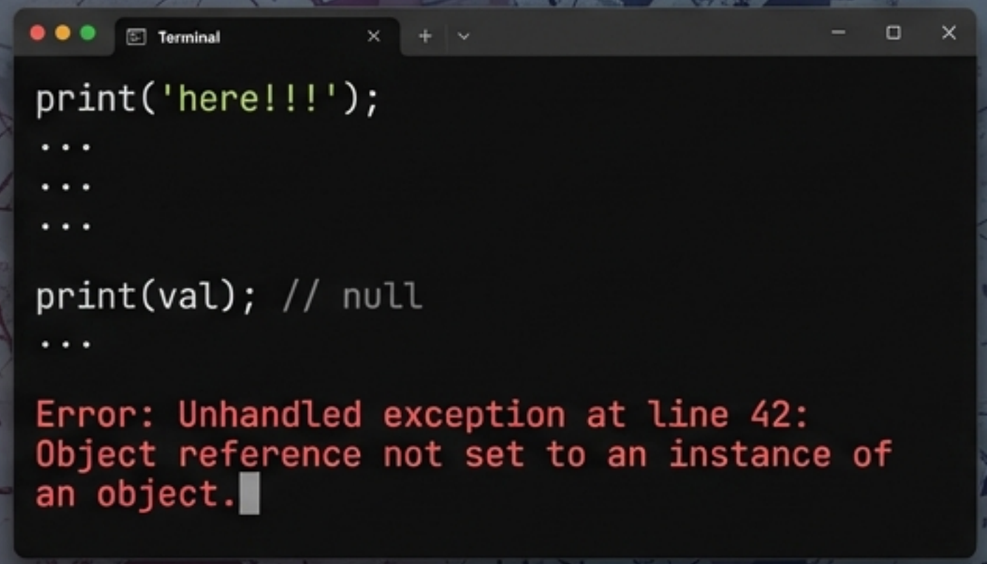
Flutter Uygulamalarını Test Etme

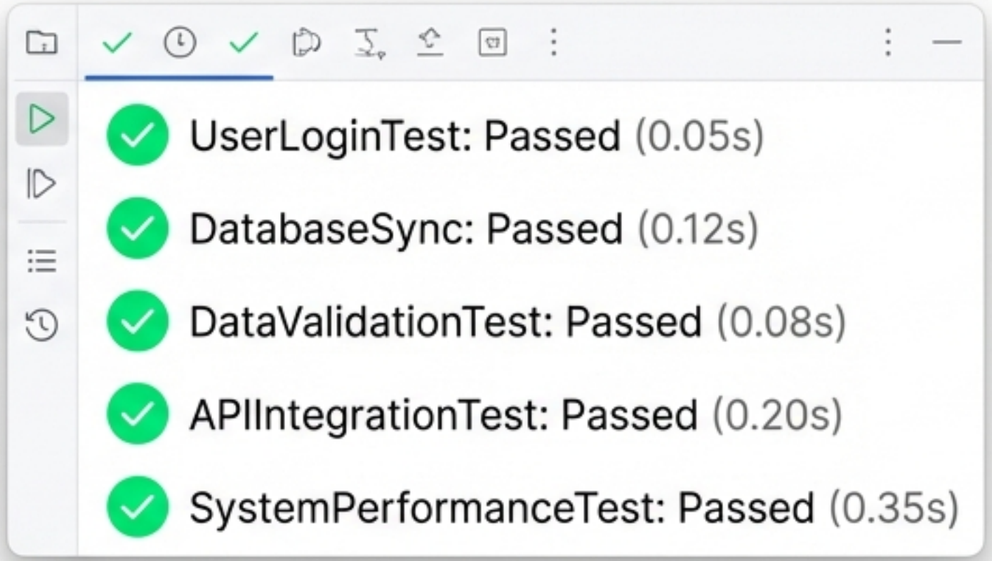
Kod Kalitesini ve Güvenilirliği Garanti Altına Alın

Birim (Unit) — Widget — Entegrasyon (Integration)

‘Saf’ Hata Ayıklama vs. Sağlam Test Stratejisi



Manuel Debug & 'Refactoring Korkusu'



Otomatize Güven & Sürdürülebilirlik

Test yazmak, hataların gelecekte tekrarlanmayacağının garantisidir.
Bir hatayı düzeltirken, iki yenisini yaratmayın.

Test Stratejisinin Üç Temel Taşı

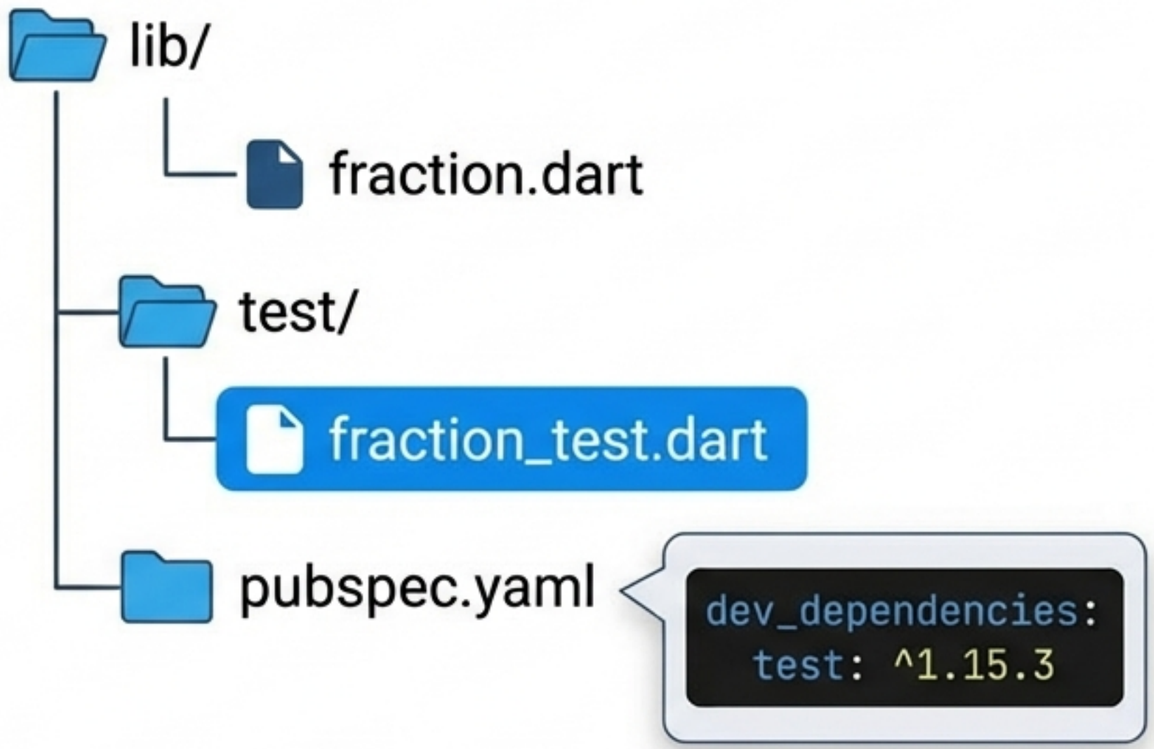
- **1. Birim Test:** Fonksiyon ve sınıfları test eder.
- **2. Widget Test:** UI bileşenlerinin render durumunu test eder.
- **3. Entegrasyon Test:** Tüm uygulamanın bir bütün olarak çalışmasını test eder.

Temel: Birim Testi (Unit Testing)

Flutter widget'larından bağımsız 'saf' Dart kodunu test eder.

UI içermez.

Hedef; sınıflar, hesaplamalar ve iş mantığıdır.



Konvansiyon: Test dosyaları her zaman **_test.dart** ile bitmelidir.

Birim Testinin Anatomisi

```dart
void main() {
  // Gruplama organizasyon sağlar
  group('Fraction Class', () {

    test('10 divided by 2 should be 5', () {
      // 1. Setup (Hazırlık)
      final fraction = Fraction(numerator: 10, denominator: 2);

      // 2. Expect (Doğrulama)
      // expect(actual, matcher)
      expect(fraction.toDouble, 5.0);
    });

  });
}
```

Test Grubu

Test Tanımı ve Callback

Sol: Gerçek Değer, Sağ: Beklenen Değer

İleri Seviye Senaryolar: Asenkron ve Hatalar

Futures (Async/Await)

```
test('Future test', () async {
  final val = await fetchValue();
  expect(val, 25);
});
```

Alternatif: `expect(future, completion(equals(25)))`

Streams (Akışlar)

```
expect(stream, emitsInOrder([
  "Loading",
  "Success",
  emitsDone
]));
```

Exceptions (Hatalar)

```
expect(
  () => calculator.divide(1, 0),
  throwsA(isA<ArgumentError>())
);
```

Dış Dünyayı İzole Etmek: Mocking

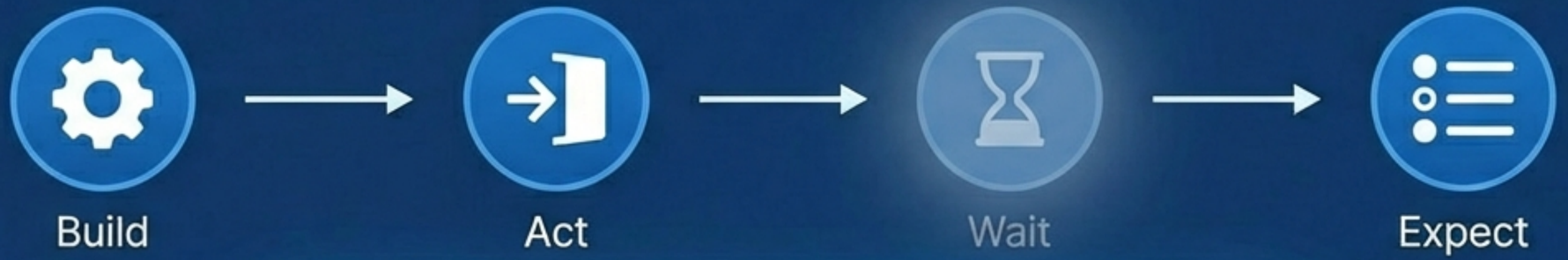
Testler sunuculara veya veritabanlarına bağımlı olmamalıdır. `mockito` paketi ile dış dünyayı taklit edin.


```dart
// 1. Mock Sınıfı Oluştur
class MockClient extends Mock implements http.Client {}

void main() {
  test('API returns success', () async {
    final mock = MockClient();
    // 2. Davranışı Belirle (Stubbing)
    when(mock.get(any)).thenReturn(http.Response('{"id": 1}', 200));
    // 3. İzole Test Et
    expect(await api.fetchData(mock), isA<Item>());
  });
}
```

State Management Testi: BLoC

`bloc_test` paketi ile okunabilir ve deklaratif testler.



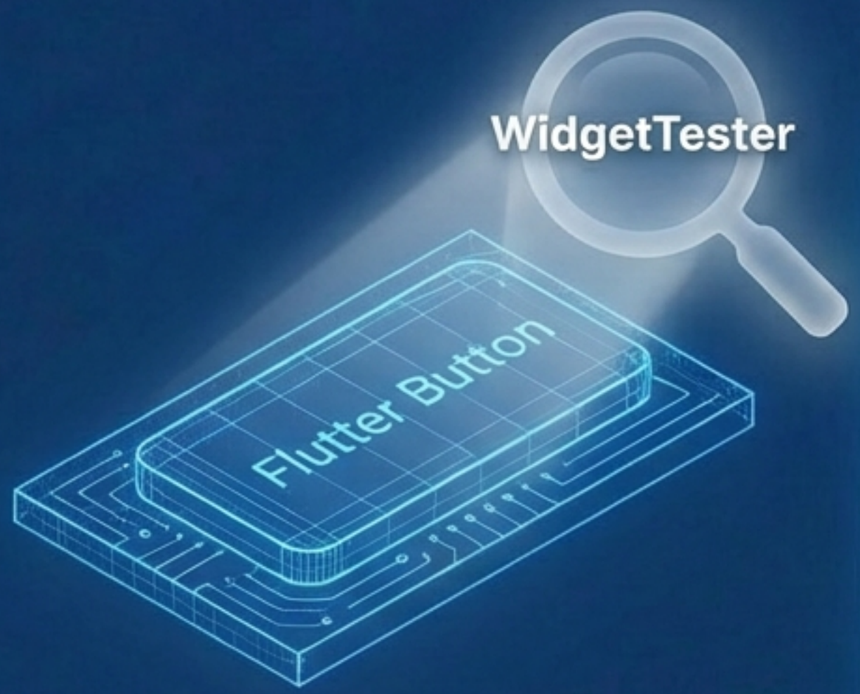

```
blocTest(
    'emits [1] when increment is added',
  // 1. Build: BLoC'u oluştur
  build: () => CounterBloc(),

  // 2. Act: Olay gönder
  act: (bloc) => bloc.add(CounterEvent.increment),

  // 3. Expect: Durumları doğrula
  expect: [1],
);
```

Görsel Katman: Widget Testi

`flutter_test` paketi kullanılır. UI bileşenlerini simülatör olmadan, sanal bir ortamda (**headless**) oluşturur ve test eder.



testWidgets(...)

```dart
testWidgets('Button tap test', (WidgetTester tester) async {
  // 1. UI'ı oluştur (Pump)
  await tester.pumpWidget(MyButton());

  // 2. Etkileşim (Tap)
  await tester.tap(find.byType(MyButton));
  await tester.pump();

  // 3. Doğrula (Expect)
  expect(find.text('Tapped!'), findsOneWidget);
});
```

Widget Test Akışı: Oluştur, Bul, Doğrula

PUMP (Oluştur)

Widget'ı sanal ağaca yerleştir.

```
await tester.pumpWidget(MyWidget());
```

FIND (Bul)

Elemanı tespit et.

```
find.text('Hello')
find.byIcon(...)
```

EXPECT (Doğrula)

Sonucu kontrol et.

```
expect(finder, findsOneWidget);
```

Etkileşimler ve Zaman

Kullanıcı Etkileşimi

Kullanıcı davranışlarını simüle edin.

```
await tester.tap(finder);
await tester.enterText(finder, 'Metin');
```

Zaman ve Rebuild

Animasyonları ve durum güncellemelerini tetikleyin.

```
await tester.pump(); (Bir frame ilerlet)
await tester.pump(Duration(seconds: 1));
(Zamanı ileri sar)
```

Zirve: Entegrasyon Testi (Integration Test)

Gerçek Cihaz & Tam Kullanıcı Deneyimi

Tool: `flutter_driver`

Ayrım: Test kodu ve Uygulama kodu iki farklı işlem (process) olarak çalışır.

Konum: `test_driver/` klasörü.

Hazırlık: Sürücü ve Anahtarlar

Derlenmiş uygulamada widget'ları bulabilmek için `Key` kullanımı zorunludur.

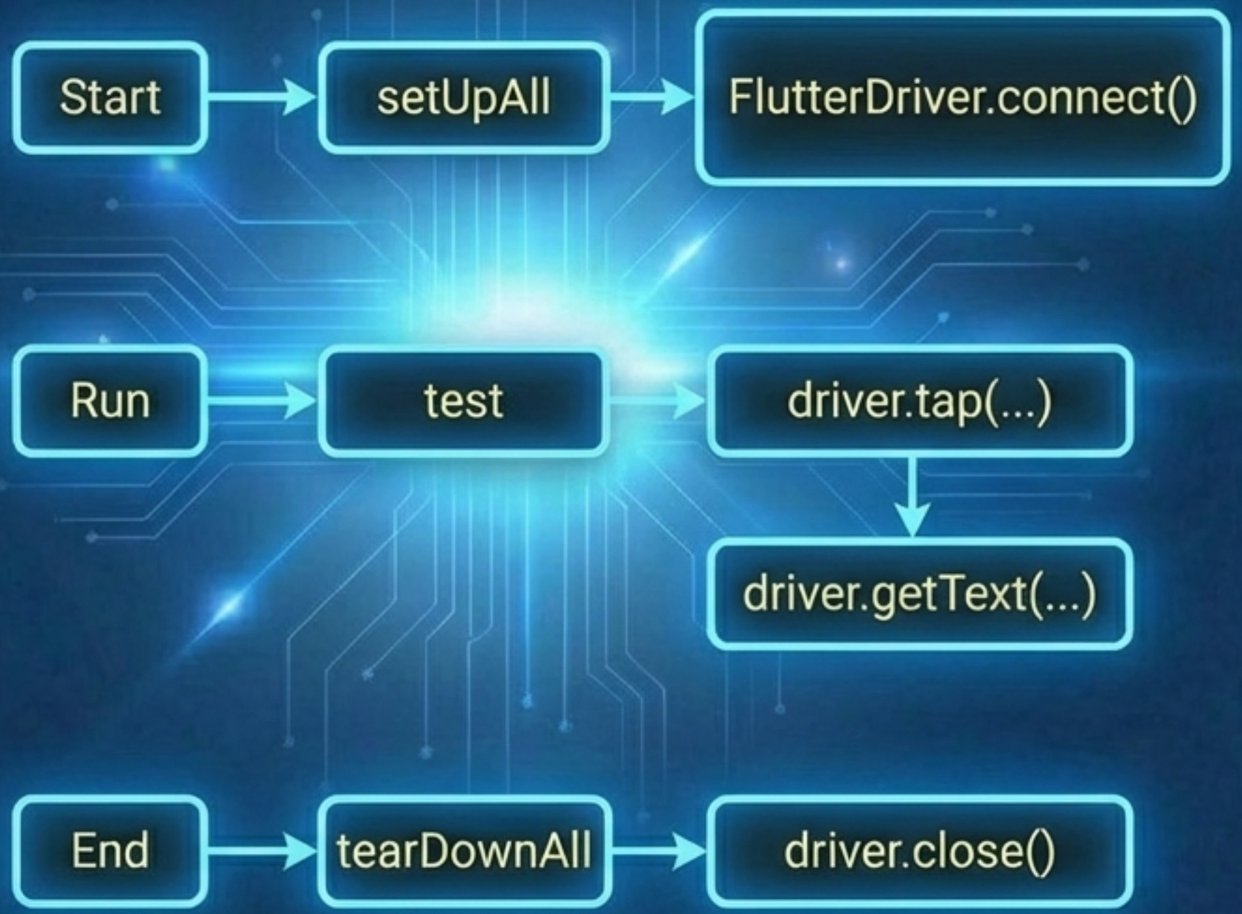
App Side (`lib/`)

```
FlatButton(
  key: Key("increment_btn"), // Kimlik
  onPressed: () => ...
);
```

Driver Side (`test_driver/app.dart`)

```
void main() {
  enableFlutterDriverExtension(); // Köprü
  runApp(MyApp());
}
```

Robotu Yönetmek: Test Senaryosu




```dart
test('Increment counter', () async {
  final btn =
    find.byValueKey('increment_btn');
  await driver.tap(btn); // Tıkla

  final result = await driver
    .getText(find.byValueKey('counter'));

  expect(result, "1"); // Kontrol Et
});
```

```
> flutter drive --target=test_driver/app.dart
```

Özet ve En İyi Uygulamalar

	Birim	Widget	Entegrasyon
Kapsam	Tekil Fonksiyon/Sınıf	Tekil UI Bileşeni	Tüm Uygulama
Hız	Çok Hızlı	Orta	Yavaş
Güven	Mantıksal Doğruluk	Görsel Doğruluk	Gerçek Kullanıcı Deneyimi
Kullanım Alanı	İş Mantığı, Hesaplamalar	Butonlar, Ekranlar	Kritik Akışlar (Ödeme, Login)

"Testler, gelecekteki hatalara karşı sigortanızdır. Hata bulunduğunda düzeltin, testini yazın ve güvenle ilerleyin."