Fiche développement multiplateforme

- **Widget**
  - La fonction main constitue le point d'entrée du programme.
  - runApp crée une instance du widget MainApp et le positionne en tant que racine de l'arborescence des widgets.
  - Build() : cette méthode est appelée à chaque fois que le widget doit être redessinée.

1. **Stateful widgets :** classes qui ne peuvent pas être changé

```
Class runner extends StetfullWidgets
{
    @override
    _runnerState createState () => _runnerState() ;
}
Class _runnerState extends State<runner>
{
    @override
    Widget build(BuildContext context)
    {
        Return Container(
            Child: Text('Hello'),
            );
    }
}
```

2. **Stateless widgets :** est une classe qui présente des objets ou des données (peuvent être modifier après la création)

```
Class runner extends StatelessWidget
{
    @override
    Widget build (BuildContext context)
    {
        Return Container(
            Child : Text('hello),
    ) ;
    }
}
```

3. **Scaffold widgets :** c'est le contour de toute la page.

```
@overide
Widget build (BuildContext context)
{
    Return Scafold (
            backgroundColor : colors.green,
            body: Container (
                    width :200,
                    height :200,
                    color :colors.white,
                    child : Text('hello ') // Pour ajouter dans le container
                    // children : [] si on a plusieurs child
)
}
```

4. **Center Widget :** centre le texte

```
Child : Center(
    Child : Text('Hello')
)
```

- **Text / Text style**

```
Child : Center(
    Child : Text('hello',
            Style : TextStyle(
                    Fontsize :20,
                    Fontweight : fontWeight.Blod
                    fontStyle : fontStyle.Italic
                ),
        ),

)
```

- **Floating Action Button**

```
FloatingActionButton: FloatingActionButton(
backgroundColor: colors.white,
child: Icon(Icons.arrow,
                    colors: Colors.black
                ),
    )
```

- **Box decoration**

```
Body: Container(
        Decoration: BoxDecoration (
        Image: DecorationImage(
                image: AssetImage(‘’),
                 fit;BoxFit.cover
                ),
        )
)
```

- **SingleChildScrollView**

```
Body: Container(
Child: SingleChildScrollView(
Child: Column(
Children:[
Container(
        Form(
                key: Column(
                        children: [
                                TextFormField(
                                        Controller: _controller,
                                        Decoration: InputDecoration,
                                        hintText: ‘name’
                                ),
                        ])))])))
```

- **StreamBuilder :** utilise pour la mise à jour dynamique de l’interface utilisateur en réponse à des données en continu.

```
Body : Center(
    Child : StreamBuilder<String>(
    Stream : _controller.strem,
    Builder : (context, snapshot){
        Return Text(${snaposhot.data}) ;
    }
    )
)
```

- **FutureBuilder :** permet de gérer l'affichage d'une interface utilisateur basé sur le résultat d'une opération asynchrone.

```
Future<String> fetchData() async {
  await Future.delayed(Duration(seconds: 2));
  return 'Données chargées avec succès!';
 }
body: Center(
        child: FutureBuilder<String>(
          future: fetchData(),
          builder: (context, snapshot) {
            return Text('Données: ${snapshot.data}');
          }
        },
      ),
```

- **ListView** : affiche une liste déroulante d'éléments.

```
body: ListView.builder(
        itemCount: items.length,
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(items[index]),
            onTap: () {
              // Gérer le tap sur l'élément de la liste
              print('Tapped on: ${items[index]}');
            },
          );
        },
      ),
```

- **Consommation des API REST**

```
import 'package:http/http.dart' as http;
Future fetchData() async {
 final response = await http.get('https://api.example.com/data');
 if (response.statusCode == 200) {
  final Map<String, dynamic> data = json.decode(response.body);
  return data;
 } else {
  print('Erreur de requête: ${response.statusCode}');
  return null;
 }}
```

```
ElevatedButton(
  onPressed: () async {
    var result = await fetchData();
    if (result != null) {
      print('Données récupérées avec succès: $result');
    }
  },
  child: Text('Récupérer les données de l\'API'),)
```

Post data

```
Future postData(String name, int age) async {
  final response = await http.post(
    'https://api.example.com/data',
    body: {
      'name': name,
      'age': age.toString(),
    },
  );

  if (response.statusCode == 200) {
    // La requête a réussi, analysez les données JSON si nécessaire
    print('Données postées avec succès: ${response.body}');
  } else {
    // La requête a échoué, imprimez l'erreur
    print('Erreur de requête: ${response.statusCode}');
  }
}
ElevatedButton(
  onPressed: () async {
    await postData('John Doe', 30);
  },
  child: Text('Envoyer les données à l\'API'),
)
```

- **Creation du projet flutter**
  1. Flutter create projet
  2. Cd projet
  3. Flutter run
- **Compiler pour une plateforme spécifique**
  Flutter build ios/apk ...