UI → Provider → Controller (Notifier) → Repository → Data Source

✅ The **Provider creates and exposes the Controller** (also called a *Notifier*).
✅ The **UI calls the Provider**,
and that Provider gives you access to the **Controller** and its **state**.
So it's like this:

```
UI → Provider → Controller (Notifier) → Repository → Data Source
```

🧩 Detailed Explanation

🧱 Step 1: You have a Controller (the logic class)

```
class UserController extends Notifier<UserState> {
  final UserRepository repo;

  UserController(this.repo);

  @override
  UserState build() => const UserState.initial();

  Future<void> fetchUser() async {
    state = const UserState.loading();
    final user = await repo.getUser();
    state = UserState.loaded(user);
  }
}
```

🧠 `UserController` = your **logic brain** —
it decides *when to load, when to show data, when to show error.*

🧩 Step 2: You create a Provider for it

```
final userControllerProvider =
    NotifierProvider<UserController, UserState>(
  () => UserController(ref.read(userRepoProvider)),
);
```

✅ This Provider:

- **Creates** the `UserController`
- **Exposes** its state (`UserState`)
- **Lets the UI watch or read** that state

🧠 So:

> *Provider = middleman between UI and Controller*

🧩 Step 3: The UI talks to the Provider (not directly to Controller)

```
class UserScreen extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final userState = ref.watch(userControllerProvider); // 👈 state
    final controller = ref.read(userControllerProvider.notifier); // 👈 controller

    return Scaffold(
      body: switch (userState) {
        UserLoading() => const CircularProgressIndicator(),
        UserLoaded(:final user) => Text(user.name),
        UserError(:final message) => Text(message),
        _ => const SizedBox.shrink(),
      },
      floatingActionButton: FloatingActionButton(
        onPressed: controller.fetchUser, // 👈 UI calls controller via provider
        child: const Icon(Icons.refresh),
      ),
    );
  }
}
```

🧠 So the Relationships Are:

Who	Talks To	Using
UI	Provider	`ref.watch` or `ref.read`
Provider	Controller	Creates it automatically
Controller	Repository	Via injected dependency
Repository	Data Source	Through API or DB

🔁 Data & Command Flow

```
🟦 UI
   ↓ triggers (ref.read(...).fetchUser)
🟩 Controller / Notifier
   ↓ uses
🟨 Repository
   ↓ calls
🟧 API / DB
   ↑ returns data
🟩 Controller updates state
```

```
    ↑
🟦 UI rebuilds (ref.watch)
```

🔐 In Short

Concept	Responsibility	Example
Controller (Notifier)	Business & UI logic	`fetchUser()`, `updateTodo()`
Provider	Manages and exposes the controller's state	`userControllerProvider`
UI (Widget)	Displays and triggers actions	`ref.watch()` and `ref.read()`
Repository	Handles data fetching	`api.getUser()`

✅ **Rule summary in one line:**

> *UI never directly creates or calls Controller — it always uses the Provider that manages it.*