# Testing Strategy

## Testing Pyramid

┌─────────────┐  
 │ E2E (10%) │ Playwright - Critical user flows  
 └─────────────┘  
 ┌───────────────┐  
 │ Integration │ Vitest - Component + DB  
 │ (30%) │  
 └───────────────┘  
 ┌─────────────────┐  
 │ Unit (60%) │ Vitest - Business logic  
 └─────────────────┘

## Unit Tests (Vitest)

// tests/unit/budget-validation.test.ts  
import { describe, it, expect } from 'vitest'  
import { validateBudget } from '@/lib/validation/budget'  
  
describe('validateBudget', () => {  
 it('should return valid for matching totals', () => {  
 const budget = {  
 year\_1: {  
 personnel\_costs: 75000,  
 equipment\_technology: 50000,  
 professional\_services: 25000,  
 training\_development: 10000,  
 materials\_supplies: 5000,  
 other\_costs: 0,  
 total: 165000,  
 },  
 funding\_sources: [  
 { source: 'General Fund', amount: 100000, status: 'secured' },  
 { source: 'EPA Grant', amount: 65000, status: 'pending' },  
 ],  
 }  
  
 const result = validateBudget(budget)  
 expect(result.valid).toBe(true)  
 })  
  
 it('should return error for mismatched totals', () => {  
 const budget = {  
 year\_1: {  
 personnel\_costs: 75000,  
 equipment\_technology: 50000,  
 professional\_services: 25000,  
 training\_development: 10000,  
 materials\_supplies: 5000,  
 other\_costs: 0,  
 total: 200000, // Wrong total  
 },  
 funding\_sources: [  
 { source: 'General Fund', amount: 100000, status: 'secured' },  
 ],  
 }  
  
 const result = validateBudget(budget)  
 expect(result.valid).toBe(false)  
 expect(result.error).toContain('Budget mismatch')  
 })  
})

## Integration Tests

// tests/integration/create-initiative.test.ts  
import { describe, it, expect, beforeAll, afterAll } from 'vitest'  
import { createClient } from '@supabase/supabase-js'  
import { createInitiative } from '@/app/actions/initiatives'  
  
describe('createInitiative', () => {  
 let supabase  
 let testGoalId  
  
 beforeAll(async () => {  
 supabase = createClient(  
 process.env.NEXT\_PUBLIC\_SUPABASE\_URL!,  
 process.env.SUPABASE\_SERVICE\_ROLE\_KEY!  
 )  
  
 // Create test goal  
 const { data: goal } = await supabase  
 .from('strategic\_goals')  
 .insert({ title: 'Test Goal', ... })  
 .select()  
 .single()  
 testGoalId = goal.id  
 })  
  
 afterAll(async () => {  
 // Cleanup  
 await supabase  
 .from('strategic\_goals')  
 .delete()  
 .eq('id', testGoalId)  
 })  
  
 it('should create initiative successfully', async () => {  
 const formData = new FormData()  
 formData.append('strategic\_goal\_id', testGoalId)  
 formData.append('name', 'Test Initiative')  
 formData.append('priority\_level', 'NEED')  
 formData.append('description', 'This is a test initiative description.')  
  
 const result = await createInitiative(formData)  
  
 expect(result.error).toBeUndefined()  
 expect(result.data).toBeDefined()  
 expect(result.data.name).toBe('Test Initiative')  
 })  
})

## End-to-End Tests (Playwright)

// tests/e2e/create-plan.spec.ts  
import { test, expect } from '@playwright/test'  
  
test.describe('Strategic Plan Creation', () => {  
 test.beforeEach(async ({ page }) => {  
 // Login  
 await page.goto('/login')  
 await page.fill('[name="email"]', 'director@carrollton.gov')  
 await page.fill('[name="password"]', 'test-password')  
 await page.click('button[type="submit"]')  
 await page.waitForURL('/dashboard')  
 })  
  
 test('should create new strategic plan', async ({ page }) => {  
 // Navigate to create plan  
 await page.click('text=Create New Plan')  
 await expect(page).toHaveURL(/\/plans\/new/)  
  
 // Fill form  
 await page.fill('[name="title"]', 'FY2026-2028 Strategic Plan')  
 await page.fill('[name="executive\_summary"]', 'This is a test summary.')  
  
 // Submit  
 await page.click('button:has-text("Create Plan")')  
  
 // Verify redirect and success  
 await page.waitForURL(/\/plans\/[a-f0-9-]+/)  
 await expect(page.locator('h1')).toContainText('FY2026-2028 Strategic Plan')  
 })  
})