Introduction au développement mobile avec Flutter

m2iformation.fr

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Introduction au développement mobile avec Flutter

Flutter

Objectif : Proposer un SDK moderne pour le dév. **cross-platform** (Android, iOS, web, desktop) avec une **UX native**.

Flutter

Événement	Détail
2015	Google commence à développer Flutter en interne.
2017 (mai)	Flutter est présenté au public lors de Google I/O (Alpha).
2018 (déc.)	Sortie de Flutter 1.0 (Stable).
2021 (mars)	Flutter 2.0 introduit le support web et desktop en stable/beta.
2023+	Flutter évolue vers un vrai SDK multi-plateforme (mobile, web, desktop, embedded).

Principes clés

Principe	Description
Single Codebase	Une seule base de code Dart pour Android, iOS, Web, Windows, macOS, Linux.
Rendering engine custom	Flutter n'utilise pas les composants natifs, mais les **reproduit** à l'identique via son propre moteur de rendu (`Skia`).
Hot Reload	Mise à jour en temps réel de l'interface, très utile en phase de développement.
Composabilité	Tout est un widget. La composition d'interface suit une logique de construction hiérarchique.
Performance native	Pas de pont natif (bridge), tout est compilé en code machine (via **AOT compilation**).

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Partie 1: Introduction au dév. mobile native pour comprendre la philosophie Flutter

Objectifs du cours

- Comprendre ce que signifie **"natif"** dans le contexte du développement mobile.
- Clarifier les notions de **composant natif**, **SDK natif**, **langage natif**, et **bridge natif**.
- Identifier les **forces et limites** du natif.
- Comparer avec les approches **hybrides** ou **cross-platform** comme Flutter.
- Préparer le terrain pour comprendre la **philosophie Flutter**.

Qu'est-ce qu'une application native ?

- C'est une application développée **spécifiquement pour un OS mobile donné**, en utilisant les **outils, les langages et les composants fournis par la plateforme**.

Plateforme	Langage(s) natif(s)	Outils	Framework UI
Android	Java, Kotlin	Android Studio, SDK Android	Jetpack Compose, XML Layout
iOS	Objective-C, Swift	Xcode, SDK iOS	SwiftUI, UIKit

Une application native est compilée en **code machine optimisé** pour la plateforme. Cela lui garantit une **intégration profonde** dans le système et un accès complet à toutes les fonctionnalités.

Les composants natifs

Les **composants natifs** sont les **éléments d'interface ou de fonctionnalités** directement fournis par l'OS mobile, comme:

- **Boutons (Button, FAB, etc.)**
- **TextInput (champ de saisie)**
- **ListView, ScrollView, TabBar**
- **APIs systèmes** : caméra, gyroscope, Bluetooth, notifications, etc.

Exemple : Un UIButton iOS n'a pas exactement le même aspect ou comportement qu'un MaterialButton sur Android.

Le pont natif (native bridge)

Un **pont natif** est un **mécanisme de communication entre du code non-natif (JavaScript…) et les APIs natives** de l'OS.
Il permet :

- d'appeler des fonctions natives depuis du code non-natif.
- de recevoir des réponses/événements de l'OS.

Fonctionnement (React Native, Cordova…):

```
[Dart/JS] ⇄ [Bridge] ⇄  [Native Code Swift/Kotlin]
```

Le pont natif (native bridge)

- En React Native, le JS envoie un message au bridge, qui appelle une méthode native.
- Cela a un **coût de performance** (asynchronisme, sérialisation, latence).

- Flutter **n’a pas de bridge JS**, mais peut quand même communiquer avec du natif via **Platform Channels**. (équivalent, mais beaucoup moins sollicité que le bridge de React Native qui agit comme goulot d'étranglement.)

SDK (Software Development Kit)

Chaque plateforme fournit un SDK qui inclut :

- Les **librairies natives**
- Les **outils de compilation**
- Les **APIs système**

Exemples :

- `AVFoundation`, `CoreLocation` pour iOS.
- `MediaPlayer`, `CameraX` pour Android.

API natives

Ce sont les interfaces de programmation spécifiques à la plateforme.

On les utilise pour :

- Gérer la caméra, les fichiers, la géolocalisation.
- Afficher des composants UI spécifiques.
- Gérer les permissions (accès aux contacts, photos…).

Développement natif : avantages et limites

✅ Avantages

- Performance maximale (code compilé pour la machine cible).
- Accès complet à toutes les fonctionnalités matérielles.
- Composants UI 100% fidèles à la plateforme.
- Bonne évolutivité (intégration des dernières APIs dès leur sortie)

Développement natif : avantages et limites

❌ Inconvénients

- Nécessité de développer **deux apps distinctes** pour Android et iOS.
- Maintenance plus coûteuse.
- Délais plus longs.
- Nécessite des compétences différentes (Kotlin/Swift).

Comparaison : natif vs cross-platform

Critère	Natif	Flutter / React Native
Codebase	2 (iOS + Android)	1 seule
Accès aux APIs	Total	Possible via bridge ou plugin
UI	Composants OS	Flutter : rendus personnalisés ; RN : composants natifs
Performance	Optimale	Très bonne (Flutter) ou moyenne (RN)
Coût/délai	Plus élevé	Réduit (x1.5 environ)
Maintenance	Complexe	Centralisée

Vocabulaire clé à retenir

Terme	Définition
Natif	Propre à une plateforme (Android ou iOS), avec ses outils et son langage.
Composant natif	Élément d'interface ou de système fourni par l'OS.
Bridge natif	Mécanisme permettant à du code non-natif de communiquer avec des APIs natives.
SDK	Ensemble d'outils pour développer pour une plateforme.
API système	Fonctionnalité accessible via une interface de programmation (GPS, Bluetooth...).
UI native	Interface conçue avec les composants officiels de l'OS.

Conclusion

Avant de travailler avec Flutter, il est **essentiel de comprendre ce que Flutter abstrait** :

- Il **remplace les composants natifs** par ses propres widgets rendus via Skia.
- Il **évite le bridge JS** pour **gagner en performance**.
- Il **simule le comportement des plateformes**, tout en permettant d'accéder aux APIs via des **plugins ou channels** si besoin.

Comprendre le natif permet de mieux intégrer Flutter dans les projets réels, notamment quand il faut interagir avec du code existant ou ajouter des fonctionnalités avancées (notifications...).

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Partie 2: L'architecture de Flutter

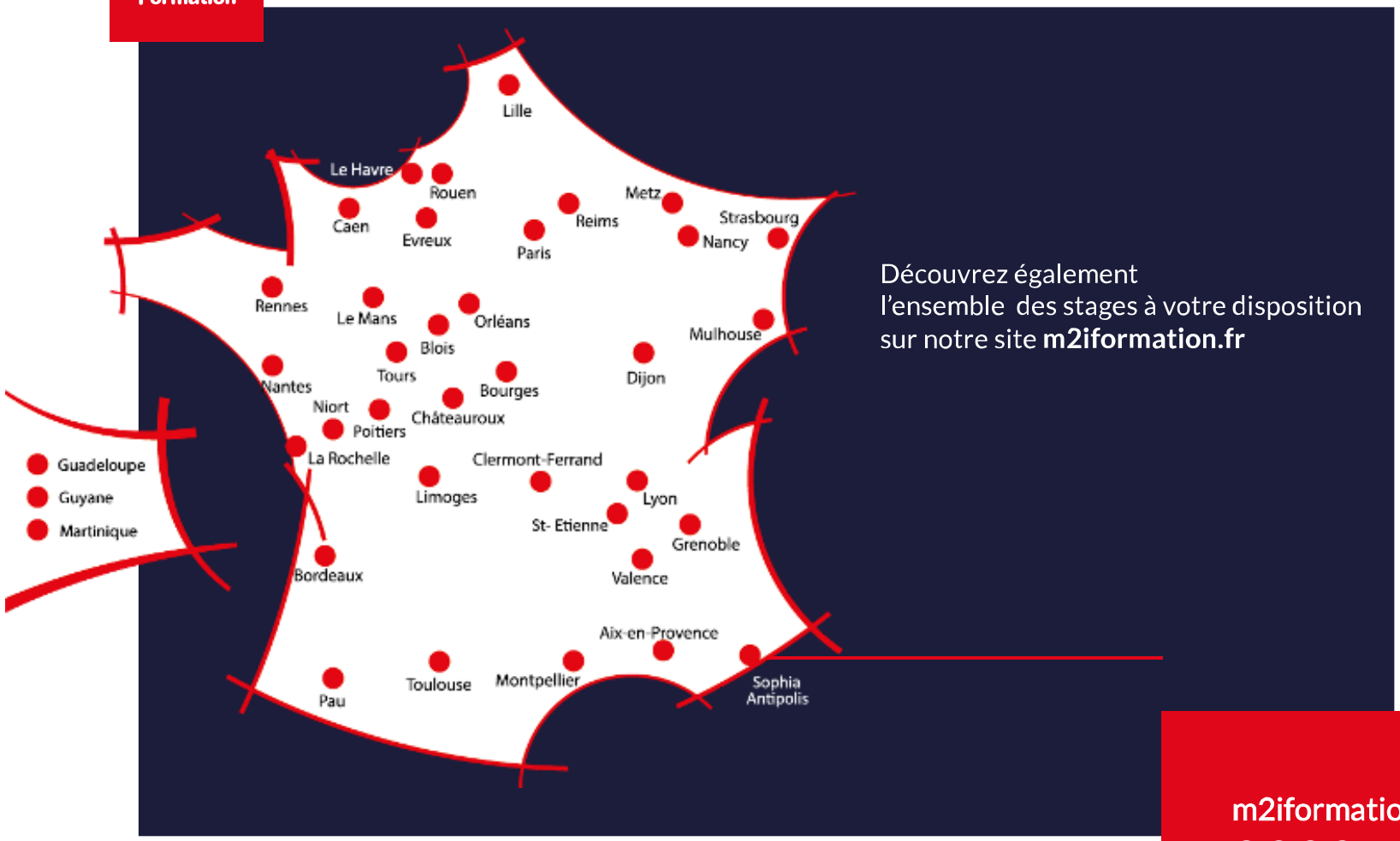
Vue d'ensemble

Fonctionnement du rendering :

- Flutter **ne** s'appuie **ni sur WebView** (affichage du contenu web directement dans une application mobile), **ni sur des composants UI natifs**.
- Il **dessine directement** les pixels via **Skia**, moteur graphique open-source (aussi utilisé dans Chrome).
- Résultat : **uniformité graphique** + **contrôle total** sur l'interface.

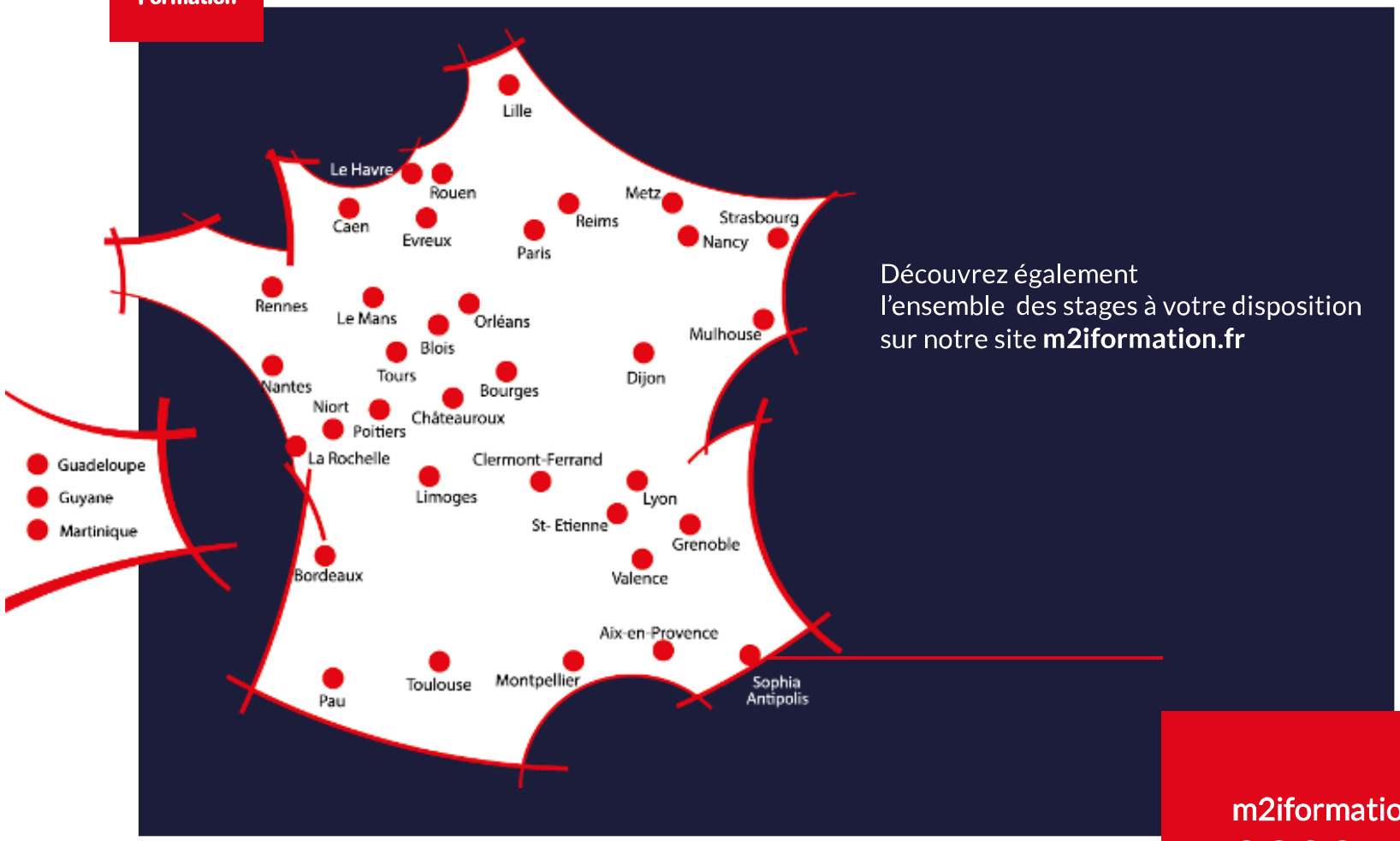
Langage Dart

- Langage conçu par Google.
- Fortement typé, orienté objet.
- Compile en **code machine (AOT)** pour les performances en production.
- Compile aussi en **JavaScript (JIT)** pour le web ou le dev rapide.



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr