Business Logic

Bilko Business Logic

> **Status:** SPECIFICATION (backend not implemented) **Last updated:** 2026-02-20

Purpose

This document defines the accounting domain rules that Bilko's backend MUST enforce. These are non-negotiable business requirements for financial accuracy and compliance.

Table of Contents

---

1. Double-Entry Bookkeeping

Core Principle

EVERY financial event creates a `Transaction` with exactly one debit and one credit.

The fundamental equation:

```
DEBITS = CREDITS
```

Double-Entry Flow

```
flowchart TD
    EVENT[Financial Event\ne.g. Invoice sent, Expense approved, Payment received]
    EVENT --> TXN[Create Transaction\ndebitAccountId + creditAccountId + amount]

    TXN --> CHK{Validate:\ndebit ≠ credit\namount > 0}
    CHK -->|FAIL| ERR[422 Validation Error]
    CHK -->|PASS| DEBIT[Debit Account\nIncrease if Asset/Expense\nDecrease if
Liability/Equity/Revenue]
    DEBIT --> CREDIT[Credit Account\nIncrease if Liability/Equity/Revenue\nDecrease if
Asset/Expense]
    CREDIT --> BAL{Trial Balance\nSum Debits = Sum Credits?}
    BAL -->|Balanced| LOCK[Lock Transaction\nappend to GL]
    BAL -->|Unbalanced| ALERT[System Alert\nCritical Error]

    style EVENT fill:#00E5A0,color:#000
    style ERR fill:#f87171,color:#fff
    style ALERT fill:#f87171,color:#fff
    style LOCK fill:#60a5fa,color:#000
```

Common Transaction Patterns

```
flowchart LR
    subgraph INV_SENT [Invoice Sent]
        IS_D[Debit: 1200 Accounts Receivable\nAsset ↑]
        IS_C[Credit: 4000 Revenue\nRevenue ↑]
        IS_D -. "amount" .- IS_C
    end

    subgraph INV_PAID [Invoice Paid]
        IP_D[Debit: 1000 Bank Account\nAsset ↑]
        IP_C[Credit: 1200 Accounts Receivable\nAsset ↓]
```

```
        IP_D -. "amount" .- IP_C
    end

    subgraph EXP_APR [Expense Approved]
        EA_D[Debit: 5100 Expense Account\nExpense ↑]
        EA_C[Credit: 2000 Accounts Payable\nLiability ↑]
        EA_D -. "amount" .- EA_C
    end

    subgraph EXP_PAID [Expense Paid]
        EP_D[Debit: 2000 Accounts Payable\nLiability ↓]
        EP_C[Credit: 1000 Bank Account\nAsset ↓]
        EP_D -. "amount" .- EP_C
    end

    style IS_D fill:#4ade80,color:#000
    style IP_D fill:#4ade80,color:#000
    style EA_D fill:#fb923c,color:#000
    style EP_D fill:#4ade80,color:#000
```

Account types and normal balances:

Account Type	Normal Balance	Increases with	Decreases with
Asset	Debit	Debit	Credit
Liability	Credit	Credit	Debit
Equity	Credit	Credit	Debit
Revenue	Credit	Credit	Debit
Expense	Debit	Debit	Credit

Transaction Rules

1. **Debit Account ≠ Credit Account**
   - A transaction cannot debit and credit the same account
   - Enforced at API validation layer
2. **Amount > 0**
   - Transaction amount must be positive
   - Sign is determined by debit/credit, not amount
3. **Balanced Entries**
   - Debit amount = Credit amount
   - No split transactions in MVP (one debit, one credit only)

4. **Locked Transactions**
   - Once `transaction.locked = true`, cannot be edited or deleted
   - Locked at end-of-period close or when reconciled

Common Transaction Patterns

1. Invoice Created (draft ? sent)

```
Debit:  1200 - Accounts Receivable (Asset)    +125,000 RSD
Credit: 4000 - Revenue (Revenue)              +125,000 RSD
```

Effect: Increases asset (money owed to us), increases revenue.

2. Invoice Paid

```
Debit:  1000 - Bank Account (Asset)           +125,000 RSD
Credit: 1200 - Accounts Receivable (Asset)    -125,000 RSD
```

Effect: Increases cash, decreases receivables (converted to cash).

3. Expense Approved

```
Debit:  5100 - Infrastructure Expense (Expense)  +850 EUR
Credit: 2000 - Accounts Payable (Liability)      +850 EUR
```

Effect: Increases expense, increases liability (we owe money).

4. Expense Paid

```
Debit:  2000 - Accounts Payable (Liability)   -850 EUR
Credit: 1000 - Bank Account (Asset)           -850 EUR
```

Effect: Decreases liability, decreases cash.

Balance Calculation

Account balance = Sum(debits) - Sum(credits) for **debit-normal accounts** (Asset, Expense)

Account balance = Sum(credits) - Sum(debits) for **credit-normal accounts** (Liability, Equity, Revenue)

Trial Balance:

- Sum of all debit balances = Sum of all credit balances
- If unbalanced, there is an error in the ledger

---

2. Invoice Workflow

Status Transitions

```
draft → sent → viewed → paid
  ↓       ↓       ↓
  └──────→ cancelled
```

```
stateDiagram-v2
    [*] --> draft : POST /invoices\n(auto-number: INV-YYYY-NNN)

    draft --> sent : PATCH /invoices/:id/status\naction=send\n[generates PDF → R2]\n[sends email via SendGrid]\n[creates Transaction:\nDR Receivable / CR Revenue]

    sent --> viewed : Email tracking pixel loaded\n[updates invoice.viewedAt]

    viewed --> paid : PATCH status action=mark-paid\n[creates Transaction:\nDR Bank / CR Receivable]

    sent --> paid : PATCH status action=mark-paid\n[creates Transaction:\nDR Bank / CR Receivable]

    draft --> cancelled : PATCH status action=cancel
    sent --> cancelled : PATCH status action=cancel\n[reverses Transaction]
    viewed --> cancelled : PATCH status action=cancel\n[reverses Transaction]

    paid --> [*]
    cancelled --> [*]

    note right of draft
        Editable: items, dates, amounts
        Invoice number locked on first save
    end note
```

```
    note right of sent
        LOCKED — cannot edit amounts
        PDF stored in Cloudflare R2
        exchangeRate locked at invoiceDate
    end note

    note right of paid
        2 GL Transactions created total:
        1. draft→sent: DR Receivable / CR Revenue
        2. paid: DR Bank / CR Receivable
    end note
```

Invoice Calculation Flow

```
flowchart TD
    ITEMS[Invoice Items\nquantity × unitPrice = lineTotal]
    ITEMS --> SUB[subtotal = SUM all lineTotals]
    SUB --> TAX[taxAmount = SUM lineTotal × taxRate/100]
    TAX --> DISC[Apply discountAmount]
    DISC --> TOTAL[totalAmount = subtotal + taxAmount - discountAmount]
    TOTAL --> BASE[baseAmount = totalAmount × exchangeRate\nexchangeRate locked at
invoiceDate]
    BASE --> LOCK[Store — NEVER recalculate\nfrom future exchange rates]

    style LOCK fill:#f87171,color:#fff
    style BASE fill:#ffd700,color:#000
```

Status rules:

From	To	Action	Transaction Created?
draft	sent	Send email	Yes (Debit Receivable, Credit Revenue)
sent	viewed	Email opened	No
viewed	paid	Mark paid	Yes (Debit Bank, Credit Receivable)
sent	paid	Mark paid	Yes (Debit Bank, Credit Receivable)
any	cancelled	Cancel	Reverses original transaction

Business Rules

Rule 1: Invoice Number Auto-Generation

- Format: `INV-YYYY-NNN` (e.g., `INV-2026-001`)
- Generated on first save (when status changes from null → draft)
- Sequential within organization per year
- NEVER reuse cancelled invoice numbers

Rule 2: Draft-Only Editing

- Can only edit invoice if `status = 'draft'`
- Once sent, cannot change line items or amounts
- Can still update notes/terms

Rule 3: Overdue Detection

- Invoice becomes `overdue` if `dueDate < today AND status != 'paid'`
- Checked automatically via scheduled job (daily at 00:00 UTC)

Rule 4: Subtotal Calculation

```
subtotal = SUM(lineTotal) for all invoice items
lineTotal = quantity * unitPrice
```

Rule 5: Tax Calculation

```
taxAmount = SUM(lineTotal * (taxRate / 100)) for all items
```

Rule 6: Total Calculation

```
totalAmount = subtotal + taxAmount - discountAmount
```

Rule 7: Base Amount Conversion

```
baseAmount = totalAmount * exchangeRate
```

- `exchangeRate` locked at `invoiceDate`
- NEVER recalculated

Rule 8: PDF Generation

- PDF generated when status → sent
- Stored in Cloudflare R2
- URL saved to `invoice.pdfUrl`

- PDF includes: org branding, line items, tax breakdown, payment terms

Rule 9: Email Delivery

- Sent to `contact.email`
- Subject: "Invoice [invoiceNumber] from [organizationName]"
- Attachment: PDF
- Tracking pixel for `viewedAt` timestamp

---

3. Expense Workflow

Status Transitions

```
pending → approved → paid
   ↓
rejected
```

```
stateDiagram-v2
    [*] --> pending : POST /expenses\n(auto-number: EXP-YYYY-NNN)\ncreatedBy: accountant/admin/owner

    pending --> approved : PATCH /expenses/:id/approve\nRoles: owner, admin ONLY\n[creates Transaction:\nDR Expense / CR Accounts Payable]

    pending --> rejected : PATCH /expenses/:id/reject\nRoles: owner, admin ONLY\n[no Transaction created]

    approved --> paid : PATCH /expenses/:id/pay\n[creates Transaction:\nDR Accounts Payable / CR Bank]

    paid --> [*]
    rejected --> [*]

    note right of pending
        Can be edited before approval
        Receipt upload optional (max 10MB)
        PDF/PNG/JPG formats
    end note
```

```
    note right of approved
        Cannot edit after approval
        Stored in Cloudflare R2 receipts/
        exchangeRate locked at expenseDate
    end note
```

Status rules:

From	To	Action	Transaction Created?
pending	approved	Approve	Yes (Debit Expense, Credit Payable)
pending	rejected	Reject	No
approved	paid	Mark paid	Yes (Debit Payable, Credit Bank)

Business Rules

Rule 1: Expense Number Auto-Generation

- Format: `EXP-YYYY-NNN` (e.g., `EXP-2026-001`)
- Generated on creation
- Sequential within organization per year

Rule 2: Approval Required

- Expenses created with `status = 'pending'`
- Only `owner` or `admin` can approve
- `accountant` can create but cannot approve
- Once approved, cannot be edited

Rule 3: Receipt Upload

- Optional but recommended
- Max file size: 10MB
- Allowed formats: PDF, PNG, JPG
- Stored in Cloudflare R2
- URL saved to `expense.receiptUrl`

Rule 4: Category Tracking

- Free-text category field
- Common categories suggested: Infrastructure, Software, Office, Travel, Marketing, Utilities
- Used for expense reports by category

Rule 5: Tax Amount

- Optional `taxAmount` field
- If provided, represents input VAT (can be deducted from output VAT)
- Used in VAT report

Rule 6: Base Amount Conversion

```
baseAmount = amount * exchangeRate
```

- `exchangeRate` locked at `expenseDate`

---

4. VAT Calculation

VAT Calculation Flow

```
flowchart TD
    subgraph OUTPUT [Output VAT — Sales]
        INV[Invoice sent to customer]
        INV --> OLINE[For each line item:\nlineTotal = qty × unitPrice\nlineTaxAmount =
lineTotal × taxRate/100]
        OLINE --> OTOT[Invoice taxAmount = SUM all lineTaxAmounts]
        OTOT --> OREC[Recorded as Output VAT\nin VAT Report]
    end

    subgraph INPUT [Input VAT — Purchases]
        EXP[Expense from vendor]
        EXP --> ETAX[expense.taxAmount field\nUser-entered or calculated]
        ETAX --> IREC[Recorded as Input VAT\nin VAT Report]
    end

    subgraph NET [Net VAT Calculation]
        OREC --> CALC[netVAT = outputVAT - inputVAT]
        IREC --> CALC
        CALC --> POS{netVAT > 0?}
        POS -->|Yes| OWE[Owe to tax authority\nFile PDV/VAT return]
        POS -->|No| REF[Tax authority owes refund\nRare for SMBs]
    end
```

```
    style OWE fill:#f87171,color:#fff
    style REF fill:#4ade80,color:#000
```

VAT Rates by Country

Country	Standard VAT	Reduced VAT	Zero VAT
Serbia (RS)	20%	10%	0%
BiH (BA)	17%	-	0%
Croatia (HR)	25%	13%	0%

Business Rules

Rule 1: Tax Rate Application

- Invoice items have `taxRate` field (percentage)
- Default to organization's country standard rate
- User can override per line item

Rule 2: Tax Amount Calculation

```
For each invoice item:
  lineTotal = quantity * unitPrice
  lineTaxAmount = lineTotal * (taxRate / 100)

For invoice:
  subtotal = SUM(lineTotal)
  taxAmount = SUM(lineTaxAmount)
  totalAmount = subtotal + taxAmount - discountAmount
```

Rule 3: Output VAT (Sales)

- VAT collected on invoices sent to customers
- Recorded when invoice status → sent
- Included in VAT report as "Output VAT"

Rule 4: Input VAT (Purchases)

- VAT paid on expenses from vendors
- Recorded from `expense.taxAmount` field
- Included in VAT report as "Input VAT"

Rule 5: Net VAT Calculation

```
netVAT = outputVAT - inputVAT
```

- If positive: owe VAT to tax authority
- If negative: tax authority owes refund (rare for small businesses)

VAT Report Structure

```
interface VATReport {
  period: { from: string, to: string }

  outputVAT: {
    total: Decimal                    // Total VAT collected
    invoices: Array<{
      invoiceNumber: string
      customerName: string
      invoiceDate: string
      baseAmount: Decimal             // Subtotal
      vatAmount: Decimal              // Tax amount
      vatRate: Decimal                // Tax rate %
    }>
  }

  inputVAT: {
    total: Decimal                    // Total VAT paid
    expenses: Array<{
      expenseNumber: string
      vendorName: string
      expenseDate: string
      baseAmount: Decimal
      vatAmount: Decimal
      vatRate: Decimal
    }>
  }

  netVAT: Decimal                     // outputVAT - inputVAT

  reconciliationStatus: {
    allInvoicesPaid: boolean          // All invoices in period are paid
```

```
    allExpensesApproved: boolean      // All expenses in period are approved
    unmatchedTransactions: number     // Unreconciled bank transactions
  }
}
```

---

5. Multi-Currency

Supported Currencies

MVP:

- EUR (Euro) — default
- RSD (Serbian Dinar)
- BAM (Bosnian Mark)
- HRK (Croatian Kuna)
- USD (US Dollar)

Exchange Rate Locking

CRITICAL RULE: Exchange rates are locked at transaction date.

Why:

- Financial reports must be consistent over time
- Cannot recalculate historical transactions with current rates
- Accounting standards require rate at transaction date

How it works:

1. **Invoice created on 2026-02-20:**
   - `currencyCode = 'RSD'`
   - `exchangeRate = 117.50` (EUR to RSD rate on 2026-02-20)
   - `totalAmount = 125,000 RSD`
   - `baseAmount = 125,000 / 117.50 = 1,063.83 EUR` (locked)
2. **Today (2026-03-15), rate is now 120.00:**
   - Invoice `baseAmount` stays 1,063.83 EUR
   - NEVER recalculated to `125,000 / 120.00 = 1,041.67 EUR`

Exchange Rate Sources

Primary: European Central Bank (ECB) API

- Free
- Daily updates
- Reliable

Fallback: fixer.io API

- Freemium (1000 requests/month free)
- Real-time rates

Manual Entry:

- If API unavailable, user can enter rate manually
- Stored with `source = 'manual'`

Base Currency Conversion

All reports displayed in organization's `baseCurrency`.

Example:

- Organization baseCurrency = EUR
- Invoice 1: 125,000 RSD → 1,063.83 EUR (rate 117.50)
- Invoice 2: 3,500 EUR → 3,500 EUR (rate 1.0)
- Expense 1: 850 USD → 794.39 EUR (rate 1.07)

Total Revenue: 1,063.83 + 3,500 = 4,563.83 EUR

---

6. Bank Reconciliation

Purpose

Match bank transactions (from statements) to general ledger transactions (from invoices/expenses).

```
flowchart TD
    CSV[Bank Statement CSV\nDate, Description, Amount, Reference]
    CSV --> PARSE[Parse & validate CSV\nCreate BankTransaction records]
    PARSE --> LINK[Link to BankAccount]

    LINK --> MATCH[Auto-Match Algorithm\nScore 0-100]
```

```
    subgraph SCORE [Match Score Calculation]
        S1[+50 pts: Exact amount match]
        S2[+30 pts: Same date\n+20 pts: ±1 day\n+10 pts: ±3 days]
        S3[+20 pts: Reference contains\ninvoice/expense number]
    end

    MATCH --> SCORE
    SCORE --> THRESH{Score?}

    THRESH -->|≥ 90| AUTO[Auto-match\nreconciled = true]
    THRESH -->|70-89| SUGGEST[Suggest to user\nUser confirms]
    THRESH -->|< 70| MANUAL[Manual review\nUser links manually]

    SUGGEST --> CONFIRM{User\nconfirms?}
    CONFIRM -->|Yes| RECONCILE[Set reconciled = true\nmatchedTransactionId = glTxId]
    CONFIRM -->|No| MANUAL

    MANUAL --> RECONCILE

    AUTO --> RECONCILE
    RECONCILE --> REPORT[Reconciliation Report\nbalanceDiscrepancy should = 0]

    style AUTO fill:#4ade80,color:#000
    style MANUAL fill:#fb923c,color:#000
    style REPORT fill:#60a5fa,color:#000
```

Process

1. **Import bank statement (CSV):**
   - Parse CSV file
   - Create `BankTransaction` records
   - Link to `BankAccount`
2. **Auto-match transactions:**
   - Match by amount + date (within ±3 days)
   - Match by reference (invoice number in description)
   - Calculate confidence score (0-100)
3. **Manual reconciliation:**
   - User links `BankTransaction` to `Transaction`
   - Set `bankTransaction.reconciled = true`
   - Set `bankTransaction.matchedTransactionId = transaction.id`
4. **Unmatched transactions:**

- Flag in reconciliation report
- User must create manual journal entry or mark as miscellaneous

Matching Algorithm

Score calculation:

```
function calculateMatchScore(
  bankTx: BankTransaction,
  glTx: Transaction
): number {
  let score = 0

  // Exact amount match
  if (Math.abs(bankTx.amount) === glTx.amount) {
    score += 50
  }

  // Date within ±3 days
  const daysDiff = Math.abs(
    daysBetween(bankTx.transactionDate, glTx.transactionDate)
  )
  if (daysDiff === 0) score += 30
  else if (daysDiff <= 1) score += 20
  else if (daysDiff <= 3) score += 10

  // Reference contains invoice/expense number
  if (glTx.referenceType === 'invoice' && bankTx.description?.includes(glTx.referenceId)) {
    score += 20
  }

  return score
}
```

Auto-match threshold:

- Score ≥ 90 → Auto-match
- Score 70-89 → Suggest match (user confirms)
- Score < 70 → No match suggested

Reconciliation Report

```
interface ReconciliationReport {
  bankAccount: {
    id: string
    name: string
    currentBalance: Decimal
  }

  period: { from: string, to: string }

  bankTransactions: {
    total: number
    reconciled: number
    unreconciled: number
    totalAmount: Decimal
  }

  glTransactions: {
    total: number
    reconciled: number
    unreconciled: number
    totalAmount: Decimal
  }

  unmatchedBankTransactions: Array<BankTransaction>
  unmatchedGLTransactions: Array<Transaction>

  balanceDiscrepancy: Decimal       // Should be 0 when fully reconciled
}
```

---

7. Chart of Accounts

Structure

Hierarchical account codes:

- 1xxx = Assets
- 2xxx = Liabilities
- 3xxx = Equity
- 4xxx = Revenue
- 5xxx = Expenses

Example Serbian Chart of Accounts:

```
1000  Assets
  1100  Current Assets
    1110  Cash
    1120  Bank Accounts
      1121  Intesa RSD Account
      1122  Raiffeisen EUR Account
    1200  Accounts Receivable
  1500  Fixed Assets
    1510  Equipment
    1520  Vehicles

2000  Liabilities
  2100  Current Liabilities
    2110  Accounts Payable
    2120  VAT Payable
  2500  Long-term Liabilities
    2510  Loans Payable

3000  Equity
  3100  Share Capital
  3900  Retained Earnings

4000  Revenue
  4100  Service Revenue
  4200  Product Sales

5000  Expenses
  5100  Operating Expenses
    5110  Salaries
    5120  Rent
    5130  Utilities
  5200  Cost of Goods Sold
```

Business Rules

Rule 1: Account Hierarchy

- Parent account codes must exist before creating child accounts
- Cannot delete parent account if child accounts exist
- Sub-account balance rolls up to parent

Rule 2: Account Deactivation

- Cannot deactivate account with transactions
- Deactivated accounts hidden from dropdowns but visible in reports

Rule 3: Reserved Accounts

- System creates default accounts on organization registration
- Cannot delete: Cash, Bank Account, Accounts Receivable, Accounts Payable, Revenue, Expense

---

8. Fiscal Year

Definition

Fiscal year: 12-month period for financial reporting.

Default: January 1 - December 31

Configurable: Organization can set custom fiscal year start (e.g., April 1 for UK-style fiscal year)

Business Rules

Rule 1: Year-End Close

- At fiscal year end, close Revenue and Expense accounts
- Transfer net profit/loss to Retained Earnings
- Lock all transactions for closed fiscal year
- Cannot edit locked transactions

Rule 2: Period-Based Reports

- Profit & Loss: always for a period (from → to)
- Balance Sheet: always as of a date (point in time)

- Cash Flow: always for a period

---

9. Audit Trail

Purpose

Immutable log of all data changes for:

- Compliance (GDPR, financial regulations)
- Debugging (track down errors)
- Rollback simulation (undo mistakes)

What is Logged

ALL INSERT/UPDATE/DELETE operations on:

- Invoices
- Expenses
- Transactions
- Contacts
- Users
- Organization settings

Captured data:

- Table name
- User ID (who made the change)
- Timestamp
- Action (INSERT, UPDATE, DELETE)
- Old values (before change)
- New values (after change)
- Client IP address

Implementation

Via Prisma Middleware:

```
prisma.$use(async (params, next) => {
  const result = await next(params)
```

```
  if (['create', 'update', 'delete'].includes(params.action)) {
    await prisma.loggedAction.create({
      data: {
        tableName: params.model,
        userId: getCurrentUserId(),
        action: params.action.toUpperCase(),
        rowData: params.action === 'delete' ? params.args.where : null,
        changedFields: params.action === 'update' ? params.args.data : null,
        clientIp: getClientIp(),
        applicationName: 'bilko-api'
      }
    })
  }

  return result
})
```

Retention Policy

- Audit logs retained for **7 years** (financial compliance requirement)
- After 7 years, archived to cold storage (AWS S3 Glacier)
- NEVER deleted

Summary of Critical Business Rules

1. **Double-entry:** Every transaction has one debit and one credit
2. **Debits = Credits:** Ledger must always balance
3. **Exchange rate locking:** Rates locked at transaction date, NEVER recalculated
4. **Invoice workflow:** draft → sent → paid (creates 2 transactions)
5. **Expense workflow:** pending → approved → paid (creates 2 transactions)
6. **VAT calculation:** `taxAmount = lineTotal * (taxRate / 100)`
7. **Account hierarchy:** Parent-child relationships in Chart of Accounts
8. **Audit trail:** ALL changes logged immutably
9. **Fiscal year close:** Lock transactions, transfer P&L to Retained Earnings
10. **Reconciliation:** Match bank transactions to GL transactions

End of Business Logic Documentation

Revision #3
Created 2026-02-23 10:47:53 UTC by John
Updated 2026-05-31 20:02:37 UTC by John