

# Persistência local de dados

**Danilo Perez**

Full Stack Developer

 **perez-danilo**

 **Fala devs**

# Objetivo Geral

Objetivo deste é conhecer formas gravar dados na memória do dispositivo. Será abordado formas de gravar de forma chave e valor fazendo uso do `shared_preferences`.

Passando para utilização do Hive que proporciona além de gravar chave valor, poderemos gravar objetos complexos.

Finalizando passaremos pelo SQLite que é a gravação de dados fazendo uso de linguagem SQL.

# Percurso

- Etapa 1** Armazenando de dados com shared\_preferences
- Etapa 2** Armazenando dados com banco de dados Hive
- Etapa 3** Armazenando dados com banco de dados SQLite

## Etapa 1

# Flutter

// Armazenando de dados com shared\_preferences

# Shared Preferences

O recurso SharedPreferences é usado para armazenar dados no formato chave-valor.

Storage location by platform

Platform	Location
Android	SharedPreferences
iOS	NSUserDefaults
Linux	In the XDG_DATA_HOME directory
macOS	NSUserDefaults
Web	LocalStorage
Windows	In the roaming AppData directory

	Android	iOS	Linux	macOS	Web	Windows
<b>Support</b>	SDK 16+	9.0+	Any	10.11+	Any	Any

# Shared Preferences


Write data

```
// Obtain shared preferences.
final prefs = await SharedPreferences.getInstance();

// Save an integer value to 'counter' key.
await prefs.setInt('counter', 10);
// Save an boolean value to 'repeat' key.
await prefs.setBool('repeat', true);
// Save an double value to 'decimal' key.
await prefs.setDouble('decimal', 1.5);
// Save an String value to 'action' key.
await prefs.setString('action', 'Start');
// Save an list of strings to 'items' key.
await prefs.setStringList('items', <String>['Earth', 'Moon', 'Sun']);
```


# Shared Preferences

## Read data



```
// Try reading data from the 'counter' key. If it doesn't exist, returns null.  
final int? counter = prefs.getInt('counter');  
// Try reading data from the 'repeat' key. If it doesn't exist, returns null.  
final bool? repeat = prefs.getBool('repeat');  
// Try reading data from the 'decimal' key. If it doesn't exist, returns null.  
final double? decimal = prefs.getDouble('decimal');  
// Try reading data from the 'action' key. If it doesn't exist, returns null.  
final String? action = prefs.getString('action');  
// Try reading data from the 'items' key. If it doesn't exist, returns null.  
final List<String>? items = prefs.getStringList('items');
```

## Remove an entry



```
// Remove data for the 'counter' key.  
final success = await prefs.remove('counter');
```

# Percurso

**Etapa 1** ~~Armazenando de dados com shared\_preferences~~

**Etapa 2** Armazenando dados com banco de dados Hive

**Etapa 3** Armazenando dados com banco de dados SQLite



## Etapa 2

# Flutter

// Armazenando dados com banco de dados Hive

# Hive

O Hive é um banco de dados NoSQL rápido e leve, para aplicativos flutter e dart.

O Hive é realmente útil se você precisar de um banco de dados de chave-valor realmente simples de usar.

É um banco de dados offline (armazenar dados em dispositivos locais).

# Hive

```
var box = Hive.box('myBox');  
box.put('name', 'David');  
var name = box.get('name');  
print('Name: $name');
```

# Hive

```
@HiveType(typeId: 0)
class Person extends HiveObject {

    @HiveField(0)
    String name;

    @HiveField(1)
    int age;
}
```

```
var box = await Hive.openBox('myBox');

var person = Person()
  ..name = 'Dave'
  ..age = 22;
box.add(person);

print(box.getAt(0)); // Dave - 22

person.age = 30;
person.save();

print(box.getAt(0)) // Dave - 30
```

# Percurso

**Etapa 1** ~~Armazenando de dados com shared\_preferences~~

**Etapa 2** ~~Armazenando dados com banco de dados Hive~~

**Etapa 3** Armazenando dados com banco de dados SQLite

## Etapa 3

# Flutter

// Armazenando dados com banco de dados SQLite

# SQLite

O SQLite com certeza é um dos bancos locais mais populares do mercado

Além disso o uso do SQLite no Flutter é relativamente simples e nativamente ele já nos provê diversos recursos como proteção contra SQL Injection e até mesmo a dispensa na escrita de código SQL.

# SQLite

```
// Get a location using getDatabasesPath
var databasesPath = await getDatabasesPath();
String path = join(databasesPath, 'demo.db');

// Delete the database
await deleteDatabase(path);

// open the database
Database database = await openDatabase(path, version: 1,
  onCreate: (Database db, int version) async {
    // When creating the db, create the table
    await db.execute(
      'CREATE TABLE Test (id INTEGER PRIMARY KEY, name TEXT, value INTEGER, num REAL)');
  });

// Insert some records in a transaction
await database.transaction((txn) async {
  int id1 = await txn.rawInsert(
    'INSERT INTO Test(name, value, num) VALUES("some name", 1234, 456.789)');
  print('inserted1: $id1');
  int id2 = await txn.rawInsert(
    'INSERT INTO Test(name, value, num) VALUES(?, ?, ?)',
    ['another name', 12345678, 3.1416]);
  print('inserted2: $id2');
});
```



# Links Úteis

- [digitalinnovationone/dio-flutter \(github.com\)](#)
- [hive | Dart Package \(pub.dev\)](#)
- [shared\\_preferences | Flutter Package \(pub.dev\)](#)
- [sqlite | Dart Package \(pub.dev\)](#)

# Para saber mais

Artigos e cursos da DIO

“Fala Devs” youtube

# Dúvidas?

- > Fórum/Artigos
- > Comunidade Online (Discord)



# Desafio - IMC

- Criar classe IMC (Peso / Altura)
- Altura ler em tela de Configurações
- Ler dados no app
- Calcular IMC
- Gravar dados no Hive ou SQLite
- Exibir em uma lista

IMC	Classificação
< 16	Magreza grave
16 a < 17	Magreza moderada
17 a < 18,5	Magreza leve
18,5 a < 25	Saudável
25 a < 30	Sobrepeso
30 a < 35	Obesidade Grau I
35 a < 40	Obesidade Grau II (severa)
≥ 40	Obesidade Grau III (mórbida)

$$\text{IMC} = \frac{\text{Peso (em quilos)}}{\text{Altura}^2 \text{ (em metros)}}$$