

Flutter & Dart Complete Guide

Prepared by Eng. Yousef Salah Kassem

DAFL 3
Local Storage

Lecture 3



Chapter
03

Table Of Content

- What is Local Storage?
- Why Do We Need Local Storage?
- Types of Local Storage in Flutter



Introduction

What is Local Storage?

- Data stored **locally on the device**
- Not dependent on an internet connection

Why Do We Need Local Storage?

- Offline access
- Persist user sessions
- Store user settings, tokens, cache
- Improve performance (avoid frequent API calls)

Types of Local Storage in Flutter

- Key-value: `SharedPreferences`, `SecureStorage`
- NoSQL: `Hive`
- SQL: `SQFlite`

Shared Preferences

What is Shared Preferences?

- Simple key-value storage for **primitive data types**
- Good for storing small data: theme, login status

Supported Data Types

- `int`
- `double`
- `bool`
- `String`
- `List<String>`

Example Use Case

- **Store theme preference (dark/light)**
- **Store Authentication Status**

```
final prefs = await SharedPreferences.getInstance();  
await prefs.setBool('isDarkMode', true);
```

```
final isDark = prefs.getBool('isDarkMode') ?? false;
```

Limitations of Shared Preferences

- No encryption
- Not suitable for complex/nested data
- Not good for large datasets

Secure Storage

What is Secure Storage?

- Uses `flutter_secure_storage`
- Data stored in **Keychain (iOS)** or **Encrypted SharedPreferences (Android)**

Why Use Secure Storage?

- Store sensitive data like **tokens, passwords, private keys**

```
final storage = FlutterSecureStorage();  
await storage.write(key: 'token', value: 'abc123');  
final token = await storage.read(key: 'token');
```

When to Use Secure Storage

- Login tokens
- API keys
- Any **sensitive user information**

Comparison: Shared vs Secure Storage

Feature	SharedPrefs	SecureStorage
Encrypted	✗	✓
Complex Data	✗	✗
Usage	Settings	Tokens

Hive

What is Hive?

- Lightweight & fast NoSQL DB
- Stores objects using **boxes**

Why Use Hive?

- Fast
- Works with **custom objects**
- Suitable for **offline-first apps**

```
@HiveType(typeld: 0)
```

```
class User {
```

```
  @HiveField(0)
```

```
  String name;
```

```
  User(this.name);
```

```
}
```

```
final box = await Hive.openBox('userBox');
```

```
box.put('user1', User('Ahmed'));
```

Hive Features

- No native SQL
- Good for local cache, lists, user models
- Type adapters needed for custom classes

Hive vs SharedPrefs

Feature	Hive	SharedPrefs
Complex Models	✓	✗
Encrypted?	✓ (optional)	✗
Performance	⚡ Fast	🚀 Fast

SQLite (SQL Database)

What is SQFlite?

- SQLite plugin for Flutter
- Full-featured relational DB

Why Use SQLite?

- Structured data
- Queries, relationships, indexing
- Great for **multi-table storage**

When to Use?

- Contact list
- Inventory
- Complex app with relational data

```
final db = await openDatabase('my_db.db');  
await db.execute('CREATE TABLE users (id  
INTEGER PRIMARY KEY, name TEXT)');  
await db.insert('users', {'name': 'Ali'});
```

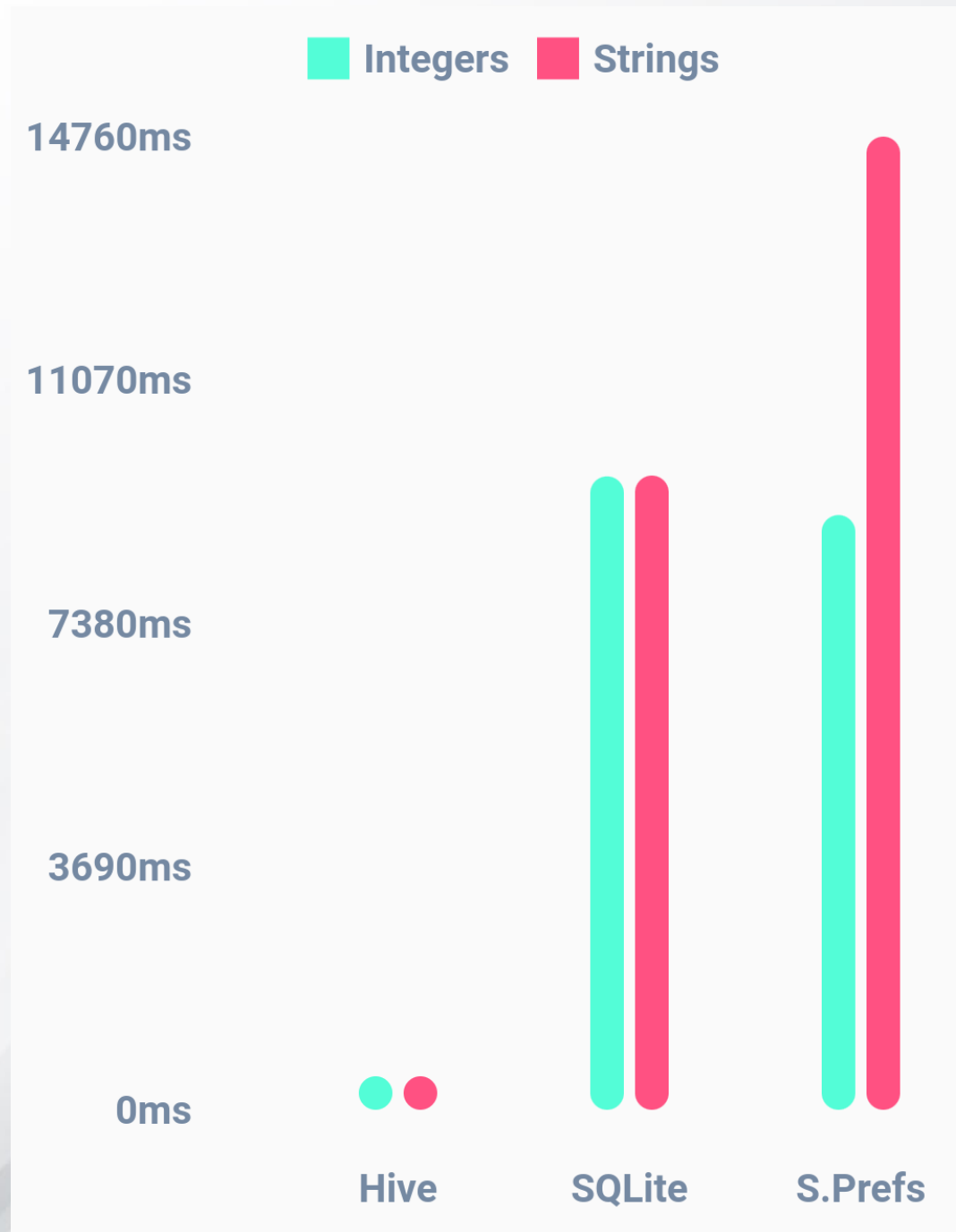
SQLite Pros and Cons

-  Powerful for structured data
-  Verbose, needs table & schema management

Choosing the Right Storage

Summary Table: Storage Types

Use Case	Storage Type
Theme/Settings	SharedPreferences
Tokens/Secrets	SecureStorage
Cached Lists	Hive
Complex Schema	SQFLite



Storage Type Decision Tree

- Flowchart for selection:
 - Small key-value → SharedPreferences
 - Sensitive? → SecureStorage
 - Complex/nested? → Hive
 - Structured relational? → SQLite

Auth Flow

- Use `SecureStorage` for token
- `SharedPrefs` for "remember me" toggle
- `Hive` for offline user profile

Offline To-Do App

- Use **Hive** to save tasks
- Use **SharedPrefs** for sorting preference

eCommerce App

- **SecureStorage**: token
- **Hive**: cart
- **SQFlite**: product catalog

Conclusion & Best Practices

- Pick storage based on **data sensitivity, structure & size**
- Use encryption when needed
- Cache wisely to improve UX

Security Best Practices

Security Best Practices for Local Storage

- Never store **tokens or passwords** in `SharedPreferences`.
- Use `flutter_secure_storage` for **authentication credentials**.
- Avoid storing **sensitive data in plain Hive/SQFLite**.
- **Encrypt Hive boxes** if used for sensitive data.
- Always provide **logout flows** that clear local storage.

What is Next ?



Thank You