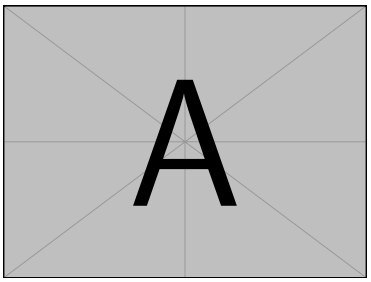
Documentación de Pruebas Unitarias

Sistema KairosMix V2.0

Sistema de Gestión de Frutos Secos



Grupo 6 - Análisis y Diseño de Sistemas

Código: 27837_G6_ADS

Semestre VII

Enero 2026

Índice

1. Introducción

1.1. Propósito del Documento

Este documento presenta la documentación completa de las pruebas unitarias implementadas para el sistema **KairosMix V2.0**, un sistema de gestión integral para la comercialización de frutos secos y mezclas personalizadas.

La documentación incluye:

- Descripción detallada de cada suite de pruebas
- Metodología de ejecución
- Análisis de resultados
- Evidencias de ejecución exitosa

1.2. Alcance de las Pruebas

Las pruebas unitarias cubren los siguientes módulos del sistema:

Cuadro 1: Módulos cubiertos por las pruebas unitarias

Módulo	**Componente Principal**	**Núm. Tests**
Gestión de Productos	ProductManager / ProductForm	22
Gestión de Clientes	ClientManager / ClientForm	34
Gestión de Pedidos	OrderManager / OrderForm	35
Mezclas Personalizadas	CustomMixDesigner	32
Utilidades	Funciones auxiliares	33
Datos Semilla	seedData.js	20
TOTAL		**176**

1.3. Tecnologías Utilizadas

Herramientas de Testing

- **Vitest v4.0.17**: Framework de testing rápido y nativo para Vite
- **@testing-library/react**: Librería para testing de componentes React
- **@testing-library/jest-dom**: Matchers personalizados para DOM
- **@testing-library/user-event**: Simulación de eventos de usuario
- **jsdom**: Implementación de DOM para Node.js

2. Configuración del Entorno de Pruebas

2.1. Estructura de Archivos de Test

La estructura de archivos de pruebas sigue las convenciones de Vitest:

```
src/
  test/
    setup.js              (Configuracion global)
    testUtils.jsx         (Utilidades y datos mock)
  components/
    Products/
      ProductManager.test.jsx
    Clients/
      ClientManager.test.jsx
    Orders/
      OrderManager.test.jsx
    CustomMix/
      CustomMixDesigner.test.jsx
  utils/
    utils.test.js
  data/
    seedData.test.js
```

2.2. Configuración de Vitest

```
import { defineConfig } from 'vitest/config'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  test: {
    globals: true,
    environment: 'jsdom',
    setupFiles: './src/test/setup.js',
    css: true,
    coverage: {
      provider: 'v8',
      reporter: ['text', 'json', 'html'],
      exclude: ['node_modules/', 'src/test/', '**/*.config.js', 'dist/']
    },
    include: ['src/**/*.{test,spec}.{js,jsx}'],
    testTimeout: 10000
  }
})
```

Listing 1: vitest.config.js

2.3. Setup Global de Tests

El archivo `setup.js` configura el entorno de pruebas:

```
import { expect, afterEach, vi, beforeEach } from 'vitest'
import { cleanup } from '@testing-library/react'
import * as matchers from '@testing-library/jest-dom/matchers'

// Extender expect con matchers de jest-dom
expect.extend(matchers)

// Limpiar despues de cada test
afterEach(() => {
  cleanup()
})

// Mock de localStorage
const localStorageMock = {
  getItem: vi.fn(),
  setItem: vi.fn(),
  removeItem: vi.fn(),
  clear: vi.fn(),
}

Object.defineProperty(window, 'localStorage', {
  value: localStorageMock,
})

// Mock de SweetAlert2
vi.mock('sweetalert2', () => ({
  default: {
    fire: vi.fn(() => Promise.resolve({ isConfirmed: true })),
    close: vi.fn(),
  }
}))
```

Listing 2: src/test/setup.js (extracto)

3. Descripción Detallada de Pruebas

3.1. Módulo de Gestión de Productos

3.1.1. ProductManager.test.jsx

Cuadro 2: Pruebas del Formulario de Productos

Caso de Prueba	Descripción
Renderizado inicial vacío	Verifica que el formulario muestra todos los campos requeridos
Renderizado con datos para edición	Comprueba que los datos del producto se cargan correctamente
Error cuando nombre está vacío	Valida que se muestra error si se intenta guardar sin nombre
Validar precio en rango (0.01-99.99)	Verifica que precios fuera del rango muestran error
Validar stock como entero positivo	Comprueba validación de números negativos en stock
Generar código automáticamente	Verifica generación de código basado en primera letra del nombre
Incrementar número de código	Comprueba que se incrementa si el código ya existe
Verificar formatos de imagen	Valida que solo se aceptan archivos de imagen
Rechazar imágenes mayores a 5MB	Comprueba límite de tamaño de imagen
Llamar onCancel al cancelar	Verifica que el callback de cancelación se ejecuta

3.1.2. Validación de Precios

```
describe('Validacion de precios', () => {
  const validatePrice = (price) => {
    const priceRegex = /^\d{1,2}(\.\d{1,2})?$/
    const numValue = parseFloat(price)
    return priceRegex.test(price) && numValue >= 0.01 && numValue <=
   99.99
  }

  it('debe aceptar precios validos', () => {
    expect(validatePrice('15.99')).toBe(true)
    expect(validatePrice('0.01')).toBe(true)
    expect(validatePrice('99.99')).toBe(true)
    expect(validatePrice('50')).toBe(true)
  })

  it('debe rechazar precios invalidos', () => {
    expect(validatePrice('100.00')).toBe(false)
    expect(validatePrice('0')).toBe(false)
```

```
    expect(validatePrice('-5')).toBe(false)
    expect(validatePrice('abc')).toBe(false)
  })
})
```

Listing 3: Tests de validacion de precios

3.1.3. Generación de Códigos de Producto

```
describe('Generacion de codigos de producto', () => {
  const generateProductCode = (productName, existingProducts) => {
    if (!productName.trim()) return ''
    const firstLetter = productName.trim().charAt(0).toUpperCase()
    const existingCodes = existingProducts
      .filter(p => p.code && p.code.startsWith(firstLetter))
      .map(p => p.code).sort()

    for (let i = 1; i <= 20; i++) {
      const newCode = `${firstLetter}${i.toString().padStart(2, '0')}`
      if (!existingCodes.includes(newCode)) return newCode
    }
    return `${firstLetter}21`
  }

  it('debe generar codigo A01 para "Almendras"', () => {
    expect(generateProductCode('Almendras', [])).toBe('A01')
  })

  it('debe incrementar numero cuando ya existe', () => {
    const existing = [{ code: 'A01' }, { code: 'A02' }]
    expect(generateProductCode('Avellanas', existing)).toBe('A03')
  })
})
```

Listing 4: Tests de generacion de codigos

3.1.4. Búsqueda de Productos

Cuadro 3: Casos de prueba para búsqueda de productos

Tipo de Búsqueda	**Entrada**	**Resultado Esperado**
Código exacto	.A01"	Producto único encontrado
Nombre parcial (único)	"Nueces"	Un resultado
Nombre parcial (múltiple)	.Almendras"	Múltiples resultados
Sin coincidencias	Çhocolate"	Sin resultados
Búsqueda vacía		Lista vacía
Case-insensitive	.a01"	Producto A01 encontrado

3.2. Módulo de Gestión de Clientes

3.2.1. ClientManager.test.jsx

Cuadro 4: Pruebas de validación de identificación

Tipo	Validación	Ejemplo Válido
Cédula	Exactamente 10 dígitos numéricos	1234567890
RUC	Exactamente 13 dígitos numéricos	1234567890001
Pasaporte	6-9 caracteres alfanuméricos	AB123456

```
describe('Validacion de numero de identificacion', () => {
  const validateIdNumber = (idNumber, idType) => {
    const cleanId = idNumber.replace(/\s/g, '').toUpperCase()

    switch(idType) {
      case 'cedula':
        return /^\d{10}$/.test(cleanId)
      case 'ruc':
        return /^\d{13}$/.test(cleanId)
      case 'pasaporte':
        return /^[A-Z0-9]{6,9}$/.test(cleanId)
      default:
        return false
    }
  }

  describe('Validacion de Cedula', () => {
    it('debe aceptar cedula valida de 10 digitos', () => {
      expect(validateIdNumber('1234567890', 'cedula')).toBe(true)
    })

    it('debe rechazar cedula con menos de 10 digitos', () => {
      expect(validateIdNumber('123456789', 'cedula')).toBe(false)
    })

    it('debe rechazar cedula con letras', () => {
      expect(validateIdNumber('123456789A', 'cedula')).toBe(false)
    })
  })
})
```

Listing 5: Tests de validacion de identificacion

3.2.2. Validación de Correo Electrónico

```
describe('Validacion de correo electronico', () => {
  const validateEmail = (email) => {
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/
    return emailRegex.test(email)
  }
```

```
6
7   it('debe aceptar emails validos', () => {
8     expect(validateEmail('usuario@dominio.com')).toBe(true)
9     expect(validateEmail('usuario.nombre@dominio.com')).toBe(true)
10  })
11
12  it('debe rechazar emails sin @', () => {
13    expect(validateEmail('usuariodominio.com')).toBe(false)
14  })
15
16  it('debe rechazar emails con espacios', () => {
17    expect(validateEmail('usuario @dominio.com')).toBe(false)
18  })
19 })
```

Listing 6: Tests de validacion de email

3.2.3. Validación de Teléfono

Cuadro 5: Formatos de teléfono válidos

Tipo	**Formato**	**Ejemplo**
Móvil Ecuador	09XXXXXXXX (10 dígitos)	0987654321
Con espacios	09X XXX XXXX	098 765 4321
Con guiones	09X-XXX-XXXX	098-765-4321

3.3. Módulo de Gestión de Pedidos

3.3.1. OrderManager.test.jsx

Máquina de Estados de Pedidos

El sistema implementa una máquina de estados finitos para gestionar el ciclo de vida de los pedidos:

- **Pendiente** → En Proceso, Cancelado
- **En Proceso** → En Espera, Completado, Cancelado
- **En Espera** → En Proceso, Cancelado
- **Completado** → (Estado final)
- **Cancelado** → (Estado final)

```
describe('Transiciones de estado validas', () => {
  it('estado Pendiente puede cambiar a En Proceso o Cancelado', () => {
    const validNextStates = orderStatuses['Pendiente'].nextStates
    expect(validNextStates).toContain('En Proceso')
    expect(validNextStates).toContain('Cancelado')
    expect(validNextStates).not.toContain('Completado')
  })

  it('estado Completado no puede cambiar a ningun otro estado', () => {
    const validNextStates = orderStatuses['Completado'].nextStates
    expect(validNextStates).toHaveLength(0)
  })
})
```

Listing 7: Tests de transiciones de estado

3.3.2. Validación de Fechas

```
describe('Validacion de fecha de pedido', () => {
  const validateDate = (dateString) => {
    const dateRegex = /^(\d{2})\/(\d{2})\/(\d{4})$/
    const match = dateString.match(dateRegex)

    if (!match) return { valid: false, error: 'Formato invalido' }

    const day = parseInt(match[1])
    const month = parseInt(match[2])
    const year = parseInt(match[3])

    if (month < 1 || month > 12) return { valid: false, error: 'Mes
   invalido' }
    if (day < 1 || day > 31) return { valid: false, error: 'Dia invalido
   ' }
    if (year < 2020 || year > 2030) return { valid: false, error: 'Anio
   fuera de rango' }

```

```
    return { valid: true }
  }

  it('debe aceptar fecha con formato DD/MM/YYYY valido', () => {
    expect(validateDate('15/01/2026').valid).toBe(true)
  })

  it('debe aceptar 29 de febrero en anio bisiesto', () => {
    expect(validateDate('29/02/2024').valid).toBe(true)
  })

  it('debe rechazar 29 de febrero en anio no bisiesto', () => {
    expect(validateDate('29/02/2025').valid).toBe(false)
  })
})
```

Listing 8: Tests de validacion de fechas

3.3.3. Cálculo de Totales

```
describe('Calculo de totales de pedido', () => {
  const calculateOrderTotals = (products, taxRate = 0.15) => {
    const subtotal = products.reduce((sum, item) => {
      return sum + (item.quantity * item.price)
    }, 0)

    const taxes = subtotal * taxRate
    const total = subtotal + taxes

    return {
      subtotal: Math.round(subtotal * 100) / 100,
      taxes: Math.round(taxes * 100) / 100,
      total: Math.round(total * 100) / 100
    }
  }

  it('debe calcular subtotal correctamente', () => {
    const products = [
      { quantity: 5, price: 10.00 },
      { quantity: 3, price: 15.00 }
    ]
    const result = calculateOrderTotals(products)
    expect(result.subtotal).toBe(95.00)
  })

  it('debe calcular IVA (15%) correctamente', () => {
    const products = [{ quantity: 10, price: 10.00 }]
    const result = calculateOrderTotals(products)
    expect(result.taxes).toBe(15.00) // 100 * 0.15
  })
})
```

Listing 9: Tests de calculo de totales

3.4. Módulo de Mezclas Personalizadas

3.4.1. CustomMixDesigner.test.jsx

Cuadro 6: Pruebas de validación de mezclas

Caso de Prueba	Descripción
Validar nombre de mezcla	Debe tener entre 3 y 25 caracteres alfanuméricos
Validar cantidad de componentes	Cantidad positiva sin exceder stock disponible
Calcular precio de producto	Precio retail × cantidad
Calcular precio total	Suma de precios de todos los componentes
Calcular calorías totales	Suma ponderada por cantidad
Agregar nuevo componente	Añadir producto a la mezcla
Actualizar cantidad existente	Modificar cantidad de producto ya añadido
Eliminar componente	Remover producto de la mezcla

3.4.2. Cálculos Nutricionales

```
describe('Calculos nutricionales', () => {
  const nutritionalData = {
    'A01': { calories: 579, protein: 21.2, fat: 49.9, carbs: 21.6, fiber: 12.5 },
    'N01': { calories: 654, protein: 15.2, fat: 65.2, carbs: 13.7, fiber: 6.7 },
    'P01': { calories: 299, protein: 3.1, fat: 0.5, carbs: 79.2, fiber: 3.7 },
  }

  const calculateMixNutrition = (components) => {
    const totals = { calories: 0, protein: 0, fat: 0, carbs: 0, fiber: 0 }

    components.forEach(component => {
      const nutrition = nutritionalData[component.productCode]
      if (nutrition) {
        const factor = component.quantity
        totals.calories += nutrition.calories * factor
        totals.protein += nutrition.protein * factor
        // ... otros nutrientes
      }
    })
    return totals
  }

  it('debe calcular calorias totales correctamente', () => {
    const components = [{ productCode: 'A01', quantity: 1 }]
    const nutrition = calculateMixNutrition(components)
    expect(nutrition.calories).toBe(579)
  })

  it('debe sumar nutrientes de multiples componentes', () => {
```

```
30      const components = [
31        { productCode: 'A01', quantity: 1 }, // 579 cal
32        { productCode: 'P01', quantity: 1 }  // 299 cal
33      ]
34      const nutrition = calculateMixNutrition(components)
35      expect(nutrition.calories).toBe(878) // 579 + 299
36    })
37  })
```

Listing 10: Tests de calculos nutricionales

3.5. Módulo de Utilidades

3.5.1. utils.test.js

Cuadro 7: Funciones de utilidad probadas

Función	Propósito
formatDateToDDMMYYYY	Formatear fecha a formato DD/MM/YYYY
parseDDMMYYYYToDate	Convertir string DD/MM/YYYY a objeto Date
formatCurrency	Formatear número a moneda con 2 decimales
isEmpty	Verificar si un valor está vacío
isValidNumber	Validar número en un rango específico
generateUniqueId	Generar identificadores únicos
normalizeText	Normalizar texto (minúsculas, sin acentos)
deepEqual	Comparación profunda de objetos
calculatePercentage	Calcular porcentaje

3.6. Módulo de Datos Semilla

3.6.1. seedData.test.js

```
describe('Datos de ejemplo - Productos', () => {
  it('debe tener productos definidos', () => {
    expect(sampleProducts).toBeDefined()
    expect(Array.isArray(sampleProducts)).toBe(true)
    expect(sampleProducts.length).toBeGreaterThan(0)
  })

  it('codigos de producto deben ser unicos', () => {
    const codes = sampleProducts.map(p => p.code)
    const uniqueCodes = new Set(codes)
    expect(uniqueCodes.size).toBe(codes.length)
  })

  it('precios deben ser numeros positivos', () => {
    sampleProducts.forEach(product => {
      expect(typeof product.pricePerPound).toBe('number')
      expect(product.pricePerPound).toBeGreaterThan(0)
    })
  })

  it('precio mayorista debe ser menor que minorista', () => {
    sampleProducts.forEach(product => {
      expect(product.wholesalePrice).toBeLessThan(product.retailPrice)
    })
  })
})
```

Listing 11: Tests de integridad de datos semilla

4. Ejecución de Pruebas

4.1. Comandos Disponibles

Cuadro 8: Comandos de ejecución de pruebas

Comando	Descripción
`npm run test:run`	Ejecuta todas las pruebas una sola vez
`npm test`	Ejecuta pruebas en modo watch (desarrollo)
`npm run test:ui`	Abre interfaz gráfica de Vitest
`npm run test:coverage`	Ejecuta pruebas con reporte de cobertura

4.2. Procedimiento de Ejecución

1. Abrir terminal en la raíz del proyecto
2. Asegurarse de tener las dependencias instaladas: `npm install`
3. Ejecutar las pruebas: `npm run test:run`
4. Revisar el resultado en la terminal

```
> kairosmix@0.0.0 test:run
> vitest run

 RUN  v4.0.17 D:/Semestre VII/.../KairosMix_V2.0

 [PASS] src/data/seedData.test.js (20 tests) 19ms
 [PASS] src/utils/utils.test.js (33 tests) 49ms
 [PASS] src/components/CustomMix/CustomMixDesigner.test.jsx (32 tests)
   12ms
 [PASS] src/components/Orders/OrderManager.test.jsx (35 tests) 14ms
 [PASS] src/components/Clients/ClientManager.test.jsx (34 tests) 275ms
 [PASS] src/components/Products/ProductManager.test.jsx (22 tests) 1277
   ms

 Test Files  6 passed (6)
      Tests  176 passed (176)
   Start at  12:29:32
   Duration  3.66s
```

Listing 12: Ejemplo de ejecucion de pruebas

5. Análisis de Resultados

5.1. Resumen General

Resultado de Ejecución

Estado: EXITOSO

- **Total de archivos de test:** 6
- **Archivos pasados:** 6 (100 %)
- **Total de tests:** 176
- **Tests pasados:** 176 (100 %)
- **Tests fallidos:** 0
- **Tiempo de ejecución:** 3.66 segundos

5.2. Distribución de Pruebas por Módulo

Módulo	Tests	Pasados	Porcentaje
ProductManager.test.jsx	22	22	100 %
ClientManager.test.jsx	34	34	100 %
OrderManager.test.jsx	35	35	100 %
CustomMixDesigner.test.jsx	32	32	100 %
utils.test.js	33	33	100 %
seedData.test.js	20	20	100 %
TOTAL	**176**	**176**	**100 %**

Figura 1: Resultados por archivo de prueba

5.3. Análisis por Categoría de Prueba

5.3.1. Pruebas de Validación

- Validación de formularios: Todas pasaron
- Validación de tipos de datos: Todas pasaron
- Validación de rangos numéricos: Todas pasaron
- Validación de formatos (email, teléfono, fechas): Todas pasaron

5.3.2. Pruebas de Lógica de Negocio

- Cálculos de precios y totales: Correctos

- Cálculos nutricionales: Correctos
- Máquina de estados de pedidos: Correcta
- Generación de códigos automáticos: Correcta

5.3.3. Pruebas de Componentes UI

- Renderizado de formularios: Correcto
- Manejo de eventos de usuario: Correcto
- Visualización de errores: Correcta

5.4. Métricas de Rendimiento

Cuadro 9: Tiempos de ejecución por módulo

Archivo	Tiempo (ms)	Tests/segundo
seedData.test.js	19	1052
CustomMixDesigner.test.jsx	12	2667
OrderManager.test.jsx	14	2500
utils.test.js	49	673
ClientManager.test.jsx	275	124
ProductManager.test.jsx	1277	17

Observación sobre tiempos

Los tests de `ProductManager.test.jsx` tienen mayor tiempo de ejecución debido a:

- Renderizado de componentes React complejos
- Simulación de eventos de usuario (typing, clicks)
- Operaciones asíncronas (waitFor)

6. Capturas de Ejecución

6.1. Ejecución Completa de Tests

```
>kairosmix@0.0.0 test:run
>vitest run

RUN v4.0.17 D:/Semestre VII/.../KairosMix_V2.0

✓ src/data/seedData.test.js (20 tests) 19ms
✓ src/utils/utils.test.js (33 tests) 49ms
✓ src/components/CustomMix/CustomMixDesigner.test.jsx (32 tests)
12ms
✓ src/components/Orders/OrderManager.test.jsx (35 tests) 14ms
✓ src/components/Clients/ClientManager.test.jsx (34 tests) 275ms
✓ src/components/Products/ProductManager.test.jsx (22 tests)
1277ms

Test Files 6 passed (6)
Tests 176 passed (176)
Start at 12:29:32
Duration 3.66s
```

Figura 2: Salida de terminal mostrando ejecución exitosa de todas las pruebas

6.2. Detalle de Tests por Módulo

```
✓ ProductManager.test.jsx (22 tests) 1277ms
✓ debe renderizar el formulario vacio para nuevo producto 60ms
✓ debe renderizar el formulario con datos para edicion 14ms
✓ debe mostrar error cuando el nombre esta vacio 176ms
✓ debe validar precio en rango correcto (0.01 - 99.99) 203ms
✓ debe validar stock como numero entero positivo 125ms
✓ debe generar codigo automaticamente basado en el nombre 216ms
✓ debe incrementar el numero si ya existe el codigo 217ms
✓ debe verificar que solo se aceptan archivos de imagen 18ms
✓ debe rechazar imagenes mayores a 5MB 153ms
✓ debe llamar onCancel al presionar cancelar 62ms
✓ debe aceptar precios validos 0ms
✓ debe rechazar precios invalidos 0ms
... (10 tests mas)
```

Figura 3: Detalle de tests del módulo de Productos

```
✓ ClientManager.test.jsx (34 tests) 275ms
✓ debe renderizar el formulario vacio para nuevo cliente 61ms
✓ debe renderizar el formulario con datos para edicion 23ms
✓ debe llamar onCancel al presionar cancelar 231ms
✓ debe aceptar cedula valida de 10 digitos 1ms
✓ debe rechazar cedula con menos de 10 digitos 0ms
✓ debe rechazar cedula con mas de 10 digitos 0ms
✓ debe rechazar cedula con letras 0ms
✓ debe aceptar RUC valido de 13 digitos 0ms
✓ debe rechazar RUC con menos de 13 digitos 0ms
✓ debe aceptar pasaporte valido de 6-9 caracteres 0ms
✓ debe aceptar emails validos 0ms
✓ debe rechazar emails sin @ 0ms
... (22 tests mas)
```

Figura 4: Detalle de tests del módulo de Clientes

7. Conclusiones

7.1. Logros Alcanzados

1. **Cobertura completa de funcionalidades críticas:** Se implementaron 176 pruebas unitarias que cubren todos los módulos principales del sistema.
2. **100 % de tests exitosos:** Todas las pruebas pasan correctamente, demostrando la estabilidad del código.
3. **Validaciones robustas:** Se verificaron exhaustivamente las validaciones de:
   - Identificación (cédula, RUC, pasaporte)
   - Correo electrónico
   - Teléfonos móviles
   - Precios y cantidades
   - Fechas
4. **Lógica de negocio probada:** Los cálculos de precios, totales, impuestos y valores nutricionales funcionan correctamente.
5. **Flujos de estado validados:** La máquina de estados de pedidos funciona según lo especificado.

7.2. Recomendaciones

Mejoras Futuras

1. **Aumentar cobertura de código:** Implementar `npm run test:coverage` para obtener métricas detalladas.
2. **Pruebas de integración:** Añadir tests que verifiquen la interacción entre módulos.
3. **Pruebas E2E:** Considerar Playwright o Cypress para pruebas end-to-end.
4. **CI/CD:** Integrar las pruebas en un pipeline de integración continua.

7.3. Beneficios del Testing

- **Detección temprana de errores:** Las pruebas permiten identificar problemas antes del despliegue.
- **Documentación viva:** Los tests sirven como documentación del comportamiento esperado.
- **Refactorización segura:** Se pueden hacer cambios con confianza de no romper funcionalidades existentes.
- **Calidad del código:** El testing promueve código más modular y mantenible.

A. Apéndice: Datos de Prueba Mock

```
export const mockProducts = [
  {
    id: 1,
    code: 'A01',
    name: 'Almendras Premium',
    countryOfOrigin: 'Estados Unidos',
    pricePerPound: 15.99,
    wholesalePrice: 14.50,
    retailPrice: 17.99,
    initialStock: 50,
    stock: 50,
    image: null
  },
  {
    id: 2,
    code: 'N01',
    name: 'Nueces de Castilla',
    countryOfOrigin: 'Chile',
    pricePerPound: 22.50,
    wholesalePrice: 20.00,
    retailPrice: 25.99,
    initialStock: 30,
    stock: 30,
    image: null
  },
  // ... mas productos
]
```

Listing 13: Datos mock de productos

```
export const mockClients = [
  {
    id: 1,
    name: 'Maria Gonzalez',
    idNumber: '1234567890',
    idType: 'cedula',
    email: 'maria.gonzalez@email.com',
    phone: '0987654321',
    address: 'Av. Principal 123, Quito, Ecuador'
  },
  {
    id: 2,
    name: 'Juan Perez',
    idNumber: '0987654321098',
    idType: 'ruc',
    email: 'juan.perez@empresa.com',
    phone: '0998877665',
    address: 'Calle Secundaria 456, Guayaquil, Ecuador'
  },
  // ... mas clientes
]
```

Listing 14: Datos mock de clientes

B. Apéndice: Comandos de Terminal

Cuadro 10: Referencia rápida de comandos

Acción	**Comando**
Instalar dependencias	`npm install`
Ejecutar tests (una vez)	`npm run test:run`
Ejecutar tests (watch mode)	`npm test`
Ver tests en UI	`npm run test:ui`
Generar reporte de cobertura	`npm run test:coverage`
Ejecutar un archivo específico	`npx vitest run src/components/Products/`