State Management in Flutter

Mobile Dev Guide · Module 3 of 8 · CHERIEDU Dev Series

CHERIEDU.COM
Dev Training Series

1. Why State Management Matters

As your app grows, sharing data between screens becomes complex. State management solutions make it clean, testable, and scalable. Choosing the right one early saves major refactoring later.

2. State Management Options

Solution	Complexity	Best For
setState	Simple	Single screen, small apps
Provider	Medium	Most apps — official recommended
Riverpod	Medium-High	Type-safe, testable apps
Bloc / Cubit	High	Large enterprise apps
GetX	Low (opinionated)	Rapid development, smaller teams

3. Provider — Recommended Pattern

```
// 1. Create a Provider class
class StudentProvider extends ChangeNotifier {
  List<Student> students = [];
  void loadStudents() async {
    students = await api.getStudents();
    notifyListeners();  // tells widgets to rebuild
  }
}

// 2. Wrap app with ChangeNotifierProvider
ChangeNotifierProvider(create: (_) => StudentProvider(), child: MyApp())

// 3. Consume in a widget
final provider = context.watch<StudentProvider>();
Text("Students: ${provider.students.length}")
```

4. Passing Data Between Screens

```
// Navigate to detail screen with data
Navigator.push(
  context,
  MaterialPageRoute(
    builder: (_) => StudentDetailScreen(student: selectedStudent),
  ),
);
```

```
// Receive in detail screen
```

```
class StudentDetailScreen extends StatelessWidget {
```

```
final Student student;
```

```
StudentDetailScreen({required this.student});
```

```
}
```

5. Persistence — Saving Data Locally

- shared_preferences: Key-value storage for settings and tokens.
- sqflite: SQLite database for structured local data.
- Hive: Fast NoSQL local storage — great for offline-first apps.
- secure_storage: Encrypted storage for passwords and JWT tokens.

PROJECT

Add a Provider to your student list app. Implement: load students from API, filter by grade, mark attendance, and persist the attendance data locally using shared_preferences.