Chapitre 2 : Développement Multiplateforme

2.1 Présentation des émulateurs et simulateurs

Définition :
Un **émulateur** est un logiciel qui imite le matériel d'un appareil réel pour exécuter des applications, permettant de tester sans un vrai appareil physique.
Un **simulateur** reproduit l'environnement logiciel d'un appareil, sans émuler son matériel à 100%.

Rôle dans le développement :

- Tester rapidement des applications sans matériel physique.
- Déboguer en conditions variées (taille d'écran, versions d'OS).
- Automatiser des tests sur plusieurs configurations.

Exemples courants :

- Android Emulator (Android Studio)
- iOS Simulator (Xcode)

2.2 Différence entre émulateurs et simulateurs

Aspect	Émulateur	Simulateur
Simulation matérielle	Oui	Non
Vitesse	Plus lent (plus lourd)	Plus rapide (plus léger)

Précision de test	Très précis (réplique matérielle)	Moins précis (ne gère pas bien les erreurs matérielles)
Plateformes principales	Android, parfois Windows	iOS principalement
Exemples	Android Emulator	iOS Simulator

Résumé pratique :

- **Android** utilise des **émulateurs**.
- **iOS** utilise des **simulateurs**.
- Choisir l'outil dépend de la précision requise pour vos tests.

2.3 Installation et configuration des émulateurs pour Android et iOS

Android :

1. **Installer Android Studio** (nécessaire pour SDK et outils).
2. **Configurer un Android Virtual Device (AVD)** :
    - Ouvrir Android Studio → AVD Manager.
    - Créer un nouvel appareil virtuel.
    - Choisir le modèle d'appareil (Pixel, Nexus, etc.).
    - Télécharger une image système (Android 13, Android 14...).
    - Lancer l'émulateur.

Commandes utiles Flutter :

bash

```
flutter emulators
flutter emulators --launch <emulator_id>
```

iOS :

1. **Installer Xcode** (nécessaire pour l'environnement de développement iOS).
2. **Accéder au simulateur** :
    - Xcode → Open Developer Tools → Simulator.
3. **Créer/Configurer un simulateur personnalisé** si besoin.

Notes importantes :

- **Xcode n'est disponible que sur macOS**.
- Pour **Flutter**, il faut accepter les licences Xcode :

bash

```
sudo xcode-select --switch /Applications/Xcode.app
sudo xcodebuild -license accept
```

2.4 Présentation de Windows Universal Device Portal

Qu'est-ce que Windows Device Portal ?
C'est un portail web qui permet de **gérer et déboguer** des appareils Windows 10 et Windows 11 via réseau (LAN ou USB).

Utilisations principales :

- Déployer et tester des applications UWP (Universal Windows Platform).
- Surveiller les performances en temps réel (CPU, mémoire).
- Accéder aux journaux et traces réseau.

- Installer des certificats ou paquets d'applications (.appx/.msix).

Activation :

1. Activer le **Mode Développeur** dans Windows.
2. Activer **Device Portal** dans les Paramètres → Mise à jour et Sécurité → Pour les développeurs.

2.5 Utilisation de Flutter pour le développement multiplateforme

Pourquoi Flutter pour du multiplateforme ?

- **Un seul code source** pour Android, iOS, Web, Windows, macOS, Linux.
- Performances proches du natif (car Flutter compile en code natif).
- Flexibilité de l'UI grâce aux **widgets personnalisables**.

Plateformes supportées par Flutter :

- Mobile : Android, iOS.
- Web : Chrome, Safari, Firefox.
- Desktop : Windows, macOS, Linux.
- Embarqué : Raspberry Pi, IoT.

2.6 Introduction à Flutter et son architecture

Présentation générale :

Flutter est un **framework UI open-source** développé par Google pour créer des applications nativement compilées.

Architecture Flutter :

- **Dart Framework** : Gère les widgets, animations, gestion d'état.
- **Engine (moteur)** : Écrit en C++, il utilise Skia pour le rendu graphique.
- **Embedder** : S'adapte à chaque plateforme cible (Android, iOS, Web, Windows...).

Cycle d'une application Flutter :

1. L'utilisateur interagit avec l'UI.
2. Les widgets capturent l'événement.
3. Flutter engine traite l'événement.
4. Le moteur demande au GPU de redessiner l'UI.

2.7 Configuration de l'environnement Flutter

Étapes principales :

1. **Télécharger Flutter SDK** :
    - Depuis flutter.dev.
2. **Configurer l'environnement système** :
    - Ajouter Flutter aux variables d'environnement (`PATH`).
3. **Installer les outils nécessaires** :
    - Android Studio + plugin Flutter + Dart.
    - Visual Studio Code + extensions Flutter et Dart.

4. **Vérifier l’installation** :

bash

```
flutter doctor
```

Ce diagnostic vérifie que tout est prêt (SDK, IDEs, émulateurs, etc.).

2.8 Concepts clés : widgets, hot reload, framework Dart

Widgets :

- **Définition** : Chaque composant visuel est un widget (texte, bouton, image...).
- **Types de widgets** :
    - **StatelessWidget** : UI statique.
    - **StatefulWidget** : UI dynamique (qui change dans le temps).

Exemple rapide :

dart

```
Text('Bonjour Flutter!')
```

Hot Reload :

- Permet de **mettre à jour instantanément l’application** sans redémarrage complet.
- Très utile pour le développement rapide et le test de modifications d’UI.
- Commande dans l'IDE ou via console :

bash

```
r (dans le terminal avec flutter run)
```

Framework Dart :

- Langage utilisé par Flutter.
- Caractéristiques :
    - Orienté objet, syntaxe proche de Java et JavaScript.
    - Gestion native des futures (asynchronisme) avec async/await.
    - Garbage collection automatique.

Exemple de fonction Dart simple :

dart

```
void saluer() {
  print('Salut le monde!');
}
```

2.9 Création d’une première application simple avec Flutter

Objectif :

Créer une application qui affiche « Bonjour le monde ! ».

Étapes détaillées :

1. **Créer un nouveau projet :**

bash

```
flutter create bonjour_flutter
cd bonjour_flutter
```

```
code .
```

2. **Modifier le fichier `lib/main.dart` :**

dart

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

void main() {
  runApp(const MonApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: const Text('Bonjour Flutter')),
        body: const Center(
           child: Text('Bonjour le monde!', style: TextStyle(fontSize:
24)),
        ),
      ),
    );
  }
}
```

3. **Lancer l'application :**

bash

```
flutter run
```

4. **Tester sur différents appareils** (émulateur Android, simulateur iOS, navigateur Web).