Flutter Riverpod Project Structure

Riverpod state management with compile-time safety. Providers, NotifierProviders, and code generation.

#flutter #dart #mobile #riverpod #state-management

PNG PDF Copy Prompt

Project Directory

```
myapp/
  pubspec.yaml
  pubspec.lock
  analysis_options.yaml
  build.yaml  Code gen config
  .gitignore
  lib/
    main.dart  ProviderScope w…
    core/
      providers/  Global providers
        dio_provider.dart
        shared_prefs_provider.dart
      constants/
      theme/
      utils/
    features/
      auth/
        data/
          auth_repository.dart
          user_model.dart
        providers/  Feature provide…
          auth_provider.dart  @riverpod annot…
          auth_provider.g.dart  Generated code
        screens/
          login_screen.dart
        widgets/
      home/
        data/
        providers/
        screens/
        widgets/
      routing/
        app_router.dart
      shared/
        widgets/
        models/
    test/
      features/
        auth/
    android/
    ios/
    assets/
```

Why This Structure?

Riverpod is a complete rewrite of Provider with compile-time safety, no BuildContext dependency, and proper testing support. With riverpod_generator, you get type-safe providers with minimal boilerplate.

Key Directories

`lib/core/providers/` - Global providers (Dio, SharedPrefs, etc.)

`lib/features/*/providers/` - Feature-scoped providers with @riverpod

`*.g.dart files` - Generated provider code (run build_runner)

Riverpod Generator

```
// lib/features/auth/providers/auth_provider.dart
@riverpod
class AuthNotifier extends _$AuthNotifier {
  @override
  AsyncValue build() => const AsyncData(null);

  Future login(String email, String password) async {
    state = const AsyncLoading();
    state = await AsyncValue.guard(() async {
      return ref.read(authRepositoryProvider).login(email,
    });
  }
}
```

Getting Started

1. `flutter pub add flutter_riverpod riverpod_annotation`
2. `flutter pub add -d riverpod_generator build_runner`
3. Wrap app with `ProviderScope`
4. Create providers with `@riverpod` annotation
5. `dart run build_runner watch`

When To Use This

- Want compile-time provider safety
- Need easy testing with provider overrides
- Prefer code generation over boilerplate
- Building medium to large apps
- Want to avoid Provider context issues

Trade-offs

`Code generation` - Must run build_runner for .g.dart files

`Learning curve` - Different mental model from Provider

`Verbose for simple cases` - Overkill for tiny apps