Testing Framework

Testing Guide

Bilko — Testing Guide

Status: Active but partially stale — implementation moved to Kotlin/Ktor API and current Playwright partitioning is in progress **Version:** 2.1 **Last Updated:** 2026-05-21 **Author:** ALAI Documentation Team

> **Canonical testing policy:** see TEST-STRATEGY.md, E2E-TEST-PLAN.md, and DEMO-TESTING-PLAN.md. Older sections in this guide may still mention Express/Prisma/API Vitest inventory and should not override the current Ktor + Playwright policy.

Table of Contents

---

1. 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** — Prefer fast unit tests and targeted integration/contract tests; use real PostgreSQL/Testcontainers where behavior depends on the database

---

2. Testing Pyramid

```
        /\
       /E2E\        ← 10% (Critical user flows only)
      /------\
     /  Integ \     ← 20% (API endpoints, mocked DB)
    /----------\
   /    Unit    \   ← 70% (Business logic, financial engine)
  /--------------\
```

Distribution:

- **70% Unit Tests** — Fast, isolated, test business logic in `@bilko/core`
- **20% Integration Tests** — Test API endpoints with mocked Prisma client
- **10% E2E Tests** — Full API integration test (single file in `tests/e2e/`)

Coverage Targets by Module

Module	Unit	Integration	Reason
`@bilko/core` accounting engine	**95%**	N/A	Double-entry errors = financial loss
`@bilko/core` tax/VAT calculations	**95%**	N/A	Tax miscalculations = regulatory penalty
`@bilko/core` multi-currency	**90%**	N/A	FX errors = revenue leakage
Auth API	85%	**90%**	Security boundary
Invoice API	80%	**90%**	Core revenue feature
Expense API	80%	**85%**	Core cost tracking
Reports API	75%	**80%**	Regulatory output

Module	Unit	Integration	Reason
Banking API	75%	**75%**	Complex matching logic
Multi-tenant isolation	N/A	**100%**	GDPR + security critical

---

3. Tech Stack

Test Type	Framework	Purpose
Unit	Vitest	`@bilko/core` business logic (financial engine)
Integration	Vitest + Supertest	API endpoint testing with mocked Prisma
E2E	Vitest + Supertest	Full API integration (`tests/e2e/api.test.ts`)

Why Vitest (not Jest)

- Native ESM support, Vite-based — faster than Jest
- Compatible with Turborepo workspace structure
- Watch mode with HMR
- Same API as Jest (easy migration)

Why Supertest (not Postman)

- Programmatic API testing within Vitest
- Works with Express app instance directly
- No running server required

---

4. Actual Test Files

`apps/api/tests/` — Mock Suite (11 test files)

Tests with mocked Prisma — fast, no database required.

File	Tests	What It Covers
`setup.ts`	—	Shared test setup: Prisma mock, JWT helpers, test data factories

File	Tests	What It Covers
`auth.test.ts`	11	Register, login, refresh token, logout, GET /me — auth flows with mocked DB
`invoices.test.ts`	11	List, create, get, update, status change (send/pay), delete invoice endpoints
`expenses.test.ts`	9	List, create, get, update, approve/reject, delete expense endpoints
`contacts.test.ts`	9	List, create, get, update, delete contact endpoints
`accounts.test.ts`	4	List, get, create, delete chart-of-accounts endpoints
`banking.test.ts`	10	Bank account management, transaction import, reconciliation endpoints
`reports.test.ts`	9	P&L, balance sheet, VAT report, trial balance endpoints
`transactions.test.ts`	9	Transaction ledger list, filter, and detail endpoints
`country.test.ts`	27	Country plugin integration — tax rates for RS/BA/HR, invoice number formats
`chatbot.test.ts`	9	Chatbot message, history, clear history — rate limit (429) test
`invoice-gl-reversal.test.ts`	6	Invoice cancellation GL reversal — double-entry stays balanced
`new-endpoints.test.ts`	~10	Receipt, VAT export PDF/XML, dashboard, security audit log, data export

Total: ~120+ mock suite test cases

`apps/api/tests/unit/` — Unit Suite (4 test files)

Service-layer tests. Mocked Prisma. No HTTP layer.

File	Tests	What It Covers
`invoice-service-calculations.test.ts`	~15	`InvoiceService.createInvoice()` arithmetic — line totals, VAT, FX

File	Tests	What It Covers
`two-factor.test.ts`	8	`TwoFactorService` — enable, verify, disable TOTP with backup codes
`sef-submission.test.ts`	~10	`SefClient` HTTP calls, `InvoiceService.submitToSef()` fire-and-forget
`vat-calculation.test.ts`	~20	Pure VAT functions from country-rs, country-ba, country-hr packages

`apps/api/tests/e2e/` — E2E Suite (2 test files)

End-to-end workflow tests. Mocked services, no real DB required.

File	Tests	What It Covers
`api.test.ts`	—	Full Express stack integration test (live server, no mocks)
`billing-flow.e2e.test.ts`	~8	Full billing workflow: contact → invoice → send → pay → P&L → credit note

`apps/api/tests/integration/` — Real DB Suite (5 test files)

Requires `docker-compose.test.yml` PostgreSQL. Run with `npm run test:integration`.

File	Tests	What It Covers
`auth.integration.test.ts`	4	Registration + login + refresh against real DB
`invoice.integration.test.ts`	5	Full invoice CRUD lifecycle against real DB
`credit-note-gl.integration.test.ts`	3	Credit note GL entries balance in real DB
`report.integration.test.ts`	3	Reports with real seeded transactions
`tenant-isolation.integration.test.ts`	5	Cross-tenant data isolation (org A cannot read org B's data)

Grand Total: ~390 tests across 27 test files

`packages/core/tests/` — Unit Tests (5 test files)

File	Tests	What It Covers
`accounting.test.ts`	20	`validateDoubleEntry`, `createJournalEntry`, `calculateTrialBalance` — double-entry engine
`chart-of-accounts.test.ts`	32	Chart of accounts operations: account creation, hierarchy, account types
`invoicing.test.ts`	22	`generateInvoiceNumber`, `calculateInvoiceTotals`, `validateLineItem`
`multi-currency.test.ts`	24	Currency conversion, exchange rate locking, precision handling
`tax.test.ts`	23	VAT calculations for RS (20%), BA (17%), HR (25%), mixed rates, edge cases

Total: 121 unit test cases

Grand Total: ~220 tests across 14 test files

---

5. Running Tests

Run All Tests (Turborepo)

```
# From project root — runs all tests in all packages
npm run test

# Or with turbo directly
npx turbo run test
```

Run API Mock Suite

```
cd apps/api
npx vitest run

# Watch mode
npx vitest
```

```
# Specific test file
npx vitest run tests/auth.test.ts
npx vitest run tests/chatbot.test.ts

# Specific test by name pattern
npx vitest run --reporter=verbose -t "POST /api/v1/auth/login"
```

Run API Unit Suite

```
cd apps/api
npx vitest run tests/unit/

# Specific service test
npx vitest run tests/unit/two-factor.test.ts
npx vitest run tests/unit/vat-calculation.test.ts
```

Run API E2E Suite

```
cd apps/api
npx vitest run tests/e2e/

# Full billing flow
npx vitest run tests/e2e/billing-flow.e2e.test.ts
```

Run Real DB Integration Suite

Requires Docker. Uses `docker-compose.test.yml` (port 5433).

```
# Start test database
docker-compose -f docker-compose.test.yml up -d

# Run integration tests
cd apps/api
npm run test:integration

# Or directly:
npx vitest run tests/integration/
```

```
# Stop test database
docker-compose -f docker-compose.test.yml down
```

Run Core Unit Tests

```
cd packages/core
npx vitest run

# Watch mode
npx vitest

# Specific file
npx vitest run tests/tax.test.ts

# With coverage
npx vitest run --coverage
```

Run in Verbose Mode

```
npx vitest run --reporter=verbose
```

Test Without Building

Tests resolve workspace packages from TypeScript source (not `dist/`) via `vitest.config.ts` aliases. No build step required before running tests.

---

6. Test Configuration

`apps/api-express/vitest.config.ts`

```
import { defineConfig } from 'vitest/config'
import path from 'path'

export default defineConfig({
  resolve: {
```

```
    alias: {
      // Resolves workspace packages from source — no dist/ build needed
      '@bilko/core': path.resolve(__dirname, '../../packages/core/src/index.ts'),
      '@bilko/domain-rs': path.resolve(__dirname, '../../packages/domain-rs/src/index.ts'),
      '@bilko/domain-ba': path.resolve(__dirname, '../../packages/domain-ba/src/index.ts'),
      '@bilko/domain-hr': path.resolve(__dirname, '../../packages/domain-hr/src/index.ts'),
      '@bilko/database': path.resolve(__dirname, '../../packages/database/src/index.ts'),
    },
  },
  test: {
    globals: false,
    environment: 'node',
    setupFiles: [], // Setup is imported per test file via tests/setup.ts
    include: ['tests/**/*.test.ts'],
    exclude: ['node_modules', 'dist'],
  },
})
```

`packages/core/vitest.config.ts`

```
import { defineConfig } from 'vitest/config'

export default defineConfig({
  test: {
    globals: true,
    root: '.',
    include: ['tests/**/*.test.ts'],
  },
})
```

Key Configuration Differences

Setting	`apps/api-express`	`packages/core`
`globals`	`false` — imports explicit	`true` — describe/it/expect global
`setupFiles`	None (per-file import of `./setup`)	None
`aliases`	Workspace package path aliases	Not needed
`environment`	`node` (explicit)	Default node

7. Test Setup & Mocking

`apps/api/tests/setup.ts`

The API integration tests use a **mocked Prisma client** — tests run without a real PostgreSQL database. The setup file:

1. Sets required environment variables (JWT secrets, rate limit config)
2. Mocks the entire `src/lib/prisma` module with `vi.mock()`
3. Provides test data factories and JWT token generators

```
// Import setup (must be first import in test file)
import {
  createTestUser,
  generateTestAccessToken,
  generateTestRefreshToken,
  TEST_USER_EMAIL,
  TEST_USER_ID,
  TEST_ORG_ID,
} from './setup'
```

Mock Prisma Pattern

```
// In test file — reference the mocked prisma
import { prisma } from '../src/lib/prisma'
const mockPrisma = prisma as any

// Configure mock for a specific test
mockPrisma.user.findUnique.mockResolvedValue(testUser)
mockPrisma.user.findUnique.mockResolvedValue(null) // simulate not found

// Clear mocks between tests
beforeEach(() => {
  vi.clearAllMocks()
})
```

Auth Token Generation

```
// Generate a valid JWT for test requests
const authToken = generateTestAccessToken()

// Generate token with specific role
const adminToken = generateTestAccessToken({ role: 'admin' })
const viewerToken = generateTestAccessToken({ role: 'viewer' })

// Generate refresh token (used in cookies)
const refreshToken = generateTestRefreshToken(TEST_USER_ID)
```

Service Mocking Pattern (for route tests)

```
// Mock entire service module
vi.mock('../src/services/invoice.service', () => {
  const mockService = {
    listInvoices: vi.fn(),
    createInvoice: vi.fn(),
    // ...
  }
  return { invoiceService: mockService }
})

import { invoiceService } from '../src/services/invoice.service'
const mockInvoiceService = invoiceService as any

// Configure per test
mockInvoiceService.createInvoice.mockResolvedValue(newInvoice)

// Verify calls
expect(mockInvoiceService.createInvoice).toHaveBeenCalledWith(
  TEST_ORG_ID,
  TEST_USER_ID,
  expect.any(Object),
)
```

---

8. CI Integration

Tests run via GitHub Actions on every push and pull request. See `.github/workflows/ci.yml`.

CI Pipeline (4 parallel jobs)

```
# .github/workflows/ci.yml
on:
  push:
    branches: ['*']
  pull_request:
    branches: [main, staging]

jobs:
  lint: # npx turbo run lint
  type-check: # npx turbo run type-check (after prisma generate)
  test: # npx turbo run test (unit + integration)
  build: # npx turbo run build (needs lint, type-check, test)
```

Job Details

Job	Command	Needs Prisma Generate	Blocks
`lint`	`npx turbo run lint`	No	`build`
`type-check`	`npx turbo run type-check`	Yes	`build`
`test`	`npx turbo run test`	Yes	`build`
`build`	`npx turbo run build`	Yes	Deploy

Prisma Client Generation (required in CI)

```
- name: Generate Prisma Client
  run: npx prisma generate --schema=packages/database/prisma/schema.prisma
```

This is required before `type-check` and `test` because the API source references `@prisma/client` types.

Concurrency

```
concurrency:
  group: ci-${{ github.ref }}
```

```
  cancel-in-progress: true
```

Cancels in-flight CI runs on the same branch when new commits are pushed.

CI Gate Rules

Gate	Condition	Blocks
Lint	Any lint error	PR merge + build
Type check	Any TypeScript error	PR merge + build
Tests	Any test failure	PR merge + build
Build	Lint/type-check/test all pass	Deploy

---

9. Writing New Tests

API Integration Test — Pattern

```
// apps/api/tests/my-feature.test.ts
import { describe, it, expect, vi, beforeEach } from 'vitest'
import request from 'supertest'
import { generateTestAccessToken, TEST_ORG_ID, TEST_USER_ID } from './setup'
import app from '../src/app'

// Mock the service
vi.mock('../src/services/my-feature.service', () => {
  return {
    myFeatureService: {
      listItems: vi.fn(),
      createItem: vi.fn(),
    },
  }
})

import { myFeatureService } from '../src/services/my-feature.service'
const mockService = myFeatureService as any
const authToken = generateTestAccessToken()
```

```
describe('GET /api/v1/my-feature', () => {
  beforeEach(() => {
    vi.clearAllMocks()
  })

  it('returns 200 with paginated list', async () => {
    mockService.listItems.mockResolvedValue({ data: [], total: 0, page: 1, pageSize: 20 })

    const res = await request(app)
      .get('/api/v1/my-feature')
      .set('Authorization', `Bearer ${authToken}`)

    expect(res.status).toBe(200)
    expect(res.body).toHaveProperty('data')
  })

  it('returns 401 without auth', async () => {
    const res = await request(app).get('/api/v1/my-feature')
    expect(res.status).toBe(401)
  })
})
```

Core Unit Test — Pattern

```
// packages/core/tests/my-calculation.test.ts
import { describe, it, expect } from 'vitest'
import Decimal from 'decimal.js'
import { myCalculation } from '../src/my-module/index'

describe('myCalculation', () => {
  it('returns correct result for standard input', () => {
    const result = myCalculation('100', '20')
    expect(result.eq(new Decimal('20'))).toBe(true)
  })

  it('handles zero input', () => {
    const result = myCalculation('0', '20')
    expect(result.eq(new Decimal('0'))).toBe(true)
  })
```

```
  it('throws for negative amounts', () => {
    expect(() => myCalculation('-100', '20')).toThrow()
  })
})
```

Naming Conventions

```
// Describe block: HTTP method + route OR function name
describe('POST /api/v1/invoices', () => { ... })
describe('calculateInvoiceTotals', () => { ... })

// It block: be specific about scenario and expected outcome
it('returns 201 with created invoice when data is valid', ...)
it('returns 400 when customerId is missing', ...)
it('calculates Serbian VAT at 20% on 100 RSD as 20 RSD', ...)
it('throws for negative amounts', ...)
```

---

10. Coverage Reporting

Generate Coverage (core unit tests)

```
cd packages/core
npx vitest run --coverage
```

Coverage Output Format

```
File                    | % Stmts | % Branch | % Funcs | % Lines
------------------------|---------|----------|---------|--------
All files               |   XX.X  |   XX.X   |   XX.X  |  XX.X
 accounting/index.ts    |   95.0  |   90.0   |  100.0  |  95.0
 invoicing/index.ts     |   88.2  |   80.5   |   85.0  |  88.2
 tax/index.ts           |   92.0  |   88.0   |  100.0  |  92.0
```

Coverage Targets

Category	Target	Rationale
Financial logic (`@bilko/core`)	**>95%**	Critical for correctness
API routes (`apps/api-express`)	**>80%**	Integration-level coverage
Services	**>80%**	Business logic layer

---

11. Testing Best Practices

1. Arrange-Act-Assert (AAA)

```
it('creates invoice with correct totals', async () => {
  // ARRANGE
  const newInvoice = mockInvoice({ status: 'draft' })
  mockInvoiceService.createInvoice.mockResolvedValue(newInvoice)

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

  // ASSERT
  expect(res.status).toBe(201)
  expect(res.body.invoiceNumber).toBe('INV-2026-001')
})
```

2. Test Behavior, Not Implementation

```
// BAD — tests internal mock call order
expect(mockService.createInvoice.mock.calls[0][0]).toBe(orgId)

// GOOD — tests observable API behavior
expect(res.status).toBe(201)
expect(res.body).toHaveProperty('invoiceNumber')
```

3. Always Clear Mocks

```
beforeEach(() => {
  vi.clearAllMocks()
})
```

4. Test Edge Cases for Financial Logic

Always test in core unit tests:

- Zero amounts (`calculateInvoiceTotals([])`)
- Empty input arrays
- Negative values (should throw)
- Decimal precision (`quantity: '3', unitPrice: '33.33'`)
- Large amounts (overflow protection)

5. Never Hard-Code UUIDs in Tests

```
// BAD
const orgId = '123e4567-e89b-12d3-a456-426614174000'

// GOOD
const orgId = TEST_ORG_ID // from setup.ts, generated per-run
```

---

12. Debugging Tests

Run in Verbose Mode

```
npx vitest run --reporter=verbose
```

Run Single Test by Name

```
npx vitest run -t "returns 201 with created invoice"
```

Debug Mode (Node Inspector)

```
npx vitest --inspect-brk
# Then attach VS Code debugger or open chrome://inspect
```

Print Mock Call History

```
// In a test, add temporarily:
console.log(mockService.createInvoice.mock.calls)
```

VS Code Launch Configuration

```
{
  "type": "node",
  "request": "launch",
  "name": "Debug Vitest",
  "runtimeExecutable": "npx",
  "runtimeArgs": ["vitest", "--inspect-brk", "--no-coverage"],
  "console": "integratedTerminal",
  "cwd": "${workspaceFolder}/apps/api-express"
}
```

Related Documents

- Test Inventory: TEST-INVENTORY.md
- Backend Architecture: ../backend/BACKEND-ARCHITECTURE.md
- CI/CD Pipeline: ../infrastructure/CI-CD.md

Last Updated: 2026-03-02 **Status:** Active — ~390 tests across 27 files (mock + unit + E2E + real-DB suites) **Coverage Target:** >95% for `@bilko/core` financial logic, >80% for API routes

Test Inventory

Bilko — Test Inventory

Status: Partially stale — recount required after Kotlin/Ktor migration and Playwright partitioning
Version: 2.1 **Last Updated:** 2026-05-21 **Author:** ALAI Documentation Team

This inventory catalogs implemented tests in Bilko, organized by package and file. It currently contains historical Express/Prisma-era details and must not be used as the source of truth for current test counts. For current policy, use TEST-STRATEGY.md, E2E-TEST-PLAN.md, and DEMO-TESTING-PLAN.md.

Summary

> **Refresh note (2026-05-21):** quick filesystem inventory found `apps/api/src/test/kotlin` with many Kotlin test files, `packages/core/tests`, and `apps/e2e/tests`. The table below is historical until regenerated from the current tree.

Package	Test Files	Test Cases	Status
`apps/api/tests/` (mock suite)	11 test files	~130	Implemented
`apps/api/tests/unit/` (unit suite)	4 test files	~60	Implemented
`apps/api/tests/e2e/` (E2E suite)	2 test files	~30	Implemented
`apps/api/tests/integration/` (real DB)	5 test files	~50	Implemented
`packages/core/tests/` (unit)	5 test files	121	Implemented
Total	**27**	**~390**	Active

Test Category Breakdown

Category	Description	Requires DB
Mock suite (`tests/*.test.ts`)	API endpoint tests with mocked Prisma — fast, no DB	No
Unit suite (`tests/unit/`)	Service-layer tests, financial logic, SEF client	No
E2E suite (`tests/e2e/`)	End-to-end billing flows with mocked services	No
Integration suite (`tests/integration/`)	Real DB tests (docker-compose.test.yml)	**Yes** — PostgreSQL
Core unit (`packages/core/`)	Pure business logic — no HTTP, no DB	No

`packages/core/tests/` — Unit Tests

Pure unit tests for the `@bilko/core` financial engine. No database, no HTTP. Uses `globals: true` (no explicit imports of `describe`/`it`/`expect`).

`accounting.test.ts` — 20 tests

Tests double-entry bookkeeping engine: `validateDoubleEntry`, `createJournalEntry`, `calculateTrialBalance`.

Test Name	What It Tests	Status
`validateDoubleEntry - returns true for balanced entry`	Debit total equals credit total	Implemented
`validateDoubleEntry - returns false for unbalanced entry`	Debit ≠ credit	Implemented
`validateDoubleEntry - returns false for fewer than 2 lines`	Minimum 2 lines required	Implemented
`validateDoubleEntry - returns false for empty lines`	Empty array → false	Implemented
`validateDoubleEntry - returns false for negative amounts`	Negative amounts invalid	Implemented
`validateDoubleEntry - returns false for zero amounts`	Zero amounts invalid	Implemented
`validateDoubleEntry - handles multiple lines that sum to balanced`	Multi-line balanced entry	Implemented
`validateDoubleEntry - handles decimal amounts with precision`	NUMERIC(19,4) precision	Implemented
`createJournalEntry - returns entry when valid and balanced`	Happy path	Implemented

Test Name	What It Tests	Status
`createJournalEntry - throws for fewer than 2 lines`	Error: "at least 2 lines"	Implemented
`createJournalEntry - throws for empty lines array`	Error: "at least 2 lines"	Implemented
`createJournalEntry - throws for missing description`	Error: "must have a description"	Implemented
`createJournalEntry - throws for whitespace-only description`	Whitespace = missing	Implemented
`createJournalEntry - throws for missing date`	Error: "must have a date"	Implemented
`createJournalEntry - throws for unbalanced entry`	Error shows debit vs credit amounts	Implemented
`createJournalEntry - throws for negative amount lines`	Negative amounts rejected	Implemented
`calculateTrialBalance - returns balanced from balanced entries`	isBalanced=true, totals correct	Implemented
`calculateTrialBalance - groups by account number`	Rows aggregated per account	Implemented
`calculateTrialBalance - returns empty rows for no transactions`	Empty array → zero totals	Implemented
`calculateTrialBalance - sorts rows by account number`	Rows sorted ascending	Implemented

`chart-of-accounts.test.ts` — 32 tests

Tests chart of accounts operations: account creation, parent-child hierarchy, account type validation.

Test Area	Test Count	Status
Account creation (valid + invalid inputs)	~10	Implemented
Account hierarchy (parent-child relationships)	~8	Implemented
Account type validation (Asset/Liability/Equity/Revenue/Expense)	~7	Implemented
Account code format validation (1xxx-5xxx range)	~7	Implemented

---

`invoicing.test.ts` — 22 tests

Tests invoice number generation, total calculations, and line item validation.

Test Name	What It Tests	Status
`generateInvoiceNumber - generates INV-YYYY-NNN format`	Standard format	Implemented
`generateInvoiceNumber - increments from last number`	Sequential numbering	Implemented
`generateInvoiceNumber - pads number to 3 digits`	Zero-padding	Implemented
`generateInvoiceNumber - handles numbers beyond 999`	No truncation for 1000+	Implemented
`generateInvoiceNumber - throws for empty prefix`	Error: "prefix is required"	Implemented
`generateInvoiceNumber - throws for whitespace-only prefix`	Whitespace = invalid	Implemented
`generateInvoiceNumber - throws for negative lastNumber`	Error: "non-negative integer"	Implemented
`generateInvoiceNumber - throws for non-integer lastNumber`	Float rejected	Implemented
`calculateInvoiceTotals - calculates line item total`	quantity × unitPrice	Implemented
`calculateInvoiceTotals - calculates subtotal as sum of lines`	Sum of all line totals	Implemented
`calculateInvoiceTotals - calculates tax per line item`	lineTotal × taxRate / 100	Implemented
`calculateInvoiceTotals - calculates total = subtotal + tax`	Final total	Implemented
`calculateInvoiceTotals - handles items without taxRate`	Missing tax = 0	Implemented
`calculateInvoiceTotals - returns zeros for empty items`	Empty array → all zeros	Implemented
`calculateInvoiceTotals - handles multiple items with different rates`	Mixed 20%/10% tax	Implemented
`calculateInvoiceTotals - maintains decimal precision`	3 × 33.33 = 99.99	Implemented
`validateLineItem - returns true for valid item`	Happy path	Implemented
`validateLineItem - returns false for zero quantity`	qty=0 invalid	Implemented
`validateLineItem - returns false for negative quantity`	qty<0 invalid	Implemented
`validateLineItem - returns false for negative unitPrice`	price<0 invalid	Implemented
`validateLineItem - returns false for empty description`	Description required	Implemented
`validateLineItem - returns false for whitespace-only description`	Whitespace = empty	Implemented

`multi-currency.test.ts` — 24 tests

Tests currency conversion, exchange rate locking, and NUMERIC precision handling.

Test Area	Test Count	Status
Currency conversion (EUR/RSD/BAM)	~8	Implemented
Exchange rate locking at transaction date	~6	Implemented
NUMERIC(19,4) precision (no float drift)	~5	Implemented
Edge cases (zero rate, missing rate, same currency)	~5	Implemented

`tax.test.ts` — 23 tests

Tests VAT calculations for all supported countries and edge cases.

Test Area	Test Count	Status
Serbia PDV (20% standard, 10% reduced, 0% exempt)	~6	Implemented
BiH PDV (17% standard)	~4	Implemented
Croatia PDV (25% standard, 13% reduced, 5% super-reduced)	~5	Implemented
Mixed tax rates on single invoice	~3	Implemented
Zero-rate exports	~2	Implemented
Reverse VAT / gross-to-net	~3	Implemented

`apps/api/tests/` — Mock API Tests

Integration tests for Express API endpoints. Tests use **mocked Prisma client** — no real database required. Setup in `tests/setup.ts`.

`setup.ts` — Test Infrastructure (not a test file)

Provides:

- Prisma client mock via `vi.mock('../src/lib/prisma')`
- Environment variable setup (JWT secrets, rate limits)
- `createTestUser()` — factory for test user objects
- `generateTestAccessToken()` — valid JWT for authenticated requests
- `generateTestRefreshToken()` — valid refresh token
- Constants: `TEST_ORG_ID`, `TEST_USER_ID`, `TEST_USER_EMAIL`

`auth.test.ts` — 11 tests

Test Name	Route	Status
`returns 201 with user, organization, and tokens`	POST /auth/register	Implemented
`returns 400 for duplicate email`	POST /auth/register	Implemented
`returns 200 with tokens for valid credentials`	POST /auth/login	Implemented
`returns 401 for invalid password`	POST /auth/login	Implemented
`returns 401 for non-existent email`	POST /auth/login	Implemented
`returns 200 with new access token for valid refresh token`	POST /auth/refresh	Implemented
`returns 401 when no refresh token cookie is present`	POST /auth/refresh	Implemented
`returns 204 and clears cookie`	POST /auth/logout	Implemented
`returns 200 with current user when authenticated`	GET /auth/me	Implemented
`returns 401 when no token provided`	GET /auth/me	Implemented
`returns 401 for invalid token`	GET /auth/me	Implemented

`invoices.test.ts` — 11 tests

Test Name	Route	Status
`returns 200 with paginated list`	GET /invoices	Implemented
`returns 401 without auth`	GET /invoices	Implemented
`returns 201 with created invoice`	POST /invoices	Implemented
`returns 200 with full invoice`	GET /invoices/:id	Implemented
`returns 404 when invoice not found`	GET /invoices/:id	Implemented
`returns 200 when updating draft invoice`	PUT /invoices/:id	Implemented

Test Name	Route	Status
`returns 400 when updating sent invoice`	PUT /invoices/:id	Implemented
`returns 200 when sending invoice (draft -> sent)`	PATCH /invoices/:id/status	Implemented
`returns 200 when marking invoice as paid (sent -> paid)`	PATCH /invoices/:id/status	Implemented
`returns 204 when deleting draft invoice`	DELETE /invoices/:id	Implemented
`returns 400 when deleting sent invoice`	DELETE /invoices/:id	Implemented

`expenses.test.ts` — 9 tests

Test Area	Route	Status
List expenses (200 + auth)	GET /expenses	Implemented
Create expense (201)	POST /expenses	Implemented
Get expense by ID (200 + 404)	GET /expenses/:id	Implemented
Update expense (200 + 400 for non-pending)	PUT /expenses/:id	Implemented
Approve expense (200 + role check)	PATCH /expenses/:id/approve	Implemented
Delete expense (204 + 400 for approved)	DELETE /expenses/:id	Implemented

---

`contacts.test.ts` — 9 tests

Test Area	Route	Status
List contacts (200 + auth)	GET /contacts	Implemented
Create contact (201 + validation)	POST /contacts	Implemented
Get contact (200 + 404)	GET /contacts/:id	Implemented
Update contact (200)	PUT /contacts/:id	Implemented
Delete contact (204)	DELETE /contacts/:id	Implemented

`accounts.test.ts` — 4 tests

Test Area	Route	Status
List chart of accounts (200)	GET /accounts	Implemented
Get account by ID (200 + 404)	GET /accounts/:id	Implemented
Create account (201)	POST /accounts	Implemented
Delete account (204)	DELETE /accounts/:id	Implemented

`banking.test.ts` — 10 tests

Test Area	Route	Status
List bank accounts (200)	GET /banking/accounts	Implemented
Create bank account (201)	POST /banking/accounts	Implemented
Import bank statement (200 + validation)	POST /banking/accounts/:id/import	Implemented
List bank transactions (200)	GET /banking/accounts/:id/transactions	Implemented
Reconcile transaction (200 + 400 for mismatch)	POST /banking/accounts/:id/reconcile	Implemented

---

`reports.test.ts` — 9 tests

Test Area	Route	Status
Profit & Loss report (200 + date range)	GET /reports/profit-loss	Implemented
Balance sheet (200)	GET /reports/balance-sheet	Implemented
VAT report for RS (200 + correct rates)	GET /reports/vat	Implemented
Trial balance (200)	GET /reports/trial-balance	Implemented
Auth required on all report endpoints	All report routes	Implemented

---

`transactions.test.ts` — 9 tests

Test Area	Route	Status
List transactions (200 + pagination)	GET /transactions	Implemented
Filter by account (200)	GET /transactions?accountId=	Implemented

Test Area	Route	Status
Get transaction by ID (200 + 404)	GET /transactions/:id	Implemented
Auth required	All transaction routes	Implemented

`country.test.ts` — 27 tests

Tests the country plugin integration — routes that return country-specific tax configuration.

Test Area	Route	Status
Serbian PDV rates (20/10/0) for RS org	GET /country/tax-rates	Implemented
Bosnian PDV rate (17) for BA org	GET /country/tax-rates	Implemented
Croatian PDV rates (25/13/5/0) for HR org	GET /country/tax-rates	Implemented
Invoice number format per country	GET /country/invoice-format	Implemented
Auth required	All country routes	Implemented
Unknown country code (400)	GET /country/tax-rates	Implemented

---

`chatbot.test.ts` — Chatbot API Tests

Test Name	Route	Status
`returns 200 with assistant response`	POST /chatbot/message	Implemented
`returns 400 when message is empty`	POST /chatbot/message	Implemented
`returns 429 when rate limit exceeded`	POST /chatbot/message	Implemented
`returns 401 without auth`	POST /chatbot/message	Implemented
`returns 200 with conversation history`	GET /chatbot/history	Implemented
`returns empty array when no history exists`	GET /chatbot/history	Implemented
`returns 401 without auth on history`	GET /chatbot/history	Implemented
`returns 204 when history cleared`	DELETE /chatbot/history	Implemented
`returns 401 without auth on clear history`	DELETE /chatbot/history	Implemented

`invoice-gl-reversal.test.ts` — Invoice GL Reversal Tests

Tests `InvoiceService.cancelInvoice()` — when a SENT invoice is cancelled, reversing double-entry GL entries are created to undo the original booking.

Test Area	Status
Cancels draft invoice (sets status to cancelled, no GL reversal)	Implemented
Cancels sent invoice (creates reversing GL transactions)	Implemented
Reversal debits = original credits (GL stays balanced)	Implemented
Throws 404 when invoice not found	Implemented
Throws 400 when invoice is already cancelled	Implemented
Throws 400 when invoice is paid (cannot cancel paid invoices)	Implemented

---

`new-endpoints.test.ts` — Additional Endpoint Tests

Tests for supplemental endpoints not covered in the main mock suite.

Test Area	Route	Status
Receipt not attached (404)	GET /expenses/:id/receipt	Implemented
Receipt auth required (401)	GET /expenses/:id/receipt	Implemented
VAT export PDF (200 / 404)	GET /reports/vat/export/pdf	Implemented
VAT export XML (200 / 404)	GET /reports/vat/export/xml	Implemented
Dashboard metrics (200)	GET /reports/dashboard	Implemented
Dashboard auth required (401)	GET /reports/dashboard	Implemented
Audit log (200 + owner/admin only)	GET /security/audit-log	Implemented
Audit log role check (403 for viewer)	GET /security/audit-log	Implemented
Data export (200 + owner only)	POST /security/data-export	Implemented
Data export role check (403 for admin)	POST /security/data-export	Implemented

---

`e2e/api.test.ts` — Full E2E (no mocks, live server)

End-to-end API integration test. Exercises the full Express application stack with a live server.

Status
Implemented

`e2e/billing-flow.e2e.test.ts` — Billing Workflow E2E

Tests the full billing flow through HTTP endpoints with mocked services (no real DB required):

1. Create contact
2. Create invoice
3. Send invoice (draft → sent)
4. Mark invoice paid (sent → paid)
5. Check P&L shows revenue
6. Verify trial balance returns `isBalanced=true`
7. Multi-currency invoice in EUR with country VAT rates
8. Credit note creation

Status
Implemented

`apps/api/tests/unit/` — Unit Tests (service layer)

Service-level unit tests with mocked Prisma. No HTTP layer. Tests business logic in individual service classes.

`invoice-service-calculations.test.ts` — Invoice Arithmetic

Tests `InvoiceService.createInvoice()` arithmetic at the service layer. Verifies:

- `lineTotal = quantity × unitPrice`
- `lineTax = lineTotal × taxRate / 100`
- `subtotal = sum(lineTotals)`
- `taxAmount = sum(lineTaxes)`
- `total = subtotal + taxAmount`
- `baseAmount = total × exchangeRate`

Test Area	Status
Single-line invoice arithmetic	Implemented
Multi-line invoice with mixed tax rates	Implemented
Multi-currency exchange rate application	Implemented
Zero-quantity line items rejected	Implemented
NUMERIC(19,4) precision maintained	Implemented

`two-factor.test.ts` — Two-Factor Authentication Service

Tests `TwoFactorService` at the service level. Mocks: bcryptjs, speakeasy, qrcode, Prisma.

Test Name	Status
`enable - wrong password → throws unauthorized`	Implemented
`enable - correct password → returns secret + QR data URL + 10 codes`	Implemented
`enable - backup codes are hashed before storing`	Implemented
`enable - plaintext backup codes returned once only`	Implemented
`verify - valid TOTP + window=1 → activates 2FA`	Implemented
`verify - invalid TOTP → throws badRequest`	Implemented
`disable - correct password → clears secret + backup codes`	Implemented
`disable - wrong password → throws unauthorized`	Implemented

`sef-submission.test.ts` — SEF (Serbia E-Invoicing) Client

Tests `SefClient` class and `InvoiceService.submitToSef()` fire-and-forget behavior. HTTP calls are mocked via `vi.spyOn(global, 'fetch')`.

Test Area	Status
`SefClient` submits UBL XML to SEF sandbox URL	Implemented
`SefClient` uses production URL in production env	Implemented
`SefClient` retries on network failure	Implemented
`createSefClientFromEnv()` reads credentials from env vars	Implemented
`generateSEFInvoiceXML()` produces valid UBL 2.1 XML	Implemented
`InvoiceService.submitToSef()` never re-throws errors	Implemented

`vat-calculation.test.ts` — VAT Calculation Tests (Country Packages)

Tests pure calculation functions from `@bilko/country-rs`, `@bilko/country-ba`, and `@bilko/country-hr`. No Prisma, no HTTP — stateless math functions.

Country	Functions Tested	Status
Serbia	`calculateSerbianPDV`, `calculateNetFromGrossPDV`, `qualifiesForPausalRegime`, `requiresVATRegistration`, `calculateSerbianCIT`	Implemented
Bosnia	`calculateBosnianPDV`, `calculateNetFromGrossPDV`, `requiresVATRegistration`	Implemented
Croatia	`calculateCroatianPDV`, `calculateNetFromGrossPDV`, `requiresVATRegistration`	Implemented
All rates	Standard, reduced, zero, super-reduced rates per country	Implemented

---

`apps/api/tests/integration/` — Real Database Tests

Integration tests that run against a **real PostgreSQL database** via `docker-compose.test.yml`. Requires Docker.

Run with:

```
cd apps/api
docker-compose -f ../../docker-compose.test.yml up -d
npm run test:integration
```

`auth.integration.test.ts` — Auth Integration

Full registration + login + refresh flow against real DB.

Test Area	Status
Register new organization + owner user	Implemented
Login with correct credentials	Implemented
Refresh access token via cookie	Implemented
Reject duplicate email registration	Implemented

`invoice.integration.test.ts` — Invoice CRUD Integration

Invoice lifecycle against real DB: create → read → update → status change.

Test Area	Status
Create invoice with line items	Implemented
Read invoice by ID	Implemented
Update draft invoice	Implemented
Send invoice (draft → sent)	Implemented
Mark paid (sent → paid)	Implemented

`credit-note-gl.integration.test.ts` — Credit Note GL Integration

Tests credit note creation against real DB — verifies reversing GL entries balance.

Test Area	Status
Credit note creates reversing journal entries	Implemented
Reversing entries balance (debits = credits)	Implemented

Test Area	Status
Original invoice marked as credited	Implemented

`report.integration.test.ts` — Reports Integration

Report generation against real DB with seeded transactions.

Test Area	Status
P&L report returns revenue/expense data	Implemented
VAT report returns correct tax amounts	Implemented
Trial balance returns `isBalanced=true`	Implemented

`tenant-isolation.integration.test.ts` — Multi-Tenant Security

Verifies multi-tenant isolation: Organization A cannot read or modify Organization B's data.

Test Area	Status
Org A cannot list Org B's invoices	Implemented
Org A cannot read Org B's invoice by ID	Implemented
Org A cannot update Org B's invoice	Implemented
Org A cannot delete Org B's contact	Implemented
Cross-tenant requests return 404 (not 403 — no data leakage)	Implemented

---

Coverage Tracking

Module	Current Status	Target
`@bilko/core` accounting engine	Tests implemented (20 cases)	>95%
`@bilko/core` invoicing	Tests implemented (22 cases)	>95%
`@bilko/core` tax/VAT	Tests implemented (23 cases)	>95%

Module	Current Status	Target
`@bilko/core` multi-currency	Tests implemented (24 cases)	>90%
`@bilko/core` chart of accounts	Tests implemented (32 cases)	>90%
Auth API	Tests implemented (11 cases)	>85%
Invoices API	Tests implemented (11 cases)	>80%
Expenses API	Tests implemented (9 cases)	>80%
Banking API	Tests implemented (10 cases)	>75%
Reports API	Tests implemented (9 cases)	>75%
Country/Tax-Rates API	Tests implemented (27 cases)	>80%
Chatbot API	Tests implemented (9 cases)	>80%
Security/Audit API	Tests implemented (via new-endpoints)	>80%
Invoice GL Reversal (service layer)	Tests implemented	>90%
Two-Factor Auth (service layer)	Tests implemented (8 cases)	>90%
SEF Client + Submission	Tests implemented	>85%
VAT Calculations (country packages)	Tests implemented	>95%
Multi-tenant isolation (real DB)	Tests implemented (5 scenarios)	100%
Invoice CRUD (real DB)	Tests implemented	>80%
Credit note GL (real DB)	Tests implemented	>90%

Test Execution Commands

```
# All tests (from project root — mock + unit + E2E suites)
npm run test

# Core unit tests only
cd packages/core && npx vitest run

# API mock suite only (no DB required)
cd apps/api-express && npx vitest run

# API unit suite only
cd apps/api-express && npx vitest run tests/unit/
```

```
# API E2E suite only (mocked services, no DB)
cd apps/api-express && npx vitest run tests/e2e/

# Real DB integration tests (requires docker-compose.test.yml)
docker-compose -f docker-compose.test.yml up -d
cd apps/api-express && npm run test:integration

# Watch mode (re-run on change)
cd packages/core && npx vitest
cd apps/api-express && npx vitest

# Specific file
cd apps/api-express && npx vitest run tests/auth.test.ts
cd apps/api-express && npx vitest run tests/unit/two-factor.test.ts

# With coverage
cd packages/core && npx vitest run --coverage

# Verbose output
npx vitest run --reporter=verbose
```

Related Documents

- Testing Guide: TESTING-GUIDE.md
- Backend Architecture: ../backend/BACKEND-ARCHITECTURE.md

Last Updated: 2026-03-02 **Status:** Active **Total Tests:** ~390 across 27 test files (mock + unit + E2E + integration)

Test Plan

Bilko — Test Plan

Version: 1.1 **Date:** 2026-05-21 **Project ID:** bbd77cc0 **Status:** Partially stale — use `docs/testing/TEST-STRATEGY.md`, `docs/testing/E2E-TEST-PLAN.md`, and `docs/testing/DEMO-TESTING-PLAN.md` for current policy

> **Update note:** this long-form plan still contains historical Express/Prisma examples. Current Bilko policy is layered testing: focused unit tests, strong Ktor/PostgreSQL integration/contract tests, critical Playwright E2E, non-destructive real-demo smoke, and resettable full-demo rehearsal.

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`
`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

Test Case	Assertion
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
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)');
  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 1 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
```

8.9 Accessibility Testing

Bilko serves SMB users across the Balkans, including users with disabilities and users on assistive technologies. WCAG 2.1 AA compliance is the target for the Bilko web application.

Tools

Tool	Purpose	When Run
axe-core (via `@axe-core/playwright`)	Automated WCAG 2.1 AA audit	Every E2E test run; CI on PR
Lighthouse Accessibility audit	Aggregate accessibility score	Weekly + pre-release
Manual screen reader test (NVDA/VoiceOver)	Verify keyboard navigation, ARIA	Pre-launch; after major UI changes

Automated Accessibility Tests (Playwright + axe-core)

```
// e2e/tests/accessibility.spec.ts
import { test, expect } from '@playwright/test'
import AxeBuilder from '@axe-core/playwright'

test.describe('WCAG 2.1 AA Compliance', () => {
  test.beforeEach(async ({ page }) => {
    // Login
    await page.goto('/login')
    await page.fill('[name="email"]', 'demo@bilko.io')
    await page.fill('[name="password"]', 'Demo123!')
    await page.click('button[type="submit"]')
    await page.waitForURL('/dashboard')
  })

  test('Dashboard has no WCAG 2.1 AA violations', async ({ page }) => {
    await page.goto('/dashboard')
```

```
    const accessibilityScanResults = await new AxeBuilder({ page })
      .withTags(['wcag2a', 'wcag2aa', 'wcag21a', 'wcag21aa'])
      .analyze()
    expect(accessibilityScanResults.violations).toEqual([])
  })

  test('Invoice create form has no WCAG 2.1 AA violations', async ({ page }) => {
    await page.goto('/invoices/new')
    const accessibilityScanResults = await new AxeBuilder({ page })
      .withTags(['wcag2a', 'wcag2aa'])
      .analyze()
    expect(accessibilityScanResults.violations).toEqual([])
  })

  test('Invoices list page has no WCAG 2.1 AA violations', async ({ page }) => {
    await page.goto('/invoices')
    const accessibilityScanResults = await new AxeBuilder({ page })
      .withTags(['wcag2a', 'wcag2aa'])
      .analyze()
    expect(accessibilityScanResults.violations).toEqual([])
  })

  test('Reports page has no WCAG 2.1 AA violations', async ({ page }) => {
    await page.goto('/reports')
    const accessibilityScanResults = await new AxeBuilder({ page })
      .withTags(['wcag2a', 'wcag2aa'])
      .analyze()
    expect(accessibilityScanResults.violations).toEqual([])
  })

  test('Settings page has no WCAG 2.1 AA violations', async ({ page }) => {
    await page.goto('/settings')
    const accessibilityScanResults = await new AxeBuilder({ page })
      .withTags(['wcag2a', 'wcag2aa'])
      .analyze()
    expect(accessibilityScanResults.violations).toEqual([])
  })
})
```

Key WCAG 2.1 AA Requirements for Bilko

Requirement	Criterion	Bilko Implementation
Color contrast	1.4.3 — minimum 4.5:1 for normal text, 3:1 for large text	Tailwind colors verified for contrast ratios
Keyboard navigation	2.1.1 — all functions accessible by keyboard	Focus management in modals, invoice wizard steps
Focus visible	2.4.7 — visible focus indicators	Tailwind `focus:ring` classes on all interactive elements
Labels on inputs	1.3.1 — form inputs have programmatic labels	shadcn/ui `<Label>` components linked to inputs
Error identification	3.3.1 — form errors identified and described	Zod validation errors surfaced in accessible error messages
Resize text	1.4.4 — content usable at 200% zoom	Tested at 200% browser zoom
ARIA landmark roles	1.3.1 — navigation, main, form landmarks	`<nav>`, `<main>`, `<form role="form">` in layout
Alt text for icons	1.1.1 — non-text content has text alternative	Lucide icon buttons have `aria-label`; decorative icons have `aria-hidden`
Meaningful link text	2.4.6 — links have descriptive text	"View invoice INV-2026-001" not "click here"
Language of page	3.1.1 — HTML lang attribute	`<html lang="sr">`, `<html lang="bs">`, `<html lang="hr">` per user locale

Accessibility CI Integration

```
# In .github/workflows/test.yml — add to e2e-tests job
- name: Run accessibility tests
  run: npx playwright test e2e/tests/accessibility.spec.ts
  env:
    PLAYWRIGHT_BASE_URL: http://localhost:3000

- name: Run Lighthouse accessibility audit
  run: |
    npm install -g lighthouse
    lighthouse http://localhost:3000/dashboard \
      --only-categories=accessibility \
      --output=json \
      --output-path=lighthouse-a11y.json
    node -e "
```

```
      const report = require('./lighthouse-a11y.json');
      const score = report.categories.accessibility.score * 100;
      if (score < 90) {
        console.error('Lighthouse accessibility score: ' + score + ' (required: >= 90)');
        process.exit(1);
      }
      console.log('Lighthouse accessibility score: ' + score + ' ✓');
    "
```

Acceptance Criteria — Accessibility

- **Zero** axe-core violations on: dashboard, invoice list, invoice create, reports, settings
- **Lighthouse accessibility score ≥ 90** on all main pages
- **All form inputs** have associated `<label>` elements
- **All icon buttons** have `aria-label` attributes
- **Focus trap** correctly implemented in modals (invoice create wizard, expense create)
- **Keyboard shortcut** for most common action: `N` to create new invoice (when not in input field)

---

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/                          ← EXISTS (~99 integration tests, mocked Prisma)
│       ├── setup.ts                    ← Shared test setup, JWT helpers, data factories
│       ├── auth.test.ts                ← 11 tests: register, login, refresh, logout, me
```

```
│       ├── invoices.test.ts          ← 11 tests: CRUD + send/pay lifecycle
│       ├── expenses.test.ts          ← 9 tests: CRUD + approve/reject
│       ├── contacts.test.ts          ← 9 tests: CRUD
│       ├── accounts.test.ts          ← 4 tests: chart of accounts
│       ├── banking.test.ts           ← 10 tests: accounts, import, reconcile
│       ├── reports.test.ts           ← 9 tests: P&L, balance sheet, VAT, trial balance
│       ├── transactions.test.ts      ← 9 tests: ledger list, filter, detail
│       ├── country.test.ts           ← 27 tests: RS/BA/HR tax rates, invoice formats
│       └── e2e/
│           └── api.test.ts           ← Full end-to-end API test (no mocks)
│
└── e2e/                              ← To be created (Playwright browser tests)
    ├── 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

Module	Unit Coverage	Integration Coverage
`@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%
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.