Gestion de données locales avec Hive dans Flutter

Gestion de données locales avec Hive dans Flutter



1. **Objectif**
    - Apprendre à utiliser Hive, une base de données locale rapide et légère, pour stocker et gérer des notes/tâches dans une application Flutter.
2. **Hive pour Flutter : une base de données locale rapide et légère**
    1. **Introduction à Hive**
        - Hive est une base de données NoSQL révolutionnaire, spécialement conçue pour les applications Dart et Flutter.
        - Créée par Simon Leier, Hive répond au besoin d'une solution de stockage local simple, rapide et sécurisée, sans la complexité des bases de données SQL traditionnelles.
        - C'est un excellent choix pour stocker localement de petites quantités de données, telles que les préférences utilisateur, les scores de jeu ou les paniers d'achat.
        - Hive est également une bonne option pour les applications nécessitant une utilisation hors ligne.
        - Contrairement à d'autres solutions de bases de données qui s'appuient sur du code natif ou nécessitent des procédures de configuration complexes, Hive est

apcpedagogie.com

entièrement écrit en Dart pur. Cela signifie qu'il fonctionne parfaitement sur toutes les plateformes prises en charge par Flutter : iOS, Android, Web, Desktop et même les applications Dart côté serveur.

2. **Caractéristiques**

   - **Rapide** : Hive est très rapide, avec des opérations de lecture et d'écriture généralement beaucoup plus rapides que les autres bases de données locales pour Flutter.
   - **Léger**: Hive est très léger, ce qui en fait un bon choix pour les applications qui doivent économiser de la mémoire.
   - **Multiplateforme**: Hive est multiplateforme, il peut donc être utilisé pour créer des applications pour Android, iOS et le Web.
   - **Type sécurisé**: Hive est un type sécurisé, ce qui permet d'éviter les erreurs et d'améliorer la qualité du code.
   - Zéro configuration : aucun schéma à définir, aucune migration à écrire, aucune configuration complexe. Ouvrez une boîte, stockez vos données, et le tour est joué.

3. **Les HiveBox**

   - Une `HiveBox` est comparable à une table SQL :
     - Chaque box stocke un type de données spécifique
     - Vous pouvez avoir plusieurs boxes dans une app
     - Chaque élément dans une box a une clé unique

3. Configuration de Hive dans votre projet Flutter

1. **Étape 1 : Créer un projet Flutter**

   - Créer un projet avec `flutter create hive_app`.
   - Se déplacer dans le projet : `cd hive_app`.

2. **Étape 2 : Ajouter les dépendances**

   - Dans `pubspec.yaml`, ajoutez les dépendances suivantes pour Hive :

```
dependencies:
  flutter:
```

```
    sdk: flutter
  hive: ^2.2.3
  hive_flutter: ^1.1.0

dev_dependencies:
  flutter_test:
    sdk: flutter
  hive_generator: ^1.1.0
  build_runner: ^2.4.0
```

3. **Explication :**
   - **hive**: Le package principal
   - **hive_flutter**: Intégration avec Flutter
   - **hive_generator**: Génération automatique du code
   - **build_runner**: Outil pour générer le code
4. Exécutez `flutter pub get` pour installer les dépendances :

5. **Étape 3 : Créer le modèle de données**

   - Créez d'abord le modèle avant de générer l'adaptateur :

```
// models/todo.dart
import 'package:hive/hive.dart';

part 'todo.g.dart'; // Fichier qui sera généré automatiquement

@HiveType(typeId: 0)
class Todo {
  @HiveField(0)
  final String content;

  @HiveField(1)
  final DateTime createdAt;

  @HiveField(2)
  final bool isDone;

  Todo({
    required this.content,
    required this.createdAt,
```

apcpedagogie.com

```
    this.isDone = false,
  });
}
```

6. **Explication :**
   - **`@HiveField(entier)`** - L'identifiant des champs
   - **`@HiveType(typeId: 0)`** - L'identifiant du modèle
   - **À quoi ça sert ?**
     - Identifie de manière unique votre modèle dans Hive
     - Permet la sérialisation/désérialisation correcte
     - Doit être unique pour chaque modèle dans votre application
   - **BONNES PRATIQUES AVEC PLUSIEURS MODELES**

```
// models/constants.dart
class HiveTypeIds {
  static const todo = 0;
  static const user = 1;
  static const category = 2;
  static const settings = 3;
}

// Utilisation
@HiveType(typeId: HiveTypeIds.todo)
class Todo { /* ... */ }

@HiveType(typeId: HiveTypeIds.user)
class User { /* ... */ }
```

7. **Étape 4 : Générer les adaptateurs Hive**

   - Générez les adaptateurs Hive pour vos modèles de données :
   - Lancez cette commande dans le terminal : `flutter packages pub run build_runner build`
   - **Explication de l'adaptateur :**
     - Un adaptateur est un traducteur qui permet à Hive de comprendre comment :
     - **Sérialiser**: Convertir votre objet Dart en données binaires pour le stockage
     - **Désérialiser**: Reconvertir les données binaires en objet Dart

apcpedagogie.com

8. **Étape 5 : Initialiser Hive dans le fichier `main.dart`**

   - Dans `main.dart`, configurez la structure de base de l'application Flutter et initialisez Hive:

```
import 'package:flutter/material.dart';
import 'package:hive_flutter/hive_flutter.dart';
import 'models/todo.dart'; // Votre modèle

void main() async {
  // ? ÉTAPE 1: Initialiser le moteur Flutter
  WidgetsFlutterBinding.ensureInitialized();

  // ? ÉTAPE 2: Initialiser Hive
  await Hive.initFlutter();

  // ? ÉTAPE 3: Enregistrer l'adaptateur
  Hive.registerAdapter(TodoAdapter());

  // ? ÉTAPE 4: Ouvrir la box (base de données)
  await Hive.openBox('todos_box');

  // ? ÉTAPE 5: Lancer l'application
  runApp(MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Gestionnaire de Tâches Hive',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.red),
        useMaterial3: true,
      ),
      home: const HomePage(),
      debugShowCheckedModeBanner: false,
    );
  }
}
```

apcpedagogie.com

9. **Explications détaillées des concepts clés**

1. **WidgetsFlutterBinding.ensureInitialized()**

- **C'est quoi ?**
  - C'est une méthode qui initialise le moteur Flutter et assure que toutes les connexions entre le framework Flutter et le moteur sous-jacent sont établies.
- **Pourquoi est-ce nécessaire avant Hive ?**

```
void main() async {
  // ⚡ ESSENTIEL : "Allume le moteur Flutter" avant toute opération
asynchrone
  WidgetsFlutterBinding.ensureInitialized();

  // Maintenant on peut faire des opérations asynchrones sûres
  await Hive.initFlutter();
  // ...
}
```

2. **Analogie facile à comprendre**
   - Imaginez que vous voulez utiliser votre téléphone :
   - Sans `ensureInitialized()`: → □ Essayez d'envoyer un SMS alors que le téléphone n'est pas encore allumé
   - Avec `ensureInitialized()`: → □ Allumez d'abord le téléphone, puis envoyez le SMS
3. **Quand l'utiliser ?**
   - Toujours quand vous avez des opérations **asynchrone** (`async/await`) dans le `main()` avant `runApp()`.

4. **TodoAdapter() - L'Adaptateur Hive**

- **C'est quoi ?**
  - Un adaptateur est un traducteur qui permet à Hive de comprendre comment :
  - **Sérialiser**: Convertir votre objet Dart en données binaires pour le stockage
  - **Désérialiser**: Reconvertir les données binaires en objet Dart
- **Pourquoi en avons-nous besoin ?**

apcpedagogie.com

```
// VOTRE OBJECT DART
class Todo {
  String content;
  DateTime createdAt;
  Todo(this.content, this.createdAt);
}

// HIVE NE SAIT PAS COMMENT STOCKER CET OBJET DIRECTEMENT
// L'adaptateur lui explique comment faire !
```

5. **Analogie facile à comprendre**
   - Imaginez que vous voulez envoyer une lettre à l'étranger :
   - Votre objet Dart : → La lettre en français
   - Le stockage Hive : → Le système postal international
   - L'adaptateur : → Le traducteur qui convertit le français en anglais

10. **Étape 6 : Créer la page d'accueil**

- Fichier `home_page.dart` → interface pour ajouter, afficher et gérer les tâches.
- Utilisation de `ValueListenableBuilder` pour réactivité.

```
import 'package:flutter/material.dart';
import 'package:hive_flutter/hive_flutter.dart';
import 'package:intl/intl.dart';
import '../models/todo.dart';

class HomePage extends StatefulWidget {
  const HomePage({super.key});

  @override
  State createState() => _HomePageState();
}

class _HomePageState extends State {
  final TextEditingController _textEditingController = TextEditingController();
  late final Box _todosBox;

  @override
  void initState() {
    super.initState();
    _todosBox = Hive.box('todos_box');
```

apcpedagogie.com

```
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text(
        "Gestionnaire de Tâches",
        style: TextStyle(color: Colors.white),
      ),
      backgroundColor: Colors.red,
      actions: [
        IconButton(
          icon: const Icon(Icons.delete, color: Colors.white),
          onPressed: _clearAllTodos,
          tooltip: 'Supprimer toutes les tâches',
        ),
      ],
    ),
    body: _buildBody(),
    floatingActionButton: FloatingActionButton(
      onPressed: () => _showAddTodoDialog(context),
      backgroundColor: Colors.red,
      child: const Icon(Icons.add, color: Colors.white),
    ),
  );
}

Widget _buildBody() {
  return ValueListenableBuilder(
    valueListenable: _todosBox.listenable(),
    builder: (context, Box box, _) {
      final todos = box.values.toList();

      if (todos.isEmpty) {
        return _buildEmptyState();
      }

      // Trier par date (plus récent en premier)
      todos.sort((a, b) => b.createdAt.compareTo(a.createdAt));

      return ListView.builder(
        itemCount: todos.length,
```

apcpedagogie.com

```
          itemBuilder: (context, index) {
            final todo = todos[index];
            final key = _getKeyForTodo(todo, box);

            return _buildTodoItem(todo, key);
          },
        );
      },
    );
}

Widget _buildEmptyState() {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(Icons.note_add, size: 64, color: Colors.grey[300]),
        const SizedBox(height: 16),
        const Text(
          'Aucune tâche pour le moment',
          style: TextStyle(fontSize: 18, color: Colors.grey),
        ),
        const SizedBox(height: 8),
        const Text(
          'Appuyez sur le + pour ajouter une tâche',
          style: TextStyle(fontSize: 14, color: Colors.grey),
        ),
      ],
    ),
  );
}

Widget _buildTodoItem(Todo todo, int? key) {
  return Dismissible(
    key: Key(key?.toString() ?? todo.content),
    background: Container(color: Colors.red),
    secondaryBackground: Container(color: Colors.blue),
    confirmDismiss: (direction) async {
      if (direction == DismissDirection.endToStart) {
        _showEditTodoDialog(context, todo, key);
        return false;
      }
      return true;
```

apcpedagogie.com

```
      },
      onDismissed: (direction) async {
        if (key != null) {
          await _todosBox.delete(key);
          _showSnackBar('Tâche supprimée: ${todo.content}', Colors.red);
        }
      },
      child: Card(
        margin: const EdgeInsets.symmetric(horizontal: 8, vertical: 4),
        child: ListTile(
          title: Text(
            todo.content,
            style: TextStyle(
              decoration: todo.isDone ? TextDecoration.lineThrough : null,
              color: todo.isDone ? Colors.grey : null,
              fontWeight: FontWeight.w500,
            ),
          ),
          subtitle: Text(
            'Créé le: ${DateFormat('dd/MM/yyyy HH:mm').format(todo.createdAt)}',
            style: const TextStyle(fontSize: 12),
          ),
          trailing: Checkbox(
            value: todo.isDone,
            onChanged: (value) {
              if (key != null) {
                final updated = Todo(
                  content: todo.content,
                  createdAt: todo.createdAt,
                  isDone: value ?? false,
                );
                _todosBox.put(key, updated);
              }
            },
          ),
          onTap: () => _showTodoDetails(context, todo),
        ),
      ),
    );
  }

  Future _showAddTodoDialog(BuildContext context) async {
    _textEditingController.clear();
```

apcpedagogie.com

```
    return showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          title: const Text('Ajouter une tâche'),
          content: TextField(
            controller: _textEditingController,
            decoration: const InputDecoration(
              hintText: "Saisissez votre tâche...",
              border: OutlineInputBorder(),
            ),
            autofocus: true,
            maxLines: 3,
          ),
          actions: [
            TextButton(
              onPressed: () => Navigator.pop(context),
              child: const Text('Annuler'),
            ),
            ElevatedButton(
              style: ElevatedButton.styleFrom(backgroundColor: Colors.red),
              onPressed: () async {
                final text = _textEditingController.text.trim();
                if (text.isNotEmpty) {
                  final newTodo = Todo(
                    content: text,
                    createdAt: DateTime.now(),
                    isDone: false,
                  );
                  await _todosBox.add(newTodo);
                  _textEditingController.clear();

                  if (mounted) {
                    Navigator.pop(context);
                    _showSnackBar('Tâche ajoutée avec succès!', Colors.green);
                  }
                }
              },
              child: const Text('Ajouter', style: TextStyle(color: Colors.white)),
            ),
          ],
        );
      },
```

apcpedagogie.com

```
  );
}

void _showEditTodoDialog(BuildContext context, Todo todo, int? key) {
  final editController = TextEditingController(text: todo.content);

  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: const Text('Modifier la tâche'),
        content: TextField(
          controller: editController,
          decoration: const InputDecoration(
            hintText: "Modifiez votre tâche...",
            border: OutlineInputBorder(),
          ),
          maxLines: 3,
        ),
        actions: [
          TextButton(
            onPressed: () => Navigator.pop(context),
            child: const Text('Annuler'),
          ),
          ElevatedButton(
            style: ElevatedButton.styleFrom(backgroundColor: Colors.blue),
            onPressed: () async {
              final newText = editController.text.trim();
              if (newText.isNotEmpty && key != null) {
                final updated = Todo(
                  content: newText,
                  createdAt: todo.createdAt,
                  isDone: todo.isDone,
                );
                await _todosBox.put(key, updated);

                if (mounted) {
                  Navigator.pop(context);
                  _showSnackBar('Tâche modifiée avec succès!', Colors.blue);
                }
              }
            },
            child: const Text('Modifier', style: TextStyle(color: Colors.white)),
```

apcpedagogie.com

```
            ),
          ],
        );
      },
    );
  }

  void _showTodoDetails(BuildContext context, Todo todo) {
    showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          title: const Text('Détails de la tâche'),
          content: Column(
            mainAxisSize: MainAxisSize.min,
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Text('Tâche: ${todo.content}',
                style: const TextStyle(fontWeight: FontWeight.bold)),
              const SizedBox(height: 12),
              Text('Statut: ${todo.isDone ? "□ Terminé" : "□ En cours"}'),
              const SizedBox(height: 8),
              Text('Créé le: ${DateFormat('dd/MM/yyyy à
HH:mm').format(todo.createdAt)}'),
            ],
          ),
          actions: [
            TextButton(
              onPressed: () => Navigator.pop(context),
              child: const Text('Fermer'),
            ),
          ],
        );
      },
    );
  }

  Future _clearAllTodos() async {
    showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          title: const Text('Confirmation'),
```

apcpedagogie.com

```
          content: const Text('Êtes-vous sûr de vouloir supprimer toutes les tâches
?'),
          actions: [
            TextButton(
              onPressed: () => Navigator.pop(context),
              child: const Text('Annuler'),
            ),
            ElevatedButton(
              style: ElevatedButton.styleFrom(backgroundColor: Colors.red),
              onPressed: () async {
                await _todosBox.clear();
                if (mounted) {
                  Navigator.pop(context);
                  _showSnackBar('Toutes les tâches ont été supprimées', Colors.red);
                }
              },
              child: const Text('Supprimer tout', style: TextStyle(color:
Colors.white)),
            ),
          ],
        );
      },
    );
  }

  void _showSnackBar(String message, Color color) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text(message),
        backgroundColor: color,
        duration: const Duration(seconds: 2),
      ),
    );
  }

  int? _getKeyForTodo(Todo todo, Box box) {
    try {
      final map = box.toMap();
      for (var entry in map.entries) {
        if (entry.value.content == todo.content &&
            entry.value.createdAt == todo.createdAt) {
          return entry.key as int;
        }
```

apcpedagogie.com

```
      }
    } catch (e) {
      print('Erreur lors de la recherche de la clé: $e');
    }
    return null;
  }

  @override
  void dispose() {
    _textEditingController.dispose();
    super.dispose();
  }
}
```

11. **Étape 7 : Fonctionnalités avancées**

    - **Gestion des clés Hive :** Les clés sont auto-générées avec `add()`
    - **Persistance des données :** Les données survivent aux redémarrages de l'app
    - **Interface réactive :** Mise à jour automatique avec `ValueListenableBuilder`
    - **Opérations CRUD complètes :** Create, Read, Update, Delete

12. **Avantages de cette implémentation**

    - **Type-safe**: Sécurité des types avec les génériques
    - **Performance**: Hive est plus rapide que SharedPreferences
    - **Maintenable**: Code structuré et facile à comprendre
    - **Extensible**: Facile à modifier et enrichir
    - **UI réactive**: Mise à jour automatique de l'interface

4. ? Bonnes pratiques

- Toujours ouvrir les `boxes` avant utilisation.
- Utiliser `ValueListenableBuilder` pour une UI réactive.
- Éviter de stocker des objets trop complexes → préférer SQLite/Drift.
- Bien choisir les `typeId` uniques pour les adapters.

5. ? Hive vs SQLite vs Drift

Critère	Hive (NoSQL)	SQLite (sqflite)	Drift (ORM)

apcpedagogie.com

Performance	? Très rapide	Bonne	Bonne mais typée
Structure	Clé-valeur	Relationnelle	Relationnelle typée
Simplicité	Très simple	Moyenne (SQL à écrire)	Plus complexe (build_runner)
Idéal pour	Cache, favoris, settings	Données structurées	Projets complexes

6. ? Conclusion

- Utiliser **Hive** pour une persistance **rapide et simple** (favoris, cache, settings).
- Préférer **SQLite** ou **Drift** pour des données relationnelles complexes.

Cours Flutter

Source : https://apcpedagogie.com/gestion-de-donnees-locales-avec-hive-dans-flutter · téléchargé le 07/11/2026 · © APC Pédagogie · apcpedagogie.com

apcpedagogie.com