Cours 1

Problématiques du Dev Mobile

- **Autonomie**, la batterie est limitée.
- **Robustesse**, Les utilisateurs n'ont pas toujours accès à internet.
- Utilisation de fonctionnalités **bas-niveau**. GPS, appareil photo,
- Beaucoup d'**environnements différents** !

Quelques exemples :

- **Autonomie** : une application de fitness qui suit les activités sportives de l'utilisateur et optimise la consommation d'énergie en fonction de l'utilisation de la batterie. L'application peut utiliser des capteurs de mouvement pour détecter le niveau d'activité physique de l'utilisateur et ajuster la luminosité de l'écran, la fréquence de rafraîchissement, etc. pour économiser de l'énergie.
- **Robustesse** : une application de messagerie instantanée qui fonctionne même en cas de perte de connexion Internet. L'application peut utiliser une base de données locale pour stocker les messages en attente d'envoi et les envoyer dès que la connexion est rétablie. Elle peut également utiliser des mécanismes de réplication de données pour synchroniser les messages entre plusieurs appareils.
- **Fonctionnalités bas-niveau** : une application de réalité augmentée qui utilise des API bas-niveau pour accéder à la caméra, aux capteurs de mouvement, etc. de l'appareil. L'application peut utiliser des bibliothèques comme ARKit ou ARCore pour accéder aux fonctionnalités bas-niveau de l'appareil et créer des expériences de réalité augmentée immersives.

Quels langages pour les App mobiles ?

Du plus performant au moins performant

- **Dev natif.** Android (Java, Kotlin), ios (Objective C, Swift) . +Plus de contrôle, +outils de debug, et fonctionnalités spécifiques. +meilleures performances. -Temps de développement et maintenance
- **Flutter** est basé sur des Widgets, +Chaque Widget est ensuite compilé en composant natif pour chaque plateforme. +Accès aux fonctionnalités bas-niveau +Google maintient des librairies de base pour accéder à l'appareil photo, accéléromètre, ... -Langage spécifique, pas forcément très utilisé.
- **Cross-plateforme JS** : React Native, Cordova, Ionic. un seul code en langage web. -Performances moins bonnes, Traduire le DOM pour manipuler les éléments. +Développement rapide -limitations des accès spécifiques aux environnements.

React Native est quand même utilisé par des grosses applis.

Intro à Flutter

Flutter SDK pour **Mobile et Web**.

Basé sur **Dart** langage objet typé, développé et maintenu par Google.

Dart et Flutter sont **open Source**.

Un exemple

```
class Person {
          String name;
          int age;

          Person(this.name, this.age); // Constructeur

          void sayHello() {
                    print("Bonjour, je m'appelle $name et j'ai $age ans.");
          }
}

void main() {
          Person p = new Person("Alice", 30);
          p.sayHello();
}
```

Une **fonction main** dans le code est nécessaire. Des **points-virgule**.

Des interfaces

```
abstract class Animal {
  void makeSound();
}

class Chat implements Animal {
  @override
  void makeSound() {
    print("Miaou!");
  }
}

class Chien implements Animal {
  @override
  void makeSound() {
    print("Ouaf!");
  }
}

void main() {
  Animal chat = new Chat();
  Animal chien = new Chien();
```

```
  chat.makeSound(); // "Miaou!"
  chien.makeSound(); // "Ouaf!"
}
```

L'héritage

```
class Animal {
          String name;
          List<String> foodList;

          Animal(this.name, this.foodList);

          void makeSound() {
                    print("L'animal $name fait un bruit inconnu.");
          }

          void eat() {
                    print("$name mange ${foodList.join(', ')}.");
          }
}

class Chat extends Animal {
          Chat(String name, List<String> foodList) : super(name, foodList);

          @override
          void makeSound() {
                    print("Le chat $name miaule!");
          }
}

void main() {
          Animal animal = new Animal("anonyme", ["viande", "poisson"]);
          Chat chat = new Chat("Felix", ["croquettes", "thon"]);

          animal.makeSound(); // Affiche "L'animal anonyme fait un bruit
inconnu."
          animal.eat(); // Affiche "anonyme mange viande, poisson."
          chat.makeSound(); // Affiche "Le chat Felix miaule!"
          chat.eat(); // Affiche "Felix mange croquettes, thon."
}
```

Flutter : Les Widgets

Un **Widget** est une classe qui définit un morceau d'interface utilisateur

Tout est widget :
Styles, animations, lists, text, buttons

En React, on travaille avec des **composants**. En Flutter, des **Widgets**.

Pour faire une app, on assemble plusieurs Widgets.

Quelques exemples de widget

Des widget simples

1. `Text` : Affiche du texte sur l'écran.

```
Text(
  'Bonjour tout le monde!',
  style: TextStyle(fontSize: 24),
),
```

2. `ElevatedButton` : Affiche un bouton avec un effet de relief lorsqu'il est pressé.

```
ElevatedButton(
          onPressed: () {
                    // Action à effectuer lorsqu'on appuie sur le bouton
          },
          child: Text('Appuyez ici'),
),
```

3. `Icon` : Affiche une icône sur l'écran.

```
Icon(
          Icons.favorite,
          color: Colors.red,
),
```

4. `Image` : Affiche une image sur l'écran.

```
Image.asset(
  'images/mon_image.png',
),
```

5. `TextField` : Permet à l'utilisateur de saisir du texte.

```
TextField(
          decoration: InputDecoration(
                    hintText: 'Entrez votre nom',
          ),
),
```

6. `ListView` : Affiche une liste d'éléments défilables.

```
ListView(
          children: <Widget>[
                    Text('Elément 1'),
                    Text('Elément 2'),
                    Text('Elément 3'),
```

```
            ],
),
```

Des Widgets qui permettent d'organiser une page

7. `Container` : Permet de mettre en forme et de styliser le contenu qu'il contient.

```
Container(
            margin: EdgeInsets.all(10),
            padding: EdgeInsets.all(20),
            decoration: BoxDecoration(
                        color: Colors.blue,
                        borderRadius: BorderRadius.circular(10),
            ),
            child: Text('Contenu à l\'intérieur'),
),
```

8. `Column` : Organise les widgets enfants en colonne.

```
Column(
            children: <Widget>[
                        Text('Première colonne'),
                        Text('Deuxième colonne'),
                        Text('Troisième colonne'),
            ],
),
```

9. `Row` : Organise les widgets enfants en ligne.

```
Row(
            children: <Widget>[
                        Text('Première rangée'),
                        Text('Deuxième rangée'),
                        Text('Troisième rangée'),
            ],
),
```

10. `Stack` : Empile des widgets les uns sur les autres.

```
Stack(
            children: <Widget>[
                        Text('Contenu au premier plan'),
                        Image.asset('images/mon_image.png'),
            ],
),
```

11. `Scaffold` : Fournit une structure de base pour une application en incluant une barre d'applications, un tiroir de navigation, un corps et un fond d'écran.

```
Scaffold(
            appBar: AppBar(
                        title: Text('Mon application'),
```

```
        ),
        body: Text('Contenu de l\'application'),
),
```

12. `AppBar` : Affiche une barre d'applications en haut de l'écran.

```
AppBar(
  title: Text('Mon application'),
),
```

13. `BottomNavigationBar` : Affiche une barre de navigation en bas de l'écran.

```
BottomNavigationBar(
        items: [
                BottomNavigationBarItem(
                        icon: Icon(Icons.home),
                        title: Text('Accueil'),
                ),
                BottomNavigationBarItem(
                        icon: Icon(Icons.dashboard),
                        title: Text('Tableau de bord'),
                ),
        ],
),
```

14. `Card` : Affiche une carte avec un contenu personnalisé à l'intérieur.

```
Card(
        child: Column(
                children: <Widget>[
                        Image.asset('images/mon_image.png'),
                        Text('Titre de la carte'),
                        Text('Contenu de la carte'),
                ],
        ),
),
```

Définir des nouveaux Widget

Comme React on entoure les widgets de base des nouveaux éléments qu'on veut mettre autour.

Pour faire un widget de bouton centré :

```
class AddToCartButton extends StatelessWidget {

@override
Widget build(BuildContext context) {
        return Center(
                child: FlatButton(
                        onPressed: () {
                                print("Bonjour");
```

```
                        },
                        child: Text('Add to Cart'),
                    ),
            );
}
```

Chaque classe possède une méthode **build** qui retourne un widget.

Statefull ou Stateless ?

On peut définir un Widget à partir de ces deux classes.

Stateless = Aucune information n'est contenue dans ce widget **Stateful** = Possède un state associé qui contient une info. -> une méthode setState qui déclenche le rendu quand elle est appelée.