PRUEBAS ENDPOINTS - Documento 5: Errores, Casos Límite y Seguridad

Descripción

Este documento contiene pruebas para **casos límite, manejo de errores, validaciones de seguridad y escenarios excepcionales**.

---

1. Validaciones de Entrada - Ataques y Edge Cases

1.1 Prueba - SQL Injection en Consecutivo

Request (intento de inyección):

```
GET /api/facturas/00100010010000000001; DROP TABLE invoices;--
```

Response (400 Bad Request):

```
{
  "success": false,
  "error": "Formato de consecutivo inválido",
  "details": {
    "provided": "00100010010000000001; DROP TABLE invoices;--",
    "expected": "20 dígitos numéricos solamente"
  }
}
```

Validaciones esperadas:

- ✅ Rechazo completo del valor
- ✅ Validación strict antes de procesar
- ✅ No ejecuta operaciones en BD

1.2 Prueba - XSS en Campo de Texto

Request:

```
POST /api/facturas/emitir
Content-Type: application/json

{
  "emisor": {
    "nombre": "<script>alert('XSS')</script>",
    "identificacion": "3101234567"
  },
  "receptor": {
    "nombre": "Cliente",
    "identificacion": "123456789"
```

```
  },
  "detalleServicio": [
    {
      "numeroLinea": 1,
      "detalle": "<img src=x onerror=alert('XSS')>",
      "cantidad": 1,
      "precioUnitario": 100,
      "montoTotalLinea": 100,
      "impuestos": []
    }
  ],
  "resumenFactura": {
    "totalVenta": 100,
    "totalComprobante": 100,
    "codigoMoneda": "CRC"
  }
}
```

Response (400 Bad Request):

```
{
  "success": false,
  "error": "Datos de factura inválidos",
  "details": [
    {
      "field": "emisor.nombre",
      "message": "El nombre no puede contener caracteres especiales de script"
    },
    {
      "field": "detalleServicio[0].detalle",
      "message": "El detalle contiene caracteres potencialmente peligrosos"
    }
  ]
}
```

Validaciones esperadas:

- ✅ Rechazo de caracteres peligrosos
- ✅ Sanitización de entrada
- ✅ Prevención de XSS

1.3 Prueba - Buffer Overflow - Nombre Muy Largo

Request:

```
POST /api/facturas/emitir
Content-Type: application/json

{
  "emisor": {
    "nombre": "AAAAAAA...AAAAAAA[5000 caracteres]",
```

```
    "identificacion": "3101234567"
  },
  "receptor": {
    "nombre": "Cliente",
    "identificacion": "123456789"
  },
  "detalleServicio": [],
  "resumenFactura": {
    "totalVenta": 0,
    "totalComprobante": 0,
    "codigoMoneda": "CRC"
  }
}
```

Response (400 Bad Request):

```
{
  "success": false,
  "error": "Datos de factura inválidos",
  "details": [
    {
      "field": "emisor.nombre",
      "message": "El nombre no puede exceder 100 caracteres"
    }
  ]
}
```

Validaciones esperadas:

- ✅ Límite de caracteres aplicado
- ✅ No crash de servidor
- ✅ Error descriptivo

1.4 Prueba - JSON Payload Malformado

Request:

```
POST /api/facturas/emitir
Content-Type: application/json

{
  "emisor": {
    "nombre": "Test",
    "identificacion": "3101234567"
  },
  "receptor": {
    "nombre": "Cliente"
    // Falta coma aquí - JSON inválido
    "identificacion": "123456789"
```

```
  }
}
```

Response (400 Bad Request):

```
{
  "success": false,
  "error": "JSON inválido",
  "message": "El payload contiene JSON malformado",
  "details": {
    "line": 7,
    "near": "\"identificacion\""
  }
}
```

Validaciones esperadas:

- ✅ Validación de JSON antes de procesar
- ✅ Error específico del parsing
- ✅ No llega al validador de negocio

1.5 Prueba - Valores Numéricos Extremos

Request:

```
POST /api/facturas/emitir
Content-Type: application/json

{
  "emisor": {
    "nombre": "Test",
    "identificacion": "3101234567"
  },
  "receptor": {
    "nombre": "Cliente",
    "identificacion": "123456789"
  },
  "detalleServicio": [
    {
      "numeroLinea": 1,
      "cantidad": 999999999999999999.99,
      "detalle": "Producto",
      "precioUnitario": -1000,
      "montoTotalLinea": -999999999999999999.99,
      "impuestos": []
    }
  ],
  "resumenFactura": {
    "totalVenta": -999999999999999999.99,
    "totalComprobante": -999999999999999999.99,
    "codigoMoneda": "CRC"
```

```
  }
}
```

Response (400 Bad Request):

```
{
  "success": false,
  "error": "Datos de factura inválidos",
  "details": [
    {
      "field": "detalleServicio[0].cantidad",
      "message": "La cantidad debe ser un número positivo entre 0 y 999999999.99"
    },
    {
      "field": "detalleServicio[0].precioUnitario",
      "message": "El precio debe ser un número positivo"
    },
    {
      "field": "resumenFactura.totalVenta",
      "message": "El total no puede ser negativo"
    }
  ]
}
```

Validaciones esperadas:

- ✅ Rechazo de números negativos donde no aplique
- ✅ Límites máximos validados
- ✅ Precisión decimal controlada

1.6 Prueba - Null/Undefined en Campos Obligatorios

Request:

```
POST /api/facturas/emitir
Content-Type: application/json

{
  "emisor": null,
  "receptor": undefined,
  "detalleServicio": null,
  "resumenFactura": {}
}
```

Response (400 Bad Request):

```
{
  "success": false,
  "error": "Datos de factura inválidos",
  "details": [
```

```
    {
      "field": "emisor",
      "message": "El emisor es requerido"
    },
    {
      "field": "receptor",
      "message": "El receptor es requerido"
    },
    {
      "field": "detalleServicio",
      "message": "Debe haber al menos una línea de detalle"
    }
  ]
}
```

---

2. Casos Límite de Lógica de Negocio

2.1 Prueba - Factura con Un Céntimo

Request:

```
POST /api/facturas/emitir
Content-Type: application/json

{
  "emisor": {
    "nombre": "Prueba Céntimo",
    "identificacion": "3101234567"
  },
  "receptor": {
    "nombre": "Cliente",
    "identificacion": "123456789"
  },
  "detalleServicio": [
    {
      "numeroLinea": 1,
      "cantidad": 1,
      "detalle": "Producto",
      "precioUnitario": 0.01,
      "montoTotalLinea": 0.01,
      "impuestos": []
    }
  ],
  "resumenFactura": {
    "totalVenta": 0.01,
    "totalImpuesto": 0.00,
    "totalComprobante": 0.01,
    "codigoMoneda": "CRC"
  }
}
```

Response (201 Created):


```
{
  "success": true,
  "consecutivo": "00100010010000000100",
  "clave": "50602272020110310000100010010000000100",
  "estado": "EMISSION_SUCCESS",
  "timestamp": "2025-11-21T11:10:00.000Z"
}
```


Validaciones esperadas:

- ✅ Aceptación de montos mínimos
- ✅ Precisión decimal correcta
- ✅ Sin redondeos inesperados

2.2 Prueba - Múltiples Impuestos Acumulados

Request:

```
POST /api/facturas/emitir
Content-Type: application/json

{
  "emisor": {
    "nombre": "Test",
    "identificacion": "3101234567"
  },
  "receptor": {
    "nombre": "Cliente",
    "identificacion": "123456789"
  },
  "detalleServicio": [
    {
      "numeroLinea": 1,
      "cantidad": 1,
      "detalle": "Producto",
      "precioUnitario": 100,
      "montoTotalLinea": 100,
      "impuestos": [
        { "codigo": "01", "tarifa": 13.00, "monto": 13.00 },
        { "codigo": "02", "tarifa": 0.00, "monto": 0.00 },
        { "codigo": "05", "tarifa": 2.50, "monto": 2.50 },
        { "codigo": "07", "tarifa": 1.00, "monto": 1.00 }
      ]
    }
  ],
  "resumenFactura": {
    "totalVenta": 100,
    "totalImpuesto": 16.50,
    "totalComprobante": 116.50,
```

```
    "codigoMoneda": "CRC"
  }
}
```

Response (201 Created):

```
{
  "success": true,
  "consecutivo": "00100010010000000101",
  "clave": "50602272020110310000100010010000000101",
  "estado": "EMISSION_SUCCESS"
}
```

Validaciones esperadas:

- ✅ Múltiples impuestos soportados
- ✅ Suma correcta de impuestos (13 + 0 + 2.5 + 1 = 16.50)
- ✅ Total comprobante correcto (100 + 16.50 = 116.50)

2.3 Prueba - Descuentos Mayores que Total

Request:

```
POST /api/facturas/emitir
Content-Type: application/json

{
  "emisor": {
    "nombre": "Test",
    "identificacion": "3101234567"
  },
  "receptor": {
    "nombre": "Cliente",
    "identificacion": "123456789"
  },
  "detalleServicio": [
    {
      "numeroLinea": 1,
      "cantidad": 1,
      "detalle": "Producto",
      "precioUnitario": 100,
      "montoTotalLinea": 100,
      "descuentoMonto": 150,  // Descuento mayor que monto
      "impuestos": []
    }
  ],
  "resumenFactura": {
    "totalVenta": 100,
    "totalDescuentos": 150,
    "totalVentaNeta": -50,
    "totalImpuesto": 0,
```

```
    "totalComprobante": -50,
    "codigoMoneda": "CRC"
  }
}
```

Response (400 Bad Request):

```
{
  "success": false,
  "error": "Datos de factura inválidos",
  "details": [
    {
      "field": "detalleServicio[0].descuentoMonto",
      "message": "El descuento no puede ser mayor al monto de la línea"
    },
    {
      "field": "resumenFactura.totalVentaNeta",
      "message": "El total neto no puede ser negativo"
    }
  ]
}
```

Validaciones esperadas:

- ✅ Validación de descuentos
- ✅ Prevención de totales negativos
- ✅ Validación de coherencia en resumen

2.4 Prueba - Cálculos Inconsistentes en Resumen

Request:

```
POST /api/facturas/emitir
Content-Type: application/json

{
  "emisor": {
    "nombre": "Test",
    "identificacion": "3101234567"
  },
  "receptor": {
    "nombre": "Cliente",
    "identificacion": "123456789"
  },
  "detalleServicio": [
    {
      "numeroLinea": 1,
      "cantidad": 2,
      "detalle": "Producto",
      "precioUnitario": 50,
      "montoTotalLinea": 100,  // 2 × 50 = 100 ✓
```

```
      "impuestos": [
        { "codigo": "01", "tarifa": 13.00, "monto": 13.00 }
      ]
    }
  ],
  "resumenFactura": {
    "totalVenta": 100,
    "totalDescuentos": 0,
    "totalVentaNeta": 100,
    "totalImpuesto": 15.00,  // Dice 15 pero debe ser 13
    "totalComprobante": 115,
    "codigoMoneda": "CRC"
  }
}
```

Response (400 Bad Request):

```
{
  "success": false,
  "error": "Datos de factura inválidos",
  "details": [
    {
      "field": "resumenFactura.totalImpuesto",
      "message": "La suma de impuestos (13.00) no coincide con el total especificado
(15.00)"
    },
    {
      "field": "resumenFactura.totalComprobante",
      "message": "Total comprobante debe ser totalVentaNeta (100.00) + totalImpuesto (13.00)
= 113.00, no 115.00"
    }
  ]
}
```

---

3. Límites de Rate Limiting y DoS

3.1 Prueba - Rate Limiting en Emisión

Escenario: 100 requests en 10 segundos

```
# PowerShell
for ($i = 0; $i -lt 100; $i++) {
  Invoke-WebRequest -Uri "http://localhost:3000/api/facturas/emitir" `
    -Method POST `
    -Headers @{"Content-Type"="application/json"} `
    -Body '{"emisor":{"nombre":"Test","identificacion":"3101234567"},...}'
}
```

Response (Requests 1-50):

```
{ "success": true, "consecutivo": "...", ... }
```

Response (Request 51+, after rate limit):

```
HTTP/1.1 429 Too Many Requests
Retry-After: 60

{
  "success": false,
  "error": "Demasiadas solicitudes",
  "message": "Ha excedido el límite de 50 requests por minuto",
  "retryAfter": 60
}
```

Validaciones esperadas:

- ✅ Rate limiting activo (50 req/min sugerido)
- ✅ Header `Retry-After` presente
- ✅ HTTP 429 correcto
- ✅ No procesa más allá del límite

3.2 Prueba - Protección contra Payloads Grandes

Request (Payload de 50MB):

```
POST /api/facturas/emitir
Content-Type: application/json
Content-Length: 52428800

{
  "emisor": {...},
  "receptor": {...},
  "detalleServicio": [
    ... [millones de líneas de detalle] ...
  ]
}
```

Response (413 Payload Too Large):

```
{
  "success": false,
  "error": "Payload demasiado grande",
  "message": "El tamaño máximo permitido es 10MB",
  "maxAllowed": "10MB",
  "received": "50MB"
}
```

Validaciones esperadas:

- ✅ Límite de tamaño aplicado (10MB por defecto)

- ✅ HTTP 413 correcto
- ✅ Información de límite en respuesta

---

4. Concurrencia y Condiciones de Carrera

4.1 Prueba - Emisiones Simultáneas del Mismo Cliente

Request 1 y 2 - Simultáneamente (diferencia < 100ms):

```
POST /api/facturas/emitir
Content-Type: application/json

{
  "emisor": {
    "nombre": "Empresa Simultánea",
    "identificacion": "3101234567"
  },
  ...
}
```

Response 1:

```
{
  "success": true,
  "consecutivo": "00100010010000000200",
  "clave": "50602272020110310000100010010000000200"
}
```

Response 2:

```
{
  "success": true,
  "consecutivo": "00100010010000000201",
  "clave": "50602272020110310000100010010000000201"
}
```

Validaciones esperadas:

- ✅ Consecutivos únicos generados
- ✅ Sin colisiones
- ✅ Ambas facturas guardadas correctamente
- ✅ No hay corrupción de datos

4.2 Prueba - Lectura Mientras se Escribe

Thread 1: Emitir factura (POST /api/facturas/emitir) **Thread 2**: Listar facturas (GET /api/facturas) - iniciado 50ms después

Resultado esperado:

- ✅ GET puede ver o no la factura que se está escribiendo

- ✅ Si la ve, datos están completos
- ✅ Si no la ve, solo responde las previas
- ✅ Sin partial data o corruption

---

5. Recuperación de Errores

5.1 Prueba - Factura Parcialmente Guardada

Escenario: Conexión falla después de generar clave pero antes de guardar

Comportamiento esperado:

- ✅ Metadata existe pero archivo incompleto
- ✅ GET devuelve datos consistentes
- ✅ Sistema puede recuperarse sin corrupción
- ✅ Reintentos funcionan sin duplicados

5.2 Prueba - Recuperación de Storage Corrupto

Escenario: Archivo JSON inválido en directorio

Resultado esperado:

- ✅ GET /api/facturas lista otros archivos válidos
- ✅ GET /api/facturas/:consecutivo (corrupto) → 400 Bad Request
- ✅ Log registra el error
- ✅ Sistema continúa funcionando

---

6. Casos Especiales de Validación

6.1 Prueba - Identificaciones Válidas Límite

Casos válidos:

- 9 dígitos: "123456789" ✅
- 12 dígitos: "123456789012" ✅

Casos inválidos:

- 8 dígitos: "12345678" ❌
- 13 dígitos: "1234567890123" ❌

```
{
  "success": false,
  "error": "La identificación debe contener entre 9 y 12 dígitos"
}
```

6.2 Prueba - Moneda Específica de Costa Rica

Monedas válidas:

- "CRC" ✅

- "USD" ✅ (si está soportado)

Monedas inválidas:

- "MXN" ❌
- "EUR" ❌
- "XYZ" ❌

```
{
  "success": false,
  "error": "Moneda no válida",
  "details": {
    "provided": "MXN",
    "allowed": ["CRC", "USD"]
  }
}
```

6.3 Prueba - Tipos de Identificación

Válidos:

- "01" = Cédula Física ✅
- "02" = Cédula Jurídica ✅
- "03" = DIMEX ✅
- "04" = NITE ✅

Inválidos:

- "05" ❌
- "10" ❌
- "AA" ❌

Resumen de Pruebas - Documento 5

Categoría	Pruebas	Esperado	Estado
SQL Injection	1	Rechazo	✅
XSS	1	Rechazo	✅
Buffer Overflow	1	Rechazo	✅
JSON Malformado	1	400 Bad Request	✅
Números Extremos	1	Rechazo	✅
Null/Undefined	1	Rechazo	✅
Céntimos	1	201 Created	✅
Múltiples Impuestos	1	201 Created	✅
Descuentos Excesivos	1	Rechazo	✅

Cálculos Inconsistentes	1	Rechazo	✅
Rate Limiting	1	429 Too Many	✅
Payloads Grandes	1	413 Payload	✅
Concurrencia	2	Consistencia	✅
Recuperación	2	Robustez	✅
Validaciones Límite	3	Correctas	✅

Conclusiones

✅ **Seguridad**: Sistema resiste ataques comunes (SQL injection, XSS) ✅ **Robustez**: Maneja casos límite y excepcionales ✅ **Validación**: Entrada sanitizada y validada a fondo ✅ **Concurrencia**: Genera IDs únicos sin colisiones ✅ **Recuperación**: Recuperable ante errores inesperados ✅ **Performance**: Rate limiting previene DoS ✅ **Lógica**: Validaciones de negocio estrictas

---

FIN DE DOCUMENTACIÓN DE PRUEBAS

Todos los 5 documentos cubre:

1. **Emisión** de facturas (ES/EN)
2. **Validación y Envío** a Hacienda
3. **Consulta y Listado** de facturas
4. **Estado del Sistema** y monitoreo
5. **Errores, Límites y Seguridad**

Próximo paso: **Ejecutar tests automáticos** con Jest/Supertest