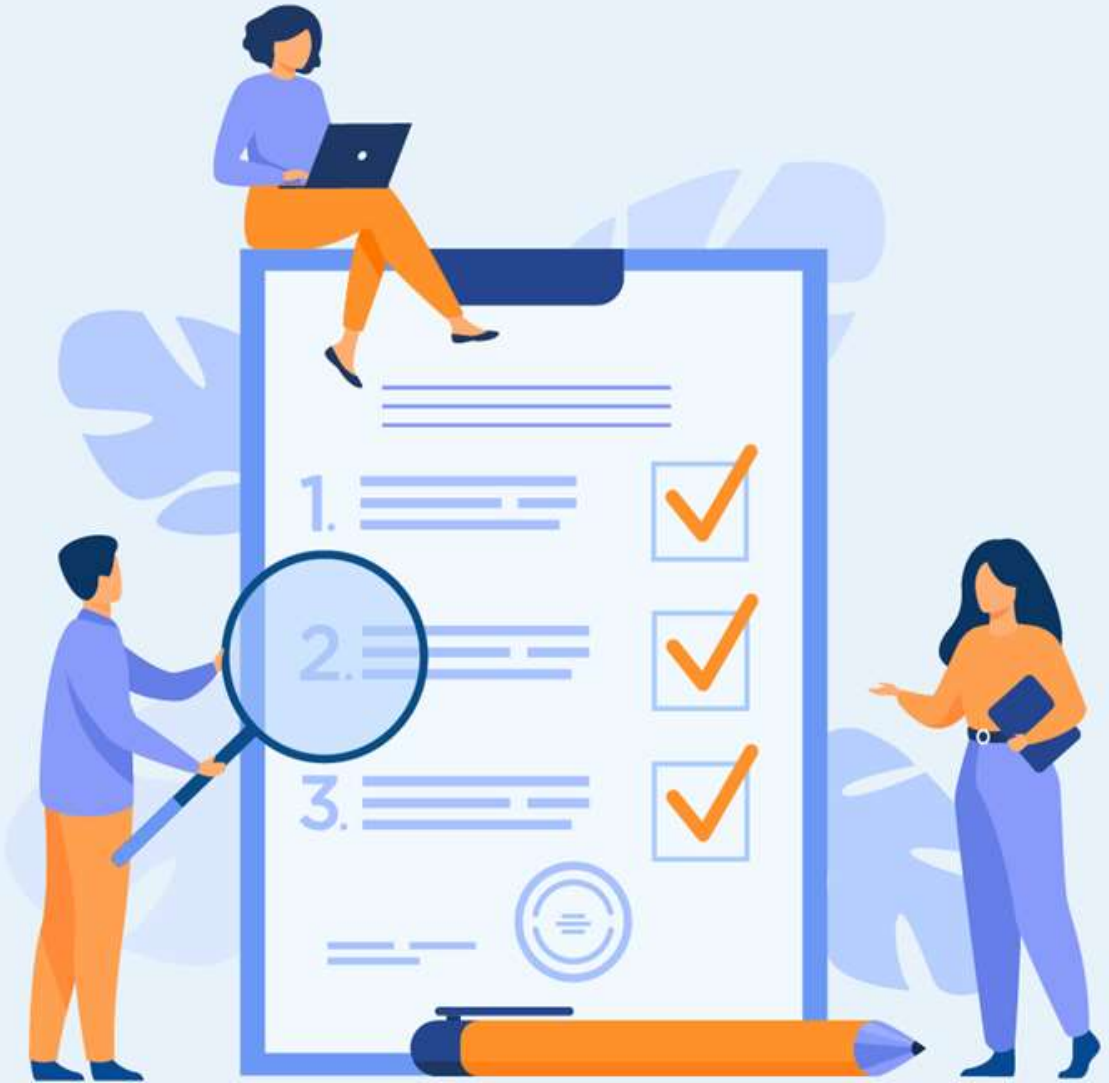
- **Form**
- **FormField**



Remarque

- Flutter fournit également un ensemble de widgets de style iOS appelés **Cupertino**.

Copyright - Tout droit réservé - OFPPT



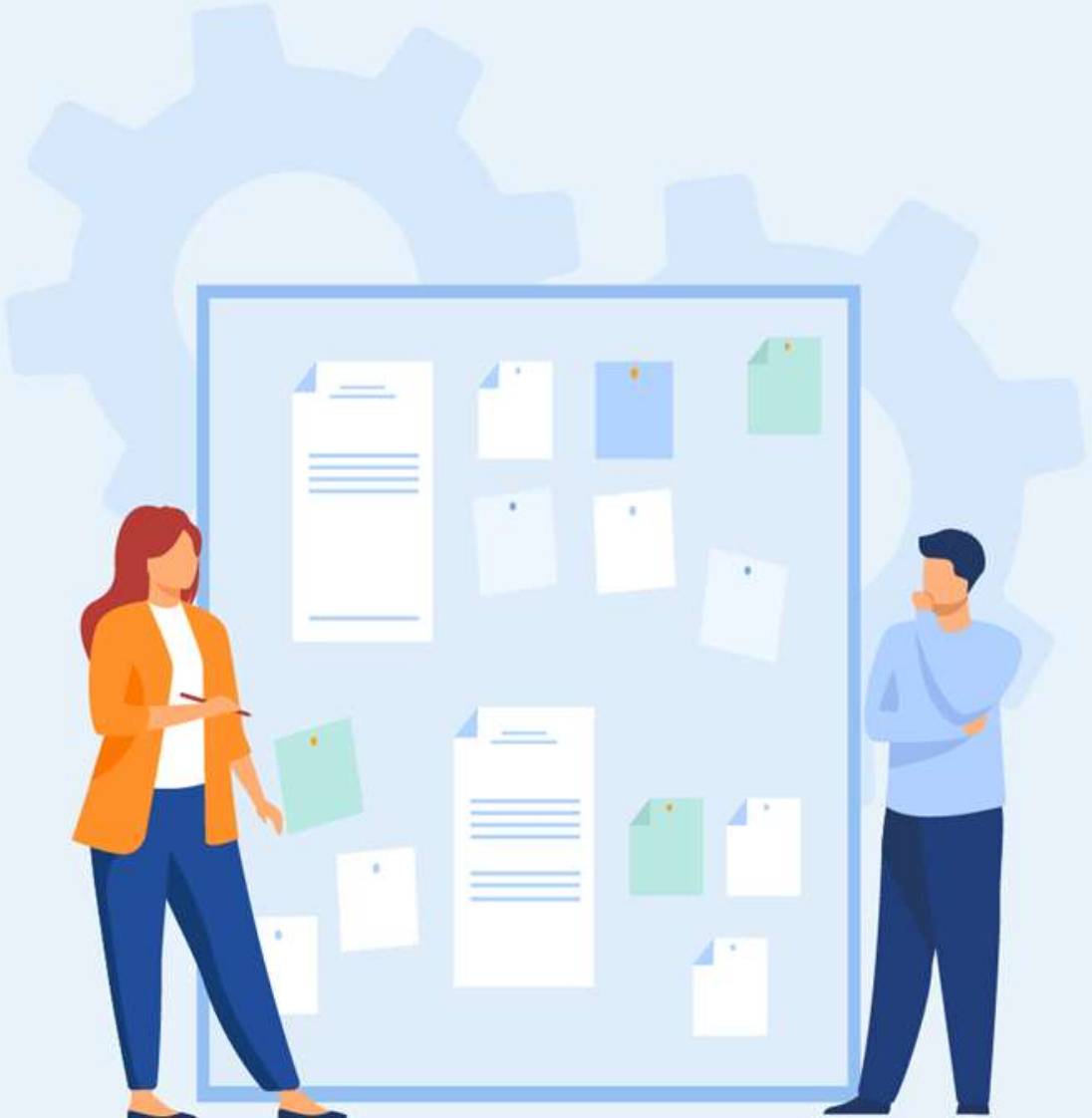
Chapitre 2

Naviguer entre deux écrans

Ce que vous allez apprendre dans ce chapitre :

- La navigation entre deux écrans : MaterialPageRoute, méthodes push() et pop()
- La création d'une nouvelle route

02 heures



Chapitre 2

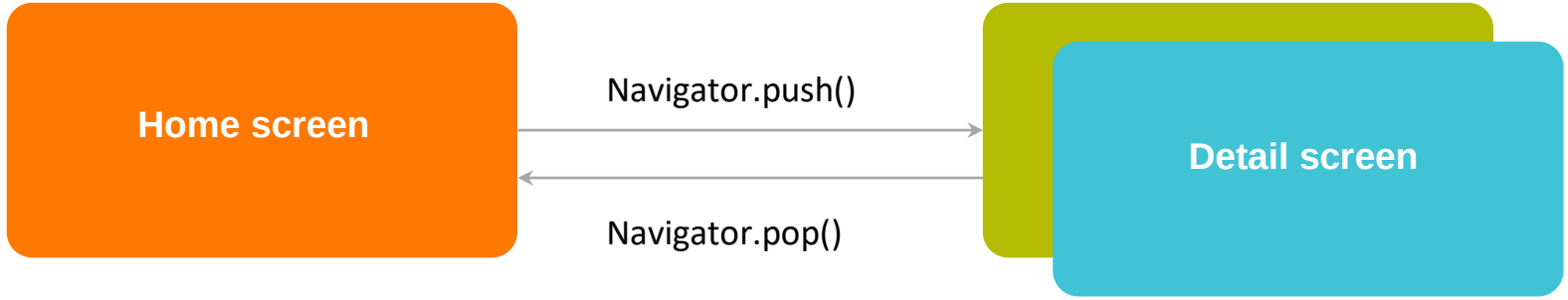
Naviguer entre deux écrans

1. **Navigation entre deux écrans : MaterialPageRoute, méthodes push() et pop()**
2. Création d’une nouvelle route

Navigation entre deux écrans

Flutter fournit un système complet pour naviguer entre les écrans. Les petites applications sans liens profonds complexes peuvent utiliser **Navigator**, tandis que les applications avec des exigences spécifiques en matière de liens profonds et de navigation doivent utiliser le **Router** pour gérer correctement les liens profonds sur Android et iOS.

Le widget **Navigator** affiche les écrans sous forme de pile en utilisant les animations de transition correctes pour la plate-forme cible. Pour naviguer vers un nouvel écran, accédez aux méthodes **Navigator** via la route **BuildContext** et appelez des méthodes impératives telles que **push()** or **pop():**



Copyright - Tout droit réservé - OFPPT

Navigation entre deux écrans

```
onPressed: () {
  Navigator.of(context).push(
    MaterialPageRoute(
      builder: (context) => const SongScreen(song: song),
    ),
  );
},
child: Text(song.name),
```

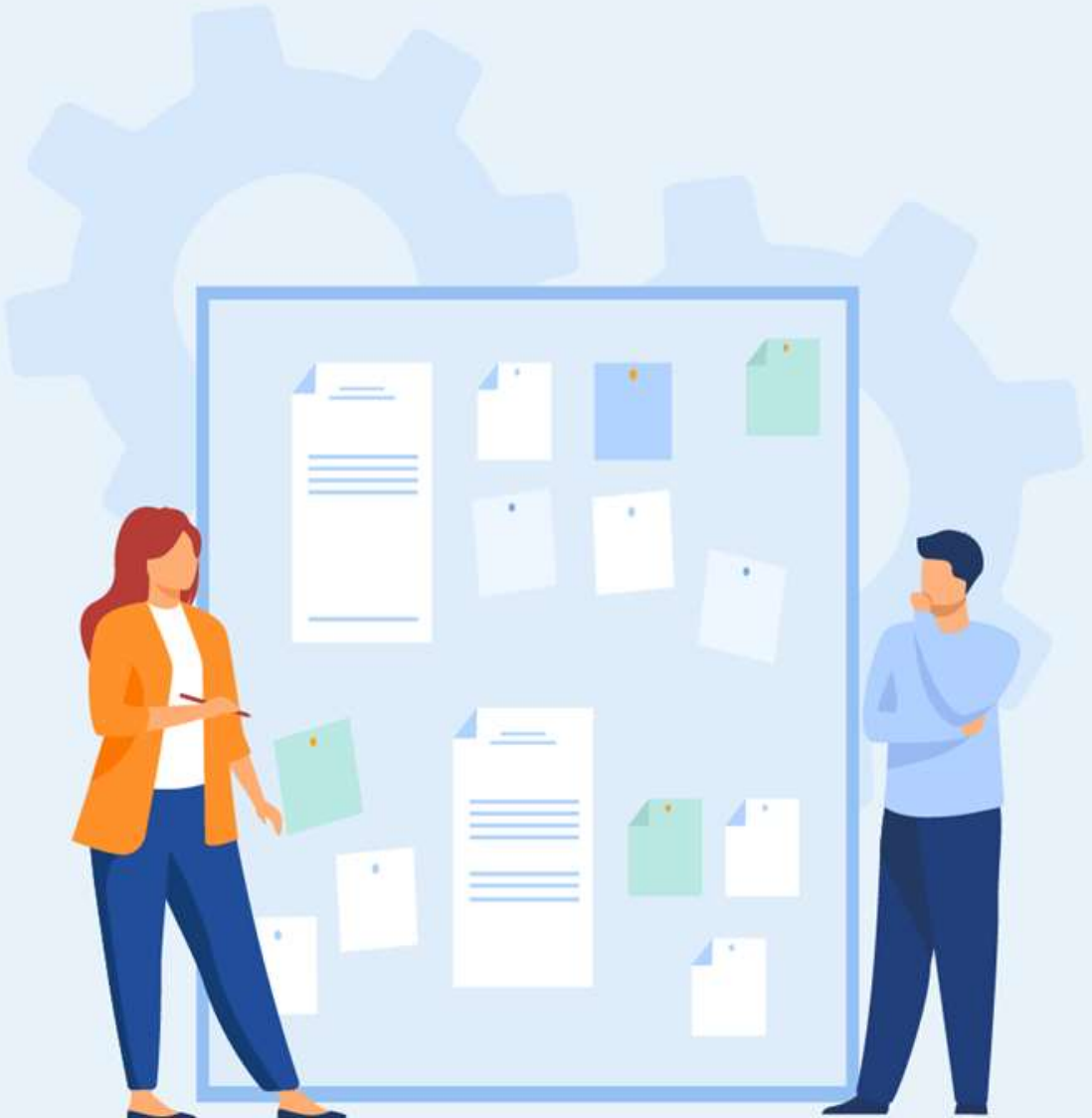
Parce que **Navigator** conserve une pile de **Route** objet (représentant la pile d'historique), la **push()** méthode prend également un objet **Route**.



Remarque

- MaterialPageRoute objet est une sous-classe de Route qui spécifie les animations de transition pour Material Design.

Copyright - Tout droit réservé - OFPPT



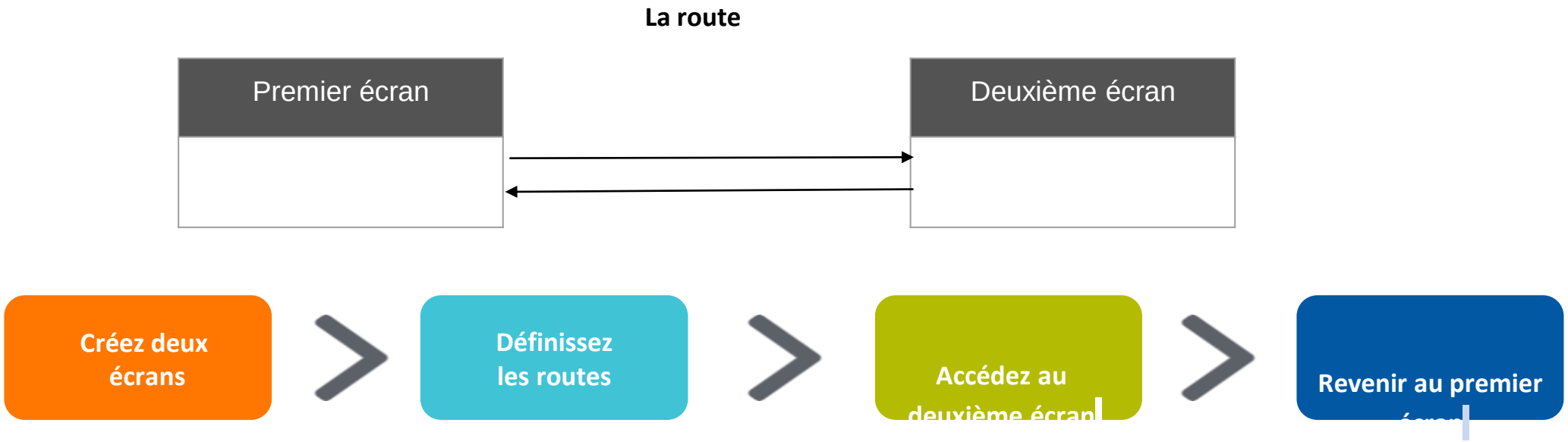
Chapitre 2

Naviguer entre deux écrans

1. Navigation entre deux écrans : MaterialPageRoute, méthodes push() et pop()
2. **Création d’une nouvelle route**

Création d'une nouvelle route

- Dans Flutter, les écrans et les pages sont appelés routes .
  Le reste de cette recette fait référence aux itinéraires, sous Android, un itinéraire équivaut à une activité.
  sous iOS, une route équivaut à un ViewController.
- Dans Flutter, une route n'est qu'un widget.



Copyright - Tout droit réservé - OFPPT