Testing & QA

- Test Plan
- Testing Guide
- Test Inventory

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`
`packages/core/tests/tax.test.ts`	`calculateVAT`, `getDefaultVATRate`, `getVATRates`, `calculateNetFromGross`, `calculateCIT`

Test File	Coverage
`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
Decimal base amounts (123.45 at 20%)	tax=24.69, total=148.14

Test Case	Assertion
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
Time to Interactive (TTI)	< 3.5s
Cumulative Layout Shift (CLS)	< 0.1

Metric	Target
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.

Testing Guide

Bilko — Testing Guide

Status: NO TESTS EXIST YET — This document defines the testing strategy for implementation.

Testing Philosophy

Financial software has a higher correctness bar than typical web apps. Bilko's testing strategy prioritizes:

1. **Financial Logic Accuracy** — VAT calculations, double-entry bookkeeping, currency conversion
2. **Data Integrity** — No lost transactions, no balance discrepancies
3. **Regression Prevention** — Once fixed, bugs stay fixed
4. **Fast Feedback** — Tests run in <5 minutes locally

Testing Pyramid

```
        /\
       /E2E\        ← 10% (Critical user flows)
      /------\
     /  Integ \     ← 30% (API endpoints, DB queries)
    /----------\
   /    Unit    \   ← 60% (Business logic, utilities)
  /--------------\
```

Distribution:

- **60% Unit Tests** — Fast, isolated, test business logic
- **30% Integration Tests** — Test API + database together
- **10% E2E Tests** — Test full user flows (expensive, slow)

```
graph TD
    subgraph STACK["Bilko Testing Stack"]
        direction TB

        subgraph E2E_LAYER["E2E Layer — 10% — Playwright"]
            PW1["invoice-flow.spec.ts<br/>Create → Send → Paid"]
            PW2["expense-flow.spec.ts<br/>Add → Approve → Pay"]
            PW3["report-flow.spec.ts<br/>P&L → Export PDF"]
            PW4["auth-flow.spec.ts<br/>Register → Login → Logout"]
        end

        subgraph INT_LAYER["Integration Layer — 30% — Supertest"]
            ST1["auth.routes.test.ts<br/>register / login / refresh / logout"]
            ST2["invoices.routes.test.ts<br/>CRUD + status transitions"]
            ST3["expenses.routes.test.ts<br/>CRUD + approve/pay"]
            ST4["reports.routes.test.ts<br/>P&L / trial-balance / VAT"]
            ST5["isolation.test.ts<br/>Cross-org data access prevention"]
        end

        subgraph UNIT_LAYER["Unit Layer — 60% — Vitest"]
            VT1["accounting.test.ts<br/>validateDoubleEntry,
createJournalEntry<br/>calculateTrialBalance"]
            VT2["tax.test.ts<br/>calculateVAT: RS 20% / BA 17% / HR 25%<br/>calculateCIT,
getVATRates"]
            VT3["multi-currency.test.ts<br/>convertCurrency,
lockExchangeRate<br/>calculateForexGainLoss"]
            VT4["bank-import.test.ts<br/>parseCSV, detectDuplicates"]
            VT5["chart-of-accounts.test.ts<br/>Structure validation"]
        end
    end

    E2E_LAYER --> INT_LAYER
    INT_LAYER --> UNIT_LAYER

    style E2E_LAYER fill:#ffc107,stroke:#e0a800
    style INT_LAYER fill:#fd7e14,color:#fff,stroke:#e8690b
    style UNIT_LAYER fill:#198754,color:#fff,stroke:#157347
```

Tech Stack

Test Type	Framework	Purpose
Unit	Vitest	Business logic, utilities, components
Integration	Supertest	API endpoint testing
E2E	Playwright	Browser automation, user flows
Coverage	c8 (built into Vitest)	Code coverage reporting

Why These Tools?

Vitest (not Jest)

- Faster (ESM native, Vite-based)
- Compatible with Vite/Turborepo
- Watch mode with HMR
- Same API as Jest (easy migration if needed)

Supertest (not Postman)

- Programmatic API testing
- Works with Express
- Can test without starting server

Playwright (not Cypress)

- Multi-browser (Chromium, Firefox, WebKit)
- Auto-wait (no flaky tests from race conditions)
- Parallel execution
- Video recording on failure

Unit Tests (Vitest)

Scope

Test pure functions and business logic in isolation:

- Invoice calculations (subtotal, tax, discount, total)
- VAT calculations (Serbia 20%, BiH 17%, Croatia 25%)
- Currency conversion (exchange rate locking)

- Double-entry validation (debit = credit)
- Date utilities (fiscal year, due date calculation)
- Number formatting (currency display)

File Structure

```
apps/api/src/
├── services/
│   ├── invoice.service.ts
│   └── invoice.service.test.ts  ← Unit test
├── utils/
│   ├── vat.ts
│   └── vat.test.ts  ← Unit test
```

Example: VAT Calculation Test

```
// apps/api/src/utils/vat.test.ts
import { describe, it, expect } from 'vitest';
import { calculateVAT } from './vat';

describe('calculateVAT', () => {
  it('calculates Serbia VAT (20%)', () => {
    const result = calculateVAT(100, 20);
    expect(result).toBe(20);
  });

  it('calculates BiH VAT (17%)', () => {
    const result = calculateVAT(100, 17);
    expect(result).toBe(17);
  });

  it('calculates Croatia VAT (25%)', () => {
    const result = calculateVAT(100, 25);
    expect(result).toBe(25);
  });

  it('handles zero VAT', () => {
    const result = calculateVAT(100, 0);
    expect(result).toBe(0);
```

```
  });

  it('handles decimal amounts', () => {
    const result = calculateVAT(123.45, 20);
    expect(result).toBe(24.69);
  });

  it('rounds to 2 decimal places', () => {
    const result = calculateVAT(10.01, 20);
    expect(result).toBe(2.00); // Not 2.002
  });
});
```

Running Unit Tests

```
# Run all unit tests
npm run test:unit

# Watch mode (re-run on file change)
npm run test:unit -- --watch

# Coverage report
npm run test:unit -- --coverage

# Specific file
npm run test:unit -- vat.test.ts
```

Coverage Requirements

Category	Target	Rationale
Financial logic	>95%	Critical for correctness
Utilities	>90%	Reused across codebase
Services	>80%	Business logic layer
Controllers	>60%	Thin layer (tested via integration)
Overall	>80%	Industry standard

Integration Tests (Supertest)

Scope

Test API endpoints with real database:

- Auth flow (register, login, refresh, logout)
- CRUD operations (invoices, expenses, contacts)
- Data validation (Zod schemas)
- Error handling (400, 401, 403, 404, 500)
- Database transactions
- Organization scoping (can't access other org's data)

File Structure

```
apps/api/src/
├── routes/
│   ├── auth.routes.ts
│   └── auth.routes.test.ts  ← Integration test
├── routes/
│   ├── invoices.routes.ts
│   └── invoices.routes.test.ts  ← Integration test
```

Test Database Setup

Use separate test database:

```
# .env.test
DATABASE_URL=postgresql://bilko_test:bilko_test@localhost:5432/bilko_test
```

Setup/teardown:

```
// apps/api/src/test/setup.ts
import { PrismaClient } from '@prisma/client';
import { beforeAll, afterAll, beforeEach } from 'vitest';

const prisma = new PrismaClient();

beforeAll(async () => {
```

```
  // Run migrations on test DB
  await execSync('npx prisma migrate deploy');
});

beforeEach(async () => {
  // Clear all tables before each test
  await prisma.$transaction([
    prisma.invoice.deleteMany(),
    prisma.expense.deleteMany(),
    prisma.contact.deleteMany(),
    prisma.user.deleteMany(),
    prisma.organization.deleteMany(),
  ]);
});

afterAll(async () => {
  await prisma.$disconnect();
});
```

Example: Invoice API Test

```
// apps/api/src/routes/invoices.routes.test.ts
import { describe, it, expect, beforeEach } from 'vitest';
import request from 'supertest';
import { app } from '../app';
import { prisma } from '../lib/prisma';

describe('POST /api/v1/invoices', () => {
  let authToken: string;
  let organizationId: string;
  let customerId: string;

  beforeEach(async () => {
    // Create test organization
    const org = await prisma.organization.create({
      data: {
        name: 'Test Company',
        baseCurrency: 'RSD',
        country: 'RS',
```

```
    },
  });
  organizationId = org.id;

  // Create test user
  const user = await prisma.user.create({
    data: {
      organizationId,
      email: 'test@bilko.io',
      passwordHash: '$2b$12$...', // bcrypt hash
      fullName: 'Test User',
      role: 'admin',
    },
  });

  // Login to get token
  const loginRes = await request(app)
    .post('/api/v1/auth/login')
    .send({ email: 'test@bilko.io', password: 'test123' });
  authToken = loginRes.body.accessToken;

  // Create test customer
  const customer = await prisma.contact.create({
    data: {
      organizationId,
      type: 'customer',
      name: 'Test Customer',
      email: 'customer@example.com',
    },
  });
  customerId = customer.id;
});

it('creates invoice with valid data', async () => {
  const res = await request(app)
    .post('/api/v1/invoices')
    .set('Authorization', `Bearer ${authToken}`)
    .send({
      customerId,
```

```
      invoiceDate: '2026-02-20',
      dueDate: '2026-03-20',
      currencyCode: 'RSD',
      items: [
        {
          description: 'Web Development',
          quantity: 10,
          unitPrice: 5000,
          taxRate: 20,
        },
      ],
    });

  expect(res.status).toBe(201);
  expect(res.body.invoiceNumber).toMatch(/^INV-\d+$/);
  expect(res.body.subtotal).toBe(50000);
  expect(res.body.taxAmount).toBe(10000);
  expect(res.body.totalAmount).toBe(60000);
});

it('rejects invoice without auth', async () => {
  const res = await request(app)
    .post('/api/v1/invoices')
    .send({ customerId, items: [] });

  expect(res.status).toBe(401);
});

it('rejects invoice for customer in different org', async () => {
  // Create another org
  const otherOrg = await prisma.organization.create({
    data: { name: 'Other Company', baseCurrency: 'EUR', country: 'RS' },
  });

  // Create customer in other org
  const otherCustomer = await prisma.contact.create({
    data: {
      organizationId: otherOrg.id,
      type: 'customer',
```

```
        name: 'Other Customer',
      },
    });

    const res = await request(app)
      .post('/api/v1/invoices')
      .set('Authorization', `Bearer ${authToken}`)
      .send({
        customerId: otherCustomer.id,
        items: [],
      });

    expect(res.status).toBe(403); // Forbidden (can't access other org's data)
  });
});
```

Running Integration Tests

```
# Run all integration tests
npm run test:integration

# Specific file
npm run test:integration -- invoices.routes.test.ts
```

E2E Tests (Playwright)

Scope

Test critical user flows from browser:

- **Invoice Flow:** Create → Preview → Send → Mark Paid
- **Expense Flow:** Add → Upload Receipt → Approve → Pay
- **Report Flow:** Generate P&L → Export PDF
- **Auth Flow:** Register → Login → 2FA → Logout

File Structure

```
apps/e2e/
├── tests/
│   ├── invoice-flow.spec.ts
│   ├── expense-flow.spec.ts
│   ├── report-flow.spec.ts
│   └── auth-flow.spec.ts
├── fixtures/
│   └── test-data.ts
└── playwright.config.ts
```

Configuration

```
// apps/e2e/playwright.config.ts
import { defineConfig } from '@playwright/test';

export default defineConfig({
  testDir: './tests',
  timeout: 60000, // 60s per test
  retries: 1, // Retry flaky tests once
  workers: 4, // Run 4 tests in parallel
  use: {
    baseURL: 'http://localhost:3000',
    screenshot: 'only-on-failure',
    video: 'retain-on-failure',
  },
  projects: [
    { name: 'chromium', use: { browserName: 'chromium' } },
    { name: 'firefox', use: { browserName: 'firefox' } },
    { name: 'webkit', use: { browserName: 'webkit' } },
  ],
});
```

Example: Invoice E2E Test

```
// apps/e2e/tests/invoice-flow.spec.ts
import { test, expect } from '@playwright/test';

test.describe('Invoice Flow', () => {
```

```
test.beforeEach(async ({ page }) => {
  // Login
  await page.goto('/login');
  await page.fill('input[name="email"]', 'demo@bilko.io');
  await page.fill('input[name="password"]', 'demo123');
  await page.click('button[type="submit"]');
  await expect(page).toHaveURL('/dashboard');
});

test('create invoice and mark as paid', async ({ page }) => {
  // Navigate to invoices
  await page.click('a[href="/invoices"]');
  await expect(page).toHaveURL('/invoices');

  // Click "New Invoice"
  await page.click('button:has-text("New Invoice")');
  await expect(page).toHaveURL('/invoices/new');

  // Fill invoice form (6-step wizard)
  // Step 1: Customer
  await page.selectOption('select[name="customerId"]', { label: 'Acme Corp' });
  await page.click('button:has-text("Next")');

  // Step 2: Details
  await page.fill('input[name="invoiceDate"]', '2026-02-20');
  await page.fill('input[name="dueDate"]', '2026-03-20');
  await page.click('button:has-text("Next")');

  // Step 3: Items
  await page.fill('input[name="items.0.description"]', 'Web Development');
  await page.fill('input[name="items.0.quantity"]', '10');
  await page.fill('input[name="items.0.unitPrice"]', '5000');
  await page.selectOption('select[name="items.0.taxRate"]', '20');
  await page.click('button:has-text("Next")');

  // Step 4: Review
  await expect(page.locator('text=Subtotal')).toContainText('50,000.00 RSD');
  await expect(page.locator('text=Tax')).toContainText('10,000.00 RSD');
  await expect(page.locator('text=Total')).toContainText('60,000.00 RSD');
```

```
    await page.click('button:has-text("Create Invoice")');

    // Verify redirect to invoice detail
    await expect(page).toHaveURL(/\/invoices\/[a-f0-9-]+$/);
    await expect(page.locator('h1')).toContainText('INV-');

    // Mark as paid
    await page.click('button:has-text("Mark as Paid")');
    await page.click('button:has-text("Confirm")');

    // Verify status changed
    await expect(page.locator('.status-badge')).toContainText('Paid');
  });

  test('validates required fields', async ({ page }) => {
    await page.goto('/invoices/new');

    // Try to submit without customer
    await page.click('button:has-text("Next")');

    // Verify error message
    await expect(page.locator('.error')).toContainText('Customer is required');
  });
});
```

Running E2E Tests

```
# Start dev server first
npm run dev

# In another terminal:
npm run test:e2e

# Headless (CI mode)
npm run test:e2e -- --headed

# Debug mode (pause on failure)
npm run test:e2e -- --debug
```

```
# Specific browser
npm run test:e2e -- --project=firefox
```

VAT Test Matrix — Country Coverage

```
graph TD
    VAT["calculateVAT Tests<br/>@bilko/core/tax"]

    VAT --> RS["Serbia RS<br/>Standard: 20%<br/>Reduced: 10%<br/>Zero: 0% exports<br/>CIT: 15% flat<br/>Pausal: &lt; 6M RSD<br/>VAT reg: &gt;= 8M RSD"]

    VAT --> BA["Bosnia BA<br/>Standard: 17%<br/>No reduced rate<br/>Zero: 0% exports<br/>FBiH CIT: 10%<br/>RS CIT: 10%<br/>WHT FBiH: 5% div<br/>VAT reg: &gt;= 100K BAM"]

    VAT --> HR["Croatia HR<br/>Standard: 25%<br/>Reduced: 13%<br/>Super-red: 5%<br/>CIT small: 10%<br/>CIT large: 18%<br/>VAT reg: &gt;= 60K EUR"]

    RS --> RS_T["Test: calculateSerbianPDV<br/>Test: qualifiesForPausalRegime<br/>Test: requiresVATRegistration RS"]
    BA --> BA_T["Test: calculateBosnianPDV<br/>Test: calculateCITFBiH / CITRS<br/>Test: calculateDividendWHT"]
    HR --> HR_T["Test: calculateCroatianPDV<br/>Test: calculateCroatianCIT<br/>Test: requiresVATRegistration HR"]

    style VAT fill:#0d6efd,color:#fff
    style RS fill:#c0392b,color:#fff
    style BA fill:#2c3e50,color:#fff
    style HR fill:#e74c3c,color:#fff
```

Test Data Management

Factories (Recommended)

Create reusable test data generators:

```
// apps/api/src/test/factories/invoice.factory.ts
import { faker } from '@faker-js/faker';
import { prisma } from '../../lib/prisma';

export async function createInvoice(overrides = {}) {
  return prisma.invoice.create({
    data: {
      organizationId: faker.string.uuid(),
      customerId: faker.string.uuid(),
      invoiceNumber: `INV-${faker.number.int({ min: 1000, max: 9999 })}`,
      invoiceDate: faker.date.recent(),
      dueDate: faker.date.future(),
      currencyCode: 'RSD',
      subtotal: 50000,
      taxAmount: 10000,
      totalAmount: 60000,
      baseAmount: 60000,
      status: 'draft',
      ...overrides,
    },
  });
}
```

Usage:

```
const invoice = await createInvoice({ status: 'paid' });
```

Coverage Reporting

Generate Coverage Report

```
npm run test:unit -- --coverage
```

Output:

```
File             | % Stmts | % Branch | % Funcs | % Lines
-----------------|---------|----------|---------|--------
```

```
All files        |   82.5  |   75.3   |   80.1  |  82.5
 vat.ts          |   95.0  |   90.0   |  100.0  |  95.0
 invoice.ts      |   88.2  |   80.5   |   85.0  |  88.2
 currency.ts     |   78.0  |   70.0   |   75.0  |  78.0
```

Coverage Thresholds (CI)

Fail build if coverage drops below threshold:

```
// vitest.config.ts
export default defineConfig({
  test: {
    coverage: {
      provider: 'c8',
      reporter: ['text', 'json', 'html'],
      statements: 80,
      branches: 75,
      functions: 80,
      lines: 80,
    },
  },
});
```

Testing Best Practices

1. Test Behavior, Not Implementation

 Bad: Test internal state

```
it('sets status to paid', () => {
  invoice.status = 'paid';
  expect(invoice.status).toBe('paid');
});
```

 Good: Test observable behavior

```
it('marks invoice as paid', async () => {
  await invoiceService.markAsPaid(invoice.id);
  const updated = await prisma.invoice.findUnique({ where: { id: invoice.id } });
  expect(updated.status).toBe('paid');
  expect(updated.paidAt).toBeTruthy();
});
```

---

2. Use Descriptive Test Names

 Bad: Vague test name

```
it('works', () => { /* ... */ });
```

 Good: Descriptive test name

```
it('calculates Serbian VAT at 20% on €100 as €20', () => { /* ... */ });
```

---

3. Arrange-Act-Assert (AAA)

```
it('creates invoice with correct totals', async () => {
  // ARRANGE — Set up test data
  const customer = await createCustomer();
  const invoiceData = { customerId: customer.id, items: [...] };

  // ACT — Perform action
  const invoice = await invoiceService.create(invoiceData);

  // ASSERT — Verify outcome
  expect(invoice.subtotal).toBe(50000);
  expect(invoice.taxAmount).toBe(10000);
  expect(invoice.totalAmount).toBe(60000);
});
```

---

4. Test Edge Cases

Always test:

- **Empty input** — `calculateVAT(0, 20)`
- **Null/undefined** — `formatCurrency(null)`
- **Negative numbers** — `calculateDiscount(100, -10)`
- **Large numbers** — `convertCurrency(999999999999.9999, 1.2)`
- **Boundary values** — Tax rate at 0%, 100%

---

5. Avoid Test Interdependence

 Bad: Tests depend on each other

```
let invoiceId;

it('creates invoice', async () => {
  const invoice = await createInvoice();
  invoiceId = invoice.id; // Shared state
});

it('updates invoice', async () => {
  await updateInvoice(invoiceId); // Depends on previous test
});
```

 Good: Tests are independent

```
it('creates invoice', async () => {
  const invoice = await createInvoice();
  expect(invoice.id).toBeTruthy();
});

it('updates invoice', async () => {
  const invoice = await createInvoice(); // Create fresh data
  await updateInvoice(invoice.id);
});
```

CI/CD Integration

Tests run automatically on every push (GitHub Actions):

```
# .github/workflows/main.yml
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - run: npm run test:unit -- --coverage
      - run: npm run test:integration
      - run: npm run test:e2e
```

```
flowchart LR
    GIT["git push"] --> GHA["GitHub Actions"]

    GHA --> PARALLEL["Parallel Jobs"]

    PARALLEL --> U["unit-tests<br/>vitest run<br/>@bilko/core<br/>~30s"]
    PARALLEL --> I["integration-tests<br/>supertest<br/>postgres:15<br/>~2min"]
    PARALLEL --> E["e2e-tests<br/>playwright<br/>all browsers<br/>~5min"]

    U --> COV{"Coverage<br/>>80%?"}
    I --> API{"All API<br/>tests pass?"}
    E --> E2E{"All flows<br/>pass?"}

    COV -->|Pass| GATE["Merge Gate"]
    API -->|Pass| GATE
    E2E -->|Pass| GATE

    COV -->|Fail| BLOCK1["Block PR"]
    API -->|Fail| BLOCK1
    E2E -->|Fail| BLOCK1

    GATE --> MAIN["Merge to main"]
    MAIN --> DEPLOY["Deploy to Staging"]

    style GIT fill:#6c757d,color:#fff
    style MAIN fill:#198754,color:#fff
    style BLOCK1 fill:#dc3545,color:#fff
    style DEPLOY fill:#0d6efd,color:#fff
```

See CI-CD.md for full pipeline.

Debugging Tests

Unit/Integration Tests (Vitest)

```
# Debug mode (pause on debugger statement)
npm run test:unit -- --inspect-brk

# VS Code launch.json:
{
  "type": "node",
  "request": "launch",
  "name": "Debug Vitest",
  "runtimeExecutable": "npm",
  "runtimeArgs": ["run", "test:unit", "--", "--inspect-brk"],
  "console": "integratedTerminal"
}
```

E2E Tests (Playwright)

```
# Debug mode (opens inspector)
npm run test:e2e -- --debug

# Headed mode (see browser)
npm run test:e2e -- --headed

# Trace viewer (after failure)
npx playwright show-trace trace.zip
```

Performance Testing (Future)

Load Testing (k6)

Test API under load:

```
// apps/e2e/load/invoices.js
import http from 'k6/http';
import { check } from 'k6';

export let options = {
  vus: 100, // 100 virtual users
  duration: '30s',
};

export default function () {
  const res = http.get('http://localhost:4000/api/v1/invoices');
  check(res, { 'status is 200': (r) => r.status === 200 });
}
```

Run:

```
k6 run apps/e2e/load/invoices.js
```

Target: API handles 1,000 requests/second with <200ms p95 latency.

Status: PLANNED (Phase 2)

Related Documents

- CI/CD Pipeline: ../infrastructure/CI-CD.md
- Test Inventory: TEST-INVENTORY.md
- Security Testing: ../security/SECURITY-ARCHITECTURE.md

Last Updated: 2026-02-20 **Status:** NO TESTS EXIST YET — Implement tests during backend development **Coverage Target:** >80% overall, >95% for financial logic

Test Inventory

Bilko — Test Inventory

Status: NO TESTS EXIST YET — This document tracks planned tests and implementation status.

This inventory catalogs all tests planned for Bilko, organized by category and priority.

Test Coverage Summary

Category	Total Planned	Implemented	Coverage Target
Unit Tests	45	0	>80%
Integration Tests	35	0	>80%
E2E Tests	12	0	Critical flows
TOTAL	**92**	**0**	—

```
graph TD
    subgraph COVERAGE["Test Coverage Matrix — 92 Tests Planned"]
        subgraph UNIT["Unit Tests — 45"]
            FIN["Financial Calculations<br/>12 tests — P0<br/>VAT, invoice totals, precision"]
            DE["Double-Entry Validation<br/>6 tests — P0<br/>debit=credit enforcement"]
            CUR["Currency Conversion<br/>8 tests — P0<br/>exchange rate locking"]
            DATE["Date Utilities<br/>6 tests — P1<br/>due dates, fiscal year"]
            FMT["Number Formatting<br/>5 tests — P1<br/>RSD, EUR, BAM display"]
            AUTH_U["Authentication Utils<br/>8 tests — P0<br/>bcrypt, JWT, tokens"]
        end

        subgraph INTEG["Integration Tests — 35"]
            AUTH_API["Auth API<br/>10 tests — P0<br/>register/login/refresh/logout"]
            INV_API["Invoices API<br/>10 tests — P0<br/>CRUD + status + org-scope"]
            EXP_API["Expenses API<br/>8 tests — P1<br/>CRUD + approve/pay"]
            REP_API["Reports API<br/>7 tests — P1<br/>P&L / BS / VAT reports"]
        end
```

```
        subgraph E2E_["E2E Tests — 12"]
            INV_E2E["Invoice Flow<br/>4 tests — P0<br/>Create → Send → Paid"]
            EXP_E2E["Expense Flow<br/>3 tests — P1<br/>Add → Approve → Pay"]
            REP_E2E["Report Flow<br/>2 tests — P1<br/>Generate → Export PDF"]
            SET_E2E["Settings Flow<br/>2 tests — P2<br/>Org settings + Invite user"]
            AUTH_E2E["Auth Flow<br/>1 test — P1<br/>Register → Login → 2FA → Logout"]
        end
    end

    style FIN fill:#dc3545,color:#fff
    style DE fill:#dc3545,color:#fff
    style CUR fill:#dc3545,color:#fff
    style AUTH_U fill:#dc3545,color:#fff
    style AUTH_API fill:#dc3545,color:#fff
    style INV_API fill:#dc3545,color:#fff
    style INV_E2E fill:#dc3545,color:#fff
    style DATE fill:#fd7e14,color:#fff
    style FMT fill:#fd7e14,color:#fff
    style EXP_API fill:#fd7e14,color:#fff
    style REP_API fill:#fd7e14,color:#fff
    style EXP_E2E fill:#fd7e14,color:#fff
    style REP_E2E fill:#fd7e14,color:#fff
    style AUTH_E2E fill:#fd7e14,color:#fff
    style SET_E2E fill:#6c757d,color:#fff
```

Priority Legend

- **P0** — Critical (MVP blocker, financial logic)
- **P1** — High (core features, security)
- **P2** — Medium (nice-to-have, edge cases)
- **P3** — Low (future enhancements)

Unit Tests (45 total)

Financial Calculations (12 tests)

Test File	Test Name	What It Tests	Priority	Status
`vat.test.ts`	`calculateVAT - Serbia 20%`	VAT calculation for Serbia	P0	⬜ Not implemented
`vat.test.ts`	`calculateVAT - BiH 17%`	VAT calculation for BiH	P0	⬜ Not implemented
`vat.test.ts`	`calculateVAT - Croatia 25%`	VAT calculation for Croatia	P0	⬜ Not implemented
`vat.test.ts`	`calculateVAT - zero rate`	VAT at 0% (exports)	P0	⬜ Not implemented
`vat.test.ts`	`calculateVAT - decimal amounts`	VAT on €123.45	P0	⬜ Not implemented
`vat.test.ts`	`calculateVAT - rounding`	Rounds to 2 decimal places	P0	⬜ Not implemented
`invoice-calc.test.ts`	`calculateInvoiceTotal - subtotal`	Sum of line items	P0	⬜ Not implemented
`invoice-calc.test.ts`	`calculateInvoiceTotal - tax`	Sum of tax amounts	P0	⬜ Not implemented
`invoice-calc.test.ts`	`calculateInvoiceTotal - discount`	Subtract discount from subtotal	P0	⬜ Not implemented
`invoice-calc.test.ts`	`calculateInvoiceTotal - total`	Subtotal + tax - discount	P0	⬜ Not implemented
`invoice-calc.test.ts`	`calculateInvoiceTotal - multi-item`	Multiple line items with different tax rates	P0	⬜ Not implemented
`invoice-calc.test.ts`	`calculateInvoiceTotal - precision`	NUMERIC(19,4) precision maintained	P0	⬜ Not implemented

Double-Entry Validation (6 tests)

Test File	Test Name	What It Tests	Priority	Status
`double-entry.test.ts`	`validateTransaction - debit equals credit`	Debit amount = Credit amount	P0	⬜ Not implemented
`double-entry.test.ts`	`validateTransaction - rejects unbalanced`	Throws error if debit ≠ credit	P0	⬜ Not implemented
`double-entry.test.ts`	`validateTransaction - requires both accounts`	Throws if missing debit or credit account	P0	⬜ Not implemented
`double-entry.test.ts`	`validateTransaction - multi-currency`	Validates amounts in base currency	P0	⬜ Not implemented
`double-entry.test.ts`	`validateTransaction - precision`	NUMERIC precision preserved	P0	⬜ Not implemented

Test File	Test Name	What It Tests	Priority	Status
`double-entry.test.ts`	`validateTransaction - zero amount`	Rejects zero-amount transactions	P1	⬜ Not implemented

Currency Conversion (8 tests)

Test File	Test Name	What It Tests	Priority	Status
`currency.test.ts`	`convertCurrency - EUR to RSD`	Convert at locked exchange rate	P0	⬜ Not implemented
`currency.test.ts`	`convertCurrency - RSD to EUR`	Reverse conversion	P0	⬜ Not implemented
`currency.test.ts`	`convertCurrency - same currency`	Rate = 1.0 when currency matches	P0	⬜ Not implemented
`currency.test.ts`	`convertCurrency - precision`	NUMERIC(19,4) preserved	P0	⬜ Not implemented
`currency.test.ts`	`convertCurrency - large amounts`	€999,999,999.9999	P0	⬜ Not implemented
`currency.test.ts`	`convertCurrency - rounding`	Rounds to 4 decimal places	P1	⬜ Not implemented
`currency.test.ts`	`lockExchangeRate - historical rate`	Uses rate from transaction date, not today	P0	⬜ Not implemented
`currency.test.ts`	`lockExchangeRate - missing rate`	Throws if no rate available for date	P1	⬜ Not implemented

Date Utilities (6 tests)

Test File	Test Name	What It Tests	Priority	Status
`date.test.ts`	`calculateDueDate - 30 days`	Invoice date + 30 days	P1	⬜ Not implemented
`date.test.ts`	`calculateDueDate - custom terms`	Invoice date + custom days	P1	⬜ Not implemented
`date.test.ts`	`isOverdue - past due date`	Returns true if today > due date	P1	⬜ Not implemented
`date.test.ts`	`isOverdue - not overdue`	Returns false if today <= due date	P1	⬜ Not implemented
`date.test.ts`	`getFiscalYear - starts Jan 1`	Fiscal year 2026 = Jan 1 - Dec 31	P2	⬜ Not implemented

Test File	Test Name	What It Tests	Priority	Status
`date.test.ts`	`getFiscalYear - custom start`	Fiscal year starts on custom date	P2	⬜ Not implemented

Number Formatting (5 tests)

Test File	Test Name	What It Tests	Priority	Status
`format.test.ts`	`formatCurrency - RSD`	"50,000.00 RSD" format	P1	⬜ Not implemented
`format.test.ts`	`formatCurrency - EUR`	"€50,000.00" format	P1	⬜ Not implemented
`format.test.ts`	`formatCurrency - BAM`	"50,000.00 BAM" format	P1	⬜ Not implemented
`format.test.ts`	`formatCurrency - decimal places`	Respects currency decimal places (0-4)	P1	⬜ Not implemented
`format.test.ts`	`formatCurrency - null`	Returns "-" for null/undefined	P2	⬜ Not implemented

Authentication (8 tests)

Test File	Test Name	What It Tests	Priority	Status
`auth.test.ts`	`hashPassword - bcrypt 12 rounds`	Password hashed with bcrypt	P0	⬜ Not implemented
`auth.test.ts`	`hashPassword - unique salt`	Each hash is different	P0	⬜ Not implemented
`auth.test.ts`	`verifyPassword - correct password`	Returns true for correct password	P0	⬜ Not implemented
`auth.test.ts`	`verifyPassword - incorrect password`	Returns false for wrong password	P0	⬜ Not implemented
`auth.test.ts`	`generateJWT - valid payload`	JWT contains user ID, org ID, role	P0	⬜ Not implemented
`auth.test.ts`	`generateJWT - expiry 15 min`	Access token expires in 15 min	P0	⬜ Not implemented
`auth.test.ts`	`generateRefreshToken - expiry 7 days`	Refresh token expires in 7 days	P0	⬜ Not implemented
`auth.test.ts`	`verifyJWT - expired token`	Throws error if token expired	P1	⬜ Not implemented

Integration Tests (35 total)

Auth API (10 tests)

Test File	Test Name	What It Tests	Priority	Status
`auth-api.test.ts`	`POST /auth/register - success`	Creates user, returns 201	P0	⬜ Not implemented
`auth-api.test.ts`	`POST /auth/register - duplicate email`	Returns 400 if email exists	P0	⬜ Not implemented
`auth-api.test.ts`	`POST /auth/register - weak password`	Returns 400 if password < 8 chars	P0	⬜ Not implemented
`auth-api.test.ts`	`POST /auth/login - success`	Returns access + refresh tokens	P0	⬜ Not implemented
`auth-api.test.ts`	`POST /auth/login - wrong password`	Returns 401	P0	⬜ Not implemented
`auth-api.test.ts`	`POST /auth/login - non-existent user`	Returns 401	P0	⬜ Not implemented
`auth-api.test.ts`	`POST /auth/refresh - success`	Returns new access token	P0	⬜ Not implemented
`auth-api.test.ts`	`POST /auth/refresh - expired token`	Returns 401	P1	⬜ Not implemented
`auth-api.test.ts`	`POST /auth/logout - success`	Deletes refresh token from DB	P1	⬜ Not implemented
`auth-api.test.ts`	`POST /auth/logout - already logged out`	Returns 204 (idempotent)	P2	⬜ Not implemented

Invoices API (10 tests)

Test File	Test Name	What It Tests	Priority	Status
`invoices-api.test.ts`	`POST /invoices - success`	Creates invoice, returns 201	P0	⬜ Not implemented
`invoices-api.test.ts`	`POST /invoices - validates required fields`	Returns 400 if missing customer	P0	⬜ Not implemented
`invoices-api.test.ts`	`POST /invoices - validates currency`	Returns 400 if invalid currency code	P0	⬜ Not implemented
`invoices-api.test.ts`	`POST /invoices - org scoping`	Returns 403 if customer in different org	P0	⬜ Not implemented

Test File	Test Name	What It Tests	Priority	Status
`invoices-api.test.ts`	`GET /invoices - list`	Returns paginated invoices	P0	⬜ Not implemented
`invoices-api.test.ts`	`GET /invoices - filter by status`	Returns only "paid" invoices	P1	⬜ Not implemented
`invoices-api.test.ts`	`GET /invoices/:id - success`	Returns invoice by ID	P0	⬜ Not implemented
`invoices-api.test.ts`	`GET /invoices/:id - not found`	Returns 404 if ID doesn't exist	P1	⬜ Not implemented
`invoices-api.test.ts`	`PATCH /invoices/:id - update status`	Changes status to "sent"	P0	⬜ Not implemented
`invoices-api.test.ts`	`DELETE /invoices/:id - soft delete`	Marks as deleted (not hard delete)	P1	⬜ Not implemented

Expenses API (8 tests)

Test File	Test Name	What It Tests	Priority	Status
`expenses-api.test.ts`	`POST /expenses - success`	Creates expense, returns 201	P0	⬜ Not implemented
`expenses-api.test.ts`	`POST /expenses - validates required fields`	Returns 400 if missing amount	P0	⬜ Not implemented
`expenses-api.test.ts`	`POST /expenses - org scoping`	Returns 403 if vendor in different org	P0	⬜ Not implemented
`expenses-api.test.ts`	`GET /expenses - list`	Returns paginated expenses	P0	⬜ Not implemented
`expenses-api.test.ts`	`PATCH /expenses/:id/approve - success`	Changes status to "approved"	P1	⬜ Not implemented
`expenses-api.test.ts`	`PATCH /expenses/:id/approve - requires admin`	Returns 403 if user is viewer	P1	⬜ Not implemented
`expenses-api.test.ts`	`PATCH /expenses/:id/reject - success`	Changes status to "rejected"	P1	⬜ Not implemented
`expenses-api.test.ts`	`DELETE /expenses/:id - soft delete`	Marks as deleted	P1	⬜ Not implemented

Reports API (7 tests)

Test File	Test Name	What It Tests	Priority	Status

`reports-api.test.ts`	`GET /reports/profit-loss - success`	Returns P&L with revenue, expenses, net	P1	⬜ Not implemented
`reports-api.test.ts`	`GET /reports/profit-loss - date range`	Filters by start/end date	P1	⬜ Not implemented
`reports-api.test.ts`	`GET /reports/balance-sheet - success`	Returns assets, liabilities, equity	P1	⬜ Not implemented
`reports-api.test.ts`	`GET /reports/cash-flow - success`	Returns operating, investing, financing	P1	⬜ Not implemented
`reports-api.test.ts`	`GET /reports/vat - success`	Returns sales VAT, purchase VAT, net VAT	P1	⬜ Not implemented
`reports-api.test.ts`	`GET /reports/vat - Serbia 20%`	Calculates Serbian VAT correctly	P1	⬜ Not implemented
`reports-api.test.ts`	`GET /reports/vat - export PDF`	Returns PDF file	P2	⬜ Not implemented

E2E Tests (12 total)

Invoice Flow (4 tests)

Test File	Test Name	What It Tests	Priority	Status
`invoice-flow.spec.ts`	`Create invoice via 6-step wizard`	Full invoice creation flow	P0	⬜ Not implemented
`invoice-flow.spec.ts`	`Preview invoice before sending`	Preview modal shows correct totals	P0	⬜ Not implemented
`invoice-flow.spec.ts`	`Send invoice to customer`	Email sent, status changed to "sent"	P0	⬜ Not implemented
`invoice-flow.spec.ts`	`Mark invoice as paid`	Status changed to "paid", paidAt timestamp	P0	⬜ Not implemented

Expense Flow (3 tests)

Test File	Test Name	What It Tests	Priority	Status
`expense-flow.spec.ts`	`Add expense with receipt upload`	Create expense, upload JPG	P1	⬜ Not implemented

Test File	Test Name	What It Tests	Priority	Status
`expense-flow.spec.ts`	`Approve expense`	Admin approves, status changed	P1	⬜ Not implemented
`expense-flow.spec.ts`	`Mark expense as paid`	Status changed to "paid"	P1	⬜ Not implemented

Report Flow (2 tests)

Test File	Test Name	What It Tests	Priority	Status
`report-flow.spec.ts`	`Generate P&L report`	Select date range, view report	P1	⬜ Not implemented
`report-flow.spec.ts`	`Export P&L to PDF`	Download PDF file	P1	⬜ Not implemented

Settings Flow (2 tests)

Test File	Test Name	What It Tests	Priority	Status
`settings-flow.spec.ts`	`Update organization settings`	Change org name, tax settings	P2	⬜ Not implemented
`settings-flow.spec.ts`	`Invite user to organization`	Send invite email, user accepts	P2	⬜ Not implemented

Auth Flow (1 test)

Test File	Test Name	What It Tests	Priority	Status
`auth-flow.spec.ts`	`Register → Login → 2FA → Logout`	Full auth flow with 2FA	P1	⬜ Not implemented

Critical Path Testing Flow

```
flowchart TD
    START(["Start: No Tests"]) --> P1

    subgraph P1["Phase 1 — MVP Critical (25 tests)"]
```

```
    P1_FIN["Financial Calculations<br/>12 unit tests<br/>VAT RS/BA/HR, invoice totals"]
    P1_DE["Double-Entry Validation<br/>6 unit tests<br/>debit=credit, balanced=true"]
    P1_AUTH["Auth API<br/>7 integration tests<br/>register, login, refresh"]
    P1_FIN --> P1_DE --> P1_AUTH
end

P1 --> P1_GATE{"Coverage<br/>>=50%?"}
P1_GATE -->|Yes| P2
P1_GATE -->|No| P1

subgraph P2["Phase 2 — Core Features (35 tests)"]
    P2_CUR["Currency Conversion<br/>8 unit tests"]
    P2_INV["Invoices API<br/>10 integration tests"]
    P2_E2E["Invoice + Auth + Expense E2E<br/>8 E2E tests"]
    P2_REP["Reports API<br/>7 integration tests"]
    P2_CUR --> P2_INV --> P2_E2E --> P2_REP
end

P2 --> P2_GATE{"Coverage<br/>>=70%?"}
P2_GATE -->|Yes| P3
P2_GATE -->|No| P2

subgraph P3["Phase 3 — Polish (32 tests)"]
    P3_DATE["Date + Formatting<br/>11 unit tests"]
    P3_EXP["Expenses API<br/>8 integration tests"]
    P3_EDGE["Edge cases<br/>8 tests"]
    P3_SET["Settings flow<br/>5 tests"]
    P3_DATE --> P3_EXP --> P3_EDGE --> P3_SET
end

P3 --> DONE(["Production Ready<br/>>80% coverage<br/>All critical paths tested"])

style START fill:#6c757d,color:#fff
style DONE fill:#198754,color:#fff
style P1_GATE fill:#ffc107,stroke:#e0a800
style P2_GATE fill:#ffc107,stroke:#e0a800
```

Test Implementation Roadmap

Phase 1 (MVP Critical) — 25 tests

- □ **Financial calculations** (12 unit tests)
- □ **Double-entry validation** (6 unit tests)
- □ **Auth API** (7 integration tests: register, login, refresh)

Target: Before backend MVP launch

Phase 2 (Core Features) — 35 tests

- □ **Currency conversion** (8 unit tests)
- □ **Invoices API** (10 integration tests)
- □ **Invoice E2E flow** (4 E2E tests)
- □ **Auth E2E flow** (1 E2E test)
- □ **Expense flow** (3 E2E tests)
- □ **Reports API** (7 integration tests)
- □ **Report E2E flow** (2 E2E tests)

Target: 1 month after MVP launch

Phase 3 (Polish) — 32 tests

- □ **Date utilities** (6 unit tests)
- □ **Number formatting** (5 unit tests)
- □ **Expenses API** (8 integration tests)
- □ **Settings flow** (2 E2E tests)
- □ **Remaining auth tests** (3 integration tests)
- □ **Edge cases** (8 tests across categories)

Target: 3 months after MVP launch

Test Execution Commands

Run All Tests

```
npm run test        # All tests (unit + integration + E2E)
```

Run by Category

```
npm run test:unit           # Unit tests only
npm run test:integration    # Integration tests only
npm run test:e2e            # E2E tests only
```

Run Specific Test File

```
npm run test:unit -- vat.test.ts
npm run test:integration -- invoices-api.test.ts
npm run test:e2e -- invoice-flow.spec.ts
```

Run with Coverage

```
npm run test:unit -- --coverage
```

Watch Mode

```
npm run test:unit -- --watch
```

Coverage Tracking

Current Coverage (as of 2026-02-20)

Category	Coverage	Target	Status
Financial Logic	0%	>95%	Not started
API Endpoints	0%	>80%	Not started
Utilities	0%	>90%	Not started
Overall	0%	>80%	Not started

Next Milestone: 50% coverage (25 critical tests)

Related Documents

- Testing Guide: TESTING-GUIDE.md
- CI/CD Pipeline: ../infrastructure/CI-CD.md
- Security Testing: ../security/SECURITY-ARCHITECTURE.md

Last Updated: 2026-02-20 **Status:** NO TESTS IMPLEMENTED YET **Total Tests Planned:** 92 (45 unit + 35 integration + 12 E2E) **Next Action:** Implement Phase 1 financial calculation tests (12 tests)