Chapter 1

Local Storage in Flutter

In Flutter, we have many options for storing data on our device. In this chapter, I will present some of my favorite options for storing simple key-value pairs or objects.

1.1 Shared Preferences

When it comes to storing simple key-value pair data, this is the simplest solution for most use cases. It is created by the Flutter team and supports all platforms. If you need to persist user data that can be identified by a primary keyword, Shared Preferences is a great choice.

1.1.1 Common Use Cases

Shared Preferences is commonly used for tracking user settings and history:

- **Light/Dark Mode selection**
- **Preferred language selection**
- **Tracking if the user has completed onboarding**
- **Sound settings/history**

1.1.2 Supported Data Types

Shared Preferences supports the following data types:

```
1     String, int, bool, double, List<String>
```

you can use other type by converting into string/jsonEncoding and decoding, but there are other good options for that later in this video

1.1.3 In order to use it

todo:

1.2 Flutter Secure Storage

But if you thinking about secure on local storage. I prefer using **flutter_secure_storage** simple, nice, elegant. Just like **shared_preference**, it is also a key-value pair storage solution.

1. Add the package in your **pubspec.yaml**
2. Create a storage instance like

```
    // You can cache encrypted options
    AndroidOptions _getAndroidOptions()
        => const AndroidOptions(
        encryptedSharedPreferences: true,
    );
    final storage = FlutterSecureStorage(
        aOptions: _getAndroidOptions());
```

3. to get value

```
    String value = await storage.read(key: key);
```

4. to set value

```
      await storage.write(key: key, value: value);
```

5. to delete

```
        await storage.write(key: key, value: value);
```

6. check official document