Test Strategy

Test Strategy

> **Project:** Bilko **Version:** 1.1 **Date:** 2026-05-21 **Author:** Ops Architect / ALAI Documentation Team **Status:** Active **Reviewers:** Tech Lead, Alem Bašić

Document History

Version	Date	Author	Changes
0.1	2026-02-23	Ops Architect	Initial draft
1.0	2026-02-25	ALAI Documentation Team	Finalized — approved for production use
1.1	2026-05-21	ALAI Documentation Team	Clarified industry/Spotify-style layered strategy, real-demo smoke gate, and full-demo rehearsal policy

---

1. Testing Philosophy & Principles

Financial software has a higher correctness bar than typical web apps. A bug in VAT calculation or double-entry bookkeeping is not a UX inconvenience — it's a compliance failure that could expose Bilko users to tax liability or audit findings.

Core Principles:

1. **Financial logic is P0** — VAT calculations, double-entry balance, NUMERIC precision are tested at >95% coverage before any feature ships
2. **Tests are first-class code** — reviewed, maintained, and refactored alongside production code
3. **Test the behavior, not the implementation** — tests enable safe refactoring of internals
4. **Fast feedback** — unit tests run in < 3 min; full suite < 10 min

5. **No test = no ship** — financial logic without a test is a P0 blocker for merging
6. **Isolation** — every test cleans up after itself; no test depends on another

Testing philosophy: Bilko follows an industry-standard layered strategy: focused unit coverage for financial calculations and business logic, strong integration/contract coverage for Ktor API + PostgreSQL behavior, and a small Playwright layer for critical user journeys. This is compatible with Spotify's public "testing honeycomb" direction: most confidence should come from service interaction tests and contracts, not from trying to automate every behavior through a browser. We do not aim for 100% E2E coverage.

---

2. Layered Test Strategy

```
                Playwright E2E / Demo Smoke
         Critical browser journeys + deployed health evidence


        Integration + Contract Tests (dominant)
      Ktor routes, services, PostgreSQL/Testcontainers, auth,
      RBAC, multi-tenant isolation, frontend/backend API contract


                Focused Unit Tests
      Financial engine, VAT, currency, validators, pure logic
```

Target distribution:

- Unit tests — financial logic and pure business rules, especially `packages/core` and accounting/tax code
- Integration/contract tests — API routes, DB behavior, auth/session boundaries, RBAC, org isolation, frontend/backend API compatibility
- Critical Playwright E2E — a small, maintained set for invoice, expense, report, auth, and settings flows
- Real-demo smoke — non-destructive deployed health check against `https://bilko-demo.alai.no`
- Full-demo rehearsal — resettable demo tenant/environment used for stakeholder demos; not the deploy gate

---

3. Testing Tools

Type	Tool	Version	Purpose	Config

Unit testing	Vitest + Kotlin/JUnit	Latest	Business logic, utilities, services	package configs / Gradle
Mocking	Vitest, MockK	—	Mock external deps where appropriate	Built-in / Gradle
Integration testing	Ktor test host + JUnit	Latest	API endpoint testing with PostgreSQL	`apps/api/build.gradle.kts`
Test database	PostgreSQL	15/16	Real database via local DB/Testcontainers	Gradle `integrationTest`
E2E testing	Playwright	Latest	Browser automation, critical user flows	`apps/e2e/playwright.config.ts`
Coverage	Kover + Vitest	—	Coverage reports and ratchets	Gradle Kover / package configs
Performance	k6	Latest	Load testing (PLANNED Phase 2)	`apps/e2e/load/`

Why Vitest (not Jest)

- ESM native, Vite-based → faster
- Compatible with Turborepo
- Watch mode with HMR
- Same API as Jest (easy migration)

Why Playwright (not Cypress)

- Multi-browser: Chromium, Firefox, WebKit (Safari)
- Auto-wait (no flaky tests from race conditions)
- Parallel execution (workers: 4)
- Video and trace on failure

---

4. Test Scope by Layer

4.1 Unit Tests (Vitest)

Attribute	Value
Scope	Pure functions: VAT calculation, double-entry validation, currency conversion, invoice totals, date utils, number formatting
External dependencies	Mocked — no real DB, network, or filesystem

Attribute	Value
Coverage target	> 95% for financial logic; > 90% utilities; > 80% services; > 80% overall
Execution time	< 3 minutes
Runs on	Every commit, pre-commit hook (lint + type-check only), CI on every push
Written by	Developer who writes the feature

What to unit test:

- `calculateVAT(amount, rate, country)` — Serbia 20%, BiH 17%, Croatia 25%
- `validateDoubleEntry(debit, credit)` — must be equal, error on imbalance
- `convertCurrency(amount, fromCurrency, toCurrency, exchangeRate)` — NUMERIC(19,4)
- `calculateInvoiceTotal(items)` — subtotal, tax, discount, total
- `lockExchangeRate(date, fromCurrency, toCurrency)` — historical rate, not today's

What NOT to unit test:

- Framework internals (Ktor, Next.js, Exposed, JDBC)
- Simple property getters/setters with no logic
- Full browser journeys that belong in Playwright E2E or demo smoke

4.2 Integration + Contract Tests

Attribute	Value
Scope	Ktor API routes, service boundaries, PostgreSQL behavior, contracts
External dependencies	Real PostgreSQL via local DB or Testcontainers where needed
Coverage target	All service boundaries; > 80% of integration paths
Execution time	< 10 minutes for blocking gate
Runs on	Every PR / deploy gate, blocking merge where configured
Written by	Developer who writes the API endpoint or client contract

What to integration test:

- Auth flow (register, login, refresh, logout)
- Invoice CRUD + status transitions (draft → sent → paid)
- Expense CRUD + approval flow
- Reports API (P&L, VAT, balance sheet)
- Organization scoping — org A cannot read org B's data (P0 security test)
- RBAC enforcement — viewer cannot create, owner can delete

4.3 E2E Tests (Playwright)

Attribute	Value
Scope	Critical user journeys through deployed/staging application
External dependencies	Real staging services; production/demo only for non-destructive smoke
Coverage target	Critical journeys + smoke, not exhaustive UI coverage
Execution time	< 8 minutes for critical gate; < 1 minute for real-demo smoke
Runs on	Post-staging deploy, pre-production gate, post-deploy smoke
Written by	Developer + QA collaboration

Critical journeys:

1. Auth/session Flow: Login → refresh/session validation → logout or session expiry behavior
2. Invoice Flow: Create draft in resettable staging/demo tenant → Preview → Send/Mark Paid where safe
3. Expense Flow: Add → Upload Receipt → Approve/Reject/Pay where safe
4. Report Flow: Generate P&L/VAT report → Export PDF/XLSX where safe
5. Settings Flow: Organization/settings/users page loads and key controls are visible

Rule: public real-demo tests must be non-destructive. Registration, deletion, rate-limit torture, invoice/expense creation, and expected-fail regressions belong in resettable staging/nightly suites, not the live demo smoke gate.

---

5. Test Data Management

Approach	Used For	Tool	Cleanup
Test factories	Unit + integration	`apps/api/src/test/factories/`	Per-test (beforeEach teardown)
Database seeding	E2E/full-demo rehearsal	Flyway fixtures / seed scripts / API factories	Per resettable environment run
PostgreSQL transactions	Integration tests	Test transaction or teardown helpers	Per test

Isolation rule: integration tests must create isolated organization/user fixtures and clean up deterministically. Cross-test dependence is forbidden.

Test org pattern: Each integration test creates a fresh `bilko_test` organization and user to prevent cross-test contamination.

---

6. Coverage Requirements

Layer	Lines	Branches	Functions	Enforcement
Financial logic (VAT, double-entry, currency)	≥ 95%	≥ 90%	≥ 100%	CI hard fail
Authentication utils	≥ 95%	≥ 90%	≥ 100%	CI hard fail
API handlers	≥ 80%	≥ 75%	≥ 80%	CI hard fail
Utilities	≥ 90%	≥ 85%	≥ 90%	CI hard fail
Overall minimum	**≥ 80%**	**≥ 75%**	**≥ 80%**	CI hard fail

Coverage enforcement: Vitest coverage thresholds in `vitest.config.ts`. CI pipeline fails if below threshold.

---

7. Quality Gates

PR Merge Gate

- ☐ All unit tests pass
- ☐ All integration tests pass
- ☐ Coverage ≥ minimum thresholds
- ☐ Linting passes (ESLint + Prettier)
- ☐ Type checking passes (TypeScript strict)
- ☐ No new HIGH/CRITICAL security findings

Staging Deploy Gate

- ☐ All PR gates passed
- ☐ Build artifact created successfully

Production Deploy Gate

- [ ] Critical E2E gate passes on staging/resettable environment
- [ ] Real-demo smoke passes after deploy with screenshot/video evidence
- [ ] Performance baseline not degraded > 20% for relevant changes
- [ ] Manual approval in CI pipeline

---

8. Responsibility Matrix

Test Type	Writes	Reviews	Maintains	Signs Off
Unit tests	Developer	PR reviewer	Developer	Tech Lead
Integration tests	Developer	QA / Tech Lead	Developer	Tech Lead
E2E tests	Developer	Tech Lead	Developer	Tech Lead
Performance tests	DevOps	Tech Lead	DevOps	Alem Bašić

---

9. Test Reporting & Metrics

Metric	Target
Test pass rate	≥ 99% unit, ≥ 95% E2E
Flaky test rate	< 2%
Full suite execution time	< 10 min
Coverage trend	Stable or improving per sprint
Financial logic coverage	≥ 95% at all times

---

10. Continuous Testing in CI/CD

Stage	Tests Run	Blocking
Pre-commit (local)	lint + type-check only	Recommended (Husky)
PR open/update	unit + integration + lint + type-check	Yes — blocks merge
Staging deploy	Critical E2E (Playwright, Chromium primary; other browsers scheduled/risk-based)	Yes — blocks production

Stage	Tests Run	Blocking
Production deploy	Real-demo smoke (`npm run test:real-demo-smoke`) with evidence	Yes — rollback/escalate on failure
Nightly / scheduled	Full E2E regression + destructive/resettable tests + performance	No — alerts/issues, not automatic deploy blocker

Related Documents

- Test Plan
- E2E Test Plan
- Performance Test Plan
- Definition of Done
- CI/CD Pipeline
- TESTING-GUIDE.md
- TEST-INVENTORY.md
- Demo Testing Plan

Approval

Role	Name	Date	Signature
Author	Ops Architect	2026-02-23	
Reviewer	Tech Lead		
Approver	Alem Bašić		

Revision #12
Created 2026-02-24 22:50:55 UTC by John
Updated 2026-07-12 20:00:53 UTC by John