

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](#)
 2. [Monorepo Structure](#)
 3. [Component Architecture](#)
 4. [Data Flow](#)
 5. [Tech Stack Rationale](#)
 6. [Multi-Tenancy Model](#)
 7. [Authentication Architecture](#)
 8. [Multi-Currency Architecture](#)
 9. [Country Plugin System](#)
 10. [Infrastructure Overview](#)
 11. [Security Model](#)
-

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 IaC – future scale migration (not active at MVP)
│   ├─ docker/        # Dockerfiles and docker-compose (local dev)
│   ├─ nginx/         # Nginx reverse proxy config (self-hosted fallback)
│   ├─ pm2/           # PM2 process manager config (self-hosted fallback)
│   └─ 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/min 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)
```

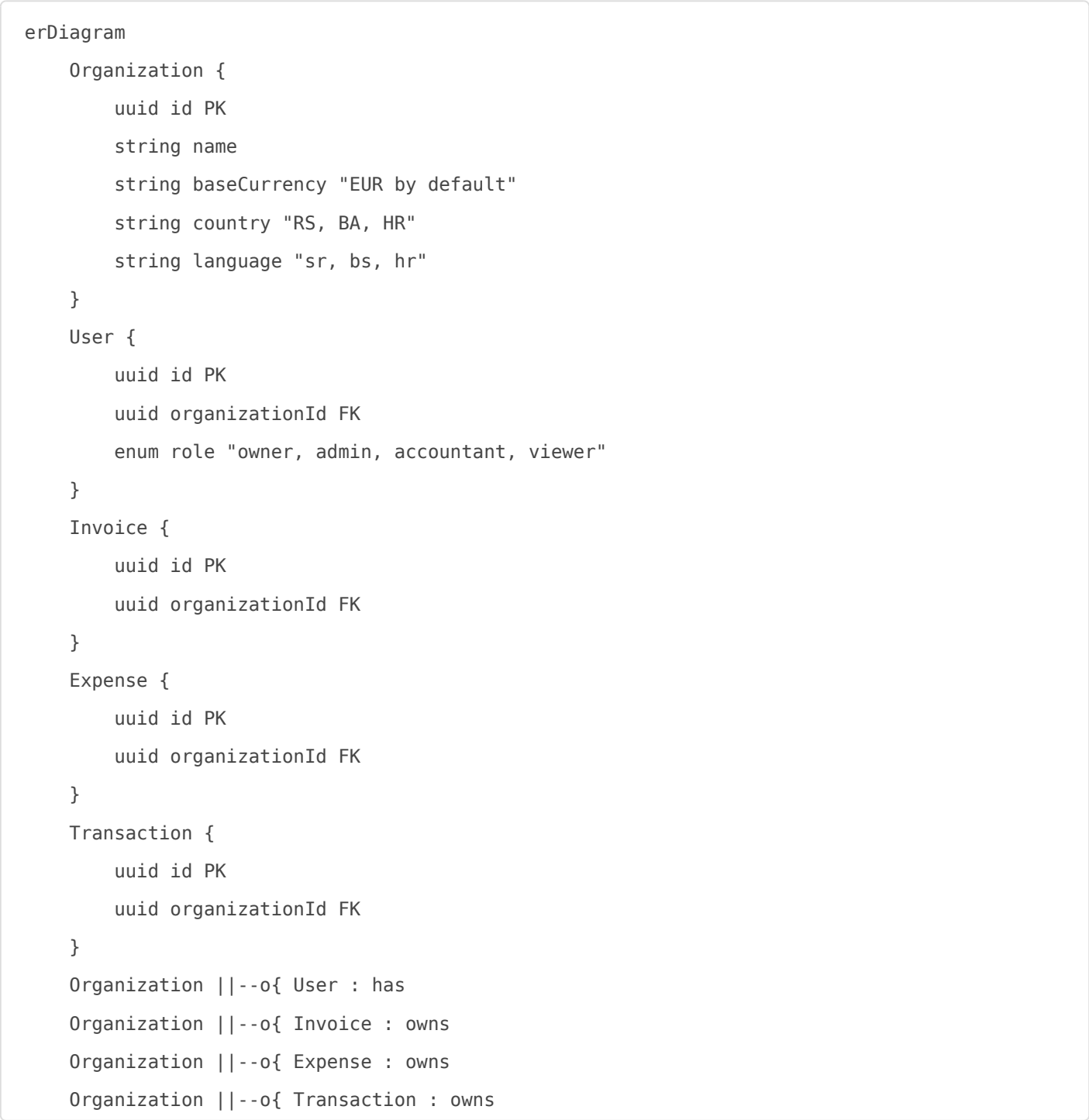
5. Tech Stack Rationale

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
Frontend Hosting	Vercel	Edge network, zero-config Next.js deployment, automatic preview deployments
Backend Hosting	Railway (EU Frankfurt)	Managed containers, automatic TLS, built-in PostgreSQL, €21/mo MVP cost
File Storage	Cloudflare R2	S3-compatible, zero egress fees, stores PDFs and receipts
IaC (future)	Terraform	Prepared for AWS migration at scale; configs in <code>infrastructure/terraform/</code>

6. Multi-Tenancy Model

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



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	Read invoices; CRUD on expenses, transactions; view 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
<code>currencyCode</code>	CHAR(3)	ISO 4217 currency of the transaction
<code>exchangeRate</code>	DECIMAL(12,6)	Rate locked at transaction date
<code>amount</code>	DECIMAL(19,4)	Amount in transaction currency
<code>baseAmount</code>	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
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

10.1 MVP Architecture (Current)

Bilko's MVP runs on **Vercel** (frontend) + **Railway EU Frankfurt** (API + PostgreSQL), chosen for developer velocity and cost efficiency at early stage. See [ADR-010](#) for the full rationale and trade-off analysis.

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

  subgraph CDN["Vercel Edge Network"]
    VCL["Vercel\n(Next.js frontend)\nglobal edge CDN"]
  end

  subgraph Railway["Railway – EU Frankfurt"]
    API["Express API\n(Node.js container)"]
  end
```

```

    PG["PostgreSQL 15\n(Railway managed)"]
end

subgraph Storage["Cloudflare R2"]
    R2["R2 Bucket\n(PDFs, receipts)\nZero egress fees"]
end

subgraph External["External APIs"]
    SEND["SendGrid\n(transactional email)"]
    ECB["ECB / Fixer.io\n(exchange rates)"]
    SEF["SEF Serbia\n(e-invoices)"]
    eRacun["eRačun Croatia\n(e-invoices)"]
end

D1 --> VCL
D2 --> API
VCL -->|"REST API calls"| API
API --> PG
API --> R2
API --> SEND
API --> ECB
API --> SEF
API --> eRacun

```

Key MVP infrastructure decisions:

Decision	Choice	Reason
Frontend hosting	Vercel	Zero-config Next.js deploy, preview deployments per PR, global CDN
API + DB hosting	Railway EU Frankfurt	Managed containers + PostgreSQL, €21/mo, GDPR-compliant EU region
File storage	Cloudflare R2	S3-compatible API, zero egress fees, invoices/receipts stored here
DNS + DDoS	Cloudflare	Free DDoS protection, CDN proxying for API origin hiding
Email	SendGrid	Reliable transactional delivery, 40K free emails/month at start
Exchange rates	ECB (free) + Fixer.io (paid fallback)	Daily EUR base rates free from ECB; Fixer for non-EUR pairs

Estimated MVP cost: €21/mo (Railway Starter: €5 API container + €5 PostgreSQL + €11 networking; Vercel: free tier; R2: free up to 10GB)

10.2 CDN & Static Assets

Vercel's edge network serves the Next.js frontend with automatic:

- Static asset caching at edge PoPs globally
- Automatic HTTPS + TLS certificate rotation
- ISR (Incremental Static Regeneration) for report pages
- Preview deployments on every pull request branch

10.3 Redis Cache (Planned — Growth Phase)

Not deployed at MVP. Planned for growth phase when session load requires it:

Use Case	Cache Key Pattern	TTL
Exchange rate lookups	<code>fx:{base}:{target}:{date}</code>	24h
Report aggregations	<code>report:{orgId}:{type}:{period}</code>	1h
User permissions	<code>rbac:{userId}:{orgId}</code>	15min

Railway provides a managed Redis add-on when needed. No code changes required in `apps/api` — add `REDIS_URL` env var and enable the cache middleware.

10.4 Scaling Path (Future — AWS)

When Bilko scales beyond Railway's limits (est. >10K active orgs), the migration path is:

MVP (Railway)	→ Growth (Railway Pro)	→ Scale (AWS eu-central-1)
Express container (€5/mo)	Express + autoscaling	ECS Fargate
Railway PostgreSQL (€5/mo)	Railway PostgreSQL Pro	RDS PostgreSQL Multi-AZ
Vercel Edge CDN	Vercel Pro	Vercel Enterprise / CloudFront
—	Redis cache (Railway add-on)	ElastiCache Redis
—	—	CloudWatch + X-Ray

Terraform configs in `infrastructure/terraform/` are pre-written for the AWS migration to avoid a cold-start when the time comes.

11. Security Model

Layer	Control
Transport	HTTPS enforced (HSTS, <code>maxAge: 31536000</code> , <code>includeSubDomains</code>)
Security headers	helmet (CSP, X-Frame-Options: deny, X-Content-Type-Options: noSniff)
CORS	Whitelist: <code>bilko.io</code> , <code>www.bilko.io</code> , <code>localhost:3000</code>
Rate limiting	100 req/min per IP (general); 5 req/min on <code>/auth/login</code> and <code>/auth/register</code>
Authentication	JWT access token (15min) + refresh token (7d, httpOnly cookie)
Authorization	RBAC checked per endpoint; <code>organizationScope</code> middleware enforces tenancy
Password storage	bcrypt, 12 salt rounds
Audit trail	<code>LoggedAction</code> 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	<code>Transaction.locked = true</code> makes records unmodifiable
SQL injection	Prisma parameterized queries — no raw SQL in business logic
Secret management	Environment variables; never committed to repository

Revision #10

Created 2026-02-23 16:21:48 UTC by John

Updated 2026-07-26 20:02:21 UTC by John