Flutter Unit Testing

Testing Models and ViewModels

Real-world Examples with Todo App

What is Unit Testing?

Unit Testing tests individual components in isolation:

Like testing individual car parts:

- **Unit Test** = Test the engine separately
- **Integration Test** = Test engine + transmission together
- **Widget Test** = Test the complete dashboard
- **End-to-End Test** = Test the complete car driving

Why Unit Testing?

Benefits:

- **Early Bug Detection** - Find problems before users do
- **Code Quality** - Forces good design patterns
- **Refactoring Safety** - Change code with confidence
- **Documentation** - Tests show how code should work
- **Faster Debugging** - Isolate problems quickly

Test Structure: Arrange-Act-Assert

Every good test follows this pattern:

```dart
test('should create a Todo with required parameters', () {
  // Arrange - Set up test data
  final expectedId = '1';
  final expectedTitle = 'Test Todo';

  // Act - Execute the code being tested
  final todo = Todo(id: expectedId, title: expectedTitle);

  // Assert - Verify the results
  expect(todo.id, expectedId);
  expect(todo.title, expectedTitle);
  expect(todo.isCompleted, false); // Default value
});
```

Testing Models: Todo Class

Model tests focus on:

- ✅ Constructor behavior
- ✅ Method functionality
- ✅ Edge cases
- ✅ Data integrity

Remember: Models are simple data classes with no business logic

In the main directory, we can use `flutter test` command to run the tests.

```
todo> flutter test test/unit/todo_model_test.dart
00:00 +10: All tests passed!
```

In this example, all the unit tests in the todo_model_test.dart are tested.

```dart
import 'package:flutter_test/flutter_test.dart';
import 'package:todo/models/todo.dart';

void main() {
  group('Todo Model Tests', () {

    test('should create a Todo with required parameters', () {...}
    ...
  }
}
```

Testing Constructor Behavior

```
import 'package:flutter_test/flutter_test.dart';
import 'package:todo/models/todo.dart';
```

```
group('Todo Model Tests', () {
  test('should create a Todo with required parameters', () {
    // Arrange & Act
    final todo = Todo(id: '1', title: 'Test Todo');

    // Assert
    expect(todo.id, '1');
    expect(todo.title, 'Test Todo');
    expect(todo.isCompleted, false); // Default value
  });

  test('should create a Todo with all parameters', () {
    // Arrange & Act
    final todo = Todo(id: '1', title: 'Test Todo', isCompleted: true);

    // Assert
    expect(todo.isCompleted, true); // Explicitly set value
  });
});
```

Testing copyWith Method

Why test `copyWith` ?

- Immutable updates are critical
- Must preserve unchanged values
- Common source of bugs

```
group('copyWith method', () {
  late Todo originalTodo;

  setUp(() {
    originalTodo = Todo(
      id: '1',
      title: 'Original Title',
      isCompleted: false,
    );
  });

  test('should create copy with updated title', () {
    // Act
    final updated = originalTodo.copyWith(
      title: 'Updated Title'
    );

    // Assert
    expect(updated.id, originalTodo.id);
    expect(updated.title, 'Updated Title');
    expect(updated.isCompleted, originalTodo.isCompleted);
  });
});
```

Testing Equality and toString

```
group('equality tests', () {
  test('should be equal when all properties match', () {
    // Arrange
    final todo1 = Todo(id: '1', title: 'Test Todo', isCompleted: true);
    final todo2 = Todo(id: '1', title: 'Test Todo', isCompleted: true);

    // Assert
    expect(todo1, equals(todo2));
    expect(todo1.hashCode, equals(todo2.hashCode));
  });

  test('should not be equal when properties differ', () {
    // Arrange
    final todo1 = Todo(id: '1', title: 'Test Todo 1');
    final todo2 = Todo(id: '2', title: 'Test Todo 2');

    // Assert
    expect(todo1, isNot(equals(todo2)));
  });
});
```

Testing Edge Cases

```
group('edge cases', () {
  test('should handle empty title', () {
    // Act
    final todo = Todo(id: '1', title: '');

    // Assert
    expect(todo.title, '');
  });

  test('should handle special characters in title', () {
    // Arrange
    const specialTitle = 'Todo with émojis 🎉';

    // Act
    final todo = Todo(id: '1', title: specialTitle);

    // Assert
    expect(todo.title, specialTitle);
  });
});

test('toString should return meaningful representation', () {
  // Arrange
  final todo = Todo(id: '1', title: 'Test Todo', isCompleted: true);

  // Act
  final stringRepresentation = todo.toString();

  // Assert
  expect(stringRepresentation, contains('Todo'));
  expect(stringRepresentation, contains('id: 1'));
});
```

Testing ViewModels: Business Logic

ViewModel tests focus on:

- ✅ Initial state
- ✅ State changes
- ✅ Business rules
- ✅ Computed properties
- ✅ Notification behavior
- ✅ Service integration

Testing Initial State with Dependency Injection

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

  setUp(() {
    todoService = TodoService();
    viewModel = TodoViewModel(todoService); // Service injection
  });

  group('Initial State', () {
    test('should have empty initial state', () {
      expect(viewModel.todos, isEmpty);
      expect(viewModel.totalTodos, 0);
      expect(viewModel.completedTodos, 0);
      expect(viewModel.pendingTodos, 0);
    });
  });
});
```

💡 **ViewModel now uses dependency injection for better testability**

Testing Business Logic: Adding Todos

```dart
group('Adding Todos', () {
  test('should add todo successfully', () {
    // Arrange
    int notificationCount = 0;
    viewModel.addListener(() => notificationCount++);

    // Act
    viewModel.addTodo('New Todo');

    // Assert
    expect(viewModel.todos.length, 1);
    expect(viewModel.todos.first.title, 'New Todo');
    expect(viewModel.todos.first.isCompleted, false);
    expect(viewModel.totalTodos, 1);
    expect(viewModel.pendingTodos, 1);
    expect(notificationCount, 1); // ChangeNotifier fired
  });
});
```

✅ **Tests both state change AND notification behavior**

Testing Data Validation

```
test('should trim whitespace from title', () {
  // Act
  viewModel.addTodo('  Trimmed Title  ');

  // Assert
  expect(viewModel.todos.first.title, 'Trimmed Title');
});

test('should ignore empty title', () {
  // Act
  viewModel.addTodo('   '); // Only whitespace

  // Assert
  expect(viewModel.todos, isEmpty);
});
```

💡 **Business rules must be tested thoroughly**

Testing Unique ID Generation

```
test('should create unique IDs for todos', () async {
  // Act
  viewModel.addTodo('Todo 1');
  await Future.delayed(const Duration(milliseconds: 2));
  viewModel.addTodo('Todo 2');

  // Assert
  expect(viewModel.todos[0].id, isNot(equals(viewModel.todos[1].id)));
});
```

⚠️ **Note:** Tests async behavior when timing matters

Testing State Transitions

```dart
group('Toggling Todos', () {
  test('should toggle todo completion successfully', () {
    // Arrange
    viewModel.addTodo('Test Todo');
    final todoId = viewModel.todos.first.id;
    expect(viewModel.todos.first.isCompleted, false);

    // Act
    viewModel.toggleTodo(todoId);

    // Assert
    expect(viewModel.todos.first.isCompleted, true);
    expect(viewModel.completedTodos, 1);
    expect(viewModel.pendingTodos, 0);
  });

  test('should toggle back to incomplete', () {
    // Arrange
    viewModel.addTodo('Test Todo');
    final todoId = viewModel.todos.first.id;
    viewModel.toggleTodo(todoId); // Mark as completed

    // Act
    viewModel.toggleTodo(todoId); // Mark as incomplete

    // Assert
    expect(viewModel.todos.first.isCompleted, false);
    expect(viewModel.completedTodos, 0);
    expect(viewModel.pendingTodos, 1);
  });
});
```

Testing Remove Operations

```dart
group('Removing Todos', () {
  test('should remove todo successfully', () {
    // Arrange
    viewModel.addTodo('Todo to Remove');
    expect(viewModel.todos.length, 1);

    // Act
    final todoId = viewModel.todos.first.id;
    viewModel.deleteTodo(todoId);

    // Assert
    expect(viewModel.todos.length, 0);
  });

  test('should remove correct todo when multiple exist', () async {
    // Arrange
    viewModel.addTodo('Todo 1');
    await Future.delayed(const Duration(milliseconds: 2));
    viewModel.addTodo('Todo 2');
    expect(viewModel.todos.length, 2);

    // Act
    final firstId = viewModel.todos[0].id;
    viewModel.deleteTodo(firstId);

    // Assert
    expect(viewModel.todos.length, 1);
    expect(viewModel.todos.first.title, 'Todo 2');
  });
});
```

Testing Bulk Operations

```
group('Clear Completed', () {
  test('should clear completed todos', () async {
    // Arrange
    viewModel.addTodo('Todo 1');
    await Future.delayed(const Duration(milliseconds: 2));
    viewModel.addTodo('Todo 2');
    expect(viewModel.todos.length, 2);

    // Complete first todo
    viewModel.toggleTodo(viewModel.todos[0].id);
    expect(viewModel.completedTodos, 1);

    // Act
    viewModel.clearCompleted();

    // Assert
    expect(viewModel.todos.length, 1);
    expect(viewModel.completedTodos, 0);
    expect(viewModel.todos.first.title, 'Todo 2');
  });

  test('should do nothing when no completed todos', () {
    // Arrange
    viewModel.addTodo('Todo 1');
    viewModel.addTodo('Todo 2');
    final originalCount = viewModel.todos.length;

    // Act
    viewModel.clearCompleted();

    // Assert
    expect(viewModel.todos.length, originalCount);
  });
});
```

Testing Error Conditions

```
test('should ignore toggle for non-existent todo', () {
  // Arrange
  viewModel.addTodo('Test Todo');
  final originalCount = viewModel.todos.length;
  final originalCompleted = viewModel.todos.first.isCompleted;

  // Act
  viewModel.toggleTodo('non-existent-id');

  // Assert
  expect(viewModel.todos.length, originalCount);
  expect(viewModel.todos.first.isCompleted, originalCompleted);
});

test('should ignore remove for non-existent todo', () {
  // Arrange
  viewModel.addTodo('Test Todo');
  final originalCount = viewModel.todos.length;

  // Act
  viewModel.deleteTodo('non-existent-id');

  // Assert
  expect(viewModel.todos.length, originalCount);
});
```

Testing Computed Properties

```
group('Computed Properties', () {
  test('should calculate counts correctly', () async {
    // Initial state
    expect(viewModel.totalTodos, 0);
    expect(viewModel.completedTodos, 0);
    expect(viewModel.pendingTodos, 0);
    expect(viewModel.completionPercentage, 0.0);

    // Add todos
    viewModel.addTodo('Todo 1');
    await Future.delayed(const Duration(milliseconds: 2));
    viewModel.addTodo('Todo 2');
    expect(viewModel.totalTodos, 2);
    expect(viewModel.pendingTodos, 2);

    // Complete one
    viewModel.toggleTodo(viewModel.todos[0].id);
    expect(viewModel.completedTodos, 1);
    expect(viewModel.completionPercentage, 50.0);
  });

  test('should handle zero division in completion percentage', () {
    expect(viewModel.completionPercentage, 0.0);
  });

  test('should calculate 100% completion', () {
    viewModel.addTodo('Todo 1');
    viewModel.toggleTodo(viewModel.todos[0].id);

    expect(viewModel.completionPercentage, 100.0);
  });
});
```

Testing ChangeNotifier Behavior

```
group('ChangeNotifier Behavior', () {
  test('should notify listeners on all state changes', () {
    // Arrange
    int notificationCount = 0;
    viewModel.addListener(() => notificationCount++);

    // Act
    viewModel.addTodo('Test Todo');
    final todoId = viewModel.todos.first.id;
    viewModel.toggleTodo(todoId);
    viewModel.deleteTodo(todoId);

    // Assert
    expect(notificationCount, 3); // One for each operation
  });

  test('should be able to remove listeners', () {
    // Arrange
    int notificationCount = 0;
    void listener() => notificationCount++;

    // Act
    viewModel.addListener(listener);
    viewModel.addTodo('Test 1');

    viewModel.removeListener(listener);
    viewModel.addTodo('Test 2');

    // Assert
    expect(notificationCount, 1); // Only first addition counted
  });
});
```

Common Testing Patterns

1. Group Related Tests

```
group('Adding Todos', () {
  // All tests related to adding functionality
});

group('Toggling Todos', () {
  // All tests related to toggling functionality
});

group('Removing Todos', () {
  // All tests related to removing functionality
});

group('edge cases', () {
  // All edge case tests
});
```

✅ **Organizes tests logically and improves readability**

2. Use setUp for Common Initialization

```
group('TodoViewModel Tests', () {
  late TodoViewModel viewModel;
  late TodoService todoService;

  setUp(() {
    todoService = TodoService();
    viewModel = TodoViewModel(todoService); // Fresh instance for each test
  });

  // All tests use the same clean viewModel
});
```

✅ **Ensures test isolation and prevents test interference**

3. Test One Thing at a Time

```
// ✅ GOOD: Tests one specific behavior
test('should trim whitespace from title', () {
  viewModel.addTodo('  Trimmed Title  ');
  expect(viewModel.todos.first.title, 'Trimmed Title');
});

// ❌ BAD: Tests multiple behaviors
test('should add todo and update counts and notify listeners', () {
  // Too many assertions for different behaviors
});
```

4. Use Descriptive Test Names

```
// ✅ GOOD: Clear what the test does
test('should ignore toggle for non-existent todo', () {
  // ...
});

test('should remove correct todo when multiple exist', () {
  // ...
});

// ❌ BAD: Unclear purpose
test('toggle test', () {
  // ...
});
```

💡 **Test names should describe the expected behavior**

Handling Timing Issues

Problem: Unique ID generation based on timestamps

```
// ❌ This might fail if executed too quickly
test('should create unique IDs', () {
  viewModel.addTodo('Todo 1');
  viewModel.addTodo('Todo 2'); // Same timestamp possible!

  expect(viewModel.todos[0].id, isNot(equals(viewModel.todos[1].id)));
});
```

✅ **Solution:** Add small delays for timing-dependent tests

```
test('should create unique IDs for todos', () async {
  viewModel.addTodo('Todo 1');
  await Future.delayed(const Duration(milliseconds: 2));
  viewModel.addTodo('Todo 2');

  expect(viewModel.todos[0].id, isNot(equals(viewModel.todos[1].id)));
});
```

Best Practices Summary

✅ DO:

- Follow Arrange-Act-Assert pattern
- Test one behavior per test
- Use descriptive test names
- Test edge cases and error conditions
- Group related tests
- Use `setUp()` for common initialization
- Test service integration with dependency injection

❌ DON'T:

- Test multiple behaviors in one test
- Rely on test execution order

Testing Benefits in Action

Before tests:

```
void toggleTodo(String id) {
  final todo = _todos.firstWhere((t) => t.id == id); // 💥 Crashes!
  // ...
}
```

After writing tests:

```
void toggleTodo(String id) {
  final index = _todos.indexWhere((todo) => todo.id == id);
  if (index != -1) { // ✅ Safe check
    _todos[index] = _todos[index].copyWith(
      isCompleted: !_todos[index].isCompleted,
    );
    notifyListeners();
  }
}
```

💡 **Tests force you to handle edge cases you might forget!**

Running Tests

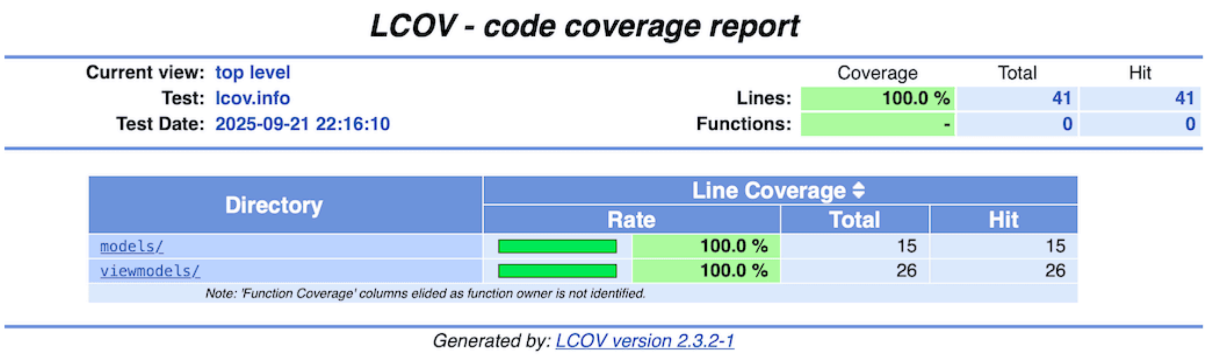
Command line:

```
flutter test test/unit/todo_model_test.dart
flutter test test/unit/todo_viewmodel_test.dart
flutter test test/unit/  # Run all unit tests
```

IDE: Click ▶️ next to test groups or individual tests

Coverage:

```
flutter test test/unit --coverage
genhtml coverage/lcov.info -o coverage/html
open coverage/html/index.html # Mac
```



LCOV - code coverage report

			Coverage	Total	Hit
Current view:	top level				
Test:	lcov.info	Lines:	100.0 %	41	41
Test Date:	2025-09-21 22:16:10	Functions:	-	0	0

Directory	Line Coverage			
	Rate		Total	Hit
models/	🟩	100.0 %	15	15
viewmodels/	🟩	100.0 %	26	26

Note: 'Function Coverage' columns elided as function owner is not identified.

Generated by: LCOV version 2.3.2-1