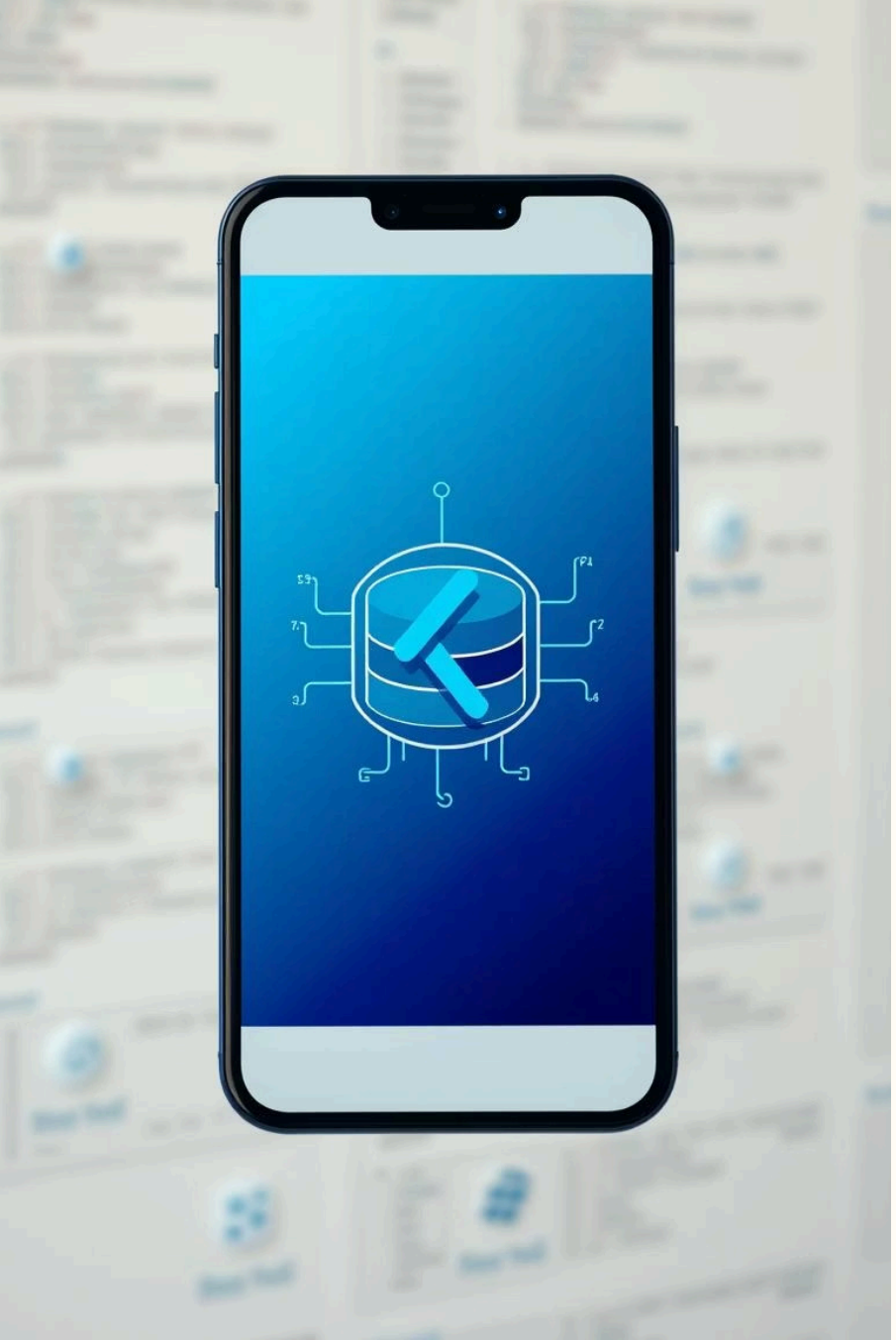

Hive in Flutter

Hive

Local Database.

- based on dart

fast

Binary storage

easy

No SQL

offline

setup

```
dependencies:
  flutter:
    sdk: flutter
  hive: ^2.2.3
  hive_flutter: ^1.1.0

dev_dependencies:
  hive_generator: ^2.0.1
  build_runner: ^2.4.6
```

Hive Initialization :

```
import 'package:hive_flutter/hive_flutter.dart';

void main() async {
 WidgetsFlutterBinding.ensureInitialized();
 await Hive.initFlutter();
 runApp(MyApp());
}
```

how to use Hive ?

Hive deals with data using **“boxes”**, which are similar to tables in relational databases or collections in NoSQL. You can open boxes to store and retrieve data.

1

create object

`await Hive.openBox('myBox');`→in main

`final Box _box = Hive.box('myBox');`→hive file

2

add data

```
box.put('name', 'karim');
box.put('age', 30);
```

3

Access data

```
var name = box.get('name');
var age = box.get('age');
```

4

delete data

```
box.delete('name');
```

normal class

```
class Person {

  final String name;
  final int age;

  Person({required this.name, required this.age});
}
```

problem with hive?
→ cant understand this normal class .

solve?
we will convert it into binary (using Adapter)

Storing Custom Objects (Custom Objects)

To store custom objects, you need to register an **Adapter** for Hive.
This adapter converts your object to and from binary so that Hive can store it.

Example of a `Person` object and its adapter:

```
import 'package:hive/hive.dart';

// This tells Hive to generate the adapter file later
3) part 'person.g.dart';



1) @HiveType(typeId: 0)
class Person {
2)  @HiveField(0) // Field index for "name"
  final String name;

  @HiveField(1) // Field index for "age"
  final int age;

  Person({required this.name, required this.age});
}

flutter pub run build_runner build
```

Remember to register the adapter in **main.dart** after initializing Hive and before opening the box that will contain these objects.

```
Hive.registerAdapter(PersonAdapter());
```