Problemas Potenciales y Soluciones v2.0 - Square to QuickBooks Integration (Versión Fortalecida)

Proyecto: Integración Personalizada Square-QuickBooks
Fecha: 2 de agosto de 2025
Versión: 2.0 (Incorporando mejoras críticas)

Mejoras Implementadas en v2.0

🔄 Jobs Idempotentes

- **Cambio:** Locks básicos → **Verificación pre-creación**
- **Razón:** Previene duplicados incluso si el worker falla depués del lock

📬 Dead Letter Queue

- **Cambio:** Polling backup → **DLQ + Polling**
- **Razón:** Repositorio centralizado para análisis y reprocesamiento dirigido

💰 Auto-ajustes Fiscales

- **Cambio:** Recomendaciones manuales → **Ajustes automáticos**
- **Razón:** Automatiza reconciliación para diferencias menores

🔒 Algoritmo Redlock

- **Cambio:** SET NX básico → **Redlock distribuido**
- **Razón:** Bloqueo más robusto para entornos distribuidos

Índice

PROBLEMAS TECNOLÓGICOS

1. API Rate Limiting y Throttling

1.1 Square API Rate Limits

Problema: Square impone límites estrictos en sus APIs: - **Orders API:** 1,000 requests/minuto por aplicación - **Catalog API:** 500 requests/minuto por aplicación - **Webhooks:** Timeout de 10 segundos para respuesta

Impacto: - Pérdida de transacciones durante picos de tráfico - Delays en sincronización - Posible suspensión temporal de la aplicación

Solución Técnica v2.0:

```
// Rate Limiter mejorado con Redlock
class AdvancedRateLimiter {
  private redis: Redis;
  private redlock: Redlock;
  private limits = {
    orders: { requests: 1000, window: 60000 }, // 1000/min
    catalog: { requests: 500, window: 60000 }  // 500/min
  };

  constructor() {
    this.redlock = new Redlock([this.redis], {
      driftFactor: 0.01,
      retryCount: 3,
      retryDelay: 200,
      retryJitter: 200
    });
  }

  async executeWithRateLimit<T>(
    apiType: 'orders' | 'catalog',
    operation: () => Promise<T>
  ): Promise<T> {
    const lockKey = `rate_limit_lock:${apiType}`;
    const lock = await this.redlock.acquire([lockKey], 5000);

    try {
      await this.waitForAvailability(apiType);
      const result = await operation();

      // Increment counter atomically
      await this.incrementCounter(apiType);

      return result;
    } finally {
      await lock.release();
    }
  }

  private async incrementCounter(apiType: 'orders' | 'catalog'): Promise<void>
{
    const key = `square_rate_limit:${apiType}`;
    const pipeline = this.redis.pipeline();

    pipeline.incr(key);
    pipeline.expire(key, this.limits[apiType].window / 1000);

    await pipeline.exec();
  }
}
```

Plan de Contingencia v2.0: 1. **Immediate:** Queue con prioridad + exponential backoff 2. **Short-term:** Distributed rate limiting con Redlock 3. **Long-term:** Multiple Square apps para distribución de carga

Métricas de Monitoreo: - `square_api_rate_limit_hits_total` - `square_api_request_duration_seconds` - `square_api_queue_depth` -

`square_api_redlock_acquisitions_total` (nuevo)

1.2 QuickBooks API Rate Limits

Problema: QuickBooks Online tiene límites más restrictivos: - **Sandbox:** 100 requests/minuto - **Production:** 500 requests/minuto por app - **Burst limit:** 100 requests en 60 segundos

Solución Técnica v2.0:

```
class QuickBooksRateLimiter {
  private tokenBucket: Map<string, TokenBucket> = new Map();
  private circuitBreaker: CircuitBreaker;

  constructor() {
    this.circuitBreaker = new CircuitBreaker(this.executeRequest.bind(this), {
      timeout: 30000,
      errorThresholdPercentage: 50,
      resetTimeout: 60000
    });
  }

  async executeWithRateLimit<T>(
    companyId: string,
    operation: () => Promise<T>
  ): Promise<T> {
    return this.circuitBreaker.fire(companyId, operation);
  }

  private async executeRequest<T>(
    companyId: string,
    operation: () => Promise<T>
  ): Promise<T> {
    const bucket = this.getOrCreateBucket(companyId);

    await bucket.waitForTokens(1);

    try {
      const result = await operation();
      bucket.resetRetryCount();
      return result;
    } catch (error) {
      if (error.status === 429) {
        const retryAfter = parseInt(error.headers['retry-after'] || '60');
        await new Promise(resolve => setTimeout(resolve, retryAfter * 1000));
        bucket.incrementRetryCount();

        if (bucket.retryCount < 3) {
          return this.executeRequest(companyId, operation);
        }
      }
      throw error;
    }
  }
}
```

2. Webhook Reliability Issues (Mejorado)

2.1 Webhook Delivery Failures con Dead Letter Queue

Problema: - Network timeouts (>10 segundos) - Server downtime durante mantenimiento - Webhooks duplicados o fuera de orden - Signature validation failures

Impacto: - Pérdida de transacciones críticas - Inconsistencia de datos - Necesidad de reconciliación manual

Solución Técnica v2.0 con DLQ:

```
class WebhookProcessorV2 {
  private orderQueue: Queue;
  private deadLetterQueue: Queue;
  private processedWebhooks: Set<string> = new Set();

  constructor() {
    // Main processing queue
    this.orderQueue = new Queue('order-processing', {
      connection: redisConnection,
      defaultJobOptions: {
        attempts: 3,
        backoff: { type: 'exponential', delay: 2000 },
        removeOnComplete: 100,
        removeOnFail: false // Keep failed jobs for DLQ
      }
    });

    // Dead Letter Queue for failed webhooks
    this.deadLetterQueue = new Queue('webhook-dlq', {
      connection: redisConnection,
      defaultJobOptions: {
        removeOnComplete: false, // Keep all DLQ items
        removeOnFail: false
      }
    });

    // Setup DLQ processing
    this.setupDeadLetterQueueProcessor();
  }

  async processWebhook(payload: SquareWebhook): Promise<void> {
    // 1. Idempotency check with TTL
    const webhookId = payload.id;
    const idempotencyKey = `webhook:${webhookId}`;

    const alreadyProcessed = await this.redis.get(idempotencyKey);
    if (alreadyProcessed) {
      logger.info('Webhook already processed', { webhookId });
      return;
    }

    // 2. Signature validation
    if (!this.validateSignature(payload)) {
      await this.deadLetterQueue.add('invalid-signature', {
        webhook: payload,
        error: 'Invalid webhook signature',
        timestamp: new Date().toISOString()
      });
      throw new Error('Invalid webhook signature');
    }

    // 3. Mark as processing
    await this.redis.setex(idempotencyKey, 3600, 'processing'); // 1 hour TTL

    // 4. Queue for processing
    try {
      await this.orderQueue.add('process-order', {
        orderId: payload.data.object.order.id,
        webhookId: payload.id,
        timestamp: payload.created_at
      });
      // Mark as queued successfully
      await this.redis.setex(idempotencyKey, 3600, 'queued');

    } catch (error) {
      // Move to DLQ if queueing fails
      await this.deadLetterQueue.add('queue-failed', {
        webhook: payload,
        error: error.message,
        timestamp: new Date().toISOString()
      });

      await this.redis.del(idempotencyKey);
      throw error;
    }
  }

  private setupDeadLetterQueueProcessor(): void {
    // Process failed jobs and move to DLQ
    this.orderQueue.on('failed', async (job, error) => {
      logger.error('Job failed, moving to DLQ', {
        jobId: job.id,
        error: error.message
      });

      await this.deadLetterQueue.add('processing-failed', {
        originalJob: job.data,
        error: error.message,
        attempts: job.attemptsMade,
        timestamp: new Date().toISOString()
      });
    });

    // DLQ processor for manual review and reprocessing
    const dlqWorker = new Worker('webhook-dlq', async (job) => {
      logger.info('Processing DLQ item', {
        jobId: job.id,
        type: job.name
      });

      // Log for manual review
      await this.logDLQItem(job);

      // Attempt automatic recovery for certain error types
      if (this.canAutoRecover(job)) {
        await this.attemptAutoRecovery(job);
      }
    });
  }

  private async attemptAutoRecovery(job: Job): Promise<void> {
    const { originalJob, error } = job.data;

    // Auto-recovery for temporary network issues
    if (error.includes('ECONNRESET') || error.includes('timeout')) {
      logger.info('Attempting auto-recovery for network issue', {
        jobId: job.id
      });

      // Wait and retry
      await new Promise(resolve => setTimeout(resolve, 30000));
      await this.orderQueue.add('process-order', originalJob, {
        attempts: 1, // Single retry attempt
        delay: 60000 // 1 minute delay
      });
    }
  }

  // Enhanced polling backup with DLQ awareness
  async pollForMissedOrders(): Promise<void> {
    const lastProcessed = await this.getLastProcessedTimestamp();
    const dlqItems = await this.getDLQOrderIds();

    const orders = await this.squareClient.ordersApi.searchOrders({
      locationIds: this.locationIds,
      query: {
        filter: {
          dateTimeFilter: {
            updatedAt: {
              startAt: lastProcessed,
              endAt: new Date().toISOString()
            }
          },
          stateFilter: {
            states: ['COMPLETED']
          }
        }
      }
    });

    for (const order of orders.result.orders || []) {
      // Skip if already in DLQ (being handled manually)
      if (dlqItems.has(order.id)) {
        continue;
      }

      await this.processOrderIfNotExists(order);
    }
  }
}
```

Plan de Contingencia v2.0: 1. **Immediate:** DLQ captura todos los fallos + polling backup 2. **Short-term:** Auto-recovery para errores temporales 3. **Long-term:** Manual review dashboard para DLQ items

3. Data Consistency Issues (Fortalecido)

3.1 Jobs Idempotentes con Verificación Pre-creación

Problema Original: - Race conditions en processing - Duplicados si worker falla después del lock - Partial failures durante sync

Solución v2.0 - Jobs Idempotentes:

```
class IdempotentOrderProcessor {
  async processOrderIdempotently(orderId: string): Promise<ProcessingResult> {
    const lockKey = `order_lock:${orderId}`;

    // Use Redlock for distributed locking
    const lock = await this.redlock.acquire([lockKey], 30000);

    try {
      // STEP 1: Check if already processed (idempotency)
      const existingReceipt = await this.findExistingQBReceipt(orderId);
      if (existingReceipt) {
        logger.info('Order already processed', {
          orderId,
          qbReceiptId: existingReceipt.qbReceiptId
        });
        return {
          status: 'already_processed',
          qbReceiptId: existingReceipt.qbReceiptId,
          skipped: true
        };
      }

      // STEP 2: Get and validate order data
      const squareOrder = await this.getSquareOrderWithValidation(orderId);

      // STEP 3: Transform data (pure function - no side effects)
      const qbSalesReceiptData = await this.transformOrderToQB(squareOrder);

      // STEP 4: Pre-flight check in QuickBooks
      const preflightResult = await this.preflightCheckQB(qbSalesReceiptData);
      if (!preflightResult.valid) {
        throw new Error(`Preflight check failed: ${preflightResult.error}`);
      }

      // STEP 5: Atomic transaction with idempotency check
      return await this.db.transaction(async (tx) => {
        // Double-check within transaction
        const doubleCheck = await tx.qbSalesReceipt.findUnique({
          where: { squareOrderId: orderId }
        });

        if (doubleCheck) {
          return {
            status: 'already_processed',
            qbReceiptId: doubleCheck.qbReceiptId,
            skipped: true
          };
        }

        // Create in QuickBooks
        const qbReceipt = await this.qbClient.createSalesReceipt(
          process.env.QB_COMPANY_ID!,
          qbSalesReceiptData
        );

        // Store in our database
        const dbRecord = await tx.qbSalesReceipt.create({
          data: {
            squareOrderId: orderId,
            qbReceiptId: qbReceipt.Id,
            qbSyncToken: qbReceipt.SyncToken,
            totalAmt: parseFloat(qbReceipt.TotalAmt),
            syncStatus: 'SYNCED',
            syncedAt: new Date(),
            rawQBData: qbReceipt
          }
        });

        return {
          status: 'processed',
          qbReceiptId: qbReceipt.Id,
          dbRecordId: dbRecord.id,
          skipped: false
        };
      });

    } finally {
      await lock.release();
    }
  }

  private async findExistingQBReceipt(orderId: string): Promise<QBSalesReceipt
| null> {
    // Check our database first
    const dbRecord = await this.db.qbSalesReceipt.findUnique({
      where: { squareOrderId: orderId }
    });

    if (dbRecord) {
      return dbRecord;
    }

    // Double-check in QuickBooks using custom field
    try {
      const qbReceipts = await this.qbClient.searchSalesReceipts(
        process.env.QB_COMPANY_ID!,
        `Square Order ID = '${orderId}'`
      );

      if (qbReceipts.length > 0) {
        // Found in QB but not in our DB - sync back
        const qbReceipt = qbReceipts[0];

        await this.db.qbSalesReceipt.create({
          data: {
            squareOrderId: orderId,
            qbReceiptId: qbReceipt.Id,
            qbSyncToken: qbReceipt.SyncToken,
            totalAmt: parseFloat(qbReceipt.TotalAmt),
            syncStatus: 'SYNCED',
            syncedAt: new Date(),
            rawQBData: qbReceipt
          }
        });

        return qbReceipt;
      }
    } catch (error) {
      logger.warn('Error checking QuickBooks for existing receipt', {
        orderId,
        error: error.message
      });
    }
    return null;
  }

  private async preflightCheckQB(salesReceiptData: any): Promise<{valid: boolean, error?: string}> {
    try {
      // Validate all referenced items exist
      for (const line of salesReceiptData.Line) {
        if (line.DetailType === 'SalesItemLineDetail') {
          const itemId = line.SalesItemLineDetail.ItemRef.value;
          const item = await this.qbClient.getItem(
            process.env.QB_COMPANY_ID!,
            itemId
          );

          if (!item) {
            return {
              valid: false,
              error: `Item ${itemId} not found in QuickBooks`
            };
          }
        }
      }

      // Validate customer exists
      if (salesReceiptData.CustomerRef) {
        const customer = await this.qbClient.getCustomer(
          process.env.QB_COMPANY_ID!,
          salesReceiptData.CustomerRef.value
        );

        if (!customer) {
          return {
            valid: false,
            error: `Customer ${salesReceiptData.CustomerRef.value} not found`
          };
        }
      }

      return { valid: true };
    } catch (error) {
      return {
        valid: false,
        error: `Preflight check error: ${error.message}`
      };
    }
  }
}
```

3.2 Redlock Implementation para Distributed Locking

```
class DistributedLockManager {
  private redlock: Redlock;

  constructor(redisInstances: Redis[]) {
    this.redlock = new Redlock(redisInstances, {
      // The expected clock drift; for more details see:
      // http://redis.io/topics/distlock
      driftFactor: 0.01, // multiplied by lock ttl to determine drift time

      // The max number of times Redlock will attempt to lock a resource
      // before erroring.
      retryCount: 10,

      // the time in ms between attempts
      retryDelay: 200, // time in ms

      // the max time in ms randomly added to retries
      // to improve performance under high contention
      retryJitter: 200, // time in ms

      // The minimum remaining time on a lock before an extension is
automatically
      // attempted with the `using` API.
      automaticExtensionThreshold: 500, // time in ms
    });
  }

  async withLock<T>(
    resource: string,
    ttl: number,
    operation: () => Promise<T>
  ): Promise<T> {
    const lock = await this.redlock.acquire([resource], ttl);

    try {
      return await operation();
    } finally {
      await lock.release();
    }
  }
}
```

PROBLEMAS DE NEGOCIO

1. Data Mapping Complexity

1.1 Modifier Mapping Ambiguity

Problema: - Múltiples estrategias de mapeo para mismo modifier - Inconsistencias en naming conventions - Business rules complejas para pricing

Impacto: - Reportes incorrectos - Reconciliación manual requerida - Pérdida de información granular

Solución de Negocio v2.0:

```
// Enhanced mapping strategies with validation
interface ModifierMappingStrategyV2 {
  id: string;
  name: string;
  description: string;
  validation: (modifier: SquareModifier) => ValidationResult;
  implementation: (modifier: SquareModifier) => Promise<QBLineItem[]>;
  rollback?: (qbLineItems: QBLineItem[]) => Promise<void>;
}

const mappingStrategiesV2: ModifierMappingStrategyV2[] = [
  {
    id: 'separate_line_item_v2',
    name: 'Línea Separada con Validación',
    description: 'Cada modifier como línea independiente con verificación
previa',
    validation: (modifier) => {
      if (!modifier.catalogObjectId) {
        return { valid: false, error: 'Modifier missing catalog ID' };
      }
      if (modifier.basePriceMoney.amount < 0) {
        return { valid: false, error: 'Negative modifier price not allowed' };
      }
      return { valid: true };
    },
    implementation: async (modifier) => {
      // Verify QB item exists or create it
      const qbItem = await this.getOrCreateModifierItem(modifier);

      return [{
        Amount: modifier.basePriceMoney.amount / 100,
        DetailType: "SalesItemLineDetail",
        SalesItemLineDetail: {
          ItemRef: { value: qbItem.Id, name: qbItem.Name },
          Qty: 1,
          UnitPrice: modifier.basePriceMoney.amount / 100
        },
        Description: `${modifier.name} (Modifier)`
      }];
    }
  },
  {
    id: 'smart_bundling',
    name: 'Agrupación Inteligente',
    description: 'Agrupa modifiers similares para simplificar QB',
    validation: (modifier) => ({ valid: true }),
    implementation: async (modifier) => {
      // Group similar modifiers together
      const similarModifiers = await this.findSimilarModifiers(modifier);

      if (similarModifiers.length > 1) {
        const totalAmount = similarModifiers.reduce(
          (sum, m) => sum + m.basePriceMoney.amount, 0
        );

        return [{
          Amount: totalAmount / 100,
          DetailType: "SalesItemLineDetail",
          SalesItemLineDetail: {
            ItemRef: { value: "MODIFIER_BUNDLE", name: "Modifier Bundle" },
            Qty: 1,
            UnitPrice: totalAmount / 100
          },
          Description: `Bundle: ${similarModifiers.map(m => m.name).join(', 
')}`
        }];
      }

      // Fall back to individual line item
      return this.mappingStrategiesV2[0].implementation(modifier);
    }
  }
];
```

2. Financial Reconciliation Issues (Mejorado)

2.1 Auto-ajustes Fiscales

Problema: - Different tax calculation methods - Tax exemptions not syncing - Multi-jurisdiction tax complexity - Manual reconciliation required

Solución v2.0 - Auto-ajustes:

```
 class AutoTaxReconciliationService {
  private readonly MINOR_VARIANCE_THRESHOLD = 0.05; // $0.05
  private readonly MODERATE_VARIANCE_THRESHOLD = 1.00; // $1.00
  private readonly TAX_ADJUSTMENT_ACCOUNT = "90"; // Tax Adjustments account

  async validateAndAutoAdjustTax(order: SquareOrder):
Promise<TaxReconciliationResult> {
    const squareTax = this.calculateSquareTax(order);
    const qbTax = await this.calculateQBTax(order);

    const variance = Math.abs(squareTax - qbTax);

    if (variance <= this.MINOR_VARIANCE_THRESHOLD) {
      // Auto-adjust minor differences
      const adjustmentResult = await this.createAutoTaxAdjustment(
        order.id,
        variance,
        squareTax > qbTax ? 'increase' : 'decrease'
      );

      return {
        valid: true,
        variance,
        squareTax,
        qbTax,
        autoAdjusted: true,
        adjustmentId: adjustmentResult.Id,
        action: 'auto_adjusted'
      };
    } else if (variance <= this.MODERATE_VARIANCE_THRESHOLD) {
      // Create adjustment but flag for review
      const adjustmentResult = await this.createTaxAdjustmentForReview(
        order.id,
        variance,
        squareTax,
        qbTax
      );

      return {
        valid: false,
        variance,
        squareTax,
        qbTax,
        autoAdjusted: false,
        adjustmentId: adjustmentResult.Id,
        action: 'flagged_for_review',
        recommendation: 'Moderate tax variance - created adjustment entry for
review'
      };
    } else {
      // Significant variance - manual review required
      await this.flagForManualReview(order.id, variance, squareTax, qbTax);

      return {
        valid: false,
        variance,
        squareTax,
        qbTax,
        autoAdjusted: false,
        action: 'manual_review_required',
        recommendation: 'Significant tax calculation difference - manual review
```

```
required'
      };
    }
  }

  private async createAutoTaxAdjustment(
    orderId: string,
    variance: number,
    direction: 'increase' | 'decrease'
  ): Promise<any> {
    const journalEntry = {
      Line: [
        {
          Amount: variance,
          DetailType: "JournalEntryLineDetail",
          JournalEntryLineDetail: {
            PostingType: direction === 'increase' ? "Debit" : "Credit",
            AccountRef: {
              value: "79", // Sales Tax Payable
              name: "Sales Tax Payable"
            }
          },
          Description: `Auto tax adjustment for Square order ${orderId}`
        },
        {
          Amount: variance,
          DetailType: "JournalEntryLineDetail",
          JournalEntryLineDetail: {
            PostingType: direction === 'increase' ? "Credit" : "Debit",
            AccountRef: {
              value: this.TAX_ADJUSTMENT_ACCOUNT,
              name: "Tax Rounding Adjustments"
            }
          },
          Description: `Tax rounding adjustment - Square order ${orderId}`
        }
      ],
      TxnDate: new Date().toISOString().split('T')[0],
      PrivateNote: `Automatic tax adjustment for variance of ```math
{variance.toFixed(2)}`
    };

    const result = await this.qbClient.createJournalEntry(
      process.env.QB_COMPANY_ID!,
      journalEntry
    );

    // Log the auto-adjustment
    logger.info('Auto tax adjustment created', {
      orderId,
      variance,
      direction,
      journalEntryId: result.Id
    });

    // Send notification for audit trail
    await this.notifyTaxAdjustment({
      orderId,
      variance,
      direction,
      journalEntryId: result.Id,
      type: 'auto_adjustment'
```

~~~
    });

    return result;
  }

  private async createTaxAdjustmentForReview(
    orderId: string,
    variance: number,
    squareTax: number,
    qbTax: number
  ): Promise<any> {
    // Create the adjustment entry but mark it for review
    const journalEntry = {
      Line: [
        {
          Amount: variance,
          DetailType: "JournalEntryLineDetail",
          JournalEntryLineDetail: {
            PostingType: squareTax > qbTax ? "Debit" : "Credit",
            AccountRef: {
              value: "79", // Sales Tax Payable
              name: "Sales Tax Payable"
            }
          },
          Description: `Tax adjustment for review - Square order ${orderId}`
        },
        {
          Amount: variance,
          DetailType: "JournalEntryLineDetail",
          JournalEntryLineDetail: {
            PostingType: squareTax > qbTax ? "Credit" : "Debit",
            AccountRef: {
              value: "91", // Tax Adjustments Pending Review
              name: "Tax Adjustments - Pending Review"
            }
          },
          Description: `Tax variance pending review - Square order ${orderId}`
        }
      ],
      TxnDate: new Date().toISOString().split('T')[0],
      PrivateNote: `Tax adjustment pending review - Variance:
```{variance.toFixed(2)}, Square: ```math
{squareTax.toFixed(2)}, QB:
```{qbTax.toFixed(2)}`
    };

    const result = await this.qbClient.createJournalEntry(
      process.env.QB_COMPANY_ID!,
      journalEntry
    );

    // Create review task
    await this.createReviewTask({
      orderId,
      variance,
      squareTax,
      qbTax,
      journalEntryId: result.Id,
      priority: variance > 0.50 ? 'high' : 'medium'
    });

    return result;
~~~

```
  }

  private async notifyTaxAdjustment(adjustment: {
    orderId: string;
    variance: number;
    direction: string;
    journalEntryId: string;
    type: string;
  }): Promise<void> {
    // Send to audit log
    await this.auditLogger.log({
      action: 'TAX_AUTO_ADJUSTMENT',
      entityType: 'order',
      entityId: adjustment.orderId,
      details: adjustment,
      timestamp: new Date()
    });

    // Send email notification for larger adjustments
    if (adjustment.variance > 0.10) {
      await this.emailService.send({
        to: process.env.ACCOUNTING_EMAIL!,
        subject: `Tax Auto-Adjustment: ```math
{adjustment.variance.toFixed(2)}`,
        template: 'tax-adjustment',
        data: adjustment
      });
    }
  }
}
```

2.2 Enhanced Fee Reconciliation

```
 class EnhancedFeeReconciliationManager {
   async processDailyReconciliationV2(date: Date):
Promise<ReconciliationReportV2> {
    // Get all transactions and fees for the day
    const transactions = await this.getTransactionsForDate(date);
    const fees = await this.getSquareFeesForDate(date);
    const deposits = await this.getBankDepositsForDate(date);

    // Calculate expected vs actual with fee breakdown
    const grossSales = transactions.reduce((sum, t) => sum + t.totalMoney, 0);
    const totalFees = fees.reduce((sum, f) => sum + f.amount, 0);
    const expectedDeposit = grossSales - totalFees;
    const actualDeposit = deposits.reduce((sum, d) => sum + d.amount, 0);

    const variance = expectedDeposit - actualDeposit;

    // Auto-reconcile small variances
    if (Math.abs(variance) <= 0.05) {
      await this.createAutoReconciliationEntry(variance, date);

      return {
        date,
        transactionCount: transactions.length,
        grossSales,
        totalFees,
        expectedDeposit,
        actualDeposit,
        variance,
        reconciled: true,
        autoReconciled: true,
        feeBreakdown: this.categorizeFees(fees)
      };
    }

    // Create manual reconciliation entry for larger variances
    if (Math.abs(variance) > 0.05) {
      await this.createManualReconciliationEntry(variance, date, {
        grossSales,
        totalFees,
        expectedDeposit,
        actualDeposit
      });

      return {
        date,
        transactionCount: transactions.length,
        grossSales,
        totalFees,
        expectedDeposit,
        actualDeposit,
        variance,
        reconciled: false,
        autoReconciled: false,
        requiresReview: true,
        feeBreakdown: this.categorizeFees(fees)
      };
    }
```

~~~
    return {
      date,
      transactionCount: transactions.length,
      grossSales,
      totalFees,
      expectedDeposit,
      actualDeposit,
      variance: 0,
      reconciled: true,
      autoReconciled: false,
      feeBreakdown: this.categorizeFees(fees)
    };
  }

  private async createAutoReconciliationEntry(variance: number, date: Date):
Promise<void> {
    if (Math.abs(variance) === 0) return;

    const journalEntry = {
      Line: [
        {
          Amount: Math.abs(variance),
          DetailType: "JournalEntryLineDetail",
          JournalEntryLineDetail: {
            PostingType: variance > 0 ? "Debit" : "Credit",
            AccountRef: { value: "1", name: "Checking Account" }
          },
          Description: `Auto reconciliation adjustment -
${date.toISOString().split('T')[0]}`
        },
        {
          Amount: Math.abs(variance),
          DetailType: "JournalEntryLineDetail",
          JournalEntryLineDetail: {
            PostingType: variance > 0 ? "Credit" : "Debit",
            AccountRef: { value: "92", name: "Bank Reconciliation Adjustments"
}
          },
          Description: `Bank reconciliation variance -
${date.toISOString().split('T')[0]}`
        }
      ],
      TxnDate: date.toISOString().split('T')[0],
      PrivateNote: `Auto-reconciliation for variance of
```{Math.abs(variance).toFixed(2)}`
    };

    await this.qbClient.createJournalEntry(journalEntry);

    logger.info('Auto reconciliation entry created', {
      date: date.toISOString().split('T')[0],
      variance
    });
  }
}
~~~

PLANES DE CONTINGENCIA v2.0

1. Plan de Contingencia para Fallos de API (Mejorado)

Nivel 1: Degradación Graceful con DLQ

Trigger: API response time > 5 segundos o error rate > 5%

Acciones Automáticas: 1. Activar circuit breaker con Redlock 2. Mover requests fallidos a Dead Letter Queue 3. Activar polling backup intensivo 4. Notificar al equipo de monitoreo

Código de Implementación v2.0:

```typescript
class EnhancedCircuitBreaker {
  private state: 'CLOSED' | 'OPEN' | 'HALF_OPEN' = 'CLOSED';
  private failureCount = 0;
  private lastFailureTime = 0;
  private threshold = 5;
  private timeout = 60000; // 1 minute
  private deadLetterQueue: Queue;

  constructor() {
    this.deadLetterQueue = new Queue('api-failures-dlq');
  }

  async execute<T>(
    operation: () => Promise<T>,
    fallbackToDLQ: boolean = true
  ): Promise<T> {
    if (this.state === 'OPEN') {
      if (Date.now() - this.lastFailureTime > this.timeout) {
        this.state = 'HALF_OPEN';
      } else {
        if (fallbackToDLQ) {
          await this.deadLetterQueue.add('circuit-breaker-blocked', {
            operation: operation.toString(),
            timestamp: new Date().toISOString(),
            reason: 'Circuit breaker is OPEN'
          });
        }
        throw new Error('Circuit breaker is OPEN - request moved to DLQ');
      }
    }

    try {
      const result = await operation();
      this.onSuccess();
      return result;
    } catch (error) {
      this.onFailure();

      if (fallbackToDLQ) {
        await this.deadLetterQueue.add('operation-failed', {
          operation: operation.toString(),
          error: error.message,
          timestamp: new Date().toISOString()
        });
      }

      throw error;
    }
  }

  private onSuccess(): void {
    this.failureCount = 0;
    this.state = 'CLOSED';
  }

  private onFailure(): void {
    this.failureCount++;
    this.lastFailureTime = Date.now();

    if (this.failureCount >= this.threshold) {
      this.state = 'OPEN';
      // Trigger intensive polling backup
      this.triggerIntensivePolling();
    }
  }

  private async triggerIntensivePolling(): Promise<void> {
    // Increase polling frequency during API issues
    await this.redis.setex('intensive_polling_mode', 300, 'true'); // 5 minutes

    logger.warn('Circuit breaker OPEN - activated intensive polling mode');
  }
}
```

Nivel 2: Modo de Emergencia con Auto-Recovery

Trigger: API completamente inaccesible por > 15 minutos

Acciones Automáticas: 1. Activar modo de emergencia completo 2. Procesar todos los items de DLQ 3. Intensificar polling a cada 30 segundos 4. Notificar a stakeholders automáticamente

Nivel 3: Disaster Recovery con Rollback

Trigger: Fallo completo del servicio por > 1 hora

Acciones: 1. Activar infraestructura de backup automáticamente 2. Restaurar desde último backup con verificación 3. Procesar DLQ completa 4. Comunicación automática a clientes

2. Plan de Contingencia para Inconsistencias de Datos

(Fortalecido)

Detección Automática con Jobs Idempotentes

~~~
class EnhancedDataConsistencyChecker {
  async runConsistencyCheckV2(): Promise<ConsistencyReportV2> {
    const issues: ConsistencyIssueV2[] = [];

    // Check 1: Orders in Square but not in QB (with idempotency verification)
    const orphanedOrders = await this.findOrphanedOrdersV2();
    for (const order of orphanedOrders) {
      // Verify it's truly orphaned (not just processing)
      const isProcessing = await this.redis.get(`order_lock:${order.id}`);
      if (!isProcessing) {
        issues.push({
          type: 'ORPHANED_ORDER',
          severity: 'HIGH',
          orderId: order.id,
          description: `Order ${order.id} exists in Square but not in
QuickBooks`,
          recommendedAction: 'REPROCESS_ORDER',
          autoRecoverable: true
        });
      }
    }

    // Check 2: Amount discrepancies with tolerance
    const amountDiscrepancies = await this.findAmountDiscrepanciesV2();
    for (const discrepancy of amountDiscrepancies) {
      const severity = discrepancy.variance > 1.00 ? 'HIGH' : 'MEDIUM';
      issues.push({
        type: 'AMOUNT_MISMATCH',
        severity,
        orderId: discrepancy.orderId,
        description: `Amount mismatch: Square ```math
{discrepancy.squareAmount}, QB
```{discrepancy.qbAmount}`,
        recommendedAction: discrepancy.variance < 0.05 ? 'AUTO_ADJUST' :
'MANUAL_REVIEW',
        autoRecoverable: discrepancy.variance < 0.05
      });
    }

    // Check 3: Stuck processing jobs
    const stuckJobs = await this.findStuckProcessingJobs();
    for (const job of stuckJobs) {
      issues.push({
        type: 'STUCK_PROCESSING',
        severity: 'MEDIUM',
        orderId: job.orderId,
        description: `Job stuck in processing for ${job.duration} minutes`,
        recommendedAction: 'RESTART_PROCESSING',
        autoRecoverable: true
      });
    }

    const report: ConsistencyReportV2 = {
      timestamp: new Date(),
~~~

```
      totalIssues: issues.length,
      autoRecoverableIssues: issues.filter(i => i.autoRecoverable).length,
      issuesBySeverity: this.groupBySeverity(issues),
      issues
    };

    // Auto-resolve recoverable issues
    await this.autoResolveIssuesV2(report);

    return report;
  }

  async autoResolveIssuesV2(report: ConsistencyReportV2): Promise<void> {
    const autoRecoverableIssues = report.issues.filter(i => i.autoRecoverable);

    for (const issue of autoRecoverableIssues) {
      try {
        switch (issue.recommendedAction) {
          case 'REPROCESS_ORDER':
            await this.reprocessOrderIdempotently(issue.orderId!);
            break;
          case 'AUTO_ADJUST':
            await this.createAutoAdjustment(issue);
            break;
          case 'RESTART_PROCESSING':
            await this.restartStuckProcessing(issue.orderId!);
            break;
        }

        logger.info('Auto-resolved consistency issue', {
          type: issue.type,
          orderId: issue.orderId,
          action: issue.recommendedAction
        });
      } catch (error) {
        logger.error('Failed to auto-resolve issue', {
          issue,
          error: error.message
        });
      }
    }
  }

  private async reprocessOrderIdempotently(orderId: string): Promise<void> {
    // Use the idempotent processor to safely reprocess
    const processor = new IdempotentOrderProcessor();
    await processor.processOrderIdempotently(orderId);
  }
}
```

MÉTRICAS Y MONITOREO v2.0

1. Métricas de Sistema Mejoradas

1.1 Métricas de Performance con DLQ

```
// Enhanced Prometheus metrics
const metricsV2 = {
  // Existing metrics
  apiRequestDuration: new prometheus.Histogram({
    name: 'api_request_duration_seconds',
    help: 'Duration of API requests',
    labelNames: ['method', 'route', 'status_code'],
    buckets: [0.1, 0.5, 1, 2, 5, 10]
  }),

  // NEW: Dead Letter Queue metrics
  dlqItemsTotal: new prometheus.Counter({
    name: 'dlq_items_total',
    help: 'Total number of items in Dead Letter Queue',
    labelNames: ['queue_name', 'reason']
  }),

  dlqProcessingDuration: new prometheus.Histogram({
    name: 'dlq_processing_duration_seconds',
    help: 'Time to process DLQ items',
    labelNames: ['queue_name', 'outcome'],
    buckets: [1, 5, 10, 30, 60, 300]
  }),

  // NEW: Idempotency metrics
  idempotentJobsSkipped: new prometheus.Counter({
    name: 'idempotent_jobs_skipped_total',
    help: 'Number of jobs skipped due to idempotency',
    labelNames: ['job_type']
  }),

  // NEW: Auto-adjustment metrics
  autoAdjustmentsCreated: new prometheus.Counter({
    name: 'auto_adjustments_created_total',
    help: 'Number of automatic adjustments created',
    labelNames: ['adjustment_type', 'severity']
  }),

  autoAdjustmentAmount: new prometheus.Histogram({
    name: 'auto_adjustment_amount_dollars',
    help: 'Amount of automatic adjustments in dollars',
    labelNames: ['adjustment_type'],
    buckets: [0.01, 0.05, 0.10, 0.50, 1.00, 5.00]
  }),

  // NEW: Redlock metrics
  redlockAcquisitions: new prometheus.Counter({
    name: 'redlock_acquisitions_total',
    help: 'Number of Redlock acquisitions',
    labelNames: ['resource', 'outcome']
  }),

  redlockWaitTime: new prometheus.Histogram({
    name: 'redlock_wait_time_seconds',
    help: 'Time waiting to acquire Redlock',
    labelNames: ['resource'],
    buckets: [0.1, 0.5, 1, 2, 5, 10]
  })
};
```

1.2 Enhanced Health Checks

```
class HealthCheckerV2 {
  async checkHealthV2(): Promise<HealthStatusV2> {
    const checks = await Promise.allSettled([
      this.checkDatabase(),
      this.checkRedis(),
      this.checkSquareAPI(),
      this.checkQuickBooksAPI(),
      this.checkDeadLetterQueue(), // NEW
      this.checkIdempotencySystem(), // NEW
      this.checkAutoAdjustmentSystem() // NEW
    ]);

    const status = checks.every(check => check.status === 'fulfilled')
      ? 'healthy'
      : 'unhealthy';

    return {
      status,
      timestamp: new Date(),
      checks: checks.map((check, index) => ({
        name: [
          'database',
          'redis',
          'square_api',
          'quickbooks_api',
          'dead_letter_queue',
          'idempotency_system',
          'auto_adjustment_system'
        ][index],
        status: check.status === 'fulfilled' ? 'up' : 'down',
        responseTime: check.status === 'fulfilled' ? check.value.responseTime :
null,
        error: check.status === 'rejected' ? check.reason.message : null,
        details: check.status === 'fulfilled' ? check.value.details : null
      })),
      dlqStats: await this.getDLQStats(),
      idempotencyStats: await this.getIdempotencyStats()
    };
  }

  private async checkDeadLetterQueue(): Promise<HealthCheckResult> {
    const start = Date.now();

    try {
      const dlqDepth = await this.deadLetterQueue.count();
      const oldestItem = await this.deadLetterQueue.getJobs(['waiting'], 0, 1);

      const responseTime = Date.now() - start;

      return {
        responseTime,
        details: {
          queueDepth: dlqDepth,
          oldestItemAge: oldestItem.length > 0
            ? Date.now() - oldestItem[0].timestamp
            : 0
        }
      };
    } catch (error) {
      throw new Error(`DLQ health check failed: ${error.message}`);
    }
  }

  private async checkIdempotencySystem(): Promise<HealthCheckResult> {
    const start = Date.now();

    try {
      // Test idempotency key creation and retrieval
      const testKey = `health_check:${Date.now()}`;
      await this.redis.setex(testKey, 10, 'test');
      const retrieved = await this.redis.get(testKey);
      await this.redis.del(testKey);

      if (retrieved !== 'test') {
        throw new Error('Idempotency key test failed');
      }

      const responseTime = Date.now() - start;

      return {
        responseTime,
        details: {
          redisConnected: true,
          idempotencyKeysWorking: true
        }
      };
    } catch (error) {
      throw new Error(`Idempotency system check failed: ${error.message}`);
    }
  }
}
```

2. Alerting Rules v2.0

2.1 Enhanced Critical Alerts

```yaml
# Enhanced Prometheus alerting rules
groups:
  - name: square_quickbooks_critical_v2
    rules:
      - alert: HighErrorRate
        expr: rate(errors_total[5m]) > 0.1
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "High error rate detected"
          description: "Error rate is {{ $value }} errors per second"

      - alert: DeadLetterQueueBacklog
        expr: dlq_items_total > 100
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "Dead Letter Queue backlog detected"
          description: "DLQ has {{ $value }} items pending"

      - alert: IdempotentJobsHighSkipRate
        expr: rate(idempotent_jobs_skipped_total[10m]) > 0.5
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "High idempotent job skip rate"
          description: "{{ $value }} jobs per second being skipped due to
idempotency"

      - alert: AutoAdjustmentSpike
        expr: rate(auto_adjustments_created_total[5m]) > 10
        for: 2m
        labels:
          severity: warning
        annotations:
          summary: "Spike in automatic adjustments"
          description: "{{ $value }} auto-adjustments per second - possible
data quality issue"

      - alert: RedlockContentionHigh
        expr: histogram_quantile(0.95, redlock_wait_time_seconds) > 5
        for: 3m
        labels:
          severity: warning
        annotations:
          summary: "High Redlock contention"
          description: "95th percentile Redlock wait time is {{ $value }}
seconds"
```

2.2 Business Alerts v2.0

```
class BusinessAlertingV2 {
  async checkBusinessMetricsV2(): Promise<void> {
    // Check for processing delays with DLQ awareness
    const oldestUnprocessedOrder = await this.getOldestUnprocessedOrder();
    if (oldestUnprocessedOrder &&
        Date.now() - oldestUnprocessedOrder.createdAt.getTime() > 300000) { //
5 minutes

      // Check if it's in DLQ
      const inDLQ = await this.isOrderInDLQ(oldestUnprocessedOrder.id);

      await this.sendAlert({
        type: 'PROCESSING_DELAY',
        severity: inDLQ ? 'MEDIUM' : 'HIGH',
        message: `Order $`{oldestUnprocessedOrder.id} has been unprocessed for
`${
          Math.round((Date.now() - oldestUnprocessedOrder.createdAt.getTime())
/ 60000)
        } minutes${inDLQ ? ' (in DLQ)' : ''}`
      });
    }

    // Check for excessive auto-adjustments
    const recentAutoAdjustments = await this.getRecentAutoAdjustments(24); //
24 hours
    if (recentAutoAdjustments.length > 50) {
      await this.sendAlert({
        type: 'EXCESSIVE_AUTO_ADJUSTMENTS',
        severity: 'MEDIUM',
        message: `${recentAutoAdjustments.length} auto-adjustments in last 24
hours - possible data quality issue`
      });
    }

    // Check for DLQ items requiring attention
    const oldDLQItems = await this.getOldDLQItems(3600000); // 1 hour
    if (oldDLQItems.length > 0) {
      await this.sendAlert({
        type: 'DLQ_ITEMS_AGING',
        severity: 'MEDIUM',
        message: `${oldDLQItems.length} items in DLQ for over 1 hour - manual
review needed`
      });
    }

    // Enhanced reconciliation variance check
    const reconciliationVariance = await this.getDailyReconciliationVariance();
    if (Math.abs(reconciliationVariance) > 100) { // $100 variance
      const autoAdjustmentsPossible = Math.abs(reconciliationVariance) < 1000;

      await this.sendAlert({
        type: 'RECONCILIATION_VARIANCE',
        severity: autoAdjustmentsPossible ? 'MEDIUM' : 'HIGH',
        message: `Daily reconciliation variance: $${reconciliationVariance}${
          autoAdjustmentsPossible ? ' - auto-adjustment possible' : ' - manual
review required'
        }`
      });
    }
  }
}
```

Esta versión 2.0 del documento de Problemas y Soluciones incorpora todas las mejoras críticas identificadas, resultando en un sistema significativamente más robusto, confiable y automatizado.