Test Plan

Bilko — Test Plan

Version: 1.0 **Date:** 2026-02-23 **Project ID:** bbd77cc0 **Status:** Current — reflects actual codebase and target testing strategy as of 2026-02-23

Table of Contents

---

1. Test Philosophy

1.1 Existing Tests

The `@bilko/core` package has unit tests written with **Vitest**:

Test File	Coverage
`packages/core/tests/accounting.test.ts`	`validateDoubleEntry`, `createJournalEntry`, `calculateTrialBalance`

Test File	Coverage
`packages/core/tests/tax.test.ts`	`calculateVAT`, `getDefaultVATRate`, `getVATRates`, `calculateNetFromGross`, `calculateCIT`
`packages/core/tests/multi-currency.test.ts`	`convertCurrency`, `lockExchangeRate`, `calculateForexGainLoss`
`packages/core/tests/invoicing.test.ts`	Invoice calculation helpers
`packages/core/tests/chart-of-accounts.test.ts`	Chart structure validation

1.2 Test Pyramid

```
          ┌──────────┐
          │   E2E    │  ← Fewer, slower, critical user flows
          │  Tests   │
          └────┬─────┘
          ┌────┴─────┐
          │  Integ   │  ← API endpoints with real test DB
          │  Tests   │
          └────┬─────┘
     ┌─────────┴──────────┐
     │     Unit Tests     │  ← Core engine, services, validators (fast, many)
     └────────────────────┘
```

```
graph TD
    subgraph PYRAMID["Bilko Test Pyramid"]
        E2E["E2E Tests — 10%<br/>Playwright<br/>5 critical flows<br/>Staging environment<br/>~60s per test"]
        INT["Integration Tests — 30%<br/>Supertest + Vitest<br/>Real PostgreSQL<br/>API endpoints<br/>~5s per test"]
        UNIT["Unit Tests — 60%<br/>Vitest<br/>@bilko/core engine<br/>Pure functions<br/>~50ms per test"]
    end

    E2E --> INT
    INT --> UNIT

    UNIT --> U1["accounting.test.ts"]
    UNIT --> U2["tax.test.ts"]
    UNIT --> U3["multi-currency.test.ts"]
```

```
    UNIT --> U4["bank-import.test.ts"]
    UNIT --> U5["chart-of-accounts.test.ts"]

    INT --> I1["auth.test.ts"]
    INT --> I2["invoices.test.ts"]
    INT --> I3["expenses.test.ts"]
    INT --> I4["reports.test.ts"]
    INT --> I5["isolation.test.ts"]

    E2E --> E1["invoice-lifecycle.spec.ts"]
    E2E --> E2["expense-flow.spec.ts"]
    E2E --> E3["bank-reconciliation.spec.ts"]
    E2E --> E4["reports.spec.ts"]
    E2E --> E5["auth.spec.ts"]

    style PYRAMID fill:#f8f9fa,stroke:#dee2e6
    style E2E fill:#dc3545,color:#fff,stroke:#c82333
    style INT fill:#fd7e14,color:#fff,stroke:#e8690b
    style UNIT fill:#198754,color:#fff,stroke:#157347
```

1.3 Non-Negotiable Rules

1. **Money is never JavaScript `number`** — all monetary tests use `Decimal.js` or string assertions
2. **Double-entry always balanced** — every test that creates a financial transaction verifies debit = credit
3. **Organization isolation** — cross-org data access must be impossible (tested explicitly)
4. **Immutability** — locked transactions cannot be modified (must throw/fail)
5. **Audit trail** — mutations must create `LoggedAction` entries (tested in integration)

---

2. Unit Test Strategy

Framework: Vitest (already configured in `packages/core/vitest.config.ts`) **Run:** `cd packages/core && npx vitest`

2.1 Core Accounting Engine (`@bilko/core`)

`accounting/index.ts` — Double-Entry Engine

File: `packages/core/tests/accounting.test.ts` (EXISTS)

Test Case	Assertion
Balanced entry: debit = credit	`validateDoubleEntry` returns `true`
Unbalanced entry: debit ≠ credit	Returns `false`
Less than 2 lines	Returns `false`
Negative amounts	Returns `false`
Zero amounts	Returns `false`
Multiple lines summing to balanced	Returns `true`
Decimal amounts with 4dp precision	Returns `true`
`createJournalEntry` with valid data	Returns entry unchanged
Missing description	Throws "must have a description"
Missing date	Throws "must have a date"
Unbalanced amounts in error message	Error shows actual debit/credit totals
`calculateTrialBalance` from balanced entries	`isBalanced = true`, sums correct
`calculateTrialBalance` groups by account number	Same account accumulated correctly
`calculateTrialBalance` empty input	`isBalanced = true`, empty rows
`calculateTrialBalance` sorts by account number	Rows sorted ascending

Additional tests needed:

```
describe('Immutable transaction locking', () => {
  it('locked transactions cannot have amount changed');
  it('locked transactions cannot change debit/credit accounts');
  it('locked = true after period close');
});
```

`tax/index.ts` — VAT/CIT Calculator

File: `packages/core/tests/tax.test.ts` (EXISTS)

Test Case	Assertion
Serbia PDV 20% on 1000	base=1000, tax=200, total=1200
BiH PDV 17% on 1000	base=1000, tax=170, total=1170
Croatia PDV 25% on 1000	base=1000, tax=250, total=1250
Zero rate	tax=0, total=base

Test Case	Assertion
Decimal base amounts (123.45 at 20%)	tax=24.69, total=148.14
Negative amount	Throws "non-negative"
Negative rate	Throws "non-negative"
Large amounts (999,999,999.9999)	No precision loss
`Decimal` input accepted	Same result as string
`getDefaultVATRate('RS')`	Returns 20
`getDefaultVATRate('BA')`	Returns 17
`getDefaultVATRate('HR')`	Returns 25
Unsupported country	Throws "Unsupported country"
`getVATRates('RS')`	3 rates: 20, 10, 0
`getVATRates('BA')`	2 rates: 17, 0
`getVATRates('HR')`	3 rates: 25, 13, 0
Returns copies (immutable)	Mutation doesn't affect originals
Reverse VAT (BiH 1170 gross)	base≈1000, tax≈170
CIT at 15%	100000 → 15000

Additional tests needed (country modules):

```
// packages/country-rs/src/tax/index.ts
describe('Serbian tax specifics', () => {
  it('calculateSerbianPDV standard 20%');
  it('calculateSerbianPDV reduced 10%');
  it('calculateSerbianPDV zero rate');
  it('calculateSerbianCIT 15% flat');
  it('qualifiesForPausalRegime: revenue < 6M RSD → true');
  it('qualifiesForPausalRegime: revenue >= 6M RSD → false');
  it('requiresVATRegistration: revenue >= 8M RSD → true');
  it('requiresVATRegistration: revenue < 8M RSD → false');
});

// packages/country-ba/src/tax/index.ts
describe('Bosnian tax specifics', () => {
  it('calculateBosnianPDV single 17% rate');
  it('calculateCITFBiH 10%');
  it('calculateCITRS 10%');
  it('calculateDividendWHT FBiH: dividends 5%');
```

```
  it('calculateDividendWHT RS: dividends 10%');
  it('requiresVATRegistration: >= 100000 BAM → true');
});

// packages/country-hr/src/tax/index.ts
describe('Croatian tax specifics', () => {
  it('calculateCroatianPDV standard 25%');
  it('calculateCroatianPDV reduced 13%');
  it('calculateCroatianPDV superReduced 5%');
  it('calculateCroatianCIT: revenue < 1M EUR → 10%');
  it('calculateCroatianCIT: revenue >= 1M EUR → 18%');
  it('qualifiesForPausalni: revenue < 60000 EUR → true');
  it('requiresVATRegistration: revenue >= 60000 EUR → true');
});
```

`multi-currency/index.ts` — Currency Conversion

File: `packages/core/tests/multi-currency.test.ts` (EXISTS)

Test Case	Assertion
Same currency	Rate = 1, no conversion
RSD to EUR at rate 0.0086	Correct base amount
`lockExchangeRate` returns ExchangeRate object	Correct fields
`lockExchangeRate` same currency	Throws error
`lockExchangeRate` rate ≤ 0	Throws "must be positive"
`convertCurrency` with zero fromRate	Throws
`calculateForexGainLoss` gain scenario	`gain > 0`, `loss = 0`
`calculateForexGainLoss` loss scenario	`gain = 0`, `loss > 0`
`isSupportedCurrency('EUR')`	`true`
`isSupportedCurrency('XYZ')`	`false`
Precision: toFixed(4) on result	4 decimal places

`bank-import/index.ts` — CSV Parser

File: `packages/core/tests/bank-import.test.ts` (MISSING — needs creation)

```
describe('parseCSV', () => {
  it('parses ISO date format YYYY-MM-DD');
  it('parses Balkan dot format DD.MM.YYYY');
  it('parses slash format DD/MM/YYYY');
  it('skips header line');
  it('skips empty lines');
  it('returns empty array for empty string');
  it('returns empty array for header-only CSV');
  it('sets direction: inbound by default');
  it('sets direction: outbound when field is "outbound"');
  it('handles quoted fields with commas');
  it('generates deterministic IDs for dedup');
});

describe('detectDuplicates', () => {
  it('detects exact duplicate by date+amount+currency+reference');
  it('returns empty array when no duplicates');
  it('returns empty array when either list is empty');
  it('does NOT flag as duplicate if amount differs');
  it('does NOT flag as duplicate if date differs');
  it('does NOT flag as duplicate if reference differs (but amount/date same)');
});
```

2.2 Validator Unit Tests

Framework: Vitest **Location:** `apps/api/src/validators/*.ts`

```
describe('Invoice validators (createInvoiceSchema)', () => {
  it('valid invoice passes');
  it('missing customerId fails');
  it('invalid date format fails');
  it('negative unitPrice fails');
  it('empty items array fails');
  it('taxRate > 100 fails');
  it('invalid currencyCode (5 chars) fails');
  it('invalid UUID for customerId fails');
});

describe('Auth validators (registerSchema)', () => {
```

```
  it('valid registration passes');
  it('invalid email fails');
  it('password too short fails (< 8 chars)');
  it('invalid country code (not RS/BA/HR) fails');
  it('missing organizationName fails');
});
```

Integration Test Architecture

```
sequenceDiagram
    participant TC as Test Case
    participant ST as Supertest
    participant APP as Express App
    participant MID as Middleware<br/>(Auth + RBAC)
    participant SVC as Service Layer
    participant PRI as Prisma ORM
    participant DB as Test PostgreSQL

    TC->>ST: HTTP request + Bearer token
    ST->>APP: Forward request
    APP->>MID: Authenticate JWT
    MID->>MID: Verify organizationId scope
    MID->>SVC: Authorized request
    SVC->>PRI: DB query (org-scoped)
    PRI->>DB: Parameterized SQL
    DB-->>PRI: Result rows
    PRI-->>SVC: Typed objects
    SVC-->>APP: Response data
    APP-->>ST: HTTP response
    ST-->>TC: Assert status + body

    Note over DB: beforeEach: seed<br/>afterEach: truncate<br/>(reverse FK order)
```

---

3. Integration Test Strategy

Framework: Supertest + Vitest (or Jest) **Database:** Test PostgreSQL instance (separate from dev/prod) **Setup:** Prisma migrations applied before tests; data seeded per test suite; truncated after each test

3.1 Test Database Setup

```
// test/setup.ts
import { prisma } from '../src/lib/prisma';
import { execSync } from 'child_process';

beforeAll(async () => {
  // Apply migrations to test DB
  execSync('npx prisma migrate deploy', { env: { DATABASE_URL: process.env.TEST_DATABASE_URL }
});
});

beforeEach(async () => {
  // Seed minimal data: 1 org, 1 owner user, default accounts
  await seedTestOrg();
});

afterEach(async () => {
  // Clean up in reverse FK order
  await prisma.loggedAction.deleteMany();
  await prisma.bankTransaction.deleteMany();
  await prisma.bankAccount.deleteMany();
  await prisma.transaction.deleteMany();
  await prisma.invoiceItem.deleteMany();
  await prisma.invoice.deleteMany();
  await prisma.expense.deleteMany();
  await prisma.contact.deleteMany();
  await prisma.account.deleteMany();
  await prisma.user.deleteMany();
  await prisma.organization.deleteMany();
});
```

3.2 Auth Endpoints

```
describe('POST /api/v1/auth/register', () => {
  it('creates org + owner user, returns tokens');
  it('returns 409 for duplicate email');
  it('returns 400 for missing required fields');
  it('password is hashed (not stored plain)');
  it('sets refreshToken httpOnly cookie');
  it('org baseCurrency defaults to EUR');
});

describe('POST /api/v1/auth/login', () => {
  it('returns accessToken + sets cookie on valid credentials');
  it('returns 401 for wrong password');
  it('returns 401 for non-existent email');
  it('updates lastLoginAt on success');
  it('rememberMe=true extends cookie to 30 days');
});

describe('POST /api/v1/auth/refresh', () => {
  it('returns new accessToken from valid refresh cookie');
  it('returns 401 when no cookie');
  it('returns 401 for expired refresh token');
});

describe('POST /api/v1/auth/logout', () => {
  it('clears refreshToken cookie');
  it('returns 204');
});

describe('GET /api/v1/auth/me', () => {
  it('returns user + org data for valid token');
  it('returns 401 for missing token');
  it('returns 401 for expired token');
});
```

3.3 Invoice Endpoints

```
describe('GET /api/v1/invoices', () => {
  it('returns paginated invoices for organization');
  it('does NOT return invoices from other orgs');
```

```
  it('filters by status');
  it('filters by customerId');
  it('filters by date range');
  it('returns empty data array when no invoices');
  it('returns 401 without auth');
});

describe('POST /api/v1/invoices', () => {
  it('creates invoice in draft status');
  it('auto-generates invoice number INV-YYYY-001');
  it('increments invoice number sequentially');
  it('calculates subtotal correctly from line items');
  it('calculates taxAmount at specified rate');
  it('sets baseAmount = totalAmount when currency = baseCurrency');
  it('locks exchange rate from ExchangeRate table');
  it('returns 404 for non-existent customerId');
  it('returns 400 for contact that is vendor only (not customer)');
});

describe('PATCH /api/v1/invoices/:id/status → send', () => {
  it('changes status from draft to sent');
  it('creates Transaction: DR Receivable / CR Revenue');
  it('transaction.amount = invoice.totalAmount');
  it('transaction.referenceType = invoice, referenceId = invoice.id');
  it('returns 400 if invoice already sent');
  it('returns 400 if required accounts not in chart of accounts');
});

describe('PATCH /api/v1/invoices/:id/status → mark-paid', () => {
  it('changes status from sent to paid');
  it('creates Transaction: DR Bank / CR Receivable');
  it('sets paidAt to provided date');
  it('returns 400 if invoice is still draft');
});

describe('DELETE /api/v1/invoices/:id', () => {
  it('deletes draft invoice');
  it('returns 400 when trying to delete sent invoice');
  it('returns 404 for non-existent invoice');
```

```
  it('cannot delete invoice from another org');
});
```

3.4 Expense Endpoints

```
describe('POST /api/v1/expenses', () => {
  it('creates expense in pending status');
  it('auto-generates expense number EXP-YYYY-001');
  it('stores taxAmount separately');
  it('locks exchange rate at expenseDate');
});

describe('PATCH /api/v1/expenses/:id/approve', () => {
  it('changes status from pending to approved');
  it('creates Transaction: DR Expense / CR Payable');
  it('returns 400 for non-pending expense');
});

describe('PATCH /api/v1/expenses/:id/pay', () => {
  it('changes status from approved to paid');
  it('creates Transaction: DR Payable / CR Bank');
  it('returns 400 for non-approved expense');
});
```

3.5 Transaction Endpoints

```
describe('POST /api/v1/transactions (manual journal)', () => {
  it('accountant can create manual transaction');
  it('viewer cannot create manual transaction (403)');
  it('debit and credit account must be different (422)');
  it('debit account must belong to same org (404)');
  it('credit account must belong to same org (404)');
  it('creates transaction with correct amounts');
  it('referenceType = manual');
});

describe('GET /api/v1/transactions', () => {
  it('filters by accountId (both debit and credit sides)');
```

```
  it('filters by referenceType');
  it('filters by date range');
  it('does not return transactions from other orgs');
});
```

3.6 Report Endpoints

```
describe('GET /api/v1/reports/trial-balance', () => {
  it('returns balanced trial balance (totalDebits = totalCredits)');
  it('returns balanced = true when no transactions');
  it('includes all accounts with transactions');
  it('debit-normal accounts: balance = debit - credit');
  it('credit-normal accounts: balance = credit - debit');
});

describe('GET /api/v1/reports/profit-loss', () => {
  it('revenue accounts (type=4) in revenue section');
  it('expense accounts (type=5) in expenses section');
  it('netProfit = revenue - expenses');
  it('respects date range filter');
});

describe('GET /api/v1/reports/vat', () => {
  it('outputVAT sum from invoice.taxAmount for sent/paid invoices');
  it('inputVAT sum from expense.taxAmount for approved/paid expenses');
  it('netVAT = outputVAT - inputVAT');
  it('draft invoices excluded from output VAT');
  it('pending expenses excluded from input VAT');
});
```

3.7 Multi-Tenancy Isolation Tests

```
describe('Organization isolation', () => {
  let org1Token: string;
  let org2Token: string;
  let org1InvoiceId: string;

  beforeEach(async () => {
    // Create two separate organizations
```

```
    org1Token = await registerAndLogin('org1@test.rs');
    org2Token = await registerAndLogin('org2@test.rs');
    // Create invoice in org1
    const res = await createInvoice(org1Token, { ... });
    org1InvoiceId = res.body.id;
  });

  it('org2 cannot GET invoice from org1 (returns 404)');
  it('org2 cannot PUT invoice from org1 (returns 404)');
  it('org2 cannot DELETE invoice from org1 (returns 404 or 404)');
  it('org2 list invoices does not include org1 invoices');
  it('org2 cannot GET org1 contacts');
  it('org2 cannot GET org1 transactions');
  it('org2 cannot GET org1 bank accounts');
  it('org2 trial balance does not include org1 accounts');
});
```

Invoice Lifecycle — Integration Test Flow

```
stateDiagram-v2
    [*] --> Draft: POST /api/v1/invoices<br/>(test: creates draft, generates INV-YYYY-NNN)

    Draft --> Sent: PATCH /status → send<br/>(test: DR Receivable / CR Revenue)
    Draft --> Deleted: DELETE /invoices/:id<br/>(test: draft can be deleted)
    Sent --> Paid: PATCH /status → mark-paid<br/>(test: DR Bank / CR Receivable)
    Sent --> Deleted_ERR: DELETE attempt<br/>(test: returns 400 — cannot delete sent)
    Paid --> [*]: Trial balance balanced<br/>(test: Receivable = 0, balanced=true)

    state "Sent → mark-paid" as Paid {
        [*] --> TX_Created: Transaction created
        TX_Created --> GL_Updated: General Ledger updated
        GL_Updated --> Reconciled: BankTransaction matched
    }

    note right of Draft
        Auto-generates invoice number
```

```
        Locks exchange rate if foreign currency
        Validates customerId belongs to org
    end note

    note right of Sent
        Creates accounting transaction
        referenceType = invoice
        referenceId = invoice.id
    end note
```

---

4. End-to-End Test Strategy

Framework: Playwright **Target:** Critical business flows that span the full stack **Environment:** Staging environment with seeded data

4.1 User Registration and Setup

```
test('New user can register, set up org, and access dashboard', async ({ page }) => {
  // 1. Navigate to /register
  // 2. Fill in org name, country=RS, email, password
  // 3. Submit → redirected to dashboard
  // 4. Dashboard loads with zero-state (empty metrics)
  // 5. Logout → redirected to /login
  // 6. Login with same credentials → dashboard again
});
```

4.2 Complete Invoice Flow

```
test('Create invoice → send → mark paid → check P&L', async ({ page }) => {
  // Step 1: Create contact (customer)
  await page.goto('/contacts/new');
  await fillContactForm({ name: 'Test Customer', type: 'customer' });
  await page.click('button[type=submit]');

  // Step 2: Create invoice
  await page.goto('/invoices/new');
  // Fill 6-step wizard: customer, date, items (1000 RSD + 20% PDV), review
```

```
  await completeInvoiceWizard({ customer: 'Test Customer', amount: 1000, taxRate: 20 });
  // Verify: status = draft, total = 1200 RSD

  // Step 3: Send invoice
  await page.click('button:text("Send Invoice")');
  // Verify: status = sent

  // Step 4: Mark paid
  await page.click('button:text("Mark as Paid")');
  await page.fill('[name=paidAt]', '2026-02-20');
  await page.click('button:text("Confirm")');
  // Verify: status = paid, paidAt set

  // Step 5: Check P&L report
  await page.goto('/reports?from=2026-01-01&to=2026-12-31');
  await expect(page.locator('[data-testid=revenue-total]')).toContainText('1,200.00');

  // Step 6: Check trial balance (balanced)
  await page.goto('/reports/trial-balance');
  await expect(page.locator('[data-testid=balanced-indicator]')).toBeVisible();
});
```

4.3 Expense Approval Flow

```
test('Create expense → approve → pay → check trial balance', async ({ page }) => {
  // Step 1: Create expense (office supplies, 5000 RSD, 17% PDV)
  // Step 2: Approve expense → DR Office Expense / CR Accounts Payable
  // Step 3: Pay expense → DR Accounts Payable / CR Bank
  // Step 4: Verify trial balance is still balanced
  // Step 5: Verify P&L shows expense in correct category
});
```

4.4 Bank Reconciliation Flow

```
test('Import bank statement → reconcile with invoice payment', async ({ page }) => {
  // Pre-condition: Paid invoice exists (DR Bank / CR Receivable transaction)
  // Step 1: Go to Banking page
  // Step 2: Import CSV with matching payment entry
```

```
  // Step 3: Verify imported: 1, duplicates: 0
  // Step 4: Match bank transaction to GL transaction
  // Step 5: Verify BankTransaction.reconciled = true
  // Step 6: Verify Transaction.reconciled = true
});
```

4.5 VAT Report Generation

```
test('VAT report reflects invoices and expenses for period', async ({ page }) => {
  // Pre-condition: 3 sent invoices with 20% PDV, 2 approved expenses with PDV
  // Step 1: Navigate to Reports → VAT Report
  // Step 2: Set period to current month
  // Step 3: Verify: output VAT = sum of invoice tax amounts
  // Step 4: Verify: input VAT = sum of expense tax amounts
  // Step 5: Verify: net VAT = output - input
  // Step 6: Download/export VAT report (future feature)
});
```

E2E Test Flow — Complete Invoice Lifecycle

```
flowchart TD
    START([Browser: /login]) --> LOGIN[Fill credentials<br/>demo@bilko.io]
    LOGIN --> DASH[Dashboard loaded<br/>assert: zero-state metrics]

    DASH --> NEW_CONTACT["/contacts/new<br/>Create: Test Customer"]
    NEW_CONTACT --> NEW_INV["/invoices/new<br/>6-step wizard"]

    NEW_INV --> W1["Step 1: Select Customer<br/>assert: customer appears in dropdown"]
    W1 --> W2["Step 2: Set dates<br/>invoiceDate, dueDate"]
    W2 --> W3["Step 3: Add line items<br/>1000 RSD + 20% PDV = 1200 RSD total"]
    W3 --> W4["Step 4: Currency & exchange rate"]
    W4 --> W5["Step 5: Notes / payment terms"]
    W5 --> W6["Step 6: Review & Create<br/>assert: subtotal=1000, tax=200, total=1200"]

    W6 --> INV_DETAIL["Invoice detail page<br/>assert: status=draft, number=INV-YYYY-NNN"]
```

```
    INV_DETAIL --> SEND["Click: Send Invoice<br/>assert: status=sent"]
    SEND --> PAY["Click: Mark as Paid<br/>Enter paidAt date"]
    PAY --> PAID["assert: status=paid, paidAt set"]

    PAID --> PL["/reports?from=...&to=...<br/>assert: revenue-total = 1,200.00"]
    PL --> TB["/reports/trial-balance<br/>assert: balanced-indicator visible<br/>assert: Receivable balance = 0"]

    style START fill:#198754,color:#fff
    style TB fill:#0d6efd,color:#fff
    style PAID fill:#198754,color:#fff
```

---

5. Accounting Scenario Tests

These tests verify correctness of the double-entry system under real-world accounting scenarios.

5.1 Invoice ? Payment ? Reconciliation

Scenario: Company issues invoice, receives payment, reconciles bank statement

```
test('Full invoice lifecycle creates correct ledger entries', async () => {
  // 1. Create invoice: 100,000 RSD net + 20,000 RSD PDV = 120,000 RSD total
  // 2. Send invoice → Transaction: DR Receivable 120,000 / CR Revenue 120,000
  // 3. Mark paid → Transaction: DR Bank 120,000 / CR Receivable 120,000
  // 4. Trial balance: Bank +120,000 / Revenue +120,000 (balanced)
  // 5. Receivable account balance = 0 (opened and closed)
  // 6. General ledger shows both entries on Receivable account
  const trialBalance = await getTrialBalance();
  expect(trialBalance.balanced).toBe(true);
  const receivable = trialBalance.accounts.find(a => a.code.startsWith('12'));
  expect(receivable.balance).toBe('0.0000');
});
```

5.2 Multi-Currency Invoice

Scenario: RSD-based company invoices EUR customer

```
test('EUR invoice stored and reported in RSD base currency', async () => {
  // Exchange rate: 1 EUR = 117.25 RSD (locked at invoice date)
  // Invoice: 1,000 EUR + 200 EUR PDV = 1,200 EUR
  // Expected baseAmount: 1,200 × 117.25 = 140,700 RSD

  const invoice = await createInvoice({
    currencyCode: 'EUR',
    items: [{ quantity: 1, unitPrice: 1000, taxRate: 20 }],
    invoiceDate: '2026-02-01' // rate exists for this date
  });

  expect(invoice.currencyCode).toBe('EUR');
  expect(invoice.totalAmount).toBe('1200.0000');
  expect(invoice.exchangeRate).toBe('117.250000');
  expect(invoice.baseAmount).toBe('140700.0000');

  // When paid: DR Bank 140,700 RSD / CR Receivable 140,700 RSD
  await markPaid(invoice.id, '2026-02-15');
  const transaction = await getTransactionForInvoice(invoice.id, 'payment');
  expect(transaction.baseAmount).toBe('140700.0000');
});
```

5.3 VAT Calculation Accuracy

```
test('VAT calculated with Decimal precision, no float errors', async () => {
  // Known float trap: 0.1 + 0.2 ≠ 0.3 in JavaScript float
  // Test with amounts that expose float precision issues
  const result = calculateVAT('123.45', '20');
  // Expected: tax = 123.45 × 0.20 = 24.69 (not 24.690000000000003)
  expect(result.tax.toString()).toBe('24.6900');
  expect(result.total.toString()).toBe('148.1400');

  // Large amount
  const large = calculateVAT('999999.9999', '17');
  expect(large.tax.toString()).toBe('169999.9998');  // exact
});
```

5.4 Trial Balance After Multiple Transactions

```
test('Trial balance remains balanced after 10 invoices and 5 expenses', async () => {
  // Create 10 invoices (all sent + paid)
  for (let i = 0; i < 10; i++) {
    const inv = await createAndSendInvoice(orgId, 10000 + i * 100);
    await markPaid(inv.id, today);
  }
  // Create 5 expenses (all approved + paid)
  for (let i = 0; i < 5; i++) {
    const exp = await createAndApproveExpense(orgId, 5000 + i * 50);
    await payExpense(exp.id);
  }

  const tb = await getTrialBalance(orgId);
  expect(tb.balanced).toBe(true);
  // Total debits must equal total credits
  expect(new Decimal(tb.totals.debit)).toEqual(new Decimal(tb.totals.credit));
});
```

5.5 Expense Approval Double-Entry

```
test('Expense approval creates correct DR Expense / CR Payable entry', async () => {
  const expense = await createExpense({ amount: 5000, taxRate: 17 }); // 850 PDV
  await approveExpense(expense.id);

  const transactions = await getTransactionsForExpense(expense.id);
  expect(transactions).toHaveLength(1);
  const tx = transactions[0];

  // Verify debit is an expense account
  expect(tx.debitAccountCode).toMatch(/^5/);
  // Verify credit is payable
  expect(tx.creditAccountCode).toMatch(/^22/);
  // Amount matches expense amount
  expect(tx.amount).toBe('5000.0000');
});
```

---

6. Regulatory Compliance Tests

6.1 Serbia (RS)

```
describe('Serbia regulatory compliance', () => {
  it('PDV 20% standard rate applied to default supplies', async () => {
    const result = calculateSerbianPDV('10000', 'standard');
    expect(result).toBe('2000.00');
  });

  it('PDV 10% reduced rate applied to food/medicine', async () => {
    const result = calculateSerbianPDV('10000', 'reduced');
    expect(result).toBe('1000.00');
  });

  it('Business below 8M RSD threshold does not require VAT registration', () => {
    expect(requiresVATRegistration('7999999')).toBe(false);
  });

  it('Business at 8M RSD threshold requires VAT registration', () => {
    expect(requiresVATRegistration('8000000')).toBe(true);
  });

  it('Business below 6M RSD qualifies for pausal regime', () => {
    expect(qualifiesForPausalRegime('5999999')).toBe(true);
  });

  it('CIT calculated at flat 15%', () => {
    const cit = calculateSerbianCIT('100000');
    expect(cit).toBe('15000.00');
  });

  it('Invoice number follows Serbian format requirements', () => {
    // INV-YYYY-NNN format with sequential numbering
    expect(invoiceNumber).toMatch(/^INV-\d{4}-\d{3,}$/);
  });

  it('VAT report groups output and input VAT separately', async () => {
    const report = await getVATReport(orgId, { from: '2026-01-01', to: '2026-01-31' });
    expect(report).toHaveProperty('outputVAT');
    expect(report).toHaveProperty('inputVAT');
```

```
    expect(report).toHaveProperty('netVAT');
    expect(new Decimal(report.netVAT)).toEqual(
      new Decimal(report.outputVAT.total).sub(new Decimal(report.inputVAT.total))
    );
  });
});
```

6.2 Bosnia & Herzegovina (BA)

```
describe('BiH regulatory compliance', () => {
  it('Single PDV rate of 17% applied uniformly', () => {
    const result = calculateBosnianPDV('10000');
    expect(result).toBe('1700.00');
  });

  it('BiH has no reduced VAT rate (only standard and zero)', () => {
    const rates = Object.keys(bosnianVATRates);
    expect(rates).toEqual(['standard', 'zero']);
  });

  it('VAT registration required at 100,000 BAM', () => {
    expect(requiresVATRegistration('99999')).toBe(false);
    expect(requiresVATRegistration('100000')).toBe(true);
  });

  it('FBiH CIT at 10%', () => {
    expect(calculateCITFBiH('100000')).toBe('10000.00');
  });

  it('RS CIT at 10%', () => {
    expect(calculateCITRS('100000')).toBe('10000.00');
  });

  it('Dividend WHT: FBiH 5%, RS 10%', () => {
    expect(calculateDividendWHT('100000', 'fbih')).toBe('5000.00');
    expect(calculateDividendWHT('100000', 'rs')).toBe('10000.00');
  });
});
```

6.3 Croatia (HR)

```
describe('Croatia regulatory compliance', () => {
  it('Standard PDV rate is 25%', () => {
    const result = calculateCroatianPDV('10000', 'standard');
    expect(result).toBe('2500.00');
  });

  it('Reduced PDV rate is 13% (food, accommodation)', () => {
    const result = calculateCroatianPDV('10000', 'reduced');
    expect(result).toBe('1300.00');
  });

  it('Super-reduced PDV rate is 5% (books, medicines)', () => {
    const result = calculateCroatianPDV('10000', 'superReduced');
    expect(result).toBe('500.00');
  });

  it('CIT 10% for small business (revenue < 1M EUR)', () => {
    const cit = calculateCroatianCIT('50000', '900000');
    expect(cit).toBe('5000.00');
  });

  it('CIT 18% for large business (revenue >= 1M EUR)', () => {
    const cit = calculateCroatianCIT('50000', '1000000');
    expect(cit).toBe('9000.00');
  });

  it('VAT registration threshold 60,000 EUR (EU 2025 aligned)', () => {
    expect(requiresVATRegistration('59999')).toBe(false);
    expect(requiresVATRegistration('60000')).toBe(true);
  });
});
```

6.4 Audit Trail Compliance

```
describe('Immutable audit trail', () => {
  it('LoggedAction created on invoice create', async () => {
```

```
    await createInvoice(orgId, ...);
    const logs = await prisma.loggedAction.findMany({
      where: { tableName: 'invoices', action: 'INSERT' }
    });
    expect(logs).toHaveLength(1);
    expect(logs[0].rowData).toBeTruthy();
  });

  it('LoggedAction created on invoice status change', async () => {
    await sendInvoice(invoiceId);
    const logs = await prisma.loggedAction.findMany({
      where: { tableName: 'invoices', action: 'UPDATE' }
    });
    expect(logs.length).toBeGreaterThan(0);
    expect(logs[0].changedFields).toHaveProperty('status');
  });

  it('LoggedAction cannot be deleted', async () => {
    // Attempt to delete a log entry — should fail (policy enforced at app or DB level)
    await expect(prisma.loggedAction.delete({ where: { eventId: logs[0].eventId } }))
      .rejects.toThrow();
  });

  it('locked transaction cannot be updated', async () => {
    await prisma.transaction.update({
      where: { id: txId },
      data: { locked: true }
    });
    // Attempt to change amount of locked transaction via API
    const response = await request(app)
      .put(`/api/v1/transactions/${txId}`)
      .set('Authorization', `Bearer ${token}`)
      .send({ amount: 99999 });
    expect(response.status).toBe(400);
  });
});
```

6.5 Record Retention

```
describe('Record retention requirements', () => {
  it('Deleted invoices remain in LoggedAction with full row data', async () => {
    const invoice = await createInvoice(orgId);
    const invoiceId = invoice.id;
    await deleteInvoice(invoiceId);
    // Invoice deleted from invoices table
    const inv = await prisma.invoice.findUnique({ where: { id: invoiceId } });
    expect(inv).toBeNull();
    // But audit log captures the full row
    const log = await prisma.loggedAction.findFirst({
      where: { tableName: 'invoices', action: 'DELETE' }
    });
    expect(log.rowData).toBeTruthy();
    expect(JSON.parse(log.rowData).id).toBe(invoiceId);
  });
});
```

---

7. Performance Benchmarks

Tool: k6 (load testing), Lighthouse (frontend)

7.1 API Response Time Targets

Endpoint	Target (P95)	Max Acceptable
`GET /api/v1/health`	< 10ms	< 50ms
`POST /api/v1/auth/login`	< 300ms	< 1s
`GET /api/v1/invoices` (20 items)	< 200ms	< 500ms
`POST /api/v1/invoices`	< 500ms	< 1s
`GET /api/v1/reports/profit-loss`	< 500ms	< 2s
`GET /api/v1/reports/trial-balance`	< 1s	< 3s
`GET /api/v1/reports/general-ledger`	< 2s	< 5s
`POST /api/v1/bank-accounts/:id/import` (100 rows)	< 1s	< 3s

7.2 Load Test Scenarios

```
// k6 scenario: Normal business day load
export const options = {
  scenarios: {
    normal_load: {
      executor: 'constant-vus',
      vus: 50,
      duration: '5m',
    },
    spike: {
      executor: 'ramping-vus',
      startVUs: 0,
      stages: [
        { duration: '30s', target: 200 },
        { duration: '1m', target: 200 },
        { duration: '30s', target: 0 },
      ],
    }
  },
  thresholds: {
    'http_req_duration{type:api}': ['p(95)<500'],
    'http_req_failed': ['rate<0.01'], // < 1% error rate
  }
};
```

7.3 Database Performance

Operation	Target
Invoice list query (org with 10K invoices)	< 100ms
Trial balance (org with 1K accounts, 100K transactions)	< 2s
Exchange rate lookup	< 10ms (covered by index)
Audit log insert	< 5ms

7.4 Frontend Performance (Lighthouse)

Metric	Target
First Contentful Paint (FCP)	< 1.5s
Largest Contentful Paint (LCP)	< 2.5s

Metric	Target
Time to Interactive (TTI)	< 3.5s
Cumulative Layout Shift (CLS)	< 0.1
Lighthouse Performance Score	> 90

---

8. Security Tests

8.1 Authentication Security

```
describe('Authentication security', () => {
  it('rejected with 401 for missing Authorization header');
  it('rejected with 401 for malformed Bearer token');
  it('rejected with 401 for expired access token');
  it('rejected with 401 for tampered JWT signature');
  it('rejected with 401 for wrong JWT_SECRET');
  it('access token cannot be used as refresh token');
  it('refresh token cannot be used as access token');
  it('tokens have correct issuer and audience claims');
});
```

8.2 RBAC Authorization

```
describe('Role-based access control', () => {
  it('viewer cannot create invoices (403)');
  it('viewer cannot create expenses (403)');
  it('viewer cannot create manual transactions (403)');
  it('accountant cannot change user roles (403)');
  it('accountant cannot invite users (403)');
  it('admin cannot change owner role (403)');
  it('owner can change any role');
  it('user cannot change their own role');
  it('user cannot delete themselves');
});
```

8.3 SQL Injection

```
describe('SQL injection prevention', () => {
  it('invoice search with SQL payload returns 400 (Zod validation)', async () => {
    const res = await request(app)
      .get('/api/v1/invoices?customerId=\'; DROP TABLE invoices; --')
      .set('Authorization', `Bearer ${token}`);
    expect(res.status).toBe(400); // Zod rejects invalid UUID
  });

  it('Prisma parameterizes all queries (no raw SQL in services)');
});
```

8.4 Cross-Site Scripting (XSS)

```
describe('XSS prevention', () => {
  it('contact name with script tag is stored as plain text', async () => {
    const name = '<script>alert("xss")</script>';
    const contact = await createContact({ name });
    expect(contact.name).toBe(name); // stored as-is
    // API response should not execute as HTML (verified by Content-Type: application/json)
  });

  it('Content-Security-Policy header blocks inline scripts', async () => {
    const res = await request(app).get('/api/v1/health');
    const csp = res.headers['content-security-policy'];
    expect(csp).toContain("script-src 'self'");
    expect(csp).not.toContain("'unsafe-eval'");
  });
});
```

8.5 Rate Limiting

```
describe('Rate limiting', () => {
  it('general API limit: 100 requests per 15 min per IP', async () => {
    // Make 101 requests from same IP
    const responses = await makeRequests(101, '/api/v1/health');
    const lastResponse = responses[100];
    expect(lastResponse.status).toBe(429);
  });
```

```
  it('auth endpoints have stricter rate limit', async () => {
    // Make rapid login attempts — triggers auth rate limiter before general
    const responses = await makeLoginAttempts(20);
    expect(responses.some(r => r.status === 429)).toBe(true);
  });
});
```

8.6 Data Isolation / Multi-Tenant Security

```
describe('Tenant isolation security', () => {
  it('cannot access another org invoice by ID (returns 404, not 403)');
  // Note: returning 404 instead of 403 prevents enumeration attacks
  it('cannot access another org transactions by reference ID');
  it('cannot access another org users via /api/v1/users');
  it('cannot access another org bank accounts');
  it('PATCH invoice from another org returns 404');
  it('DELETE invoice from another org returns 404');
});
```

8.7 CORS

```
describe('CORS policy', () => {
  it('requests from bilko.io are allowed');
  it('requests from unknown origin are rejected with CORS error');
  it('OPTIONS preflight returns correct headers');
  it('credentials (cookies) allowed with CORS');
});
```

8.8 Security Headers

```
describe('Security headers', () => {
  it('X-Frame-Options: deny (clickjacking protection)');
  it('X-Content-Type-Options: nosniff');
  it('Strict-Transport-Security: maxAge=31536000; includeSubDomains; preload');
  it('Content-Security-Policy present');
  it('X-Powered-By header removed (helmet default)');
});
```

CI/CD Test Pipeline

```
flowchart TD
    PUSH["git push / PR opened"] --> CI["GitHub Actions triggered"]

    CI --> J1["Job: unit-tests<br/>ubuntu-latest<br/>No DB required"]
    CI --> J2["Job: integration-tests<br/>ubuntu-latest<br/>postgres:15 service"]
    CI --> J3["Job: e2e-tests<br/>ubuntu-latest<br/>Full stack startup"]

    J1 --> U1["npm ci"]
    U1 --> U2["cd packages/core && npx vitest run"]
    U2 --> U3{Coverage >= 80%?}
    U3 -->|Yes| U_OK["PASS"]
    U3 -->|No| U_FAIL["FAIL — block merge"]

    J2 --> I1["npm ci"]
    I1 --> I2["npx prisma migrate deploy<br/>(TEST_DATABASE_URL)"]
    I2 --> I3["npm run test:integration<br/>(apps/api)"]
    I3 --> I4{All assertions pass?}
    I4 -->|Yes| I_OK["PASS"]
    I4 -->|No| I_FAIL["FAIL — block merge"]

    J3 --> E1["npx playwright install --with-deps"]
    E1 --> E2["npm run dev (staging seed)"]
    E2 --> E3["npm run test:e2e"]
    E3 --> E4{All flows pass?}
    E4 -->|Yes| E_OK["PASS"]
    E4 -->|No| E_FAIL["FAIL — screenshot + video saved"]

    U_OK --> MERGE{All jobs passed?}
    I_OK --> MERGE
    E_OK --> MERGE
    MERGE -->|Yes| DEPLOY["Allow merge to main"]
    MERGE -->|No| BLOCK["Block PR merge"]

    style PUSH fill:#6c757d,color:#fff
    style DEPLOY fill:#198754,color:#fff
    style BLOCK fill:#dc3545,color:#fff
```

```
    style U_FAIL fill:#dc3545,color:#fff
    style I_FAIL fill:#dc3545,color:#fff
    style E_FAIL fill:#dc3545,color:#fff
```

---

9. Test Infrastructure

9.1 Directory Structure

```
Bilko/
├── packages/core/
│   ├── tests/                          ← Unit tests (EXISTS)
│   │   ├── accounting.test.ts
│   │   ├── tax.test.ts
│   │   ├── multi-currency.test.ts
│   │   ├── invoicing.test.ts
│   │   └── chart-of-accounts.test.ts
│   └── vitest.config.ts                ← Vitest config (EXISTS)
│
├── apps/api/
│   └── tests/                          ← To be created
│       ├── setup.ts                    ← DB setup/teardown
│       ├── helpers/
│       │   ├── auth.helper.ts          ← Login/register helpers
│       │   └── factory.ts              ← Test data factories
│       ├── integration/
│       │   ├── auth.test.ts
│       │   ├── invoices.test.ts
│       │   ├── expenses.test.ts
│       │   ├── contacts.test.ts
│       │   ├── accounts.test.ts
│       │   ├── transactions.test.ts
│       │   ├── reports.test.ts
│       │   ├── banking.test.ts
│       │   ├── settings.test.ts
│       │   └── isolation.test.ts
│       └── security/
│           ├── auth-security.test.ts
```

```
│           ├── rbac.test.ts
│           ├── injection.test.ts
│           └── headers.test.ts
│
└── e2e/                              ← To be created
    ├── playwright.config.ts
    ├── fixtures/
    │   └── test-data.ts
    └── tests/
        ├── auth.spec.ts
        ├── invoice-lifecycle.spec.ts
        ├── expense-flow.spec.ts
        ├── bank-reconciliation.spec.ts
        └── reports.spec.ts
```

9.2 Environment Variables for Testing

```
# Test environment
TEST_DATABASE_URL="postgresql://bilko_test:password@localhost:5432/bilko_test"
JWT_SECRET="test-jwt-secret-not-for-production"
JWT_REFRESH_SECRET="test-refresh-secret-not-for-production"
NODE_ENV="test"
```

9.3 CI Pipeline Integration

```
# .github/workflows/test.yml (target)
jobs:
  unit-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
      - run: npm ci
      - run: cd packages/core && npx vitest run

  integration-tests:
    runs-on: ubuntu-latest
    services:
      postgres:
```

```
        image: postgres:15
        env:
          POSTGRES_DB: bilko_test
          POSTGRES_PASSWORD: password
    steps:
      - uses: actions/checkout@v4
      - run: npm ci
      - run: npx prisma migrate deploy
        env:
          DATABASE_URL: ${{ env.TEST_DATABASE_URL }}
      - run: npm run test:integration
        working-directory: apps/api

  e2e-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: npx playwright install --with-deps
      - run: npm run test:e2e
        env:
          PLAYWRIGHT_BASE_URL: http://localhost:3000
```

---

10. Test Coverage Targets

Module	Unit Coverage	Integration Coverage
`@bilko/core` accounting	95% (near complete)	N/A
`@bilko/core` tax	95% (near complete)	N/A
`@bilko/core` multi-currency	90%	N/A
`@bilko/core` bank-import	80% (tests missing)	N/A
`@bilko/country-rs` tax	0% (tests missing)	N/A
`@bilko/country-ba` tax	0% (tests missing)	N/A
`@bilko/country-hr` tax	0% (tests missing)	N/A
API auth routes	N/A	90%
API invoice routes	N/A	90%
API expense routes	N/A	85%

Module	Unit Coverage	Integration Coverage
API report routes	N/A	80%
API banking routes	N/A	75%
API settings routes	N/A	80%
Multi-tenancy isolation	N/A	100%
Security tests	N/A	90%

Overall target: 80% line coverage across the codebase before production launch.

Revision #4
Created 2026-02-23 10:24:30 UTC by John
Updated 2026-05-31 20:02:29 UTC by John