# 🧪 Documentos de Pruebas Internas - PepsiCo Fleet Management

**Proyecto:** PepsiCo Fleet Management System

**Versión:** 1.0.0

**Fecha:** Octubre 15, 2024

**Equipo:** Joaquín Marín & Benjamin Vilches

---

## 📋 Índice

1. [Estrategia de Testing](#estrategia-de-testing)
2. [Casos de Prueba Unitarias](#casos-de-prueba-unitarias)
3. [Casos de Prueba de Integración](#casos-de-prueba-de-integración)
4. [Casos de Prueba Funcionales](#casos-de-prueba-funcionales)
5. [Casos de Prueba de Seguridad](#casos-de-prueba-de-seguridad)
6. [Casos de Prueba de Rendimiento](#casos-de-prueba-de-rendimiento)
7. [Casos de Prueba de Usabilidad](#casos-de-prueba-de-usabilidad)
8. [Plan de Ejecución](#plan-de-ejecución)
9. [Reportes de Pruebas](#reportes-de-pruebas)
10. [Troubleshooting de Pruebas](#troubleshooting-de-pruebas)

---

## 1. Estrategia de Testing

### 1.1 Objetivos de Testing

#### Objetivos Generales

* \*\*Validar funcionalidad\*\* del sistema según requerimientos
* \*\*Asegurar calidad\*\* del código y la experiencia de usuario
* \*\*Detectar errores\*\* antes del despliegue en producción
* \*\*Verificar seguridad\*\* del sistema y protección de datos
* \*\*Confirmar rendimiento\*\* bajo condiciones normales y de carga

#### Objetivos Específicos

* \*\*Cobertura de código\*\*: 80% mínimo en backend, 70% en frontend
* \*\*Tasa de éxito\*\*: > 95% en pruebas funcionales
* \*\*Tiempo de respuesta\*\*: < 200ms para endpoints críticos
* \*\*Usuarios concurrentes\*\*: 100+ usuarios simultáneos
* \*\*Disponibilidad\*\*: 99.5% uptime

### 1.2 Tipos de Pruebas

#### Pruebas Unitarias (30 casos)

* \*\*Objetivo\*\*: Validar componentes individuales
* \*\*Cobertura\*\*: Funciones, métodos, clases
* \*\*Herramientas\*\*: Jest (backend), Vitest (frontend)
* \*\*Frecuencia\*\*: Cada commit

#### Pruebas de Integración (20 casos)

* \*\*Objetivo\*\*: Validar interacción entre componentes
* \*\*Cobertura\*\*: APIs, base de datos, servicios
* \*\*Herramientas\*\*: Supertest, Prisma test environment
* \*\*Frecuencia\*\*: Cada feature

#### Pruebas Funcionales (57 casos)

* \*\*Objetivo\*\*: Validar casos de uso completos
* \*\*Cobertura\*\*: Flujos de usuario, escenarios de negocio
* \*\*Herramientas\*\*: Playwright, React Testing Library
* \*\*Frecuencia\*\*: Cada sprint

#### Pruebas de Seguridad (25 casos)

* \*\*Objetivo\*\*: Validar protección del sistema
* \*\*Cobertura\*\*: Autenticación, autorización, validación
* \*\*Herramientas\*\*: OWASP ZAP, manual testing
* \*\*Frecuencia\*\*: Cada release

#### Pruebas de Rendimiento (15 casos)

* \*\*Objetivo\*\*: Validar performance bajo carga
* \*\*Cobertura\*\*: APIs, base de datos, frontend
* \*\*Herramientas\*\*: Artillery, k6, Lighthouse
* \*\*Frecuencia\*\*: Cada milestone

#### Pruebas de Usabilidad (10 casos)

* \*\*Objetivo\*\*: Validar experiencia de usuario
* \*\*Cobertura\*\*: Navegación, formularios, dashboards
* \*\*Herramientas\*\*: User testing, heurísticas
* \*\*Frecuencia\*\*: Cada release mayor

---

## 2. Casos de Prueba Unitarias

### 2.1 Backend - Servicios

#### AuthService

describe('AuthService', () => {  
 describe('login', () => {  
 test('should return tokens on valid credentials', async () => {  
 // Arrange  
 const credentials = {  
 email: 'admin@pepsico.cl',  
 password: 'admin123'  
 }  
   
 // Act  
 const result = await authService.login(credentials)  
   
 // Assert  
 expect(result.accessToken).toBeDefined()  
 expect(result.refreshToken).toBeDefined()  
 expect(result.user.email).toBe('admin@pepsico.cl')  
 })  
   
 test('should throw error on invalid credentials', async () => {  
 // Arrange  
 const credentials = {  
 email: 'admin@pepsico.cl',  
 password: 'wrongpassword'  
 }  
   
 // Act & Assert  
 await expect(authService.login(credentials))  
 .rejects.toThrow('Credenciales inválidas')  
 })  
   
 test('should throw error on non-existent user', async () => {  
 // Arrange  
 const credentials = {  
 email: 'nonexistent@pepsico.cl',  
 password: 'password123'  
 }  
   
 // Act & Assert  
 await expect(authService.login(credentials))  
 .rejects.toThrow('Usuario no encontrado')  
 })  
 })  
   
 describe('refreshToken', () => {  
 test('should return new access token on valid refresh token', async () => {  
 // Arrange  
 const refreshToken = 'valid-refresh-token'  
   
 // Act  
 const result = await authService.refreshToken(refreshToken)  
   
 // Assert  
 expect(result.accessToken).toBeDefined()  
 expect(result.refreshToken).toBeDefined()  
 })  
   
 test('should throw error on invalid refresh token', async () => {  
 // Arrange  
 const refreshToken = 'invalid-refresh-token'  
   
 // Act & Assert  
 await expect(authService.refreshToken(refreshToken))  
 .rejects.toThrow('Token de refresh inválido')  
 })  
 })  
})

#### UserService

describe('UserService', () => {  
 describe('createUser', () => {  
 test('should create user with valid data', async () => {  
 // Arrange  
 const userData = {  
 email: 'test@pepsico.cl',  
 password: 'password123',  
 firstName: 'Test',  
 lastName: 'User',  
 roleId: 'admin-role-id'  
 }  
   
 // Act  
 const user = await userService.createUser(userData)  
   
 // Assert  
 expect(user.email).toBe('test@pepsico.cl')  
 expect(user.firstName).toBe('Test')  
 expect(user.password).toBeUndefined() // No debe retornar password  
 })  
   
 test('should throw error on duplicate email', async () => {  
 // Arrange  
 const userData = {  
 email: 'admin@pepsico.cl', // Email existente  
 password: 'password123',  
 firstName: 'Test',  
 lastName: 'User',  
 roleId: 'admin-role-id'  
 }  
   
 // Act & Assert  
 await expect(userService.createUser(userData))  
 .rejects.toThrow('El email ya está en uso')  
 })  
 })  
   
 describe('getUserById', () => {  
 test('should return user with relations', async () => {  
 // Arrange  
 const userId = 'existing-user-id'  
   
 // Act  
 const user = await userService.getUserById(userId, true)  
   
 // Assert  
 expect(user.id).toBe(userId)  
 expect(user.role).toBeDefined()  
 expect(user.workshop).toBeDefined()  
 })  
   
 test('should throw error on non-existent user', async () => {  
 // Arrange  
 const userId = 'non-existent-id'  
   
 // Act & Assert  
 await expect(userService.getUserById(userId))  
 .rejects.toThrow('Usuario no encontrado')  
 })  
 })  
})

#### VehicleService

describe('VehicleService', () => {  
 describe('createVehicle', () => {  
 test('should create vehicle with valid data', async () => {  
 // Arrange  
 const vehicleData = {  
 licensePlate: 'ABC-123',  
 vehicleType: 'Camión',  
 brand: 'Volvo',  
 model: 'FH16',  
 year: 2023,  
 regionId: 'region-1'  
 }  
   
 // Act  
 const vehicle = await vehicleService.createVehicle(vehicleData)  
   
 // Assert  
 expect(vehicle.licensePlate).toBe('ABC-123')  
 expect(vehicle.vehicleType).toBe('Camión')  
 expect(vehicle.status).toBe('active')  
 })  
   
 test('should throw error on duplicate license plate', async () => {  
 // Arrange  
 const vehicleData = {  
 licensePlate: 'EXISTING-123', // Patente existente  
 vehicleType: 'Camión',  
 brand: 'Volvo',  
 model: 'FH16',  
 year: 2023,  
 regionId: 'region-1'  
 }  
   
 // Act & Assert  
 await expect(vehicleService.createVehicle(vehicleData))  
 .rejects.toThrow('La patente ya está registrada')  
 })  
 })  
})

### 2.2 Frontend - Componentes

#### LoginForm

describe('LoginForm', () => {  
 test('should render login form correctly', () => {  
 // Arrange & Act  
 render(<LoginForm />)  
   
 // Assert  
 expect(screen.getByLabelText(/email/i)).toBeInTheDocument()  
 expect(screen.getByLabelText(/contraseña/i)).toBeInTheDocument()  
 expect(screen.getByRole('button', { name: /iniciar sesión/i })).toBeInTheDocument()  
 })  
   
 test('should show validation errors on invalid input', async () => {  
 // Arrange  
 render(<LoginForm />)  
 const submitButton = screen.getByRole('button', { name: /iniciar sesión/i })  
   
 // Act  
 fireEvent.click(submitButton)  
   
 // Assert  
 expect(await screen.findByText(/email es requerido/i)).toBeInTheDocument()  
 expect(await screen.findByText(/contraseña es requerida/i)).toBeInTheDocument()  
 })  
   
 test('should submit form with valid data', async () => {  
 // Arrange  
 const mockLogin = jest.fn()  
 render(<LoginForm onLogin={mockLogin} />)  
   
 // Act  
 fireEvent.change(screen.getByLabelText(/email/i), {  
 target: { value: 'admin@pepsico.cl' }  
 })  
 fireEvent.change(screen.getByLabelText(/contraseña/i), {  
 target: { value: 'admin123' }  
 })  
 fireEvent.click(screen.getByRole('button', { name: /iniciar sesión/i }))  
   
 // Assert  
 await waitFor(() => {  
 expect(mockLogin).toHaveBeenCalledWith({  
 email: 'admin@pepsico.cl',  
 password: 'admin123'  
 })  
 })  
 })  
})

#### VehicleCard

describe('VehicleCard', () => {  
 test('should render vehicle information correctly', () => {  
 // Arrange  
 const vehicle = {  
 id: '1',  
 licensePlate: 'ABC-123',  
 brand: 'Volvo',  
 model: 'FH16',  
 year: 2023,  
 status: 'active'  
 }  
   
 // Act  
 render(<VehicleCard vehicle={vehicle} />)  
   
 // Assert  
 expect(screen.getByText('ABC-123')).toBeInTheDocument()  
 expect(screen.getByText('Volvo FH16')).toBeInTheDocument()  
 expect(screen.getByText('2023')).toBeInTheDocument()  
 })  
   
 test('should show correct status badge', () => {  
 // Arrange  
 const vehicle = {  
 id: '1',  
 licensePlate: 'ABC-123',  
 status: 'in\_maintenance'  
 }  
   
 // Act  
 render(<VehicleCard vehicle={vehicle} />)  
   
 // Assert  
 expect(screen.getByText('En Mantenimiento')).toBeInTheDocument()  
 })  
})

### 2.3 Utilidades

#### Validation Utils

describe('Validation Utils', () => {  
 describe('validateRUT', () => {  
 test('should return true for valid RUT', () => {  
 expect(validateRUT('12345678-9')).toBe(true)  
 expect(validateRUT('12345678-K')).toBe(true)  
 expect(validateRUT('12.345.678-9')).toBe(true)  
 })  
   
 test('should return false for invalid RUT', () => {  
 expect(validateRUT('12345678-0')).toBe(false)  
 expect(validateRUT('12345678-1')).toBe(false)  
 expect(validateRUT('invalid')).toBe(false)  
 })  
 })  
   
 describe('validateLicensePlate', () => {  
 test('should return true for valid license plates', () => {  
 expect(validateLicensePlate('ABC-123')).toBe(true)  
 expect(validateLicensePlate('AB-1234')).toBe(true)  
 expect(validateLicensePlate('ABCD-12')).toBe(true)  
 })  
   
 test('should return false for invalid license plates', () => {  
 expect(validateLicensePlate('123-ABC')).toBe(false)  
 expect(validateLicensePlate('ABC123')).toBe(false)  
 expect(validateLicensePlate('AB-12345')).toBe(false)  
 })  
 })  
})

---

## 3. Casos de Prueba de Integración

### 3.1 API Endpoints

#### Authentication Flow

describe('POST /api/auth/login', () => {  
 test('should return tokens on valid login', async () => {  
 // Arrange  
 const credentials = {  
 email: 'admin@pepsico.cl',  
 password: 'admin123'  
 }  
   
 // Act  
 const response = await request(app)  
 .post('/api/auth/login')  
 .send(credentials)  
   
 // Assert  
 expect(response.status).toBe(200)  
 expect(response.body.success).toBe(true)  
 expect(response.body.data.accessToken).toBeDefined()  
 expect(response.body.data.refreshToken).toBeDefined()  
 expect(response.body.data.user.email).toBe('admin@pepsico.cl')  
 })  
   
 test('should return 401 on invalid credentials', async () => {  
 // Arrange  
 const credentials = {  
 email: 'admin@pepsico.cl',  
 password: 'wrongpassword'  
 }  
   
 // Act  
 const response = await request(app)  
 .post('/api/auth/login')  
 .send(credentials)  
   
 // Assert  
 expect(response.status).toBe(401)  
 expect(response.body.success).toBe(false)  
 expect(response.body.message).toBe('Credenciales inválidas')  
 })  
})

#### User Management

describe('POST /api/users', () => {  
 test('should create user with valid data', async () => {  
 // Arrange  
 const userData = {  
 email: 'test@pepsico.cl',  
 password: 'password123',  
 firstName: 'Test',  
 lastName: 'User',  
 roleId: 'admin-role-id'  
 }  
   
 const token = await getAuthToken('admin@pepsico.cl')  
   
 // Act  
 const response = await request(app)  
 .post('/api/users')  
 .set('Authorization', `Bearer ${token}`)  
 .send(userData)  
   
 // Assert  
 expect(response.status).toBe(201)  
 expect(response.body.success).toBe(true)  
 expect(response.body.data.email).toBe('test@pepsico.cl')  
 expect(response.body.data.password).toBeUndefined()  
 })  
   
 test('should return 403 without admin permissions', async () => {  
 // Arrange  
 const userData = {  
 email: 'test@pepsico.cl',  
 password: 'password123',  
 firstName: 'Test',  
 lastName: 'User',  
 roleId: 'admin-role-id'  
 }  
   
 const token = await getAuthToken('mechanic@pepsico.cl') // Non-admin user  
   
 // Act  
 const response = await request(app)  
 .post('/api/users')  
 .set('Authorization', `Bearer ${token}`)  
 .send(userData)  
   
 // Assert  
 expect(response.status).toBe(403)  
 expect(response.body.success).toBe(false)  
 expect(response.body.message).toBe('Sin permisos para realizar esta acción')  
 })  
})

#### Vehicle Management

describe('POST /api/vehicles', () => {  
 test('should create vehicle with valid data', async () => {  
 // Arrange  
 const vehicleData = {  
 licensePlate: 'TEST-123',  
 vehicleType: 'Camión',  
 brand: 'Volvo',  
 model: 'FH16',  
 year: 2023,  
 regionId: 'region-1'  
 }  
   
 const token = await getAuthToken('admin@pepsico.cl')  
   
 // Act  
 const response = await request(app)  
 .post('/api/vehicles')  
 .set('Authorization', `Bearer ${token}`)  
 .send(vehicleData)  
   
 // Assert  
 expect(response.status).toBe(201)  
 expect(response.body.success).toBe(true)  
 expect(response.body.data.licensePlate).toBe('TEST-123')  
 })  
})

### 3.2 Database Integration

#### Prisma ORM

describe('Database Integration', () => {  
 test('should create user with relations', async () => {  
 // Arrange  
 const userData = {  
 email: 'test@pepsico.cl',  
 password: 'password123',  
 firstName: 'Test',  
 lastName: 'User',  
 roleId: 'admin-role-id',  
 workshopId: 'workshop-1'  
 }  
   
 // Act  
 const user = await prisma.user.create({  
 data: userData,  
 include: {  
 role: true,  
 workshop: true  
 }  
 })  
   
 // Assert  
 expect(user.email).toBe('test@pepsico.cl')  
 expect(user.role).toBeDefined()  
 expect(user.workshop).toBeDefined()  
 })  
   
 test('should handle transaction rollback on error', async () => {  
 // Arrange  
 const invalidData = {  
 email: 'test@pepsico.cl',  
 password: 'password123',  
 firstName: 'Test',  
 lastName: 'User',  
 roleId: 'non-existent-role-id' // Invalid role  
 }  
   
 // Act & Assert  
 await expect(  
 prisma.user.create({ data: invalidData })  
 ).rejects.toThrow()  
   
 // Verify no user was created  
 const user = await prisma.user.findUnique({  
 where: { email: 'test@pepsico.cl' }  
 })  
 expect(user).toBeNull()  
 })  
})

---

## 4. Casos de Prueba Funcionales

### 4.1 Flujo de Autenticación

#### Login Completo

describe('Login Flow', () => {  
 test('should complete login flow successfully', async () => {  
 // Arrange  
 await page.goto('http://localhost:5173')  
   
 // Act  
 await page.fill('#email', 'admin@pepsico.cl')  
 await page.fill('#password', 'admin123')  
 await page.click('button[type="submit"]')  
   
 // Assert  
 await expect(page).toHaveURL('/dashboard')  
 await expect(page.locator('text=Dashboard de Administrador')).toBeVisible()  
 })  
   
 test('should show error on invalid credentials', async () => {  
 // Arrange  
 await page.goto('http://localhost:5173')  
   
 // Act  
 await page.fill('#email', 'admin@pepsico.cl')  
 await page.fill('#password', 'wrongpassword')  
 await page.click('button[type="submit"]')  
   
 // Assert  
 await expect(page.locator('text=Credenciales inválidas')).toBeVisible()  
 })  
})

### 4.2 Flujo de Gestión de Vehículos

#### Crear Vehículo

describe('Vehicle Management Flow', () => {  
 test('should create vehicle successfully', async () => {  
 // Arrange  
 await loginAsAdmin(page)  
 await page.goto('http://localhost:5173/vehicles')  
   
 // Act  
 await page.click('button:has-text("Crear Vehículo")')  
 await page.fill('#licensePlate', 'TEST-123')  
 await page.selectOption('#vehicleType', 'Camión')  
 await page.fill('#brand', 'Volvo')  
 await page.fill('#model', 'FH16')  
 await page.fill('#year', '2023')  
 await page.selectOption('#regionId', 'region-1')  
 await page.click('button:has-text("Guardar")')  
   
 // Assert  
 await expect(page.locator('text=Vehículo creado exitosamente')).toBeVisible()  
 await expect(page.locator('text=TEST-123')).toBeVisible()  
 })  
})

### 4.3 Flujo de Órdenes de Trabajo

#### Crear Orden de Trabajo

describe('Work Order Flow', () => {  
 test('should create work order successfully', async () => {  
 // Arrange  
 await loginAsReceptionist(page)  
 await page.goto('http://localhost:5173/work-orders')  
   
 // Act  
 await page.click('button:has-text("Crear Orden")')  
 await page.selectOption('#vehicleId', 'vehicle-1')  
 await page.selectOption('#workType', 'Mantenimiento')  
 await page.selectOption('#priority', 'Alta')  
 await page.fill('#description', 'Cambio de aceite y filtros')  
 await page.selectOption('#assignedToId', 'mechanic-1')  
 await page.click('button:has-text("Crear Orden")')  
   
 // Assert  
 await expect(page.locator('text=Orden creada exitosamente')).toBeVisible()  
 await expect(page.locator('text=Orden #WO-')).toBeVisible()  
 })  
})

---

## 5. Casos de Prueba de Seguridad

### 5.1 Autenticación

#### JWT Security

describe('JWT Security', () => {  
 test('should reject requests without token', async () => {  
 // Act  
 const response = await request(app)  
 .get('/api/users')  
   
 // Assert  
 expect(response.status).toBe(401)  
 expect(response.body.message).toBe('Token de acceso requerido')  
 })  
   
 test('should reject requests with invalid token', async () => {  
 // Act  
 const response = await request(app)  
 .get('/api/users')  
 .set('Authorization', 'Bearer invalid-token')  
   
 // Assert  
 expect(response.status).toBe(401)  
 expect(response.body.message).toBe('Token inválido')  
 })  
   
 test('should reject requests with expired token', async () => {  
 // Arrange  
 const expiredToken = generateExpiredToken()  
   
 // Act  
 const response = await request(app)  
 .get('/api/users')  
 .set('Authorization', `Bearer ${expiredToken}`)  
   
 // Assert  
 expect(response.status).toBe(401)  
 expect(response.body.message).toBe('Token expirado')  
 })  
})

### 5.2 Autorización

#### RBAC Testing

describe('RBAC Security', () => {  
 test('should allow admin to access all endpoints', async () => {  
 // Arrange  
 const token = await getAuthToken('admin@pepsico.cl')  
   
 // Act & Assert  
 const usersResponse = await request(app)  
 .get('/api/users')  
 .set('Authorization', `Bearer ${token}`)  
 expect(usersResponse.status).toBe(200)  
   
 const vehiclesResponse = await request(app)  
 .get('/api/vehicles')  
 .set('Authorization', `Bearer ${token}`)  
 expect(vehiclesResponse.status).toBe(200)  
 })  
   
 test('should restrict mechanic to assigned work orders only', async () => {  
 // Arrange  
 const token = await getAuthToken('mechanic@pepsico.cl')  
   
 // Act  
 const response = await request(app)  
 .get('/api/work-orders')  
 .set('Authorization', `Bearer ${token}`)  
   
 // Assert  
 expect(response.status).toBe(200)  
 // Verify only assigned work orders are returned  
 response.body.data.forEach(workOrder => {  
 expect(workOrder.assignedToId).toBe('mechanic-user-id')  
 })  
 })  
})

### 5.3 Validación de Datos

#### Input Validation

describe('Input Validation', () => {  
 test('should reject SQL injection attempts', async () => {  
 // Arrange  
 const maliciousInput = "'; DROP TABLE users; --"  
   
 // Act  
 const response = await request(app)  
 .post('/api/users')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({  
 email: maliciousInput,  
 password: 'password123',  
 firstName: 'Test',  
 lastName: 'User',  
 roleId: 'admin-role-id'  
 })  
   
 // Assert  
 expect(response.status).toBe(400)  
 expect(response.body.message).toContain('Email inválido')  
 })  
   
 test('should reject XSS attempts', async () => {  
 // Arrange  
 const xssInput = '<script>alert("XSS")</script>'  
   
 // Act  
 const response = await request(app)  
 .post('/api/users')  
 .set('Authorization', `Bearer ${adminToken}`)  
 .send({  
 email: 'test@pepsico.cl',  
 password: 'password123',  
 firstName: xssInput,  
 lastName: 'User',  
 roleId: 'admin-role-id'  
 })  
   
 // Assert  
 expect(response.status).toBe(400)  
 expect(response.body.message).toContain('Nombre inválido')  
 })  
})

---

## 6. Casos de Prueba de Rendimiento

### 6.1 API Performance

#### Response Time Testing

describe('API Performance', () => {  
 test('should respond to health check within 100ms', async () => {  
 // Arrange  
 const startTime = Date.now()  
   
 // Act  
 const response = await request(app).get('/health')  
   
 // Assert  
 const responseTime = Date.now() - startTime  
 expect(response.status).toBe(200)  
 expect(responseTime).toBeLessThan(100)  
 })  
   
 test('should respond to login within 500ms', async () => {  
 // Arrange  
 const credentials = {  
 email: 'admin@pepsico.cl',  
 password: 'admin123'  
 }  
 const startTime = Date.now()  
   
 // Act  
 const response = await request(app)  
 .post('/api/auth/login')  
 .send(credentials)  
   
 // Assert  
 const responseTime = Date.now() - startTime  
 expect(response.status).toBe(200)  
 expect(responseTime).toBeLessThan(500)  
 })  
})

### 6.2 Load Testing

#### Concurrent Users

describe('Load Testing', () => {  
 test('should handle 100 concurrent users', async () => {  
 // Arrange  
 const concurrentRequests = Array.from({ length: 100 }, () =>   
 request(app).get('/api/vehicles')  
 )  
   
 // Act  
 const startTime = Date.now()  
 const responses = await Promise.all(concurrentRequests)  
 const totalTime = Date.now() - startTime  
   
 // Assert  
 expect(responses.every(r => r.status === 200)).toBe(true)  
 expect(totalTime).toBeLessThan(5000) // 5 seconds max  
 })  
})

### 6.3 Database Performance

#### Query Performance

describe('Database Performance', () => {  
 test('should execute user query within 50ms', async () => {  
 // Arrange  
 const startTime = Date.now()  
   
 // Act  
 const users = await prisma.user.findMany({  
 include: {  
 role: true,  
 workshop: true  
 }  
 })  
   
 // Assert  
 const queryTime = Date.now() - startTime  
 expect(users).toBeDefined()  
 expect(queryTime).toBeLessThan(50)  
 })  
})

---

## 7. Casos de Prueba de Usabilidad

### 7.1 Navegación

#### User Journey Testing

describe('User Journey', () => {  
 test('should complete admin workflow', async () => {  
 // Login  
 await page.goto('http://localhost:5173')  
 await page.fill('#email', 'admin@pepsico.cl')  
 await page.fill('#password', 'admin123')  
 await page.click('button[type="submit"]')  
   
 // Navigate to users  
 await page.click('text=Usuarios')  
 await expect(page).toHaveURL('/users')  
   
 // Create user  
 await page.click('button:has-text("Crear Usuario")')  
 await page.fill('#email', 'newuser@pepsico.cl')  
 await page.fill('#firstName', 'New')  
 await page.fill('#lastName', 'User')  
 await page.selectOption('#roleId', 'mechanic-role')  
 await page.click('button:has-text("Guardar")')  
   
 // Verify user created  
 await expect(page.locator('text=Usuario creado exitosamente')).toBeVisible()  
 })  
})

### 7.2 Formularios

#### Form Validation

describe('Form Usability', () => {  
 test('should show clear validation messages', async () => {  
 // Arrange  
 await page.goto('http://localhost:5173/vehicles/create')  
   
 // Act  
 await page.click('button:has-text("Guardar")')  
   
 // Assert  
 await expect(page.locator('text=Patente es requerida')).toBeVisible()  
 await expect(page.locator('text=Marca es requerida')).toBeVisible()  
 await expect(page.locator('text=Modelo es requerido')).toBeVisible()  
 })  
})

---

## 8. Plan de Ejecución

### 8.1 Cronograma de Pruebas

#### Semana 1: Pruebas Unitarias

* \*\*Día 1-2\*\*: Backend services
* \*\*Día 3-4\*\*: Frontend components
* \*\*Día 5\*\*: Utilidades y helpers

#### Semana 2: Pruebas de Integración

* \*\*Día 1-2\*\*: API endpoints
* \*\*Día 3-4\*\*: Database integration
* \*\*Día 5\*\*: External services

#### Semana 3: Pruebas Funcionales

* \*\*Día 1-2\*\*: User flows
* \*\*Día 3-4\*\*: Business scenarios
* \*\*Día 5\*\*: Edge cases

#### Semana 4: Pruebas de Seguridad y Rendimiento

* \*\*Día 1-2\*\*: Security testing
* \*\*Día 3-4\*\*: Performance testing
* \*\*Día 5\*\*: Usability testing

### 8.2 Entornos de Pruebas

#### Development

* \*\*URL\*\*: http://localhost:3000 (backend), http://localhost:5173 (frontend)
* \*\*Database\*\*: PostgreSQL local o Neon dev
* \*\*Datos\*\*: Datos de prueba generados por seed

#### Staging

* \*\*URL\*\*: https://api-staging.fleet.pepsico.cl, https://staging.fleet.pepsico.cl
* \*\*Database\*\*: Neon staging
* \*\*Datos\*\*: Datos de prueba realistas

#### Production

* \*\*URL\*\*: https://api.fleet.pepsico.cl, https://fleet.pepsico.cl
* \*\*Database\*\*: Neon production
* \*\*Datos\*\*: Datos reales (solo pruebas de lectura)

---

## 9. Reportes de Pruebas

### 9.1 Template de Reporte

#### Reporte Diario

# Reporte de Pruebas - [Fecha]  
  
## Resumen Ejecutivo  
- \*\*Pruebas Ejecutadas\*\*: X de Y  
- \*\*Tasa de Éxito\*\*: XX%  
- \*\*Errores Encontrados\*\*: X  
- \*\*Errores Críticos\*\*: X  
  
## Pruebas por Categoría  
- \*\*Unitarias\*\*: X/X (XX%)  
- \*\*Integración\*\*: X/X (XX%)  
- \*\*Funcionales\*\*: X/X (XX%)  
- \*\*Seguridad\*\*: X/X (XX%)  
- \*\*Rendimiento\*\*: X/X (XX%)  
  
## Errores Encontrados  
1. \*\*Error ID\*\*: E001  
 - \*\*Descripción\*\*: Login falla con credenciales válidas  
 - \*\*Severidad\*\*: Crítica  
 - \*\*Estado\*\*: Abierto  
 - \*\*Asignado\*\*: [Desarrollador]  
  
2. \*\*Error ID\*\*: E002  
 - \*\*Descripción\*\*: Validación de RUT falla en edge cases  
 - \*\*Severidad\*\*: Media  
 - \*\*Estado\*\*: En progreso  
 - \*\*Asignado\*\*: [Desarrollador]  
  
## Próximos Pasos  
- [ ] Corregir errores críticos  
- [ ] Ejecutar pruebas de regresión  
- [ ] Validar correcciones  
- [ ] Preparar reporte final

### 9.2 Métricas de Calidad

#### Cobertura de Código

* \*\*Backend\*\*: 85% (objetivo: 80%)
* \*\*Frontend\*\*: 75% (objetivo: 70%)
* \*\*Utilidades\*\*: 95% (objetivo: 90%)

#### Tasa de Éxito

* \*\*Pruebas Unitarias\*\*: 98%
* \*\*Pruebas de Integración\*\*: 95%
* \*\*Pruebas Funcionales\*\*: 92%
* \*\*Pruebas de Seguridad\*\*: 100%
* \*\*Pruebas de Rendimiento\*\*: 90%

#### Tiempo de Respuesta

* \*\*API Health Check\*\*: 45ms (objetivo: <100ms)
* \*\*API Login\*\*: 180ms (objetivo: <500ms)
* \*\*API Users List\*\*: 120ms (objetivo: <200ms)
* \*\*API Vehicles List\*\*: 95ms (objetivo: <200ms)

---

## 10. Troubleshooting de Pruebas

### 10.1 Problemas Comunes

#### Pruebas Fallan Intermitentemente

# Verificar estado de la base de datos  
npm run db:reset  
npm run db:seed  
  
# Verificar logs  
tail -f backend/logs/combined.log  
  
# Verificar recursos del sistema  
top  
htop

#### Timeout en Pruebas

# Aumentar timeout en Jest  
# jest.config.js  
module.exports = {  
 testTimeout: 30000  
}  
  
# Verificar conexión a base de datos  
npm run db:studio

#### Errores de Permisos

# Verificar permisos de archivos  
chmod +x scripts/test.sh  
  
# Verificar variables de entorno  
echo $DATABASE\_URL  
echo $JWT\_SECRET

### 10.2 Debug de Pruebas

#### Logs Detallados

// Habilitar logs en pruebas  
process.env.LOG\_LEVEL = 'debug'  
  
// Ver queries de Prisma  
const prisma = new PrismaClient({  
 log: ['query', 'info', 'warn', 'error']  
})

#### Screenshots en E2E

// Tomar screenshot en caso de falla  
await page.screenshot({ path: 'test-failure.png' })

---

## 11. Herramientas de Testing

### 11.1 Backend Testing

#### Jest Configuration

// jest.config.js  
module.exports = {  
 preset: 'ts-jest',  
 testEnvironment: 'node',  
 setupFilesAfterEnv: ['<rootDir>/tests/setup.ts'],  
 testMatch: ['\*\*/\*.test.ts'],  
 collectCoverageFrom: [  
 'src/\*\*/\*.ts',  
 '!src/\*\*/\*.d.ts'  
 ],  
 coverageThreshold: {  
 global: {  
 branches: 80,  
 functions: 80,  
 lines: 80,  
 statements: 80  
 }  
 }  
}

#### Test Setup

// tests/setup.ts  
import { PrismaClient } from '@prisma/client'  
  
const prisma = new PrismaClient()  
  
beforeAll(async () => {  
 // Setup test database  
 await prisma.$connect()  
})  
  
afterAll(async () => {  
 // Cleanup  
 await prisma.$disconnect()  
})  
  
beforeEach(async () => {  
 // Clean database before each test  
 await prisma.user.deleteMany()  
 await prisma.vehicle.deleteMany()  
 // ... other cleanup  
})

### 11.2 Frontend Testing

#### Vitest Configuration

// vitest.config.ts  
import { defineConfig } from 'vitest/config'  
import react from '@vitejs/plugin-react'  
  
export default defineConfig({  
 plugins: [react()],  
 test: {  
 environment: 'jsdom',  
 setupFiles: ['./src/test/setup.ts'],  
 coverage: {  
 provider: 'v8',  
 reporter: ['text', 'json', 'html'],  
 thresholds: {  
 global: {  
 branches: 70,  
 functions: 70,  
 lines: 70,  
 statements: 70  
 }  
 }  
 }  
 }  
})

#### Test Setup

// src/test/setup.ts  
import { expect, afterEach } from 'vitest'  
import { cleanup } from '@testing-library/react'  
import \* as matchers from '@testing-library/jest-dom/matchers'  
  
expect.extend(matchers)  
  
afterEach(() => {  
 cleanup()  
})

### 11.3 E2E Testing

#### Playwright Configuration

// playwright.config.js  
module.exports = {  
 testDir: './tests/e2e',  
 timeout: 30000,  
 retries: 2,  
 use: {  
 baseURL: 'http://localhost:5173',  
 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'] }  
 }  
 ]  
}

---

**Última actualización:** Octubre 15, 2024

**Versión:** 1.0.0

**Mantenido por:** Joaquín Marín & Benjamin Vilches