Data Persistence and Different Storage Options

What is Data Persistence?

Data Persistence refers to the ability of data to outlast the process that created it. It guarantees that data remains accessible even after the device is shut down or powered off.

Choosing a Data Persistent Solution:

Consider you app specific needs such as the complexity of the data, targeted platforms and the need for the offline accessibility, or cross-platform synchronization while choosing a data persistent solution.

Introduction to Local Storage and its types:

Local Storage is used to handle the *client-side* operations.

Types of Local Storage:

1- **SharedPreferences(for Android) / NSUserDefaults(for IOS):**
   It is ideal for storing simple data in *key value pairs*.
   They serve as *the browser base local storage* for data
   -----such as
   ----------*user settings*
   ----------*app configurations*.

2- **SQLite:**
   Powerful database help manage *complex data structures*.
   Works seamlessly with IOS and Android.
   Provides *Robust framework* for handling data through *SQL queries*.

3- **Hive:**
   Lightweight, no SQL database.
   It is written in Dart.
   No platform specific requirements.
   Fast read and write operation support.

Compatible with Mobile and desktop apps.

4- **Files:**
Allows you to read and write directly from the files.
Ideal for downloading, saving and retrieving media.
Offer maximum control over file handling.
Demands careful attention to management of :
--filepaths.
--storage permissions.

Advanced Storage Options:

1- **Cloud Firestore:**
- *Versatile* and *Scalable* solutions.
- Seamless mobile, web, desktop development
- Real-time Data *Synchronization*.

2- **PostgreSQL:**
- Ideal for *Sophisticated* applications.
- Offer *strong transactional* support.
- Implement using *Client-Side* and *cloud-based* solutions.
- Provide advanced *relational database* capabilities.

3- **Realm:**
- Strong *real-time* capabilities.
- Offline-first architecture.
- Smooth *Synchronization* with its *cloud service*.
- *Robust* support for handling *complex data models*.

Platform-Specific considerations:

- **For IOS:**
Use Apple's Specific guidlelines.
Use UserDefaults to handle simple data.
Use Core Data to handle complex data.

- **For Android:**

For lightweight storage use *SharedPreferences*.
Use SQLite for more *substantial* data sets.
Use *Room* for additional functionalities.

- **For Web Apps:**
  Flutter Web Apps use *IndexDB:*
  An Ideal solution for client-side storage
  Greater flexibility
  Since it is a new technology so may have *inconsistent support* across different *browsers*.

- **For Desktop:**
  Use SQLite
  Direct Files Storage to offer similar *mobile experiences* .