Flutter Integration Testing

End-to-End User Workflows

Real-world Examples with Complete Todo App Flows

What is Integration Testing?

Integration Testing tests complete user workflows across multiple components:

Real-world analogy:

Like testing a **complete car journey**:

- **Unit Tests** = Test engine separately
- **Widget Tests** = Test dashboard separately
- **Integration Tests** = Test complete driving experience

💡 **Integration tests verify the entire app works together**

Testing Pyramid in Detail

```
        ▲ E2E Tests (Few, Very Slow)
      ▲ ▲ Integration Tests (Some, Slow)
    ▲ ▲ ▲ ▲ Widget Tests (Many, Fast)
  ▲ ▲ ▲ ▲ ▲ ▲ ▲ Unit Tests (Most, Fastest)
```

Test Type	Speed	Dependencies	Purpose
Unit	⚡ Fastest	None	Business logic
Widget	🚀 Fast	UI framework	UI components
Integration	🐌 Slow	Full app	User workflows
E2E	🐢 Slowest	Everything	Complete system

Why Integration Testing?

✅ **Catches issues that unit/widget tests miss:**

- Navigation between screens
- State management across components
- Real user interaction flows
- Platform-specific behavior
- Performance under real conditions

✅ **Builds confidence each components can communicate properly:**

- App works as users expect
- Critical paths function correctly
- Changes don't break workflows

Integration vs Other Test Types

Unit Test:

```
test('should add todo', () {
  viewModel.addTodo('Test');
  expect(viewModel.todos.length, 1);
});
```

Tests isolated logic

Integration Test:

```
testWidgets('should complete todo flow', (tester) async {
  // Start real app
  app.main();
  await tester.pumpAndSettle();

  // Complete user workflow
  await addTodo(tester, 'Test');
  await completeTodo(tester, 'Test');
  await deleteTodo(tester, 'Test');
});
```

Tests complete user journey

```
// UNIT TEST - One box
 ┌──────────────┐
 │ ViewModel    │   ← Only this
 └──────────────┘

// INTEGRATION TEST - Multiple connected boxes
 ┌──────────┐   ┌──────────┐   ┌──────────┐
 │   UI     │ ←→ │ ViewModel │ ←→ │  State   │
 └──────────┘   └──────────┘   └──────────┘
      ↑              ↑              ↑
  All of these components working together
```

Project Setup

1. Add to `pubspec.yaml`:

```yaml
dev_dependencies:
  flutter_test:
    sdk: flutter
  integration_test:
    sdk: flutter  # Add this line
```

2. Create directory structure:

```
project/
├── integration_test/
│   └── app_test.dart
├── test/
│   ├── unit/
│   └── view/
└── lib/
```

Basic Integration Test Setup

```dart
// integration_test/app_test.dart
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:integration_test/integration_test.dart';
import 'package:provider/provider.dart';
import 'package:todo/main.dart' as app;
import 'package:todo/viewmodels/todo_viewmodel.dart';
import 'package:todo/views/todo_view.dart';

void main() {
  IntegrationTestWidgetsFlutterBinding.ensureInitialized();

  group('Todo App Integration Tests', () {
    // Tests go here
  });
}
```

`IntegrationTestWidgetsFlutterBinding.ensureInitialized()` is required!

Two Approaches: Test Widget vs Full App

Approach 1: Test Specific Widget

```
testWidgets('test TodoView', (tester) async {
  await tester.pumpWidget(
    ChangeNotifierProvider(
      create: (context) => TodoViewModel(),
      child: const MaterialApp(
        home: TodoView(),
      ),
    ),
  );

  // Test TodoView in isolation
});
```

Approach 2: Test Full App

```
testWidgets('test complete app', (tester) async {
  // Start the real app
  app.main();
  await tester.pumpAndSettle();

  // Test complete app including
  // navigation, routing, etc.
});
```

Simple Integration Test Example

```
testWidgets('should add and display todo', (tester) async {
  // Arrange - Start the app
  await tester.pumpWidget(
    ChangeNotifierProvider(
      create: (context) => TodoViewModel(),
      child: const MaterialApp(home: TodoView()),
    ),
  );
  await tester.pumpAndSettle();

  // Assert - Initial empty state
  expect(find.byKey(const Key('empty_state')), findsOneWidget);
  expect(find.text('Total: 0'), findsOneWidget);

  // Act - Add a todo
  await tester.enterText(find.byKey(const Key('todo_input_field')), 'Integration Test Todo');
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pumpAndSettle();

  // Assert - Todo should appear
  expect(find.text('Integration Test Todo'), findsOneWidget);
  expect(find.text('Total: 1'), findsOneWidget);
  expect(find.byKey(const Key('empty_state')), findsNothing);
});
```

Complete User Workflow Test

```
testWidgets('should complete full todo management flow', (tester) async {
  // Setup
  await tester.pumpWidget(createApp());
  await tester.pumpAndSettle();

  // Step 1: Add multiple todos
  await addTodo(tester, 'Buy groceries');
  await addTodo(tester, 'Walk the dog');
  expect(find.text('Total: 2'), findsOneWidget);

  // Step 2: Complete one todo
  final checkboxes = find.byType(Checkbox);
  await tester.tap(checkboxes.first);
  await tester.pumpAndSettle();
  expect(find.text('Completed: 1'), findsOneWidget);

  // Step 3: Clear completed todos
  await tester.tap(find.byKey(const Key('clear_completed_button')));
  await tester.pumpAndSettle();
  expect(find.text('Buy groceries'), findsNothing);
  expect(find.text('Walk the dog'), findsOneWidget);

  // Step 4: Delete remaining todo
  await tester.tap(find.byIcon(Icons.delete));
  await tester.pumpAndSettle();
  expect(find.byKey(const Key('empty_state')), findsOneWidget);
});
```

Running Integration Tests

Command line:

```
flutter test integration_test/app_test.dart
```

You'll see platform selection:

```
Connected devices:
macOS (desktop) • macos  • darwin-arm64   • macOS 15.6.1
Chrome (web)    • chrome • web-javascript • Google Chrome
iPhone (mobile) • device • ios            • iOS 17.0

[1]: macOS (macos)
[2]: Chrome (chrome)
[3]: iPhone (device)
Please choose one (or "q" to quit):
```

Platform Considerations

Platform	Pros	Cons	Best For
PC/Linux	Native performance	Input focus issues	Desktop apps
Chrome	Reliable, fast	Web limitations	Web apps, debugging
iOS/Android	Real device behavior	Slow, device needed	Mobile testing

Recommendation: Start with Chrome for reliability, test on mobile for final validation

Platform	Chrome Tests	Desktop Tests	Mobile Tests	Recommended
macOS	❌ Problematic	✅ Excellent	✅ Good	**macOS Desktop**
Windows	✅ Excellent	✅ Good	✅ Good	**Chrome**
Linux	✅ Very Good	✅ Good	✅ Good	**Chrome**

- **macOS**: Use `flutter test -d macos` due to Chrome security issues
- **Windows/Linux**: Use `flutter test -d chrome` for best reliability
- **Mobile**: All platforms support iOS/Android simulators equally

Common Issue #1: Input Field Problems

❌ The Problem:

```
Input field content: ""  // Text didn't actually get entered!
```

Real debug output from our experience:

```
=== ADDING SECOND TODO ===
Entered text: Debug Todo 2
Input field content: ""          ⬅️ Empty!
Tapped add button
```

This happens because `enterText()` can fail silently on some platforms

Solution: Robust Input Handling

```
Future<void> addTodoWithFocusHandling(WidgetTester tester, String todoText) async {
  print('Adding todo: $todoText');

  // Step 1: Focus the input field explicitly
  await tester.tap(find.byKey(const Key('todo_input_field')));
  await tester.pumpAndSettle();

  // Step 2: Clear existing content
  final inputField = find.byKey(const Key('todo_input_field'));
  final textFieldWidget = tester.widget<TextField>(inputField);
  textFieldWidget.controller?.clear();
  await tester.pump();

  // Step 3: Enter text with verification
  await tester.enterText(inputField, todoText);
  await tester.pump(const Duration(milliseconds: 100));

  // Step 4: Verify text was entered
  final currentText = textFieldWidget.controller?.text ?? '';
  if (currentText != todoText) {
    // Fallback: Direct controller manipulation
    textFieldWidget.controller?.text = todoText;
    await tester.pump();
  }

  // Step 5: Submit
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pumpAndSettle();
}
```

Common Issue #2: Timing Problems

❌ Problem: Animations not finished

```
await tester.tap(find.byKey(Key('button')));
// UI might still be animating!
expect(find.text('Result'), findsOneWidget); // Might fail
```

✅ Solution: Wait for completion

```
await tester.tap(find.byKey(Key('button')));
await tester.pumpAndSettle(); // Wait for all animations
expect(find.text('Result'), findsOneWidget); // Reliable
```

Additional timing options:

```
await tester.pump(); // Single frame
await tester.pump(Duration(milliseconds: 500)); // Specific delay
await tester.pumpAndSettle(Duration(seconds: 2)); // Max wait time
```

Common Issue #3: State Management

Problem: State doesn't update as expected

```
// Add todo
await addTodo(tester, 'Test Todo');

// State might not be immediately updated
expect(find.text('Total: 1'), findsOneWidget); // Might fail
```

Solution: Add state update delays

```
await addTodo(tester, 'Test Todo');
await tester.pump(const Duration(milliseconds: 200)); // Extra wait
expect(find.text('Total: 1'), findsOneWidget); // More reliable
```

Testing Input Validation

```
testWidgets('should handle input validation correctly', (tester) async {
  await tester.pumpWidget(createApp());
  await tester.pumpAndSettle();

  // Test 1: Empty input should be ignored
  await tester.enterText(find.byKey(const Key('todo_input_field')), '   ');
  await tester.tap(find.byKey(const Key('add_todo_button')));
  await tester.pumpAndSettle();

  expect(find.byKey(const Key('empty_state')), findsOneWidget);
  expect(find.text('Total: 0'), findsOneWidget);

  // Test 2: Valid input should work
  await addTodoWithFocusHandling(tester, 'Valid Todo');
  expect(find.text('Valid Todo'), findsOneWidget);
  expect(find.text('Total: 1'), findsOneWidget);
});
```

Testing Multiple User Interactions

```
testWidgets('should handle multiple todos efficiently', (tester) async {
  await tester.pumpWidget(createApp());
  await tester.pumpAndSettle();

  // Add multiple todos with verification
  for (int i = 1; i <= 3; i++) {
    await addTodoWithFocusHandling(tester, 'Todo $i');

    // Verify each addition worked
    expect(find.text('Todo $i'), findsOneWidget);
    expect(find.text('Total: $i'), findsOneWidget);
  }

  // Test bulk operations
  final checkboxes = find.byType(Checkbox);
  await tester.tap(checkboxes.at(0)); // Complete first
  await tester.tap(checkboxes.at(2)); // Complete third
  await tester.pumpAndSettle();

  // Verify final state
  expect(find.text('Total: 3'), findsOneWidget);
  expect(find.text('Pending: 1'), findsOneWidget);
  expect(find.text('Completed: 2'), findsOneWidget);
});
```

Testing Real App Startup

```
testWidgets('should test real app startup', (tester) async {
  // Start the actual main() function
  app.main();
  await tester.pumpAndSettle();

  // Test real app initialization
  expect(find.text('MVVM Todo App'), findsOneWidget);
  expect(find.byKey(const Key('empty_state')), findsOneWidget);
  expect(find.byKey(const Key('todo_input_field')), findsOneWidget);

  // Test that real app state management works
  await addTodoWithFocusHandling(tester, 'Real App Test');
  expect(find.text('Real App Test'), findsOneWidget);
  expect(find.text('Total: 1'), findsOneWidget);
});
```

💡 **This tests your actual app entry point and routing**

Debugging Integration Tests

1. Add comprehensive logging:

```
testWidgets('debug test', (tester) async {
  print('=== STARTING TEST ===');
  await tester.pumpWidget(createApp());

  print('=== CURRENT WIDGETS ===');
  final allText = find.byType(Text);
  for (int i = 0; i < allText.evaluate().length; i++) {
    final text = tester.widget<Text>(allText.at(i));
    print('Text $i: "${text.data}"');
  }

  print('=== ADDING TODO ===');
  await addTodo(tester, 'Debug Todo');

  print('=== FINAL STATE ===');
  // ... more logging
});
```

Visual Debugging

Take screenshots during test:

```
testWidgets('debug with screenshots', (tester) async {
  await tester.pumpWidget(createApp());

  // Take screenshot of initial state
  await tester.binding.convertFlutterSurfaceToImage();

  await addTodo(tester, 'Test');

  // Take screenshot after action
  await tester.binding.convertFlutterSurfaceToImage();
});
```

Enable visual debugging:

```
flutter test integration_test/ --reporter expanded
```

Performance Testing

```
testWidgets('should handle large number of todos', (tester) async {
  await tester.pumpWidget(createApp());
  await tester.pumpAndSettle();

  // Measure performance with many todos
  final stopwatch = Stopwatch()..start();

  for (int i = 0; i < 50; i++) {
    await addTodoWithFocusHandling(tester, 'Performance Todo $i');
  }

  stopwatch.stop();
  print('Added 50 todos in ${stopwatch.elapsedMilliseconds}ms');

  // Verify app still responsive
  expect(find.text('Total: 50'), findsOneWidget);

  // Test scrolling performance
  final listFinder = find.byKey(const Key('todos_list'));
  await tester.drag(listFinder, const Offset(0, -500));
  await tester.pumpAndSettle();
});
```

Platform-Specific Testing

```
testWidgets('should work on web platform', (tester) async {
  await tester.pumpWidget(createApp());
  await tester.pumpAndSettle();

  // Web-specific interactions
  await tester.enterText(find.byKey(const Key('todo_input_field')), 'Web Todo');

  // Test web keyboard shortcuts
  await tester.testTextInput.receiveAction(TextInputAction.done);
  await tester.pumpAndSettle();

  expect(find.text('Web Todo'), findsOneWidget);
}, variant: TargetPlatformVariant.only(TargetPlatform.web));

testWidgets('should work on mobile', (tester) async {
  // Mobile-specific tests
}, variant: TargetPlatformVariant.mobile());
```

Best Practices Summary

✅ DO:

- Test complete user workflows, not individual components
- Use `pumpAndSettle()` after interactions
- Handle input field focus issues proactively
- Add timing delays for complex state changes
- Test on multiple platforms (Chrome for development, mobile for validation)
- Use helper functions for common actions (addTodo, completeTodo)
- Add debugging output when tests fail

❌ DON'T:

- Test every edge case (use unit tests for that)
- Create overly complex test setups
- Ignore timing issues - they will cause flaky tests
- Forget to handle platform-specific input behaviors
- Test implementation details instead of user behavior

When to Write Integration Tests

✅ **Write integration tests for:**

- **Critical user journeys** (signup, checkout, main app flow)
- **Cross-component interactions** (navigation, state sharing)
- **Platform-specific behavior** (mobile gestures, web shortcuts)
- **Performance-critical paths** (large lists, complex animations)
- **End-to-end workflows** (create → edit → delete cycles)

❌ **Don't write integration tests for:**

- Individual widget behavior (use widget tests)
- Business logic edge cases (use unit tests)
- Error handling details (use unit tests)
- Every possible user path (focus on critical flows)

Integration Test Organization

```
group('Todo App Integration Tests', () {
  group('Basic Functionality', () {
    testWidgets('should add and display todos', (tester) async { /* */ });
    testWidgets('should handle input validation', (tester) async { /* */ });
  });

  group('Complete Workflows', () {
    testWidgets('should complete todo lifecycle', (tester) async { /* */ });
    testWidgets('should handle multiple todos', (tester) async { /* */ });
  });

  group('Edge Cases', () {
    testWidgets('should handle app restart', (tester) async { /* */ });
    testWidgets('should handle network issues', (tester) async { /* */ });
  });

  group('Performance', () {
    testWidgets('should handle large datasets', (tester) async { /* */ });
  });
});
```

Running Integration Tests in CI/CD

GitHub Actions example:

```yaml
- name: Run Integration Tests
  run: |
    flutter test integration_test/ -d web-server --web-port=7357
```

Local testing script:

```bash
#!/bin/bash
echo "Running integration tests on Chrome..."
flutter test integration_test/ -d chrome

echo "Running integration tests on macOS..."
flutter test integration_test/ -d macos

echo "Integration tests completed!"
```

Summary

- **Integration tests** verify complete user workflows
- **Test real scenarios** users actually perform
- **Handle platform differences** (Chrome often more reliable)
- **Focus on timing** - use `pumpAndSettle()` and delays
- **Debug systematically** with logging and screenshots
- **Test critical paths** not every edge case
- **Input field focus** is a common issue - handle proactively

Remember: Integration tests give confidence that your app works as users expect!