Flutter cheat Sheet

Shared Prefrences

Shared preferences in mobile development allow developers to store small amounts of data in key-value pairs on a user's device. This data is persistent and can be accessed even after the app has been closed. In Flutter, shared preferences are easily managed using the shared_preferences package.

- **Getting started with shared_preferences :-**

- The shared_preferences package can be easily added to your Flutter project by adding the following line to your pubspec.yaml file: yamlCopy code

- dependencies:
  shared_preferences: ^0.5.12+4

- Once you have added the dependency, you can import it into your Dart file and initialize it as follows:

- import 'package:shared_preferences/shared_preferences.dart';

- SharedPreferences prefs;

// Initialize shared preferences
- prefs = await SharedPreferences.getInstance();

Flutter cheat Sheet

Shared Prefrences

- **Saving data to shared preferences :-**

• Saving data to shared preferences is straightforward and can be done as follows:

```
// Saving a string value
• prefs.setString('key', 'value');

// Saving an integer value
• prefs.setInt('key', 42);

// Saving a double value
• prefs.setDouble('key', 3.14);

// Saving a boolean value
• prefs.setBool('key', true);
```

- **Retrieving data from shared preferences :-**

```
// Retrieving a string value
• String stringValue = prefs.getString('key');

// Retrieving an integer value
• int intValue = prefs.getInt('key');

// Retrieving a double value
• double doubleValue = prefs.getDouble('key');

// Retrieving a boolean value
• bool boolValue = prefs.getBool('key');
```

Flutter cheat Sheet

Shared Prefrences

- **Demonstrative App :-**

```
class _MyAppState extends State<MyApp> {
 String _username;
 bool _rememberMe;

 @override
 void initState() { super.initState();
 _loadPreferences(); }

 void _loadPreferences() async {
 final prefs = await SharedPreferences.getInstance();
 setState(() {
 _username = prefs.getString('username') ?? '';
 _rememberMe = prefs.getBool('rememberMe') ?? false; }); }

 @override
 Widget build(BuildContext context) { return MaterialApp(
 home: Scaffold( appBar: AppBar(
 title: Text('Shared Preferences Demo'), ),
 body: Padding(
 padding: const EdgeInsets.all(16.0),
 child: Column( children: <Widget>[
 TextField( decoration: InputDecoration(
 labelText: 'Username', hintText: 'Enter your username', ),
 onChanged: (value) { setState(() { _username = value; }); }, ),
 CheckboxListTile(
 title: Text('Remember me'), value: _rememberMe,
 onChanged: (value) { setState(() { _rememberMe = value; }); }, ),
 RaisedButton( child: Text('Save'), onPressed: () async {
 final prefs = await SharedPreferences.getInstance();
 prefs.setString('username', _username);
 prefs.setBool('rememberMe', _rememberMe); }, ), ], ), ), ), ); }  }
```

Flutter cheat Sheet

DataBase

Flutter provides many packages to work with the database. The most used and popular packages are:

- sqflite database: It allows to access and manipulate SQLite database.
- Firebase database: It will enable you to access and manipulate the cloud database.

• SQLite Database :-

SQLite is a popular database software library that provides a relational database management system for local/client storage. It is a light-weight and time-tested database engine and contains features like self-contained, server-less, zero-configuration, transactional SQL database engine.

Flutter SDK does not support SQLite directly. Instead, it provides a plugin sqflite, which performs all operations on the database as similar to the SQLite library. The sqflite provides most of the core functionalities related to the database are as follows:

- It creates or opens the SQLite database.
- It can execute SQL statements easily.
- It also provides an advanced query method to get information from the SQLite database.

- The sqflite package provides classes and functions to interact with the SQLite database.
- The path_provider package provides functions to define the location of your database on the local system, such as TemporaryDirectory and ApplicationDocumentsDirectory.

Flutter cheat Sheet

DataBase

Open the database. Here, we need to open the connection to the database. It requires two steps:

1. Set the path to the database by using the getDtabasePath() method and combined it with the path package.
2. Use openDatabase() function to open the database.

```
// It allows to open the database and store the reference.
final Future<Database> database = openDatabase(
 join(await getDatabasesPath(), 'book_database.db'),
);
```

Create the table. In this step, we have to create a table that stores information about the books. Here, we are going to create a table named books, which contains the id, title, and price of the books. They are represented as three columns in the book table.

```
final Future<Database> database = openDatabase(
 join(await getDatabasesPath(), 'book_database.db'),
 // When you create a database, it also needs to create a table to store books.
 onCreate: (db, version) {
  // Run the CREATE TABLE statement.
  return db.execute(
    "CREATE TABLE books(id INTEGER PRIMARY KEY, title TEXT, price INTEGER)",  );  },
  // Set the version to perform database upgrades and downgrades.  version: 1,  );
```

DataBase

Insert a Book into the database. Here, you have to store information on the table about the various books. Inserting a book into the table involves two steps:

- Convert the Book into a Map
- Uses insert() method

```
// Update the Book class.
class Book{
  final int id,
  final String title;
  final int price;

  Book({this.id, this.title, this.price});

   // It converts a Book into a Map. The keys correspond to the
names of the columns in the database.
  Map<String, dynamic> toMap() {   return {  'id': id,  'title': title,
     'price': price,  };  }  }

Future<void> insertBook(Book book) async {
  final Database db = await database;    await db.insert(
    'books',      book.toMap(),
    conflictAlgorithm: ConflictAlgorithm.replace,    );  }

// Create a Book and add it to the books table.
final b1 = Book(
  id: 0,    title: 'Let Us C',    price: 350,  );

await insertBook(b1);
```

Flutter cheat Sheet

DataBase

Retrieve the list of books. Now, we have stored the book into the database, and you can use a query to retrieve a particular book or list of all books. It involves two steps:

- Run a query that returns List<Map>.
- Convert the List<Map> into the List<Book>.

```
// This method retrieves all the books from the books table.
Future<List<Book>> books() async {
  final Database db = await database;

  // Use query for all Books.
  final List<Map<String, dynamic>> maps = await db.query('maps');

  return List.generate(maps.length, (i) {
    return Book(
      id: maps[i]['id'],
      title: maps[i]['title'],
      price: maps[i]['price'],
    );
  });
}

// It prints all the books.
print(await books());
```

Flutter cheat Sheet

DataBase

Update a Book in the database. You can use an update() method to update the book that you want. It involves two steps:

- Convert Book into the Map.
- Then, use where clause to update the book.

```
Future<void> updateBook(Book book) async {
  final db = await database;

  // Update the given Book.
  await db.update(
    'books',
    book.toMap(),
    // It ensure the matching id of a Book.
    where: "id = ?",
    whereArgs: [book.id],
  );
}

// Update b1 price.
await updateBook(Book(
  id: 0,
  title: 'Let Us C',
  price: 420,
));

// Print the updated results.
print(await books());
```

Flutter cheat Sheet

DataBase

Delete a Book from the database. You can use the delete() method to delete the database. For this, you need to make a function that takes an id and delete the database of the matching id.

```
Future<void> deleteBook(int id) async {
 final db = await database;

 // This function removes books from the database.
 await db.delete(
  'books',
  where: "id = ?",
  whereArgs: [id],
 );
}
```