# SKIDS ADVANCED - COMPREHENSIVE TESTING FRAMEWORK

## Development Team Testing Guide

### \*\*OVERVIEW\*\*

This document provides comprehensive testing procedures for the SKIDS Advanced integration infrastructure, including unit tests, integration tests, end-to-end testing scenarios, and performance benchmarks.

## \*\*QUICK START TESTING\*\*

### \*\*Prerequisites\*\*

# Ensure Node.js 18+ is installed
node --version # Should be 18.0.0 or higher
npm --version # Should be 8.0.0 or higher
# Install dependencies
npm install
# Set up environment variables
cp .env.example .env.local
# Configure your environment variables

### \*\*Run All Tests\*\*

# Run complete test suite
npm run test:all
# Run specific test categories
npm run test:unit # Unit tests only
npm run test:integration # Integration tests only
npm run test:e2e # End-to-end tests only
npm run test:performance # Performance tests only

### \*\*Development Server Testing\*\*

# Start development server
npm run dev
# Run tests against development server
npm run test:dev
# Run tests with coverage
npm run test:coverage

## \*\*TESTING INFRASTRUCTURE SETUP\*\*

### \*\*Test Configuration Files\*\*

#### \*\*Jest Configuration\*\* (`jest.config.js`)

module.exports = {
testEnvironment: 'jsdom',
setupFilesAfterEnv: ['<rootDir>/tests/setup.ts'],
testMatch: [
'<rootDir>/tests/\*\*/\*.test.{js,jsx,ts,tsx}',
'<rootDir>/src/\*\*/\_\_tests\_\_/\*\*/\*.{js,jsx,ts,tsx}'
],
collectCoverageFrom: [
'src/\*\*/\*.{js,jsx,ts,tsx}',
'!src/\*\*/\*.d.ts',
'!src/types/\*\*/\*',
'!src/\*\*/\*.stories.{js,jsx,ts,tsx}'
],
coverageThreshold: {
global: {
branches: 80,
functions: 80,
lines: 80,
statements: 80
}
},
moduleNameMapping: {
'^@/(.\*)$': '<rootDir>/src/$1'
}
}

#### \*\*Playwright Configuration\*\* (`playwright.config.ts`)

import { defineConfig } from '@playwright/test'
export default defineConfig({
testDir: './tests/e2e',
fullyParallel: true,
forbidOnly: !!process.env.CI,
retries: process.env.CI ? 2 : 0,
workers: process.env.CI ? 1 : undefined,
reporter: 'html',
use: {
baseURL: 'http://localhost:3001',
trace: 'on-first-retry',
screenshot: 'only-on-failure'
},
projects: [
{
name: 'chromium',
use: { ...devices['Desktop Chrome'] }
},
{
name: 'firefox',
use: { ...devices['Desktop Firefox'] }
},
{
name: 'webkit',
use: { ...devices['Desktop Safari'] }
}
],
webServer: {
command: 'npm run dev',
url: 'http://localhost:3001',
reuseExistingServer: !process.env.CI
}
})

### \*\*Test Environment Setup\*\* (`tests/setup.ts`)

import '@testing-library/jest-dom'
import { server } from './mocks/server'
// Mock environment variables
process.env.NODE\_ENV = 'test'
process.env.NEXT\_PUBLIC\_APP\_ENV = 'test'
process.env.USE\_MOCK\_PAYMENTS = 'true'
process.env.USE\_MOCK\_AI = 'true'
process.env.USE\_MOCK\_EXTERNAL\_SERVICES = 'true'
// Start MSW server
beforeAll(() => server.listen())
afterEach(() => server.resetHandlers())
afterAll(() => server.close())
// Mock Next.js router
jest.mock('next/router', () => ({
useRouter: () => ({
push: jest.fn(),
pathname: '/',
query: {},
asPath: '/'
})
}))
// Mock Clerk authentication
jest.mock('@clerk/nextjs', () => ({
useAuth: () => ({
isSignedIn: true,
userId: 'test-user-id'
}),
useUser: () => ({
user: {
id: 'test-user-id',
firstName: 'Test',
lastName: 'User',
emailAddresses: [{ emailAddress: 'test@skids.clinic' }],
publicMetadata: { role: 'provider' }
},
isSignedIn: true,
isLoaded: true
})
}))

## \*\*UNIT TESTING\*\*

### \*\*Payment Gateway Tests\*\* (`tests/unit/payment-gateway.test.ts`)

import { MockPaymentGateway, PaymentGatewayFactory } from '@/lib/payment/PaymentGateway'
import { PaymentProvider } from '@/types/payment'
describe('Payment Gateway', () => {
let gateway: MockPaymentGateway
let provider: PaymentProvider
beforeEach(() => {
provider = {
id: 'test-provider',
name: 'Test Provider',
type: 'razorpay',
isActive: true,
supportedCurrencies: ['INR', 'USD'],
supportedCountries: ['IN', 'US'],
features: [],
configuration: {
environment: 'sandbox',
customSettings: {}
},
webhookEndpoints: []
}
gateway = new MockPaymentGateway(provider)
})
describe('Payment Intent Creation', () => {
it('should create payment intent successfully', async () => {
await gateway.initialize()
const request = {
amount: 1000,
currency: 'INR',
providerId: 'test-provider',
metadata: {
userId: 'user-123',
description: 'Test payment'
}
}
const intent = await gateway.createPaymentIntent(request)
expect(intent).toMatchObject({
amount: 1000,
currency: 'INR',
providerId: 'test-provider',
status: 'pending'
})
expect(intent.id).toMatch(/^pi\_mock\_/)
})
it('should validate amount before creating intent', async () => {
await gateway.initialize()
const request = {
amount: -100,
currency: 'INR',
providerId: 'test-provider',
metadata: {
userId: 'user-123',
description: 'Invalid payment'
}
}
await expect(gateway.createPaymentIntent(request))
.rejects.toThrow('Amount must be greater than 0')
})
})
describe('Subscription Management', () => {
it('should create subscription successfully', async () => {
await gateway.initialize()
const request = {
userId: 'user-123',
carePlanId: 'plan-123',
providerId: 'test-provider',
paymentMethodId: 'pm-123',
billing: {
amount: 299,
currency: 'INR',
interval: 'monthly' as const,
intervalCount: 1
},
metadata: {
carePlanName: 'Essential Plan',
features: ['Basic care'],
autoRenewal: true
}
}
const subscription = await gateway.createSubscription(request)
expect(subscription).toMatchObject({
userId: 'user-123',
carePlanId: 'plan-123',
status: 'active',
billing: {
amount: 299,
currency: 'INR',
interval: 'monthly'
}
})
})
})
})

### \*\*Vendor Management Tests\*\* (`tests/unit/vendor-management.test.ts`)

import { vendorAPI } from '@/lib/api/vendor-management'
import { VendorStatus } from '@/types/vendor'
describe('Vendor Management API', () => {
describe('Vendor CRUD Operations', () => {
it('should create vendor successfully', async () => {
const vendorData = {
companyName: 'Test Vendor Inc.',
businessType: 'technology\_platform' as const,
registrationNumber: 'TEST123',
taxId: 'TAX123',
contactInfo: {
primaryContact: {
name: 'John Doe',
title: 'CEO',
email: 'john@testvendor.com',
phone: '+1-555-0123',
preferredContactMethod: 'email' as const
}
}
}
const vendor = await vendorAPI.createVendor(vendorData)
expect(vendor).toMatchObject({
companyName: 'Test Vendor Inc.',
businessType: 'technology\_platform',
status: 'pending\_application'
})
expect(vendor.id).toMatch(/^vendor-/)
})
it('should update vendor status', async () => {
const vendors = await vendorAPI.getVendors()
const vendor = vendors[0]
const updatedVendor = await vendorAPI.updateVendorStatus(
vendor.id,
'approved',
'Vendor meets all requirements'
)
expect(updatedVendor?.status).toBe('approved')
expect(updatedVendor?.onboarding.reviewNotes).toHaveLength(
vendor.onboarding.reviewNotes.length + 1
)
})
})
describe('Onboarding Management', () => {
it('should update onboarding step', async () => {
const vendors = await vendorAPI.getVendors()
const vendor = vendors[0]
const step = vendor.onboarding.pendingSteps[0]
const result = await vendorAPI.updateOnboardingStep(
vendor.id,
step.id,
'completed',
'All documentation verified'
)
expect(result).toBe(true)
const updatedVendor = await vendorAPI.getVendorById(vendor.id)
expect(updatedVendor?.onboarding.completedSteps).toContainEqual(
expect.objectContaining({
id: step.id,
status: 'completed'
})
)
})
})
})

### \*\*ROI Analysis Tests\*\* (`tests/unit/roi-analysis.test.ts`)

import { roiAnalysisEngine } from '@/lib/ai/roi-analysis'
import { Vendor } from '@/types/vendor'
describe('ROI Analysis Engine', () => {
let mockVendor: Vendor
beforeEach(() => {
mockVendor = {
id: 'vendor-test',
companyName: 'Test Vendor',
businessType: 'technology\_platform',
// ... other vendor properties
performance: {
overallRating: 4.5,
metrics: [],
trends: [],
benchmarks: [],
reviews: [],
incidents: [],
lastUpdated: new Date()
},
compliance: {
status: 'compliant',
certifications: [],
audits: [],
policies: [],
training: [],
violations: [],
lastReview: new Date(),
nextReview: new Date()
}
} as Vendor
})
it('should analyze vendor ROI', async () => {
const analysis = await roiAnalysisEngine.analyzeVendorROI(mockVendor)
expect(analysis).toMatchObject({
vendorId: 'vendor-test',
overallROI: expect.any(Number),
financialMetrics: expect.objectContaining({
totalInvestment: expect.any(Number),
totalRevenue: expect.any(Number),
netROI: expect.any(Number)
}),
qualityMetrics: expect.objectContaining({
serviceQualityScore: 4.5,
customerSatisfaction: 4.5
}),
recommendations: expect.arrayContaining([
expect.objectContaining({
type: expect.stringMatching(/optimize|expand|reduce|terminate|renegotiate/),
priority: expect.stringMatching(/low|medium|high|critical/)
})
])
})
})
it('should generate recommendations based on performance', async () => {
// Test low-performing vendor
mockVendor.performance.overallRating = 2.0
const analysis = await roiAnalysisEngine.analyzeVendorROI(mockVendor)
expect(analysis.recommendations).toContainEqual(
expect.objectContaining({
type: 'optimize',
priority: expect.stringMatching(/medium|high/)
})
)
})
})

## � \*\*INTEGRATION TESTING\*\*

### \*\*API Integration Tests\*\* (`tests/integration/api.test.ts`)

import { render, screen, waitFor } from '@testing-library/react'
import userEvent from '@testing-library/user-event'
import VendorManagementPage from '@/app/admin/vendor-management/page'
describe('Vendor Management Integration', () => {
it('should load vendor data and display in dashboard', async () => {
render(<VendorManagementPage />)
// Wait for loading to complete
await waitFor(() => {
expect(screen.queryByText('Loading vendor management...')).not.toBeInTheDocument()
})
// Check if vendor data is displayed
expect(screen.getByText('NutreeAI Technologies')).toBeInTheDocument()
expect(screen.getByText('technology\_platform')).toBeInTheDocument()
})
it('should handle vendor status updates', async () => {
const user = userEvent.setup()
render(<VendorManagementPage />)
await waitFor(() => {
expect(screen.queryByText('Loading vendor management...')).not.toBeInTheDocument()
})
// Find and click approve button for pending vendor
const approveButton = screen.getByText('Start Review')
await user.click(approveButton)
// Verify status change
await waitFor(() => {
expect(screen.getByText('under\_review')).toBeInTheDocument()
})
})
})

### \*\*Payment Flow Integration Tests\*\* (`tests/integration/payment-flow.test.ts`)

import { PaymentGatewayFactory } from '@/lib/payment/PaymentGateway'
import { carePlansAPI } from '@/lib/api/care-plans'
describe('Payment Flow Integration', () => {
it('should complete end-to-end payment flow', async () => {
// Create care plan
const carePlan = await carePlansAPI.createCarePlan({
name: 'Test Plan',
category: 'essential',
description: 'Test care plan',
pricing: {
basePrice: 299,
currency: 'INR',
billingCycle: 'monthly'
}
})
// Create payment gateway
const provider = {
id: 'test-razorpay',
name: 'Razorpay Test',
type: 'razorpay' as const,
isActive: true,
supportedCurrencies: ['INR'],
supportedCountries: ['IN'],
features: [],
configuration: { environment: 'sandbox' as const, customSettings: {} },
webhookEndpoints: []
}
const gateway = await PaymentGatewayFactory.createGateway(provider)
// Create payment intent
const intent = await gateway.createPaymentIntent({
amount: carePlan.pricing.basePrice,
currency: carePlan.pricing.currency,
providerId: provider.id,
metadata: {
carePlanId: carePlan.id,
userId: 'test-user',
description: `Payment for ${carePlan.name}`
}
})
expect(intent.status).toBe('pending')
// Confirm payment
const confirmedIntent = await gateway.confirmPaymentIntent(intent.id)
// Mock payment should succeed 90% of the time
expect(['succeeded', 'failed']).toContain(confirmedIntent.status)
})
})

## � \*\*END-TO-END TESTING\*\*

### \*\*E2E Test Scenarios\*\* (`tests/e2e/vendor-onboarding.spec.ts`)

import { test, expect } from '@playwright/test'
test.describe('Vendor Onboarding Workflow', () => {
test('complete vendor onboarding process', async ({ page }) => {
// Navigate to vendor management
await page.goto('/admin/vendor-management')
// Wait for page to load
await expect(page.getByText('Vendor Management System')).toBeVisible()
// Add new vendor
await page.getByText('Add Vendor').click()
// Fill vendor form (when modal is implemented)
// This would be expanded when the vendor creation modal is built
// Verify vendor appears in list
await expect(page.getByText('NutreeAI Technologies')).toBeVisible()
// Test vendor status workflow
await page.getByText('Start Review').first().click()
await expect(page.getByText('under\_review')).toBeVisible()
// Test approval workflow
await page.getByText('Approve').first().click()
await expect(page.getByText('approved')).toBeVisible()
})
test('analytics dashboard functionality', async ({ page }) => {
await page.goto('/admin/analytics')
// Check dashboard loads
await expect(page.getByText('Unified Analytics Dashboard')).toBeVisible()
// Verify real-time metrics
await expect(page.getByText('Real-time System Metrics')).toBeVisible()
// Test tab navigation
await page.getByText('Vendor Analytics').click()
await expect(page.getByText('Vendor Performance Overview')).toBeVisible()
// Test auto-refresh toggle
await page.getByText('Auto-refresh ON').click()
await expect(page.getByText('Auto-refresh OFF')).toBeVisible()
})
})

### \*\*Performance Testing\*\* (`tests/performance/load-test.spec.ts`)

import { test, expect } from '@playwright/test'
test.describe('Performance Tests', () => {
test('dashboard load performance', async ({ page }) => {
// Start performance monitoring
await page.goto('/admin/analytics')
// Measure page load time
const startTime = Date.now()
await expect(page.getByText('Unified Analytics Dashboard')).toBeVisible()
const loadTime = Date.now() - startTime
// Assert load time is under 3 seconds
expect(loadTime).toBeLessThan(3000)
})
test('vendor management responsiveness', async ({ page }) => {
await page.goto('/admin/vendor-management')
// Test search performance
const searchInput = page.getByPlaceholder('Search vendors...')
await searchInput.fill('NutreeAI')
// Verify search results appear quickly
await expect(page.getByText('NutreeAI Technologies')).toBeVisible({ timeout: 1000 })
})
})

## \*\*PERFORMANCE BENCHMARKS\*\*

### \*\*Performance Requirements\*\*

### \*\*Load Testing Configuration\*\*

// k6 load test script
import http from 'k6/http'
import { check, sleep } from 'k6'
export let options = {
stages: [
{ duration: '2m', target: 10 }, // Ramp up
{ duration: '5m', target: 10 }, // Stay at 10 users
{ duration: '2m', target: 20 }, // Ramp up to 20 users
{ duration: '5m', target: 20 }, // Stay at 20 users
{ duration: '2m', target: 0 }, // Ramp down
],
thresholds: {
http\_req\_duration: ['p(95)<2000'], // 95% of requests under 2s
http\_req\_failed: ['rate<0.1'], // Error rate under 10%
}
}
export default function() {
// Test dashboard load
let response = http.get('http://localhost:3001/admin/analytics')
check(response, {
'status is 200': (r) => r.status === 200,
'response time < 2s': (r) => r.timings.duration < 2000,
})
sleep(1)
// Test vendor management
response = http.get('http://localhost:3001/admin/vendor-management')
check(response, {
'status is 200': (r) => r.status === 200,
'response time < 2s': (r) => r.timings.duration < 2000,
})
sleep(1)
}

## \*\*AUTOMATED TESTING PIPELINE\*\*

### \*\*GitHub Actions Workflow\*\* (`.github/workflows/test.yml`)

name: Test Suite
on:
push:
branches: [ main, develop ]
pull\_request:
branches: [ main ]
jobs:
test:
runs-on: ubuntu-latest
strategy:
matrix:
node-version: [18.x, 20.x]
steps:
- uses: actions/checkout@v3
- name: Use Node.js ${{ matrix.node-version }}
uses: actions/setup-node@v3
with:
node-version: ${{ matrix.node-version }}
cache: 'npm'
- name: Install dependencies
run: npm ci
- name: Run linting
run: npm run lint
- name: Run type checking
run: npm run type-check
- name: Run unit tests
run: npm run test:unit -- --coverage
- name: Run integration tests
run: npm run test:integration
- name: Install Playwright
run: npx playwright install
- name: Run E2E tests
run: npm run test:e2e
- name: Upload coverage reports
uses: codecov/codecov-action@v3
with:
file: ./coverage/lcov.info
- name: Upload test results
uses: actions/upload-artifact@v3
if: always()
with:
name: test-results
path: |
test-results/
coverage/

## \*\*TESTING CHECKLIST\*\*

### \*\*Pre-Deployment Testing\*\*

* • All unit tests passing (90%+ coverage)
* • Integration tests passing
* • E2E tests passing across browsers
* • Performance benchmarks met
* • Security tests passing
* • Accessibility tests passing
* • Mobile responsiveness verified

### \*\*Manual Testing Scenarios\*\*

* • Complete vendor onboarding workflow
* • Staff management operations
* • Analytics dashboard functionality
* • Payment flow testing
* • Error handling and edge cases
* • Browser compatibility testing
* • Mobile device testing

### \*\*Production Readiness\*\*

* • Environment variables configured
* • Database migrations completed
* • Monitoring and alerting set up
* • Backup and recovery tested
* • Security scanning completed
* • Performance monitoring enabled

## \*\*SUPPORT & ESCALATION\*\*

### \*\*Testing Support Contacts\*\*

* • \*\*Lead Developer\*\*: development-lead@skids.clinic
* • \*\*QA Manager\*\*: qa-manager@skids.clinic
* • \*\*DevOps Engineer\*\*: devops@skids.clinic
* • \*\*Product Manager\*\*: product@skids.clinic

### \*\*Issue Escalation Process\*\*

* • \*\*Level 1\*\*: Developer self-resolution (0-2 hours)
* • \*\*Level 2\*\*: Team lead involvement (2-8 hours)
* • \*\*Level 3\*\*: Senior developer/architect (8-24 hours)
* • \*\*Level 4\*\*: External consultant/vendor (24+ hours)

*This testing framework ensures comprehensive coverage of the SKIDS Advanced integration infrastructure. For questions or issues, contact the development team.*