UNIVERSITÉ DE FRANCHE-COMTÉ

# Le développement multi-plateforme avec Flutter

Eléments avancés pour développer avec Flutter

Visualisation de sources de données, bases de données et WEB

# Comment gérer efficacement une liste d'items ?

- Classiquement l'écran d'un smartphone ne permet pas d'afficher toutes les données simultanément
  - Par exemple la liste des contacts, celle des messages échangés, …
- Il faut donc pouvoir :
  - Faire défiler les items visualisant les données
    - On ne crée que les éléments d'interface utilisateur correspondant à l'espace d'affichage disponible. Lorsqu'une donnée n'est plus visualisée en raison du défilement, l'élément d'interface utilisateur correspondant est réutilisé pour afficher une autre donnée.
    - S'il y a 1000 données seules une dizaine sont visualisées et donc une dizaine d'éléments d'interface utilisateur sont nécessaires
  - Associer un item visualisé à la structure de données contenant celles-ci
- Ceci nécessite un effort de programmation important mais heureusement on dispose de composants qui fournissent ce service :
  - Le plus utilisé/documenté est le ListView

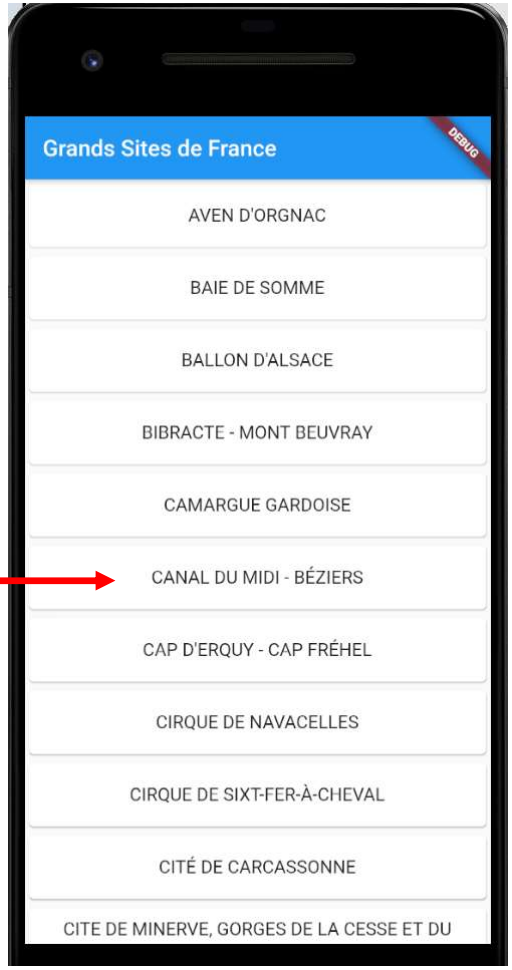
# L’exemple des grands sites de France – 1.0

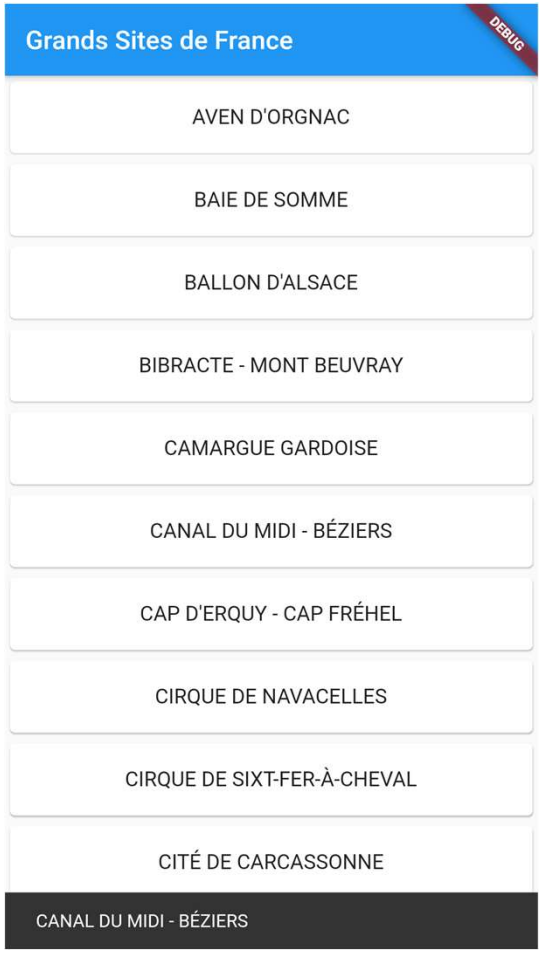

**AVEN D'ORGNAC**
**BAIE DE SOMME**
**BALLON D'ALSACE**
**…**
**MASSIF DU CANIGÓ**
**MONTSÉGUR**
**POINTE DU RAZ EN CAP SIZUN**
**PRESQU'ILE DE GIENS ET SALINS D'HYÈRES**
**PUY DE DÔME**
**PUY MARY - VOLCAN DU CANTAL**
**ROCAMADOUR**
**SAINTE-VICTOIRE**
**SALAGOU - CIRQUE DE MOURÈZE**
**SOLUTRÉ POUILLY VERGISSON**
**VALLÉE DE LA CLARÉE - VALLÉE ÉTROITEE**
**VALLÉE DE LA VÉZÈRE**
**VALLEE DU HÉRISSON - PLATEAU DES 7 LACS**
**VEZELAY**

43 données désignant des grands sites de France mais seuls 11 sont visualisés

# L’exemple des grands sites de France – 1.1

Utilisation d’un widget ListView pour visualiser les items




La sélection est affichée dans un SnackBar

# Les données sont dans une classe – 1.2

```dart
import 'dart:core';

class GrandsSitesDeFrance {
  GrandsSitesDeFrance () {
    this._addAll() ;
  }
  List<String> _grandsSites = List<String>.filled(0,"",growable: true) ;

  void _addAll () {
    _grandsSites.add ("AVEN D'ORGNAC") ;
    _grandsSites.add ("BAIE DE SOMME") ;
    _grandsSites.add ("BALLON D'ALSACE") ;
    …
    _grandsSites.add ("VALLEE DU HÉRISSON - PLATEAU DES 7 LACS") s
    _grandsSites.add ("VEZELAY") ;
  }

  List<String> get grandSites => _grandsSites ;
  // List<String> getAll() { return (_grandsSites) ; }

  String remove(int index) { return _grandsSites.removeAt(index) ; }

  void add (String siteName) { _grandsSites.add(siteName) ; }

  void insert (int position, String siteName) {_grandsSites.insert(position, siteName) ; }
}
```

Les grands sites sont ajoutés dans le constructeur via la méthode privée _addAll dans une collection de type List<String>

Le code Dart se comprend aisément

Un getter version Dart équivalente à la ligne suivante

Fonctions classiques d'ajout, suppression et insertion qui se traduisent par des appels, là aussi classiques, aux fonctions de la collection

# L'interface utilisateur en Flutter – 1.3

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext ctxt) {
    return new MaterialApp(
      home: new ListDisplay(),
    );
  }
}
```

Notre classe précédemment vue

Notre écran d'accueil est un widget sans état

Le widget retourné est un MaterialApp dont la propriété home va référencé notre instance de classe ListDisplay

StatelessWidget : sans état ce qui convient à des parties d'interfaces utilisateurs qui ne sont pas mises à jour

# L'interface utilisateur en Flutter – 1.4

```
class ListDisplay extends StatelessWidget {
  var listeDesGrandsSitesDeFrance = GrandsSitesDeFrance().grandSites;
  @override
  Widget build (BuildContext context) {
    return Scaffold(
        appBar: AppBar(title: new Text("Grands Sites de France"),),
        body: ListView.builder (
          itemCount: listeDesGrandsSitesDeFrance.length,
          itemBuilder: (BuildContext context, int index) {
            return Card(etet
                child : ListTile
                  title: Text(listeDesGrandsSitesDeFrance[index],
                    textAlign: TextAlign.center,
                  ),
                  onTap: () {
                    var snackBar = SnackBar(content: Text(listeDesGrandsSitesDeFrance[index]));
                    ScaffoldMessenger.of(context).showSnackBar(snackBar);
                  }
              )
            );
          }
        ),
    );
  }
}
```

On récupère une référence sur notre source de données

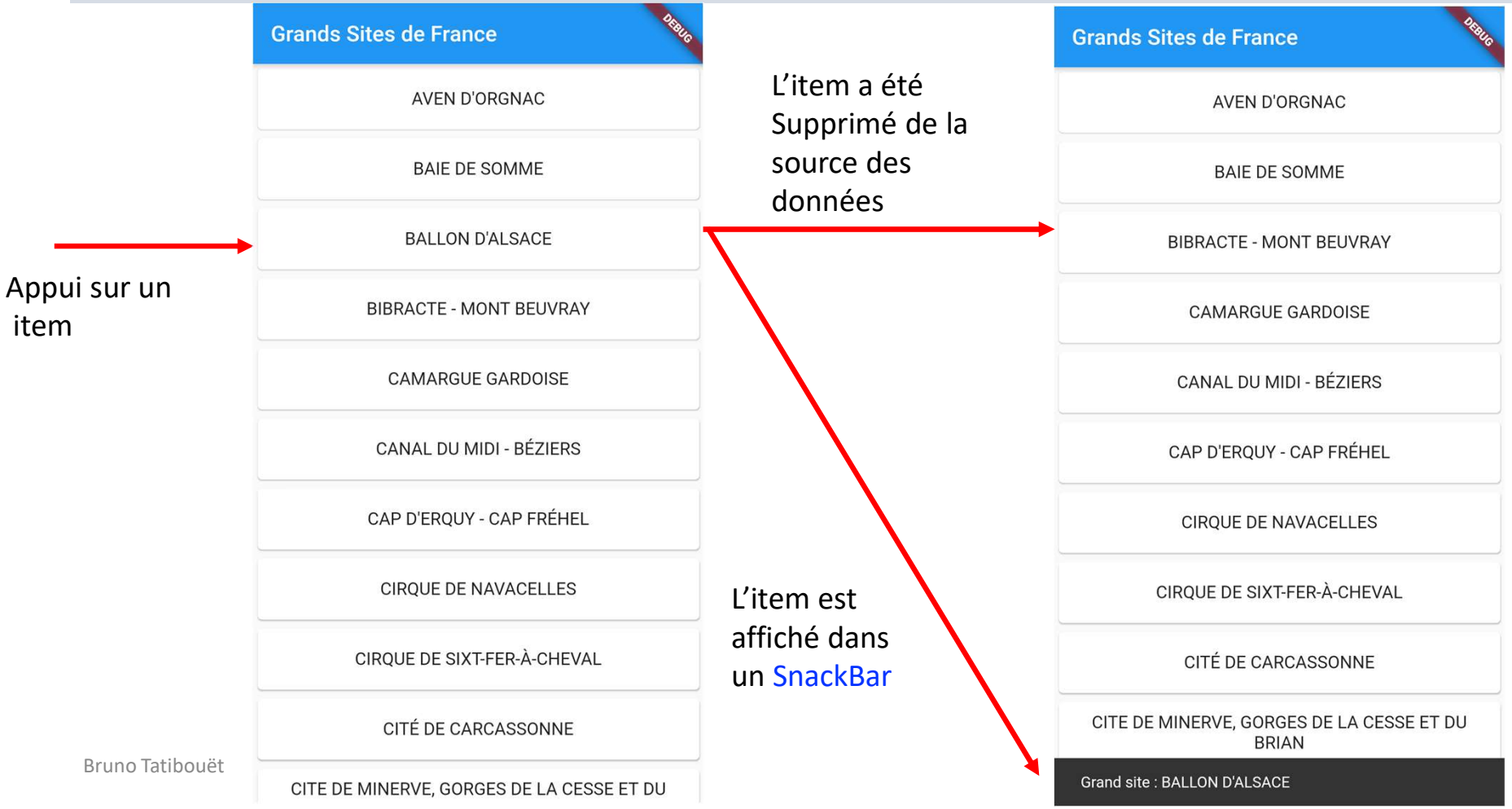
Le widget retourné est un Scaffold (widget qui va occuper tout l'espace) avec des propriétés :

- appBar qui possède un titre
- body qui est ici une ListView qui utilise un builder et qui va créer/gérer les items enfants pendant le défilement de la liste

onTap : La pression sur un texte représentant un grand site permet de l'afficher dans un snackBar (popup)

StatelessWidget : sans état ce qui convient à des parties d'interfaces utilisateurs qui ne sont pas mises à jour

# L’exemple des grands sites de France – 2.1



# Les données sont dans une classe – 2.2

Pas de changements

```
import 'dart:core';

class GrandsSitesDeFrance {
  GrandsSitesDeFrance () {
    this._addAll() ;
  }
  List<String> _grandsSites = List<String>.filled(0,"",growable: true) ;

  void _addAll () {
    _grandsSites.add ("AVEN D'ORGNAC") ;
    _grandsSites.add ("BAIE DE SOMME") ;
    _grandsSites.add ("BALLON D'ALSACE") ;
    …
    _grandsSites.add ("VALLEE DU HÉRISSON - PLATEAU DES 7 LACS") s
    _grandsSites.add ("VEZELAY") ;
  }

  List<String> get grandSites => _grandsSites ;
  // List<String> getAll() { return (_grandsSites) ; }

  String remove(int index) { return _grandsSites.removeAt(index) ; }

  void add (String siteName) { _grandsSites.add(siteName) ; }

  void insert (int position, String siteName) {_grandsSites.insert(position, siteName) ; }
}
```

Les grands sites sont ajoutés dans le constructeur via la méthode privée _addAll dans une collection de type List<String>

Le code Dart se comprend aisément

Un getter version Dart équivalente à la ligne suivante

Fonctions classiques d'ajout, suppression et insertion qui se traduisent par des appels là aussi classiques aux fonctions de la collection

# L'interface utilisateur en Flutter – 2.3

```
void main() { runApp(MyApp()); }

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext ctxt) {
    return MaterialApp(
      home: GrandsSitesDeFranceView(),
    );
  }
}

class GrandsSitesDeFranceView extends StatefulWidget {
  @override
  _GrandsSitesDeFranceStateView createState() {
    return _GrandsSitesDeFranceStateView();
  }
}
```

Notre écran d'accueil est un widget sans état

Le widget retourné est un MaterialApp dont la propriété home va référencer notre instance de classe GrandsSitesDeFranceView

StatefulWidget : l'interface va être mise à jour

```
class _GrandsSitesDeFranceStateView
        extends State<GrandsSitesDeFranceView> {

  var _gsdf = GrandsSitesDeFrance();

  void _remove(int index){
    setState(() { _gsdf.remove(index); }
    );
  }
```

# L'interface utilisateur en Flutter – 2.4

```
@override
Widget build (BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(title: new Text("Grands Sites de France"),),
    body: new ListView.builder (
        itemCount: _gsdf.grandsSites.length,
        itemBuilder: (BuildContext context, int index) {
          final item = _gsdf.grandsSites[index];
          return Card(
              child : ListTile (
                  title: Text(item,
                              textAlign: TextAlign.center),
                  onTap: () {
                    var snackBar = SnackBar(content: Text('Grand site : $item'));
                    ScaffoldMessenger.of(context).showSnackBar(snackBar);
                    _remove(index) ;
                  }
              )
          );
        }
    ),
  );
  }
}
```

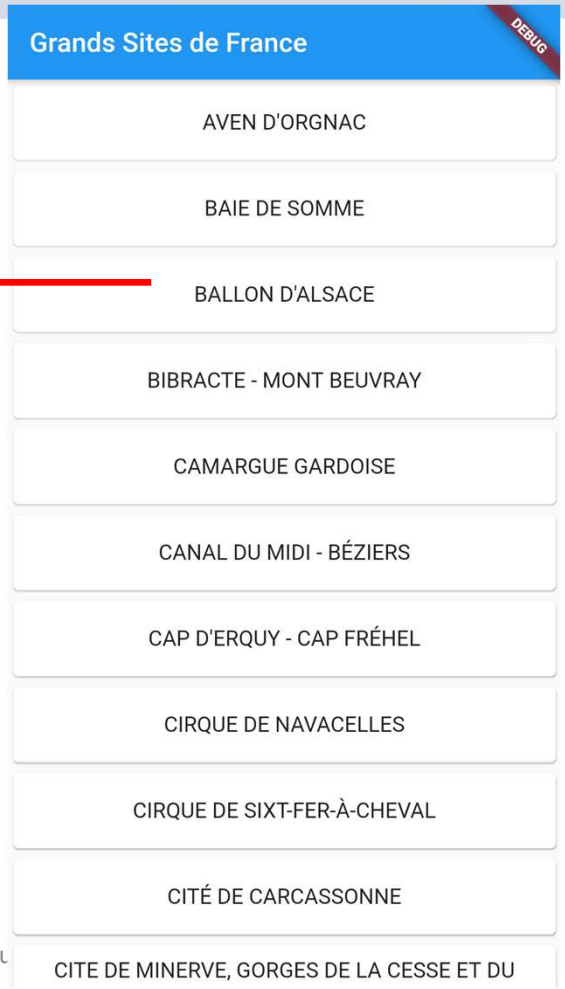
Le widget retourné est un Scaffold (widget qui va occuper tout l'espace) avec des propriétés :

- appBar qui possède un titre
- body qui est ici une ListView qui utilise un builder qui va créer/gérer les items enfants pendant le défilement de la liste

Contient un setState qui va permettre de réexécuter le build et donc de reconstruire cette partie de l'interface utilisateur

onTap : La pression sur un texte représentant un grand site permet de l'afficher dans un snackBar (popup) et de le supprimer

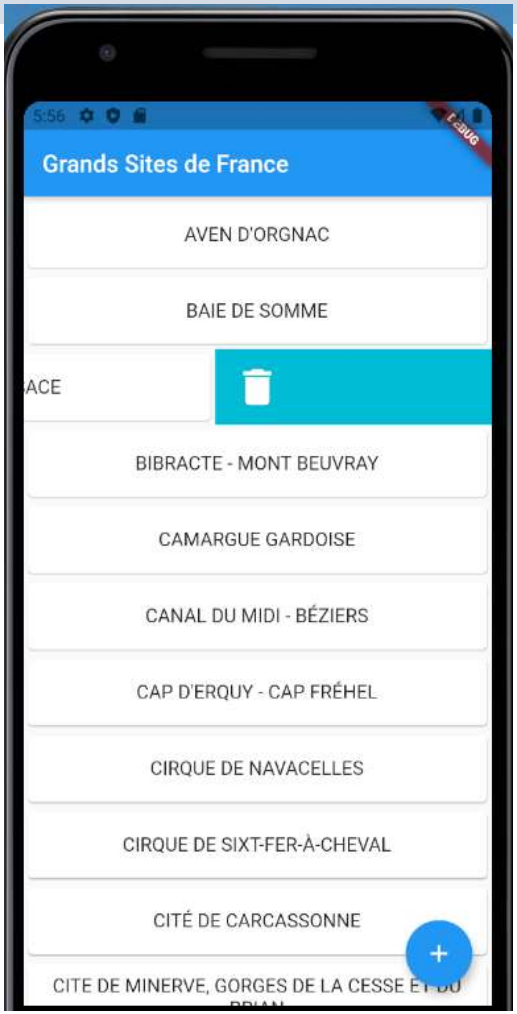
# L’exemple des grands sites de France – 3.1.1




Appui sur un item et glissement

L’item est supprimé de la source des données

L’item est affiché dans un SnackBar



# L’exemple des grands sites de France – 3.1.2

L’item supprimé est affiché dans un SnackBar
Il est possible d’annuler celle-ci


L’item supprimé est ajouté à la source des données en fin

# Les données sont dans une classe – 3.2

Pas de changements

```
import 'dart:core';

class GrandsSitesDeFrance {
  GrandsSitesDeFrance () {
    this._addAll() ;
  }
  List<String> _grandsSites = List<String>.filled(0,"",growable: true) ;

  void _addAll () {
    _grandsSites.add ("AVEN D'ORGNAC") ;
    _grandsSites.add ("BAIE DE SOMME") ;
    _grandsSites.add ("BALLON D'ALSACE") ;
    …
    _grandsSites.add ("VALLEE DU HÉRISSON - PLATEAU DES 7 LACS") s
    _grandsSites.add ("VEZELAY") ;
  }

  List<String> get grandSites => _grandsSites ;
  // List<String> getAll() { return (_grandsSites) ; }

  String remove(int index) { return _grandsSites.removeAt(index) ; }

  void add (String siteName) { _grandsSites.add(siteName) ; }

  void insert (int position, String siteName) {_grandsSites.insert(position, siteName) ; }
}
```

Les grands sites sont ajoutés dans le constructeur via la méthode privée _addAll dans une collection de type List<String>

Le code Dart se comprend aisément

Un getter version Dart équivalente à la ligne suivante

Fonctions classiques d'ajout, suppression et insertion qui se traduisent par des appels là aussi classiques aux fonctions de la collection

# L'interface utilisateur en Flutter – 3.3

Pas de changements

```
void main() { runApp(MyApp()); }

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext ctxt) {
    return MaterialApp(
      home: GrandsSitesDeFranceView(),
    );
  }
}

class GrandsSitesDeFranceView extends StatefulWidget {
  @override
  _GrandsSitesDeFranceStateView createState() {
    return _GrandsSitesDeFranceStateView();
  }
}
```

Notre écran d'accueil est un widget sans état

Le widget retourné est un MaterialApp dont la propriété home va référencer notre instance de classe GrandsSitesDeFranceView

StatefulWidget : l'interface va être mise à jour

```
class _GrandsSitesDeFranceStateView
        extends State<GrandsSitesDeFranceView> {

  var _gsdf = GrandsSitesDeFrance();

  void _remove(int index){
    setState(() { _gsdf.remove(index); }
    );
  }
```

# L'interface utilisateur en Flutter – 3.4

```
@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text("Grands Sites de France"),
    ),
    body: new ListView.builder(
        itemCount: _gsdf.grandsSites.length,
        itemBuilder: (BuildContext context, int index) {
          final item = _gsdf.grandsSites[index];
          return Dismissible(
              key: Key(item),
              background: Container(child: Icon(Icons.delete, size: 40,color: Colors.white), color: Colors.cyan),
              onDismissed: (direction) {
                _remove(index);
                var snackBar = SnackBar(
                    content: Text('Suppression du grand site $item'),
                    action: SnackBarAction(
                            label: 'Annulation suppression',
                            onPressed: () { setState(() { _gsdf.add(item); });
                }),);
                ScaffoldMessenger.of(context).showSnackBar(snackBar);
              },
              child: Card(
```

Le widget retourné est un Scaffold (widget qui va occuper tout l'espace) avec des propriétés :

- appBar qui possède un titre
- body qui est ici une ListView qui utilise un builder qui va créer/gérer les items enfants pendant le défilement de la liste
- floatingActionButton qui permettra d'ajouter un grand site

Conteneur qui permet de gérer la suppression par glissement avec l'évènement onDismissed.

Le code est identique à celui de l'exemple précédent : juste le Dismissible qui s'est intercalé

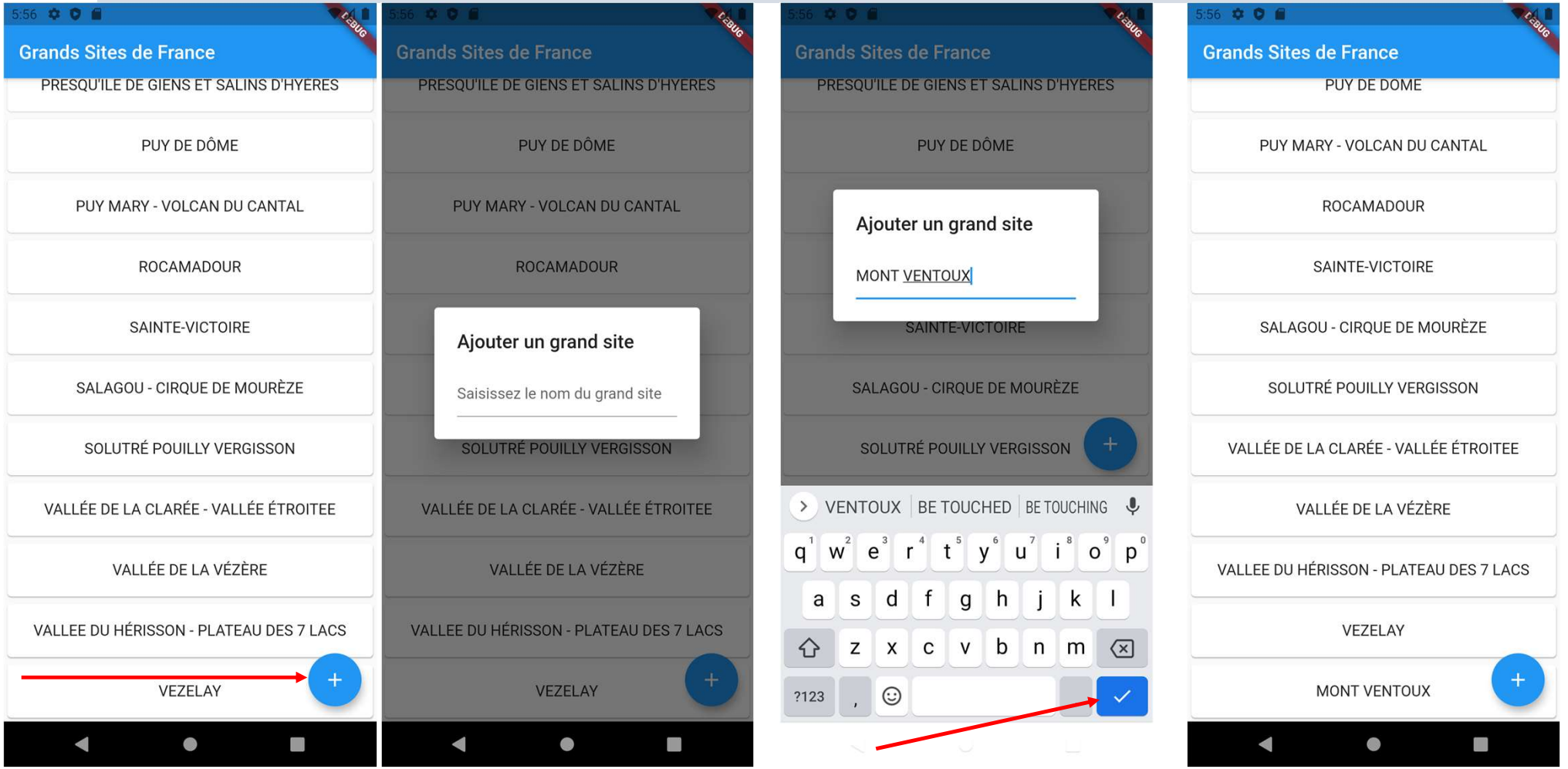
# L'interface utilisateur en Flutter – 3.5

```
}), // fin du body
  floatingActionButton: FloatingActionButton(
    onPressed: () {
      showDialog(
          context: context,
          barrierDismissible: false,
          builder: (BuildContext context) {
            return AlertDialog(
              title: Text('Ajouter un grand site'),
              content: TextField(
                  decoration: InputDecoration(
                      hintText: "Saisissez le nom du grand site"),
                  onSubmitted: (value) {
                     setState(() { _gsdf.add(value); });
                     Navigator.of(context).pop();
                  }),
            );
          });
    },
    tooltip: 'ajouter un grand site',
    child: Icon(Icons.add),
  ),);}}
```

On ajoute un grand site via une boite de dialogue qui permet la saisie du texte (cf cours précédent)

Le composant Navigator qui permet de revenir à la page précédente (cf cours précédent)

# L’exemple des grands sites de France – 3.6



# Téléchargement des données – 4.1

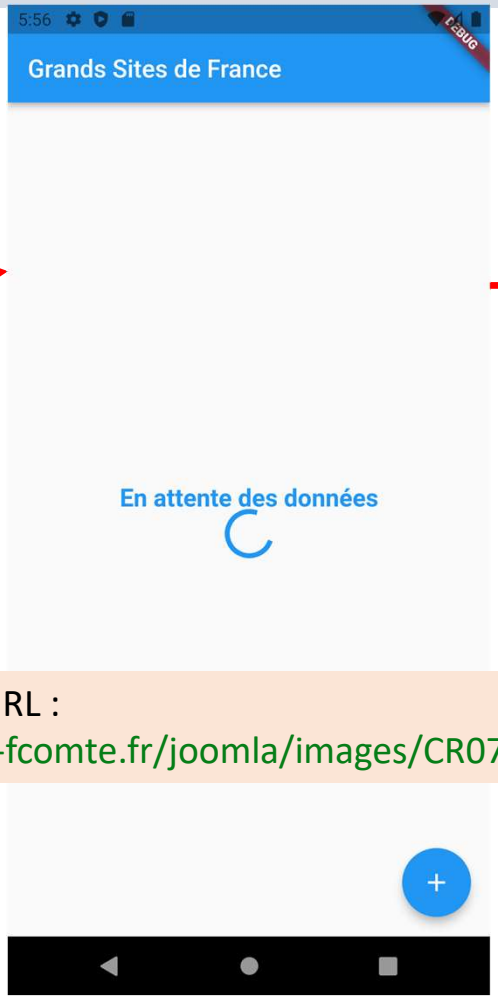
- Ici les données de notre exemple ne sont pas ajoutées initialement dans la classe GrandsSitesDeFrance mais téléchargées depuis une URL
- La fonction asynchrone _getWebListeTxtGrandsSites retourne un résultat de type Future<GrandSitesDeFrance>

```dart
import 'dart:core';

class GrandsSitesDeFrance {

  List<String> _grandsSites = List<String>.filled(0,"",growable: true) ;

  List<String> get grandSites => _grandsSites ;

  String remove(int index) { return _grandsSites.removeAt(index) ; }
  void add (String siteName) { _grandsSites.add(siteName) ; }
  void insert (int position, String siteName) {_grandsSites.insert(position, siteName) ; }
}
```

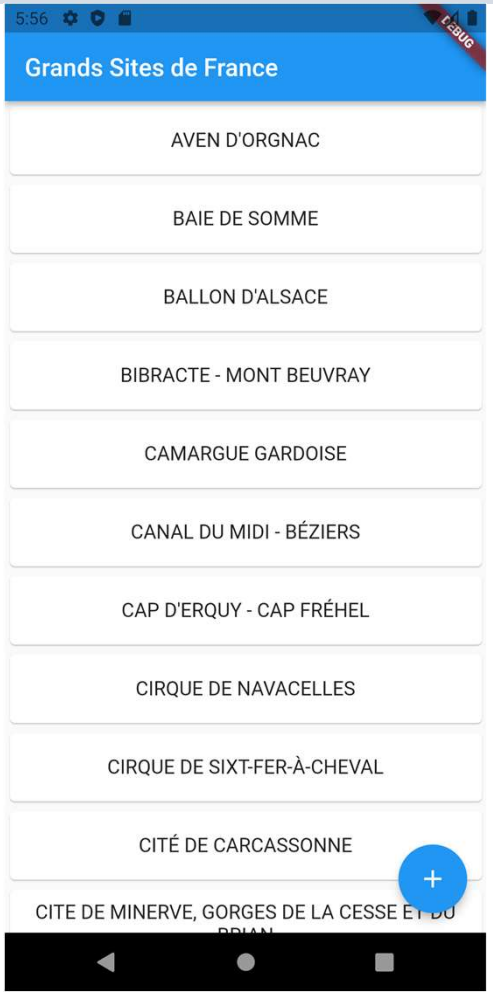
Seule la fonction addAll a disparu puisque l'on va télécharger les éléments.

# L’exemple des grands sites de France – 4.2

Attente des données




Les données téléchargées sont affichées




Les données sont à l’URL :
https://dept-info.univ-fcomte.fr/joomla/images/CR0700/GrandSites.txt

# L'interface utilisateur en Flutter – 4.4

```dart
void main() { runApp(MyApp()); }

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext ctxt) {
    return MaterialApp(
      home: GrandsSitesDeFranceView(),
    );
  }
}

class GrandsSitesDeFranceView extends StatefulWidget {
  @override
  _GrandsSitesDeFranceStateView createState() {
    return _GrandsSitesDeFranceStateView();
  }
}
```

Notre écran d'accueil est un widget sans état

Le widget retourné est un MaterialApp dont la propriété home va référencer notre instance de classe GrandsSitesDeFranceView

StatefulWidget : l'interface va être mise à jour

```dart
class _GrandsSitesDeFranceStateView
        extends State<GrandsSitesDeFranceView> {

  final url = Uri.parse('https://..GrandSites.txt');
  late Future<GrandsSitesDeFrance> _futureGrandsSites;
  late GrandsSitesDeFrance _gsdf ;

  @override
  void initState() { super.initState();
    _futureGrandsSites = _getWebListeTxtGrandsSites();
  }
```

# Le téléchargement des données – 4.5

```
Future<GrandsSitesDeFrance> _getWebListeTxtGrandsSites() async {
  var gsdf = GrandsSitesDeFrance();
  var reponse = await http.get(url);
  if (reponse.statusCode == 200) {
    String source = Utf8Decoder().convert(reponse.bodyBytes);
    LineSplitter ls = LineSplitter();
    List<String> lines = ls.convert(source);
    for (var line in lines) {
      gsdf.add(line);
    }
    return (gsdf);
  } else {
    // If the server did not return a 200 OK response,
    // then throw an exception.
    throw Exception('Failed to load listes de grands sites');
  }
}
```

On récupère la page WEB associée à l'URL

1. Si la page WEB a été obtenue sans erreur on convertit son contenu en String.
2. Cette chaîne est découpée en une liste de lignes qui sont des chaînes.
3. Chaque ligne (un nom de grand site) est insérée dans l'instance GrandSitesDeFrance

# L'interface utilisateur en Flutter – 4.6.1

```
@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text("Grands Sites de France"),
    ),
    body: FutureBuilder<GrandsSitesDeFrance>(
        future: _futureGrandsSites,
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            return ListView.builder(
              itemCount: snapshot.data!.grandsSites.length,
              itemBuilder: (BuildContext context, int index) {
                _gsdf = snapshot.data! ;
                final item = snapshot.data!.grandsSites[index];
                return Dismissible(
```

Le widget retourné est un Scaffold (widget qui va occuper tout l'espace) avec des propriétés :

- appBar qui possède un titre
- body qui est ici un widget FutureBuilder qui utilise le résultat d'un future pour se construire lui-même. Si le résultat est là c'est une ListView sinon c'est un widget indiquant la progression.
- floatingActionButton qui permettra d'ajouter un grand site

Le deuxième paramètre permet d'accéder aux résultats du téléchargement et de savoir s'ils sont disponibles.

Le code après le Dismissible est identique à celui de l'exemple précédent et celui du FloatingActionButton

# L'interface utilisateur en Flutter – 4.6.2

```
@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text("Grands Sites de France"),
    ),
    body: FutureBuilder<GrandsSitesDeFrance>(
        future: _futureGrandsSites,
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            return ListView.builder(
              …
          } else {
            return _attente();
          }
```

```
Widget _attente() {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        Text('En attente des données',
            style: TextStyle(
                color: Colors.white,
                fontSize: 20.0,
                fontWeight: FontWeight.bold)),
        const CircularProgressIndicator(),
      ],
    ),
  );
 }
}
```

Indicateur de progression.

# Stocker les données dans une BD – 5.1

- Les données de notre exemple seront insérées dans une base de données depuis la classe GrandsSitesDeFrance : ce sera son seul usage
- Le travail de gestion des données (insertion, suppression, récupération) sera réalisé au sein de la classe GrandSiteDBHelper pour créer et gérer la base de données
  - Elle utilise une classe GrandSite représentant un grand site
  - Les fonctions asynchrones l'accès à la base de données
- Le package sqlflite est utilisé. Il permet d'accéder au système de base de données sqlite disponible sur Android, les navigateurs, …

```
class GrandsSitesDeFrance {

  static void addAllToDB (GrandSiteDBHelper gsdfDB) {
    gsdfDB.insert (GrandSite("AVEN D'ORGNAC")) ;
    gsdfDB.insert (GrandSite("BAIE DE SOMME"))
    gsdfDB.insert (GrandSite("BALLON D'ALSACE")) ;
     …
 }
}
```

# Stocker les données dans une BD – 5.2

```dart
class GrandSite {
  GrandSite(this.nom) {}
  int id = 0 ;
  String nom = "" ;
  GrandSite.fromMap (Map<dynamic, dynamic> map) {
        id = map[colonneId]; nom = map[colonneNom] ;
  }
  Map<String, dynamic> toMap() {
      var map = <String, dynamic> { colonneNom: nom };
      if (id != 0) { map[colonneId] = id ; }
      return map ;
  }
}
```

```dart
final String nomBaseDonnees = 'GrandSiteDB.db' ;
final int databaseVersion = 1 ;
final String tableGrandSite = 'grandsite' ;
final String colonneId = 'id' ;
final String colonneNom = 'nom' ;
```

Une instance de GrandSite est une ligne de la table grandsite

# Stocker les données dans une BD – 5.3.1

```
class GrandSiteDBHelper {
  GrandSiteDBHelper._privateConstructor() ;

  static final GrandSiteDBHelper instance =
      GrandSiteDBHelper._privateConstructor()  ;
  static Database? _db = null ;

  static Future<bool> exists() async {
     return (_db != null) ;
  }
  Future<Database?> get db async {
    if (_db != null) { return _db ; }
    else {
      _db = await _initDatabase() ;
      return _db ;
    }
  }
  _initDatabase() async {
    Directory documentsDirectory = await getApplicationDocumentsDirectory() ;
    String path = join (documentsDirectory.path, nomBaseDonnees) ;
    return  await openDatabase(path, version: databaseVersion, onCreate: _open);
  }
```

```
final String nomBaseDonnees = 'GrandSiteDB.db' ;
final int databaseVersion = 1 ;
final String tableGrandSite = 'grandsite' ;
final String colonneId = 'id' ;
final String colonneNom = 'nom' ;
```

```
Future _open (Database db, int version) async {
   await db.execute('''
     create table $tableGrandSite (
     $colonneId integer primary key autoincrement,
     $colonneNom text)
     '''
   );
 }

 Future close() async => _db?.close() ;
```

# Stocker les données dans une BD – 5.3.2

```
Future<List<String>> getGrandsSites () async {
 Database? db = await instance.db ;
 List<Map>? maps = await db?.query (tableGrandSite);

 if (maps != null) {
   List<String> grandsSitesNoms = [] ; //List<String>.filled(maps.length, "");
   for (Map map in maps) {
     grandsSitesNoms.add(map[colonneNom]);
   }
   return grandsSitesNoms ;
 }
 return List<String>.filled(0, "");
}
```

```
final String nomBaseDonnees = 'GrandSiteDB.db' ;
final int databaseVersion = 1 ;
final String tableGrandSite = 'grandsite' ;
final String colonneId = 'id' ;
final String colonneNom = 'nom' ;
```

# Stocker les données dans une BD – 5.3.3

```dart
final String nomBaseDonnees = 'GrandSiteDB.db' ;
final int databaseVersion = 1 ;
final String tableGrandSite = 'grandsite' ;
final String colonneId = 'id' ;
final String colonneNom = 'nom' ;
```

```dart
Future<int?> insert (GrandSite gs) async {
  Database? db = await instance.db ;
  int? id = await db?.insert(tableGrandSite, gs.toMap());
  return id ;
}

Future<int?> deleteById (int id) async {
  Database? db = await instance.db ;
  return await db?.delete(tableGrandSite, where: '$colonneId = ?', whereArgs: [id]);
}

   Future<int?> deleteByName (String nom) async {
     Database? db = await instance.db ;
     return await db?.delete(tableGrandSite, where: '$colonneNom = ?', whereArgs: [nom]);
   }
```

# L'application en Flutter – 5.4.1

```dart
void main() { runApp(MyApp()); }

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext ctxt) {
    return MaterialApp(
      home: GrandsSitesDeFranceView(),
    );
  }
}

class GrandsSitesDeFranceView extends StatefulWidget {
  @override
  _GrandsSitesDeFranceStateView createState() {
    return _GrandsSitesDeFranceStateView();
  }
}
```

Notre écran d'accueil est un widget sans état

Le widget retourné est un MaterialApp dont la propriété home va référencer notre instance de classe GrandsSitesDeFranceView

StatefulWidget : l'interface va être mise à jour

# L’application en Flutter – 5.4.2

```dart
class _GrandsSitesDeFranceStateView extends State<GrandsSitesDeFranceView> {

  GrandSiteDBHelper? _gsdfDB = null;
  String _nomGrandSite = "";

  @override
  void dispose() {
    super.dispose();
    _gsdfDB?.close();
  }
```

```dart
  @override
  void initState() {
    super.initState();
    _getInstance();
  }

  Future<GrandSiteDBHelper?> _getInstance() async {
    _gsdfDB = await GrandSiteDBHelper.instance;
    GrandsSitesDeFrance.addAllToDB(_gsdfDB!) ;
    setState(() {});
    return _gsdfDB ;
  }
```

# L'interface utilisateur en Flutter – 5.5

```
@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text("Grands Sites de France"),
    ),
    body:FutureBuilder<List<String>>(
      future: _gsdfDB?.getGrandsSites(),
      builder: (context, snapshot) {
         if (snapshot.data == null)
            return _attente();
         else {
            return ListView.builder(
                itemCount: snapshot.data?.length,
                itemBuilder: (BuildContext context, int index) {
              final item = snapshot.data?[index];
              return Dismissible(
                 key: UniqueKey(),
```

Le widget retourné est un Scaffold (widget qui va occuper tout l'espace) avec des propriétés :

- appBar qui possède un titre
- body qui est ici un widget FutureBuilder qui utilise le résultat d'un future pour se construire lui-même. Si le résultat est là c'est une ListView sinon c'est un widget indiquant la progression.
- floatingActionButton qui permettra d'ajouter un grand site

Le deuxième paramètre permet d'accéder aux résultats du téléchargement et de savoir s'ils sont disponibles.

_attente est identique à celle du téléchargement

Le code après le Dismissible est identique à celui de l'exemple précédent et celui du FloatingActionButton aussi aussi à l'exception de l'utilisation des fonctions deleteByName et insert de GrandSiteDBHelper

# Télécharger les données dans une BD – 6.1

- Les données de notre exemple seront téléchargées depuis une URL et insérées dans une base de données
  - La classe GrandsSitesDeFrance disparaît au profit d'une classe GrandSite représentant un grand site et d'une classe GrandSiteDBHelper pour créer et gérer la base de données
- Les fonctions asynchrones sont utilisées à la fois pour le téléchargement et l'accès à la base de données

```
class GrandSite {
  GrandSite(this.nom) {}
  int id = 0 ;
  String nom = "" ;
  GrandSite.fromMap (Map<dynamic, dynamic> map) {
        id = map[colonneId]; nom = map[colonneNom] ;
  }
  Map<String, dynamic> toMap() {
      var map = <String, dynamic> { colonneNom: nom };
      if (id != 0) { map[colonneId] = id ; }
      return map ;
  }
}
```

```
final String nomBaseDonnees = 'GrandSiteDB.db' ;
final int databaseVersion = 1 ;
final String tableGrandSite = 'grandsite' ;
final String colonneId = 'id' ;
final String colonneNom = 'nom' ;
```

Une instance de GrandSite est une ligne de la table grandsite

# Télécharger les données dans une BD – 6.2.1

```
class GrandSiteDBHelper {
  GrandSiteDBHelper._privateConstructor() ;

  static final GrandSiteDBHelper instance =
      GrandSiteDBHelper._privateConstructor()  ;
  static Database? _db = null ;

  static Future<bool> exists() async {
     return (_db != null) ;
  }
  Future<Database?> get db async {
    if (_db != null) { return _db ; }
    else {
      _db = await _initDatabase() ;
      return _db ;
    }
  }
  _initDatabase() async {
    Directory documentsDirectory = await getApplicationDocumentsDirectory() ;
    String path = join (documentsDirectory.path, nomBaseDonnees) ;
    return  await openDatabase(path, version: databaseVersion, onCreate: _open);
  }
```

```
final String nomBaseDonnees = 'GrandSiteDB.db' ;
final int databaseVersion = 1 ;
final String tableGrandSite = 'grandsite' ;
final String colonneId = 'id' ;
final String colonneNom = 'nom' ;
```

```
Future _open (Database db, int version) async {
   await db.execute('''
     create table $tableGrandSite (
     $colonneId integer primary key autoincremen
     $colonneNom text)
     '''
   );
 }

 Future close() async => _db?.close() ;
```

# Télécharger les données dans une BD – 6.2.2

```dart
final String nomBaseDonnees = 'GrandSiteDB.db' ;
final int databaseVersion = 1 ;
final String tableGrandSite = 'grandsite' ;
final String colonneId = 'id' ;
final String colonneNom = 'nom' ;
```

```dart
Future<List<String>> getGrandsSites () async {
  Database? db = await instance.db ;
  List<Map>? maps = await db?.query (tableGrandSite);

  if (maps != null) {
    List<String> grandsSitesNoms = [] ; //List<String>.filled(maps.length, "");
    for (Map map in maps) {
      grandsSitesNoms.add(map[colonneNom]);
    }
    return grandsSitesNoms ;
  }
  return List<String>.filled(0, "");
}
```

```dart
Future<int?> insert (GrandSite gs) async {
  Database? db = await instance.db ;
  int? id = await db?.insert(tableGrandSite, gs.toMap());
  return id ;
}

Future<int?> delete (int id) async {
  Database? db = await instance.db ;
  return await db?.delete(tableGrandSite, where: '$colonneId = ?', whereArgs: [id]);
}
```

# Télécharger les données dans une BD – 6.3.1

```
void main() { runApp(MyApp()); }

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext ctxt) {
    return MaterialApp(
      home: GrandsSitesDeFranceView(),
    );
  }
}

class GrandsSitesDeFranceView extends StatefulWidget {
  @override
  _GrandsSitesDeFranceStateView createState() {
    return _GrandsSitesDeFranceStateView();
  }
}
```

Notre écran d'accueil est un widget sans état

Le widget retourné est un MaterialApp dont la propriété home va référencer notre instance de classe GrandsSitesDeFranceView

StatefulWidget : l'interface va être mise à jour

# Télécharger les données dans une BD – 6.3.2

```
class _GrandsSitesDeFranceStateView extends State<GrandsSitesDeFranceView> {
  final url = Uri.parse(
      'https://dept-info.univ-fcomte.fr/joomla/images/CR0700/GrandSites.txt');
  GrandSiteDBHelper? _gsdfDB = null;
  String _nomGrandSite = "";

  @override
  void dispose() {
    super.dispose();
    _gsdfDB?.close();
  }
```

```
@override
void initState() {
  super.initState();
  _getWebListeTxtGrandsSites();
  _getInstance();
}

Future<GrandSiteDBHelper?> _getInstance() async {
  _gsdfDB = await GrandSiteDBHelper.instance;
  return _gsdfDB;
}
```

# Télécharger les données dans une BD – 6.4

```
Future<void> _getWebListeTxtGrandsSites() async {
  bool dbExist = await GrandSiteDBHelper.exists();
  if (!dbExist) {
    var reponse = await http.get(url);
    if (reponse.statusCode == 200) {
      String source = Utf8Decoder().convert(reponse.bodyBytes);
      LineSplitter ls = new LineSplitter();
      List<String> lines = ls.convert(source);
      for (var line in lines) {
        _gsdfDB?.insert(GrandSite(line));
      }
    }
  }
  setState(() {});
}
```

On récupère la page WEB associée à l'URL

1. Si la page WEB a été obtenue sans erreur on convertit son contenu en String.
2. Cette chaîne est découpée en une liste de lignes qui sont des chaînes.
3. Chaque ligne (un nom de grand site) est insérée dans la base de données

La fonction asynchrone _getWebListeTxtGrandsSites retourne un résultat de type Future<void> puisqu'elle insère les éléments dans la base de données au fur et à mesure.

# Télécharger les données dans une BD – 6.5

```
@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text("Grands Sites de France"),
    ),
    body:FutureBuilder<List<String>>(
      future: _gsdfDB?.getGrandsSites(),
      builder: (context, snapshot) {
         if (snapshot.data == null)
            return _attente();
         else {
            return ListView.builder(
                itemCount: snapshot.data?.length,
                itemBuilder: (BuildContext context, int index) {
              final item = snapshot.data?[index];
              return Dismissible(
                 key: UniqueKey(),
```

Le widget retourné est un Scaffold (widget qui va occuper tout l'espace) avec des propriétés :

- appBar qui possède un titre
- body qui est ici un widget FutureBuilder qui utilise le résultat d'un future pour se construire lui-même. Si le résultat est là c'est une ListView sinon c'est un widget indiquant la progression.
- floatingActionButton qui permettra d'ajouter un grand site

Le deuxième paramètre permet d'accéder aux résultats du téléchargement et de savoir s'ils sont disponibles.

_attente est identique à celle du téléchargement

Le code après le Dismissible est identique à celui de l'exemple précédent et celui du FloatingActionButton aussi aussi à l'exception de l'utilisation des fonctions deleteByName et insert de GrandSiteDBHelper