High-Level Design (HLD)

Bilko — High-Level Design (HLD)

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

Table of Contents

---

1. System Overview

Bilko is a cloud-based accounting SaaS for Balkan SMBs operating in Serbia, Bosnia & Herzegovina, and Croatia. It is modeled after Fiken (Norway) — simple, compliant, and affordable.

Key design goals:

- Double-entry bookkeeping engine with immutable audit trail

- Multi-country regulatory compliance (RS, BA, HR) via pluggable country modules
- Multi-currency support with exchange rate locking at transaction date
- Organization-scoped multi-tenancy
- All monetary values stored as `NUMERIC(19,4)` — never float

Target users: 50K–500K SMBs across the Balkan region **Domains:** bilko.io (primary), bilko.rs (Serbia redirect)

---

2. Monorepo Structure

The project uses **Turborepo** for monorepo management.

```
Bilko/
├── apps/
│   ├── web/                  # Next.js 15 frontend (App Router)
│   └── api/                  # Express + TypeScript backend
├── packages/
│   ├── database/             # Prisma schema + Prisma Client (@bilko/database)
│   ├── core/                 # Accounting engine (@bilko/core)
│   ├── country-rs/           # Serbia plugin (@bilko/country-rs)
│   ├── country-ba/           # Bosnia & Herzegovina plugin (@bilko/country-ba)
│   ├── country-hr/           # Croatia plugin (@bilko/country-hr)
│   └── ui/                   # Shared UI scaffold (empty, placeholder)
├── infrastructure/
│   ├── terraform/            # AWS infrastructure as code
│   ├── docker/               # Dockerfiles and docker-compose
│   ├── nginx/                # Nginx reverse proxy config
│   ├── pm2/                  # PM2 process manager config
│   └── scripts/              # Deployment shell scripts
├── docs/                     # All documentation
│   ├── backend/              # API, auth, services, DB schema docs
│   ├── frontend/             # Pages, components, design system docs
│   ├── infrastructure/       # Deployment, CI/CD, environment docs
│   ├── regulatory/           # Country-specific accounting law summaries
│   ├── security/             # Security architecture, compliance
│   └── testing/              # Testing guides and inventory
├── CLAUDE.md                 # Project AI assistant instructions
└── PIPELINE.md               # 8-gate checklist
```

---

3. Component Architecture

```
graph TB
    subgraph Client["Client Layer"]
        Browser["Browser / Mobile"]
    end

    subgraph Frontend["apps/web — Next.js 15"]
        AppRouter["App Router"]
        Pages["Pages (Dashboard, Invoices, Expenses, Reports, Banking, Settings)"]
        Components["shadcn/ui Components"]
        MockData["lib/mock-data.ts (TEMP — replace with API calls)"]
        Zustand["Zustand Store (future)"]
    end

    subgraph Backend["apps/api — Express + TypeScript"]
        Middleware["Middleware Stack (helmet → cors → json → rate-limit → auth → validate → handler → error)"]
        Routes["Route Modules (auth, invoices, expenses, contacts, accounts, transactions, reports, banking, settings)"]
        Services["Service Layer (Invoice, Expense, Contact, Account, Banking, Report, Settings)"]
        CoreEngine["@bilko/core (accounting, tax, multi-currency, bank-import)"]
    end

    subgraph Plugins["Country Plugins"]
        RS["@bilko/country-rs (Serbia: PDV 20%, SEF, CIT 15%)"]
        BA["@bilko/country-ba (BiH: PDV 17%, IFRS, UIO)"]
        HR["@bilko/country-hr (Croatia: PDV 25%, eRačun, FINA)"]
    end

    subgraph Data["Data Layer"]
        Prisma["@bilko/database — Prisma Client"]
        PG["PostgreSQL 15 (RDS)"]
    end

    subgraph Storage["Storage"]
        R2["Cloudflare R2 (PDF storage, receipts)"]
    end
```

```
    Browser --> AppRouter
    AppRouter --> Pages
    Pages --> Components
    Pages --> MockData
    Pages --> Zustand

    Pages -->|"REST API calls (future)"| Routes
    Middleware --> Routes
    Routes --> Services
    Services --> CoreEngine
    Services --> Plugins
    Services --> Prisma
    Prisma --> PG
    Services --> R2
```

---

4. Data Flow

4.1 Standard Request Flow

```
sequenceDiagram
    participant U as User (Browser)
    participant FE as Next.js Frontend
    participant MW as Middleware Stack
    participant RT as Route Handler
    participant SV as Service Layer
    participant CE as @bilko/core
    participant PR as Prisma Client
    participant DB as PostgreSQL

    U->>FE: User Action (e.g., Create Invoice)
    FE->>MW: POST /api/v1/invoices + Bearer token
    MW->>MW: helmet (security headers)
    MW->>MW: cors (origin check)
    MW->>MW: rate-limit (100 req/15min per IP)
    MW->>MW: authGuard (verify JWT access token)
    MW->>MW: organizationScope (attach orgId to request)
```

```
    MW->>MW: validate (Zod schema check)
    MW->>RT: req.user + req.body validated
    RT->>SV: invoiceService.createInvoice(orgId, userId, data)
    SV->>CE: calculateVAT(), lockExchangeRate()
    SV->>PR: prisma.$transaction([create invoice, create items])
    PR->>DB: INSERT invoices, invoice_items
    DB-->>PR: Created records
    PR-->>SV: Invoice with items
    SV-->>RT: Formatted response
    RT-->>FE: 201 JSON response
    FE-->>U: Updated UI
```

4.2 Invoice Lifecycle with Double-Entry

```
stateDiagram-v2
    [*] --> draft: POST /api/v1/invoices
    draft --> sent: PATCH /status {action: "send"}\n→ Creates TX: DR Receivable / CR Revenue
    sent --> viewed: (future: email tracking webhook)
    viewed --> paid: PATCH /status {action: "mark-paid"}\n→ Creates TX: DR Bank / CR
Receivable
    sent --> paid: PATCH /status {action: "mark-paid"}
    draft --> cancelled: PATCH /status {action: "cancel"}
    sent --> cancelled: PATCH /status {action: "cancel"}
    viewed --> overdue: (cron job: past due date)
    overdue --> paid: PATCH /status {action: "mark-paid"}
```

4.3 Expense Lifecycle with Double-Entry

```
stateDiagram-v2
    [*] --> pending: POST /api/v1/expenses
    pending --> approved: PATCH /expenses/:id/approve\n→ Creates TX: DR Expense / CR Payable
    approved --> paid: PATCH /expenses/:id/pay\n→ Creates TX: DR Payable / CR Bank
    pending --> rejected: (future endpoint)
```

---

5. Tech Stack Rationale

Layer	Technology	Rationale

Frontend Framework	Next.js 15 (App Router)	SSR for fast initial load, SEO, file-system routing, React Server Components
Frontend Language	TypeScript 5.3	Type safety, IDE support, catch errors at compile time
Styling	Tailwind CSS 4 + shadcn/ui	Utility-first styling with accessible, unstyled Radix UI primitives
State Management	Zustand 4.5 (planned)	Lightweight global state; React hooks used currently during mock phase
Charts	Recharts 2.15	React-native chart library, composable, good TypeScript support
Icons	Lucide React	Consistent icon set, tree-shakeable, maintained fork of Feather
Backend Framework	Express + TypeScript	Minimal, battle-tested, massive ecosystem; team familiarity
ORM	Prisma	Type-safe database access, migration management, schema-as-code
Database	PostgreSQL 15	NUMERIC(19,4) for money, mature ACID compliance, full-text search
Auth	JWT (access + refresh)	Stateless, scalable; no session store needed
Validation	Zod	Runtime schema validation with full TypeScript inference
Monorepo	Turborepo	Fast incremental builds, shared packages, workspace management
Decimal Arithmetic	Decimal.js	Arbitrary-precision arithmetic — required for financial calculations
Infrastructure	AWS (EC2, RDS, S3, CloudFront)	Reliable, eu-central-1 region close to Balkan users
IaC	Terraform	Declarative infrastructure, reproducible environments

---

6. Multi-Tenancy Model

Bilko uses **organization-scoped multi-tenancy** — all business data is isolated by `organizationId`.

```
erDiagram
    Organization {
        uuid id PK
```

```
        string name
        string baseCurrency "EUR by default"
        string country "RS, BA, HR"
        string language "sr, bs, hr"
    }
    User {
        uuid id PK
        uuid organizationId FK
        enum role "owner, admin, accountant, viewer"
    }
    Invoice {
        uuid id PK
        uuid organizationId FK
    }
    Expense {
        uuid id PK
        uuid organizationId FK
    }
    Transaction {
        uuid id PK
        uuid organizationId FK
    }
    Organization ||--o{ User : has
    Organization ||--o{ Invoice : owns
    Organization ||--o{ Expense : owns
    Organization ||--o{ Transaction : owns
```

Enforcement mechanism: The `organizationScope` middleware (`apps/api/src/middleware/org-scope.ts`) attaches `req.user.organizationId` to every authenticated request. All service methods receive `organizationId` as first parameter and filter all Prisma queries with `where: { organizationId }`. Cross-organization data access is structurally impossible via the API layer.

RBAC roles:

Role	Permissions
`owner`	Full access, manage users, change roles, delete org
`admin`	Full access except role management
`accountant`	CRUD on invoices, expenses, transactions, reports
`viewer`	Read-only access to all data

---

7. Authentication Architecture

```
sequenceDiagram
    participant C as Client
    participant A as API /auth
    participant DB as PostgreSQL

    C->>A: POST /api/v1/auth/login {email, password}
    A->>DB: findUser(email) → user + passwordHash
    A->>A: bcrypt.verify(password, passwordHash)
    A->>A: signAccessToken({sub, email, role, orgId}) [15min, JWT_SECRET]
    A->>A: signRefreshToken({sub, jti}) [7d, JWT_REFRESH_SECRET]
    A-->>C: 200 {accessToken, user, org} + Set-Cookie: refreshToken (httpOnly)

    Note over C,A: Subsequent requests
    C->>A: GET /api/v1/invoices + Authorization: Bearer <accessToken>
    A->>A: authGuard: verifyAccessToken() → payload
    A->>A: organizationScope: attach orgId to req
    A-->>C: 200 {data}

    Note over C,A: Token refresh
    C->>A: POST /api/v1/auth/refresh (cookie: refreshToken)
    A->>A: verifyRefreshToken() → {sub, jti}
    A->>DB: findUser(sub) → user
    A->>A: signAccessToken(newPayload)
    A-->>C: 200 {accessToken}
```

Token storage:

- Access token: returned in response body, client stores in memory
- Refresh token: `httpOnly` cookie, path `/api/v1/auth`, `SameSite: strict`

Security:

- Passwords: bcrypt with 12 salt rounds (`apps/api/src/utils/password.ts`)
- JWT: RS256 signing, issuer/audience validation (`apps/api/src/utils/jwt.ts`)
- Optional 2FA: TOTP via `User.twoFactorSecret` (field exists, not yet wired)

---

8. Multi-Currency Architecture

All monetary amounts stored as `DECIMAL(19,4)` in PostgreSQL. The system maintains both the transaction currency amount and the base-currency equivalent.

Key fields on monetary entities:

Field	Type	Purpose
`currencyCode`	CHAR(3)	ISO 4217 currency of the transaction
`exchangeRate`	DECIMAL(12,6)	Rate locked at transaction date
`amount`	DECIMAL(19,4)	Amount in transaction currency
`baseAmount`	DECIMAL(19,4)	Amount converted to org's baseCurrency

Rate locking: When an invoice or expense is created, the exchange rate is fetched from the `ExchangeRate` table for the most recent date on or before the transaction date and locked permanently. Historical rates are never recalculated (`packages/core/src/multi-currency/index.ts`: `lockExchangeRate()`).

Supported currencies: EUR, RSD, BAM, HRK, USD, GBP, CHF

Fallback: If no exchange rate is found for a currency pair on a given date, the system logs a warning and uses 1.0. This is a known gap — exchange rate population is a prerequisite for multi-currency accuracy.

---

9. Country Plugin System

Each country is a separate npm package with the same module structure:

```
packages/country-{code}/src/
├── tax/index.ts       # VAT/PDV calculation, CIT, WHT
├── chart/index.ts     # Country-specific chart of accounts
├── fiscal/index.ts    # Fiscal year rules
├── filing/index.ts    # Tax filing periods and deadlines
├── locale/index.ts    # Language/formatting (date, currency)
└── index.ts           # Re-exports all modules
```

Country-specific data:

Country	Plugin	VAT Standard	VAT Reduced	CIT	E-Invoice
Serbia (RS)	`@bilko/country-rs`	20%	10%	15% flat	SEF (UBL 2.1) mandatory since 2023

Country	Plugin	VAT Standard	VAT Reduced	CIT	E-Invoice
Bosnia & Herzegovina (BA)	`@bilko/country-ba`	17%	none	10% (FBiH/RS both)	CPF (pending, ~2026)
Croatia (HR)	`@bilko/country-hr`	25%	13%, 5%	10%/18% progressive	eRačun (UBL 2.1) mandatory since 2026

The core engine (`@bilko/core`) provides country-agnostic accounting primitives. Country plugins extend these with jurisdiction-specific rules without modifying core logic.

---

10. Infrastructure Overview

```
graph LR
    subgraph DNS["Route 53"]
        D1["bilko.io"]
        D2["api.bilko.io"]
    end

    subgraph CDN["CloudFront"]
        CF["CloudFront Distribution\n(bilko.io → S3/Next.js)"]
    end

    subgraph Compute["EC2 (eu-central-1)"]
        NG["Nginx (reverse proxy)"]
        PM2["PM2 (process manager)"]
        API["Express API\n(Node.js)"]
        WEB["Next.js Frontend\n(standalone build)"]
    end

    subgraph Data["Data Layer"]
        RDS["RDS PostgreSQL 15\n(Multi-AZ)"]
        S3["S3 (backups, assets)"]
        R2["Cloudflare R2\n(PDFs, receipts)"]
    end

    subgraph Monitor["Monitoring"]
        CW["CloudWatch\n(logs, alarms)"]
    end
```

```
    D1 --> CF --> NG
    D2 --> NG
    NG --> PM2
    PM2 --> API
    PM2 --> WEB
    API --> RDS
    API --> R2
    API --> CW
```

Key infrastructure decisions:

- Region: `eu-central-1` (Frankfurt) — closest to Balkan users with strong data residency
- RDS Multi-AZ for database high availability
- CloudFront for global CDN caching of static frontend assets
- PM2 for Node.js process management and zero-downtime restarts
- Terraform backend: S3 state bucket + DynamoDB lock table in `eu-central-1`

---

11. Security Model

Layer	Control
Transport	HTTPS enforced (HSTS, `maxAge: 31536000`, `includeSubDomains`)
Security headers	helmet (CSP, X-Frame-Options: deny, X-Content-Type-Options: noSniff)
CORS	Whitelist: `bilko.io`, `www.bilko.io`, `localhost:3000`
Rate limiting	100 req/15min per IP (general); stricter limit on `/auth/login` and `/auth/register`
Authentication	JWT access token (15min) + refresh token (7d, httpOnly cookie)
Authorization	RBAC checked per endpoint; `organizationScope` middleware enforces tenancy
Password storage	bcrypt, 12 salt rounds
Audit trail	`LoggedAction` table — append-only, captures all INSERT/UPDATE/DELETE with user, timestamp, old/new values
Money precision	NUMERIC(19,4) everywhere; Decimal.js in business logic
Transaction immutability	`Transaction.locked = true` makes records unmodifiable
SQL injection	Prisma parameterized queries — no raw SQL in business logic

Layer	Control
Secret management	Environment variables; never committed to repository

---

Revision #3
Created 2026-02-23 10:24:30 UTC by John
Updated 2026-05-24 20:01:36 UTC by John