RADIANT Testing Guide

RADIANT Team

2025-12-28

Contents

RADIANT Testing Guide

Comprehensive guide for testing RADIANT components.

Overview

RADIANT uses a multi-layered testing strategy:

Layer	Tool	Location	Purpose
Unit Tests	Vitest	`**/__tests__/*.test.ts`	Test individual functions/components
Integration Tests	Vitest	`**/__tests__/*.integration.test.ts`	Test service interactions
E2E Tests	Playwright	`apps/admin-dashboard/e2e/`	Test user workflows
Swift Tests	XCTest	`apps/swift-deployer/Tests/`	Test Swift services

Quick Start

```
# Run all tests
pnpm test

# Run with coverage
pnpm test:coverage

# Run E2E tests
cd apps/admin-dashboard && pnpm test:e2e

# Run Swift tests
cd apps/swift-deployer && swift test
```

Unit Testing

Lambda Handler Tests

Tests for Lambda handlers are located in `__tests__/` directories:

```
packages/infrastructure/lambda/
  admin/
    __tests__/
      handler.test.ts
  billing/
    __tests__/
      handler.test.ts
  shared/
    __tests__/
      auth.test.ts
      errors.test.ts
```

services.test.ts

Running Lambda Tests

```
cd packages/infrastructure

# Run all Lambda tests
pnpm test

# Run specific handler
pnpm test -- admin
pnpm test -- billing

# Run with coverage
pnpm test:coverage

# Watch mode
pnpm test:watch
```

Writing Lambda Tests

```
import { describe, it, expect, vi, beforeEach } from 'vitest';
import type { APIGatewayProxyEvent, Context } from 'aws-lambda';

// Mock dependencies
vi.mock('../../shared/db', () => ({
  listTenants: vi.fn(),
  getTenantById: vi.fn(),
}));

import { handler } from '../handler';
import { listTenants, getTenantById } from '../../shared/db';

// Create mock context
const mockContext = {
  awsRequestId: 'test-request-id',
  functionName: 'test-handler',
  // ... other required fields
} as Context;

// Create mock event helper
function createMockEvent(overrides = {}): APIGatewayProxyEvent {
  return {
    httpMethod: 'GET',
    path: '/test',
    headers: { Authorization: 'Bearer test-token' },
    // ... default values
    ...overrides,
```

```
  };
}

describe('Handler', () => {
  beforeEach(() => {
    vi.clearAllMocks();
  });

  it('should return 200 for valid request', async () => {
    (listTenants as ReturnType<typeof vi.fn>).mockResolvedValue([]);

    const event = createMockEvent({ path: '/admin/tenants' });
    const result = await handler(event, mockContext);

    expect(result.statusCode).toBe(200);
  });
});
```

Shared Module Tests

Test shared utilities and services:

```
// packages/infrastructure/lambda/shared/__tests__/errors.test.ts
import { describe, it, expect } from 'vitest';
import {
  ValidationError,
  NotFoundError,
  isOperationalError,
  toAppError,
} from '../errors';

describe('Error Classes', () => {
  it('should create ValidationError with 400 status', () => {
    const error = new ValidationError('Invalid input');

    expect(error.statusCode).toBe(400);
    expect(error.code).toBe('VALIDATION_ERROR');
  });
});
```

E2E Testing (Admin Dashboard)

Setup

```
cd apps/admin-dashboard

# Install Playwright browsers
npx playwright install
```

```
# Run E2E tests
pnpm test:e2e

# Run with UI
pnpm test:e2e:ui

# Run specific test file
pnpm test:e2e -- dashboard.spec.ts
```

Test Structure

```
apps/admin-dashboard/e2e/
  dashboard.spec.ts      # Dashboard navigation tests
  deployment.spec.ts     # Deployment workflow tests
  fixtures/
      test-data.json     # Test fixtures
```

Writing E2E Tests

```typescript
// apps/admin-dashboard/e2e/dashboard.spec.ts
import { test, expect } from '@playwright/test';

test.describe('Dashboard', () => {
  test.beforeEach(async ({ page }) => {
    // Mock authentication
    await page.addInitScript(() => {
      localStorage.setItem('auth_token', 'test-token');
      localStorage.setItem('user', JSON.stringify({
        id: 'test-user',
        email: 'test@example.com',
        role: 'admin',
      }));
    });
  });

  test('should display dashboard home', async ({ page }) => {
    await page.goto('/');
    await expect(page.getByRole('heading', { level: 1 }))
      .toContainText('Dashboard');
  });

  test('should navigate to models page', async ({ page }) => {
    await page.goto('/');
    await page.getByRole('link', { name: 'Models' }).click();
    await expect(page).toHaveURL('/models');
  });
});
```

Swift Testing

Test Structure

```
apps/swift-deployer/Tests/
  RadiantDeployerTests.swift           # Basic unit tests
  RadiantDeployerTests/
      E2ETests/
          DeploymentE2ETests.swift     # E2E workflow tests
      ServiceTests/
          LocalStorageManagerTests.swift
          CredentialServiceTests.swift
```

Running Swift Tests

```bash
cd apps/swift-deployer

# Run all tests
swift test

# Run specific test class
swift test --filter LocalStorageManagerTests

# Run with verbose output
swift test -v

# Generate coverage (requires llvm-cov)
swift test --enable-code-coverage
```

Writing Swift Tests

```swift
import XCTest
@testable import RadiantDeployer

final class LocalStorageManagerTests: XCTestCase {
    var storageManager: LocalStorageManager!

    override func setUp() {
        super.setUp()
        storageManager = LocalStorageManager.shared
    }

    override func tearDown() {
        // Cleanup
        super.tearDown()
    }
```

```
    func testSaveAndLoadConfiguration() async throws {
        // Given
        let key = "test_config"
        let config = TestConfiguration(name: "Test", value: 42)

        // When
        try await storageManager.save(config, forKey: key)
        let loaded: TestConfiguration? = try await storageManager.load(forKey: key)

        // Then
        XCTAssertNotNil(loaded)
        XCTAssertEqual(loaded?.name, "Test")
        XCTAssertEqual(loaded?.value, 42)
    }
}
```

Test Utilities

Mock Factories

Use the shared testing utilities:

```
import {
  createMockTenant,
  createMockUser,
  createMockApiKey,
  createMockChatRequest,
  createMockChatResponse,
  createMockApiGatewayEvent,
  createMockLambdaContext,
} from '@radiant/shared/testing';

// Create mock data
const tenant = createMockTenant({ name: 'Test Corp' });
const user = createMockUser({ tenantId: tenant.id });
const event = createMockApiGatewayEvent({
  httpMethod: 'POST',
  body: JSON.stringify({ model: 'gpt-4o' }),
});
```

Assertion Helpers

```
import {
  assertDefined,
  assertEqual,
  assertMatch,
  assertContains,
  assertThrows,
```

```
  waitFor,
  sleep,
} from '@radiant/shared/testing';

// Custom assertions
assertDefined(result, 'Result should not be null');
assertMatch(response.id, /^chatcmpl_/, 'Invalid response ID format');

// Wait for async condition
await waitFor(() => service.isReady(), { timeout: 5000 });
```

Mocking Guidelines

Database Mocking

```
vi.mock('../db/client', () => ({
  executeStatement: vi.fn(),
}));

import { executeStatement } from '../db/client';

// Mock return value
(executeStatement as ReturnType<typeof vi.fn>).mockResolvedValue({
  rows: [{ id: '123', name: 'Test' }],
});
```

AWS SDK Mocking

```
vi.mock('@aws-sdk/client-s3', () => ({
  S3Client: vi.fn().mockImplementation(() => ({
    send: vi.fn(),
  })),
  PutObjectCommand: vi.fn(),
}));
```

External API Mocking

```
vi.mock('node-fetch', () => ({
  default: vi.fn(),
}));

import fetch from 'node-fetch';

(fetch as ReturnType<typeof vi.fn>).mockResolvedValue({
  ok: true,
  json: () => Promise.resolve({ data: 'mocked' }),
});
```

CI/CD Integration

Tests run automatically in GitHub Actions:

```yaml
# .github/workflows/ci.yml
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Setup Node.js
        uses: actions/setup-node@v4
      - name: Install dependencies
        run: pnpm install
      - name: Run tests
        run: pnpm test
        env:
          DATABASE_URL: postgres://test@localhost:5432/test
```

Coverage Requirements

- Minimum 80% coverage for new code
- Critical paths require 90%+ coverage
- All error handling paths must be tested

Best Practices

Do's

- Test behavior, not implementation
- Use descriptive test names
- Mock external dependencies
- Test error cases and edge cases
- Keep tests fast and isolated
- Use factory functions for test data

Don'ts

- Don't test private methods directly
- Don't share state between tests
- Don't test framework code
- Don't ignore flaky tests
- Don't hardcode test data

Test Naming Convention

```
describe('ServiceName', () => {
  describe('methodName', () => {
    it('should return X when given Y', () => {});
    it('should throw error when invalid input', () => {});
    it('should handle empty array gracefully', () => {});
  });
});
```

Debugging Tests

Vitest

```
# Run with verbose output
pnpm test -- --reporter=verbose

# Run single test
pnpm test -- -t "should return 200"

# Debug mode
node --inspect-brk node_modules/.bin/vitest run
```

Playwright

```
# Debug mode with browser
pnpm test:e2e -- --debug

# Generate trace on failure
pnpm test:e2e -- --trace on

# View trace
npx playwright show-trace trace.zip
```

Swift

```
# Run with verbose output
swift test -v

# Run single test
swift test --filter "testSaveAndLoadConfiguration"
```

See Also

- Contributing Guide
- Error Codes Reference
- API Reference