Flutter Widget Testing

Testing User Interface Components

Real-world Examples with Todo App UI

What is Widget Testing?

Widget Testing tests individual UI components and their behavior:

Real-world analogy:

Like testing a **car dashboard** before installing it:

- **Individual gauges** work correctly (widget components)
- **Button presses** respond properly (user interactions)
- **Display updates** when values change (state changes)
- **Layout renders** correctly (visual appearance)

Widget testing is essentially **"unit testing for the UI layer."**

Unit Testing	Widget Testing
Tests **individual functions**	Tests **individual UI components**
Tests **business logic** in isolation	Tests **UI behavior** in isolation
Mocks **external dependencies**	Mocks **ViewModels/data sources**
Fast, focused, reliable	Fast, focused, reliable

Testing Pyramid: Widget Tests

```
🔺 E2E Tests (Few)
🔺🔺 Integration Tests (Some)
🔺🔺🔺🔺 Widget Tests (Many)
🔺🔺🔺🔺🔺🔺🔺 Unit Tests (Most)
```

Widget tests:

- ✅ Test UI components in isolation
- ✅ Faster than full app tests
- ✅ More comprehensive than unit tests
- ✅ Verify user interactions work

Widget Tests vs Other Test Types

Test Type	What it Tests	Speed	Dependencies
Unit	Business logic only	Fastest	None
Widget	UI components + logic	Fast	UI framework
Integration	Multiple components	Medium	Real services
E2E	Complete app flow	Slowest	Everything

Basic Widget Test Structure

```dart
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:provider/provider.dart';

void main() {
  group('TodoView Tests', () {
    late TodoViewModel viewModel;

    setUp(() {
      viewModel = TodoViewModel(); // Fresh state for each test
    });

    Widget createTestWidget() {
      return MaterialApp(  // 🔥 Required wrapper
        home: ChangeNotifierProvider.value(
          value: viewModel,
          child: const TodoView(),
        ),
      );
    }

    testWidgets('should display app bar', (tester) async {
      // Test implementation
    });
  });
}
```

Widget Test Setup: createTestWidget()

Why we need a wrapper:

```
Widget createTestWidget() {
  return MaterialApp(                     // Material theme & navigation
    home: ChangeNotifierProvider.value(   // State management
      value: viewModel,
      child: const TodoView(),            // Widget under test
    ),
  );
}
```

Key Points:

- `MaterialApp` provides Material Design context
- `ChangeNotifierProvider` injects the ViewModel
- Fresh `viewModel` for each test ensures isolation

Basic Widget Finding

```dart
testWidgets('should display app bar and empty state', (tester) async {
  // Arrange & Act
  await tester.pumpWidget(createTestWidget());

  // Assert - Find widgets by text
  expect(find.text('MVVM Todo App'), findsOneWidget);
  expect(find.text('No todos yet!\nAdd one using the input above.'), findsOneWidget);

  // Assert - Find widgets by key
  expect(find.byKey(const Key('empty_state')), findsOneWidget);
  expect(find.byKey(const Key('stats_section')), findsOneWidget);
});
```

`tester.pumpWidget()` renders the widget for testing

Widget Finder Methods

Finder	Usage
`find.text('text')`	Find by text content
`find.byKey(Key('key'))`	Find by unique key
`find.byType(Widget)`	Find by widget type
`find.byIcon(icon)`	Find by icon
`find.byWidget(widget)`	Find exact widget

Widget Finder Expectations

```
// Verify widget existence
expect(find.text('Total: 0'), findsOneWidget);     // Exactly one
expect(find.byType(Checkbox), findsNWidgets(2));   // Exactly N widgets
expect(find.text('Deleted Todo'), findsNothing);   // Widget not found

// Multiple matches
expect(find.byType(ElevatedButton), findsAtLeastNWidgets(1));
expect(find.byType(Text), findsWidgets);  // At least one
```

💡 **Use specific expectations to make tests more reliable**

Testing Initial Widget State

```
testWidgets('should display stats section with zero counts', (tester) async {
  // Arrange & Act
  await tester.pumpWidget(createTestWidget());

  // Assert - Check all stat displays
  expect(find.byKey(const Key('stats_section')), findsOneWidget);
  expect(find.byKey(const Key('total_count')), findsOneWidget);
  expect(find.byKey(const Key('pending_count')), findsOneWidget);
  expect(find.byKey(const Key('completed_count')), findsOneWidget);

  // Assert - Check initial values
  expect(find.text('Total: 0'), findsOneWidget);
  expect(find.text('Pending: 0'), findsOneWidget);
  expect(find.text('Completed: 0'), findsOneWidget);
});
```

✅ **Tests the widget's default/initial state**

Testing User Input

```
testWidgets('should add todo when button is pressed', (tester) async {
  // Arrange
  await tester.pumpWidget(createTestWidget());

  // Act - Simulate user typing
  await tester.enterText(
    find.byKey(const Key('todo_input_field')),
    'New Todo'
  );

  // Act - Simulate button tap
  await tester.tap(find.byKey(const Key('add_todo_button')));

  // Act - Trigger widget rebuild
  await tester.pump();

  // Assert - Check UI updated
  expect(find.text('New Todo'), findsOneWidget);
  expect(find.text('Total: 1'), findsOneWidget);
});
```

User Interaction Methods

Method	Purpose	Example
`tester.enterText()`	Type text in field	`tester.enterText(find.byKey('input'), 'text')`
`tester.tap()`	Tap a widget	`tester.tap(find.byKey('button'))`
`tester.longPress()`	Long press widget	`tester.longPress(find.text('item'))`
`tester.drag()`	Drag gesture	`tester.drag(find.byKey('item'), offset)`
`tester.scroll()`	Scroll a scrollable	`tester.scroll(find.byType(ListView))`

Always call `await tester.pump()` after interactions to update UI

Testing Widget State Changes

```
testWidgets('should toggle todo completion when checkbox is tapped', (tester) async {
  // Arrange - Add a todo first
  await tester.pumpWidget(createTestWidget());
  await tester.enterText(find.byKey(const Key('todo_input_field')), 'Toggle Todo');
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pump();

  // Act - Tap the checkbox
  await tester.tap(find.byType(Checkbox));
  await tester.pump();

  // Assert - Check state changed
  final checkbox = tester.widget<Checkbox>(find.byType(Checkbox));
  expect(checkbox.value, true);
  expect(find.text('Completed: 1'), findsOneWidget);
  expect(find.text('Pending: 0'), findsOneWidget);
});
```

Testing Widget Properties

```
testWidgets('should show strikethrough for completed todos', (tester) async {
  // Arrange & Act - Add and complete todo
  await tester.pumpWidget(createTestWidget());
  await tester.enterText(find.byKey(const Key('todo_input_field')), 'Completed Todo');
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pump();

  await tester.tap(find.byType(Checkbox));
  await tester.pump();

  // Assert - Check text styling
  final titleText = tester.widget<Text>(find.text('Completed Todo'));
  expect(titleText.style!.decoration, TextDecoration.lineThrough);
  expect(titleText.style!.color, Colors.grey);
});
```

`tester.widget<T>()` gives access to widget properties

Testing Dynamic Widget Lists

```
testWidgets('should display todo items correctly', (tester) async {
  // Arrange
  await tester.pumpWidget(createTestWidget());

  // Act - Add multiple todos
  await tester.enterText(find.byKey(const Key('todo_input_field')), 'Todo 1');
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pump();

  await tester.enterText(find.byKey(const Key('todo_input_field')), 'Todo 2');
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pump();

  // Assert - Check list contents
  expect(find.byKey(const Key('todos_list')), findsOneWidget);
  expect(find.text('Todo 1'), findsOneWidget);
  expect(find.text('Todo 2'), findsOneWidget);
  expect(find.byType(Checkbox), findsNWidgets(2));
  expect(find.byIcon(Icons.delete), findsNWidgets(2));
});
```

Testing Button States

```
testWidgets('should enable clear completed button when todos are completed', (tester) async {
  // Arrange - Add and complete todo
  await tester.pumpWidget(createTestWidget());
  await tester.enterText(find.byKey(const Key('todo_input_field')), 'Completed Todo');
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pump();

  await tester.tap(find.byType(Checkbox));
  await tester.pump();

  // Assert - Button enabled and text updated
  expect(find.text('Clear Completed (1)'), findsOneWidget);

  final button = tester.widget<ElevatedButton>(
    find.byKey(const Key('clear_completed_button'))
  );
  expect(button.onPressed, isNotNull); // Button is enabled
});
```

Testing Input Validation

```dart
testWidgets('should ignore empty input', (tester) async {
  // Arrange
  await tester.pumpWidget(createTestWidget());

  // Act - Try to add empty todo
  await tester.enterText(find.byKey(const Key('todo_input_field')), '   ');
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pump();

  // Assert - Nothing should be added
  expect(find.byKey(const Key('empty_state')), findsOneWidget);
  expect(find.text('Total: 0'), findsOneWidget);
});

testWidgets('should clear input field after adding todo', (tester) async {
  // Arrange & Act
  await tester.pumpWidget(createTestWidget());
  await tester.enterText(find.byKey(const Key('todo_input_field')), 'Clear Test');
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pump();

  // Assert - Input field is cleared
  final textField = tester.widget<TextField>(find.byKey(const Key('todo_input_field')));
  expect(textField.controller?.text, isEmpty);
});
```

Testing Enter Key Submission

```dart
testWidgets('should add todo when enter is pressed', (tester) async {
  // Arrange
  await tester.pumpWidget(createTestWidget());

  // Act - Type and press enter
  await tester.enterText(find.byKey(const Key('todo_input_field')), 'Enter Todo');
  await tester.testTextInput.receiveAction(TextInputAction.done);
  await tester.pump();

  // Assert
  expect(find.text('Enter Todo'), findsOneWidget);
  expect(find.text('Total: 1'), findsOneWidget);
});
```

`testTextInput.receiveAction()` simulates keyboard actions

Testing Widget Deletion

```dart
testWidgets('should delete todo when delete button is tapped', (tester) async {
  // Arrange - Add a todo
  await tester.pumpWidget(createTestWidget());
  await tester.enterText(find.byKey(const Key('todo_input_field')), 'Delete Todo');
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pump();

  // Act - Delete the todo
  await tester.tap(find.byIcon(Icons.delete));
  await tester.pump();

  // Assert - Todo is gone, empty state is back
  expect(find.text('Delete Todo'), findsNothing);
  expect(find.byKey(const Key('empty_state')), findsOneWidget);
  expect(find.text('Total: 0'), findsOneWidget);
});
```

Testing Complex Workflows

```
testWidgets('should clear completed todos when button is tapped', (tester) async {
  // Arrange - Add two todos, complete one
  await tester.pumpWidget(createTestWidget());

  await tester.enterText(find.byKey(const Key('todo_input_field')), 'Completed Todo');
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pump();

  await tester.enterText(find.byKey(const Key('todo_input_field')), 'Pending Todo');
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pump();

  // Mark first as completed
  await tester.tap(find.byType(Checkbox).first);
  await tester.pump();

  // Act - Clear completed
  await tester.tap(find.byKey(const Key('clear_completed_button')));
  await tester.pump();

  // Assert - Only pending todo remains
  expect(find.text('Completed Todo'), findsNothing);
  expect(find.text('Pending Todo'), findsOneWidget);
  expect(find.text('Total: 1'), findsOneWidget);
});
```

The Importance of Keys in Widget Testing

This is why widgets keep the keys!

Without keys - unreliable:

```
// ❌ Hard to target specific widgets
await tester.tap(find.byType(Checkbox).first); // Which checkbox?
```

With keys - precise:

```
// ✅ Target exact widgets
await tester.tap(find.byKey(Key('checkbox_${todo.id}')));
expect(find.byKey(Key('title_${todo.id}')), findsOneWidget);
```

In the TodoView:

```
Checkbox(
  key: Key('checkbox_${todo.id}'), // ✅ Unique key
  value: todo.isCompleted,
  onChanged: (_) => context.read<TodoViewModel>().toggleTodo(todo.id),
)
```

Testing Widget Keys

```dart
testWidgets('should have proper test keys for all interactive elements', (tester) async {
  // Arrange
  await tester.pumpWidget(createTestWidget());
  await tester.enterText(find.byKey(const Key('todo_input_field')), 'Keyed Todo');
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pump();

  // Assert - General UI keys
  expect(find.byKey(const Key('stats_section')), findsOneWidget);
  expect(find.byKey(const Key('todo_input_field')), findsOneWidget);
  expect(find.byKey(const Key('add_todo_button')), findsOneWidget);
  expect(find.byKey(const Key('todos_list')), findsOneWidget);

  // Assert - Todo-specific keys
  final todoId = viewModel.todos.first.id;
  expect(find.byKey(Key('todo_card_$todoId')), findsOneWidget);
  expect(find.byKey(Key('checkbox_$todoId')), findsOneWidget);
  expect(find.byKey(Key('title_$todoId')), findsOneWidget);
  expect(find.byKey(Key('delete_$todoId')), findsOneWidget);
});
```

Testing Provider/Consumer Patterns

The widget under test uses Provider:

```
Consumer<TodoViewModel>(
  builder: (context, viewModel, child) {
    return Text('Total: ${viewModel.todos.length}');
  },
),
```

Test setup provides the ViewModel:

```
Widget createTestWidget() {
  return MaterialApp(
    home: ChangeNotifierProvider.value(
      value: viewModel,     // ✅ Inject test ViewModel
      child: const TodoView(),
    ),
  );
}
```

Widget Test Organization

```dart
group('TodoView Tests', () {
  late TodoViewModel viewModel;

  setUp(() {
    viewModel = TodoViewModel(); // Fresh state
  });
  group('Initial State', () {
    // Tests for initial widget state
  });
  group('Adding Todos', () {
    // Tests for add functionality
  });
  group('Todo Interactions', () {
    // Tests for toggle, delete
  });
  group('Layout and Keys', () {
    // Tests for proper widget structure
  });
});
```

Widget Testing Best Practices

✅ **DO:**

- Use meaningful test keys (`Key('add_button')`)
- Test user interactions, not just widget existence
- Verify state changes after interactions
- Use `setUp()` for fresh state
- Group related tests together
- Test edge cases (empty input, disabled buttons)

❌ **DON'T:**

- Test implementation details
- Create overly complex test setups
- Forget to call `tester.pump()` after interactions

Common Widget Testing Patterns

1. Empty State Testing:

```
// Verify empty state displays correctly
expect(find.byKey(const Key('empty_state')), findsOneWidget);
```

2. List Testing:

```
// Add items and verify list updates
expect(find.byType(Checkbox), findsNWidgets(2));
```

3. Form Testing:

```
// Test input validation and submission
await tester.enterText(find.byKey('input'), 'value');
await tester.tap(find.byKey('submit'));
```

4. Button State Testing:

Debugging Widget Tests

Common issues and solutions:

1. **Widget not found:**

```
// Add debug output
await tester.pumpWidget(createTestWidget());
print(tester.allWidgets.map((w) => w.runtimeType).toList());
```

2. **State not updating:**

```
// Ensure you call pump() after interactions
await tester.tap(find.byKey('button'));
await tester.pump(); // ✅ Required!
```

3. **Wrong widget found:**

```
// Use more specific finders
find.byKey(Key('specific_key')) // Better than find.byType()
```

Performance Considerations

Widget tests are fast, but can be optimized:

✅ **Efficient:**

```
// Single widget creation per test
Widget createTestWidget() => MaterialApp(home: MyWidget());

// Specific finders
find.byKey(Key('specific_key'))
```

❌ **Inefficient:**

```
// Complex nested MaterialApp setups
// Using find.byType() when find.byKey() would work
// Creating new widgets in every assertion
```

Widget tests should run in milliseconds, not seconds

Running Widget Tests

Command line:

```
flutter test test/view/todo_view_test.dart  # Single test file
flutter test test/view/                     # All view tests
flutter test --reporter expanded            # Verbose output
```

IDE: Click ▶️ next to test groups or individual tests

Watch mode:

```
flutter test --watch test/view/
```

Widget Tests vs Golden Tests

Test Type	Purpose	When to Use
Widget Tests	Behavior & interactions	Always
Golden Tests	Visual appearance	UI-heavy widgets

Widget tests verify functionality works

Golden tests verify UI looks correct

Both are valuable for comprehensive UI testing

Summary

- **Widget tests** verify UI components and user interactions
- `testWidgets()` provides the testing framework
- **Finders** locate widgets (`find.byKey` , `find.text`)
- **Interactions** simulate user behavior (`tap` , `enterText`)
- `tester.pump()` updates UI after changes
- **Keys** make tests reliable and maintainable
- **Provider** integration enables state testing

Remember: Widget tests bridge the gap between unit tests and full app testing!