Low-Level Design (LLD)

Bilko — Low-Level Design (LLD)

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

Table of Contents

---

1. API Endpoint Specifications

Base URL: `/api/v1` **Auth:** All endpoints except `/auth/*` and `/health` require `Authorization: Bearer <accessToken>` **Content-Type:** `application/json` **Error format:**

```
{
  "error": "Human-readable message",
  "code": "ERROR_CODE",
  "details": {}
}
```

1.1 Health

`GET /api/v1/health`

No auth required.

Response 200:

```
{ "status": "ok", "timestamp": "2026-02-23T10:00:00.000Z" }
```

1.2 Authentication (`/auth`)

Source: `apps/api/src/routes/auth.ts`

`POST /api/v1/auth/register`

Rate-limited (stricter). Creates organization + owner user in a single Prisma transaction.

Request body:

```
{
  "organizationName": "Acme DOO",
  "country": "RS",
  "baseCurrency": "RSD",
  "language": "sr",
  "registrationNumber": "12345678",
  "vatNumber": "123456789",
  "email": "user@acme.rs",
  "password": "securepassword",
  "fullName": "Marko Marković"
}
```

Response 201:

```
{
  "user": { "id": "uuid", "email": "...", "fullName": "...", "role": "owner" },
  "organization": { "id": "uuid", "name": "...", "country": "RS", "baseCurrency": "RSD" },
  "tokens": { "accessToken": "jwt...", "refreshToken": "jwt..." }
}
```

Errors: `409 DUPLICATE` (email exists), `400 VALIDATION_ERROR`

`POST /api/v1/auth/login`

Rate-limited (stricter). `rememberMe: true` extends refresh token to 30 days.

Request body:

```
{ "email": "user@acme.rs", "password": "securepassword", "rememberMe": false }
```

Response 200: Same shape as register response. Sets `refreshToken` httpOnly cookie (path: `/api/v1/auth`).

Errors: `401 UNAUTHORIZED` (invalid credentials)

`POST /api/v1/auth/refresh`

Uses `refreshToken` cookie. Issues new access token.

Response 200:

```
{ "accessToken": "jwt..." }
```

Errors: `401 NO_TOKEN`, `401 TOKEN_EXPIRED`, `401 INVALID_TOKEN`

`POST /api/v1/auth/logout`

Clears `refreshToken` cookie.

Response 204: No content.

`GET /api/v1/auth/me`

Requires `Authorization: Bearer <accessToken>`.

Response 200:

```
{
  "id": "uuid",
  "email": "...",
  "fullName": "...",
  "role": "owner",
```

```
    "twoFactorEnabled": false,
    "lastLoginAt": "2026-02-23T10:00:00.000Z",
    "organization": {
      "id": "uuid",
      "name": "...",
      "country": "RS",
      "baseCurrency": "RSD",
      "language": "sr"
    }
  }
```

1.3 Invoices (`/invoices`)

Source: `apps/api/src/routes/invoices.ts`, `apps/api/src/services/invoice.service.ts`

`GET /api/v1/invoices`

List invoices with pagination and filtering.

Query params:

Param	Type	Description
`status`	enum	`draft`, `sent`, `viewed`, `paid`, `overdue`, `cancelled`
`customerId`	uuid	Filter by customer
`fromDate`	YYYY-MM-DD	Invoice date from
`toDate`	YYYY-MM-DD	Invoice date to
`page`	int	Default 1
`perPage`	int	Default 20, max 100
`sort`	string	`invoiceDate`, `totalAmount`, `createdAt`
`order`	string	`asc`, `desc`

Response 200:

```
{
  "data": [
    {
      "id": "uuid",
      "invoiceNumber": "INV-2026-001",
```

```
      "customerId": "uuid",
      "customerName": "Acme Client",
      "invoiceDate": "2026-02-01",
      "dueDate": "2026-03-01",
      "currencyCode": "RSD",
      "totalAmount": "120000.0000",
      "status": "draft",
      "createdAt": "2026-02-01T10:00:00.000Z"
    }
  ],
  "meta": { "total": 42, "page": 1, "perPage": 20, "totalPages": 3 }
}
```

`GET /api/v1/invoices/:id`

Get single invoice with all line items.

Response 200:

```
{
  "id": "uuid",
  "invoiceNumber": "INV-2026-001",
  "customerId": "uuid",
  "customerName": "...",
  "invoiceDate": "2026-02-01",
  "dueDate": "2026-03-01",
  "currencyCode": "RSD",
  "exchangeRate": "1.000000",
  "subtotal": "100000.0000",
  "taxAmount": "20000.0000",
  "discountAmount": "0.0000",
  "totalAmount": "120000.0000",
  "baseAmount": "120000.0000",
  "status": "draft",
  "sentAt": null,
  "paidAt": null,
  "items": [
    {
      "id": "uuid",
      "lineNumber": 1,
```

```
      "description": "Consulting services",
      "quantity": "10.00",
      "unitPrice": "10000.0000",
      "taxRate": "20.00",
      "lineTotal": "100000.0000",
      "accountId": "uuid"
    }
  ],
  "notes": null,
  "terms": null,
  "pdfUrl": null,
  "createdBy": "uuid",
  "createdAt": "...",
  "updatedAt": "..."
}
```

Errors: `404 NOT_FOUND`

`POST /api/v1/invoices`

Create invoice in `draft` status. Auto-generates invoice number (`INV-YYYY-NNN`). Locks exchange rate at `invoiceDate`.

Request body:

```
{
  "customerId": "uuid",
  "invoiceDate": "2026-02-01",
  "dueDate": "2026-03-01",
  "currencyCode": "RSD",
  "items": [
    {
      "description": "Consulting",
      "quantity": 10,
      "unitPrice": 10000,
      "taxRate": 20,
      "accountId": "uuid"
    }
  ],
  "notes": "Optional notes",
  "terms": "Net 30"
```

```
}
```

Response 201: Full invoice object (same as GET /:id)

Errors: `404 NOT_FOUND` (customer), `400 VALIDATION_ERROR`

`PUT /api/v1/invoices/:id`

Update invoice. Only allowed when `status = draft`.

Request body (partial):

```
{
  "invoiceDate": "2026-02-15",
  "dueDate": "2026-03-15",
  "items": [ ... ],
  "notes": "Updated notes"
}
```

Errors: `404 NOT_FOUND`, `400 BAD_REQUEST` (not draft)

`PATCH /api/v1/invoices/:id/status`

Change invoice status. Each action triggers double-entry transaction creation.

Request body:

```
{ "action": "send" }
{ "action": "mark-paid", "paidAt": "2026-02-20" }
{ "action": "cancel" }
```

Actions and effects:

Action	From Status	To Status	Journal Entry
`send`	`draft`	`sent`	DR Accounts Receivable / CR Revenue
`mark-paid`	`sent`, `viewed`	`paid`	DR Bank / CR Accounts Receivable
`cancel`	`draft`, `sent`, `viewed`	`cancelled`	None

Errors: `404 NOT_FOUND`, `400 BAD_REQUEST` (invalid transition), `400 BAD_REQUEST` (accounts not found)

`GET /api/v1/invoices/:id/pdf`

Redirects to PDF URL in Cloudflare R2. Returns `404` if PDF not generated yet.

`POST /api/v1/invoices/:id/send`

Send invoice email to customer.

Request body:

```
{ "to": "customer@example.com", "subject": "Invoice ...", "message": "..." }
```

Response 200:

```
{ "sentAt": "...", "sentTo": "customer@example.com", "emailId": "..." }
```

Note: Email sending is a placeholder — not yet implemented.

`DELETE /api/v1/invoices/:id`

Delete invoice. Only allowed when `status = draft`.

Response 204: No content.

Errors: `404 NOT_FOUND`, `400 BAD_REQUEST` (not draft)

1.4 Expenses (`/expenses`)

Source: `apps/api/src/routes/expenses.ts`, `apps/api/src/services/expense.service.ts`

`GET /api/v1/expenses`

List with pagination.

Query params: `status`, `category`, `vendorId`, `fromDate`, `toDate`, `page`, `perPage`, `sort`, `order`

`GET /api/v1/expenses/:id`

Response 200:

```
{
  "id": "uuid",
```

```json
  "expenseNumber": "EXP-2026-001",
  "vendorId": "uuid",
  "vendorName": "Office Supplies Ltd",
  "expenseDate": "2026-02-01",
  "category": "office",
  "currencyCode": "RSD",
  "exchangeRate": "1.000000",
  "amount": "5000.0000",
  "baseAmount": "5000.0000",
  "taxAmount": "850.0000",
  "paymentMethod": "bank_transfer",
  "accountId": "uuid",
  "description": "Office supplies purchase",
  "receiptUrl": null,
  "status": "pending",
  "approvedAt": null,
  "paidAt": null,
  "createdBy": "uuid",
  "createdAt": "...",
  "updatedAt": "..."
}
```

`POST /api/v1/expenses`

Create expense in `pending` status. Auto-generates number (`EXP-YYYY-NNN`).

Request body:

```json
{
  "vendorId": "uuid",
  "expenseDate": "2026-02-01",
  "category": "office",
  "amount": 5000,
  "currencyCode": "RSD",
  "taxAmount": 850,
  "paymentMethod": "bank_transfer",
  "accountId": "uuid",
  "description": "Office supplies"
}
```

`PUT /api/v1/expenses/:id`

Update expense. Only `pending` status.

`PATCH /api/v1/expenses/:id/approve`

Approve expense. Creates double-entry: **DR Expense Account / CR Accounts Payable**

Response 200: Updated expense with `status: approved`

`PATCH /api/v1/expenses/:id/pay`

Mark expense paid. Creates double-entry: **DR Accounts Payable / CR Bank**

Response 200: Updated expense with `status: paid`

`DELETE /api/v1/expenses/:id`

Delete expense. Only `pending` status.

1.5 Contacts (`/contacts`)

Source: `apps/api/src/routes/contacts.ts`, `apps/api/src/services/contact.service.ts`

`GET /api/v1/contacts`

Query params: `type` (`customer`, `vendor`, `both`), `search`, `page`, `perPage`

`GET /api/v1/contacts/:id`

`POST /api/v1/contacts`

Request body:

```
{
  "type": "customer",
  "name": "Acme Client DOO",
  "email": "billing@acme.rs",
  "phone": "+381 11 123 4567",
  "registrationNumber": "12345678",
  "vatNumber": "123456789",
```

```
  "addressLine1": "Bulevar Kralja Aleksandra 1",
  "city": "Beograd",
  "postalCode": "11000",
  "country": "RS",
  "currencyCode": "RSD",
  "paymentTerms": 30,
  "notes": "VIP client"
}
```

`PUT /api/v1/contacts/:id`

`DELETE /api/v1/contacts/:id`

Soft-delete: sets `isActive = false`. Contact remains in database for historical records.

1.6 Accounts (Chart of Accounts) (`/accounts`)

Source: `apps/api/src/routes/accounts.ts`, `apps/api/src/services/account.service.ts`

`GET /api/v1/accounts`

Query params: `typeId`, `isActive`, `includeBalances` (boolean)

Response 200:

```
{
  "data": [
    {
      "id": "uuid",
      "code": "120",
      "name": "Potraživanja od kupaca",
      "accountTypeId": 1,
      "accountType": "Asset",
      "currencyCode": "RSD",
      "parentAccountId": null,
      "isActive": true
    }
  ]
}
```

`POST /api/v1/accounts`

Request body: `{ "code": "1201", "name": "...", "accountTypeId": 1, "currencyCode": "RSD", "parentAccountId": "uuid" }`

`PUT /api/v1/accounts/:id`

1.7 Transactions (General Ledger) (`/transactions`)

Source: `apps/api/src/routes/transactions.ts`

`GET /api/v1/transactions`

Query params: `fromDate`, `toDate`, `accountId`, `referenceType` (`invoice`, `expense`, `payment`, `manual`), `referenceId`, `page`, `perPage`, `sort`, `order`

Response 200:

```
{
  "data": [
    {
      "id": "uuid",
      "transactionDate": "2026-02-01",
      "description": "Invoice INV-2026-001",
      "debitAccountId": "uuid",
      "debitAccountCode": "120",
      "debitAccountName": "Receivables",
      "creditAccountId": "uuid",
      "creditAccountCode": "600",
      "creditAccountName": "Revenue",
      "amount": "120000.0000",
      "currencyCode": "RSD",
      "exchangeRate": "1.000000",
      "baseAmount": "120000.0000",
      "referenceType": "invoice",
      "referenceId": "uuid",
      "locked": false,
      "reconciled": false,
      "createdBy": "uuid",
      "createdAt": "..."
    }
```

```
  ],
  "meta": { "total": 100, "page": 1, "perPage": 20, "totalPages": 5 }
}
```

`GET /api/v1/transactions/:id`

Full transaction detail including account type information.

`POST /api/v1/transactions`

Manual journal entry. Requires `owner`, `admin`, or `accountant` role. Debit and credit accounts must be different.

Request body:

```
{
  "transactionDate": "2026-02-01",
  "description": "Manual adjustment",
  "debitAccountId": "uuid",
  "creditAccountId": "uuid",
  "amount": 5000,
  "currencyCode": "RSD",
  "notes": "Correction entry"
}
```

Errors: `403 FORBIDDEN` (viewer role), `422 VALIDATION_ERROR` (same debit/credit account), `404 NOT_FOUND` (accounts)

1.8 Reports (`/reports`)

Source: `apps/api/src/routes/reports.ts`, `apps/api/src/services/report.service.ts`

`GET /api/v1/reports/dashboard`

MTD metrics: cash balance, revenue, unpaid invoices, expenses, profit, monthly P&L (6 months), receivables aging, expenses by category.

`GET /api/v1/reports/profit-loss`

Query params: `from` (YYYY-MM-DD), `to` (YYYY-MM-DD)

Response 200:

```
{
  "period": { "from": "2026-01-01", "to": "2026-01-31" },
  "baseCurrency": "RSD",
  "revenue": { "total": "500000.0000", "accounts": [{ "accountCode": "600", "accountName": "Revenue", "amount": "500000.0000" }] },
  "expenses": { "total": "200000.0000", "accounts": [...] },
  "netProfit": "300000.0000"
}
```

`GET /api/v1/reports/balance-sheet`

Query params: `date` (YYYY-MM-DD, default: today)

Returns assets (current + fixed), liabilities (current + long-term), equity with account detail.

`GET /api/v1/reports/cash-flow`

Query params: `from`, `to`

Categorizes bank account transactions into operating, investing, and financing cash flows with opening/closing balance.

`GET /api/v1/reports/vat`

Query params: `from`, `to`

Returns output VAT (from invoices), input VAT (from expenses), net VAT, and reconciliation status.

`GET /api/v1/reports/trial-balance`

Query params: `date` (YYYY-MM-DD, default: today)

Returns all accounts with debit total, credit total, balance, and whether total debits equal total credits.

`GET /api/v1/reports/general-ledger`

Query params: `accountId` (optional), `from`, `to`

Returns accounts with individual transaction entries sorted by date, showing running debit/credit/counter-account.

1.9 Banking (`/bank-accounts`)

Source: `apps/api/src/routes/banking.ts`, `apps/api/src/services/banking.service.ts`

`GET /api/v1/bank-accounts`

List all bank accounts with balances.

`GET /api/v1/bank-accounts/:id`

Single bank account with recent transactions.

`POST /api/v1/bank-accounts`

Request body:

```
{
  "bankName": "UniCredit Banka",
  "accountNumber": "170-123456789-01",
  "iban": "RS35170006310000014243",
  "currencyCode": "RSD",
  "accountId": "uuid"
}
```

`GET /api/v1/bank-accounts/:id/transactions`

Query params: `fromDate`, `toDate`, `reconciled` (boolean), `page`, `perPage`

`POST /api/v1/bank-accounts/:id/import`

Import CSV bank statement. Request body: `{ "csvContent": "Date,Amount,..." }`

Response:

```
{ "imported": 45, "duplicates": 3, "errors": 0 }
```

`POST /api/v1/bank-accounts/:id/reconcile`

Request body:

```
{
  "bankTransactionId": "uuid",
  "transactionId": "uuid"
}
```

1.10 Settings

Source: `apps/api/src/routes/settings.ts`, `apps/api/src/services/settings.service.ts`

`GET /api/v1/organization`

`PUT /api/v1/organization`

Requires `owner` or `admin` role.

Request body:

```
{
  "name": "Updated Name DOO",
  "registrationNumber": "12345678",
  "vatNumber": "123456789",
  "language": "sr"
}
```

`GET /api/v1/users`

Requires `owner` or `admin` role. Returns all users in the organization.

`POST /api/v1/users/invite`

Request body:

```
{ "email": "newuser@acme.rs", "fullName": "Jana Jović", "role": "accountant" }
```

`PUT /api/v1/users/:id/role`

Requires `owner` role only.

Request body:

```
{ "role": "admin" }
```

`DELETE /api/v1/users/:id`

Requires `owner` role. Cannot delete self.

`GET /api/v1/currencies`

List all active currencies with code, name, symbol, decimal places.

`GET /api/v1/exchange-rates`

Query params: `baseCurrency`, `targetCurrency`, `date`

`GET /api/v1/settings/tax-rates`

Get org-level tax rate overrides.

`PUT /api/v1/settings/tax-rates`

Requires `owner` or `admin`.

---

2. Database Schema Documentation

Source: `packages/database/prisma/schema.prisma` **Database:** PostgreSQL 15 **ORM:** Prisma

2.1 Entity Relationship Diagram

```
erDiagram
    Organization {
        UUID id PK
        VARCHAR(255) name
        VARCHAR(50) registrationNumber
        VARCHAR(50) vatNumber
        CHAR(3) baseCurrency "default: EUR"
        CHAR(2) country
        CHAR(2) language "default: sr"
        DATE fiscalYearStart
        TIMESTAMP createdAt
        TIMESTAMP updatedAt
    }

    User {
        UUID id PK
        UUID organizationId FK
        VARCHAR(255) email UK
        VARCHAR(255) passwordHash
        VARCHAR(255) fullName
        ENUM role "owner|admin|accountant|viewer"
```

```
    BOOLEAN twoFactorEnabled
    VARCHAR(255) twoFactorSecret
    TIMESTAMP lastLoginAt
    TIMESTAMP createdAt
    TIMESTAMP updatedAt
}

AccountType {
    INT id PK "autoincrement"
    VARCHAR(50) name UK
    ENUM normalBalance "debit|credit"
    TIMESTAMP createdAt
}

Account {
    UUID id PK
    UUID organizationId FK
    VARCHAR(10) code
    VARCHAR(255) name
    INT accountTypeId FK
    CHAR(3) currencyCode
    UUID parentAccountId FK "nullable, self-reference"
    BOOLEAN isActive
    TIMESTAMP createdAt
    TIMESTAMP updatedAt
}

Contact {
    UUID id PK
    UUID organizationId FK
    ENUM type "customer|vendor|both"
    VARCHAR(255) name
    VARCHAR(255) email
    VARCHAR(50) phone
    VARCHAR(50) registrationNumber
    VARCHAR(50) vatNumber
    VARCHAR(255) addressLine1
    VARCHAR(255) addressLine2
    VARCHAR(100) city
    VARCHAR(20) postalCode
```

```
    CHAR(2) country
    CHAR(3) currencyCode
    INT paymentTerms "default: 30 days"
    TEXT notes
    BOOLEAN isActive
    TIMESTAMP createdAt
    TIMESTAMP updatedAt
}

Invoice {
    UUID id PK
    UUID organizationId FK
    UUID customerId FK
    VARCHAR(50) invoiceNumber UK
    DATE invoiceDate
    DATE dueDate
    CHAR(3) currencyCode
    DECIMAL(12_6) exchangeRate
    DECIMAL(19_4) subtotal
    DECIMAL(19_4) taxAmount
    DECIMAL(19_4) discountAmount
    DECIMAL(19_4) totalAmount
    DECIMAL(19_4) baseAmount
    ENUM status "draft|sent|viewed|paid|overdue|cancelled"
    TIMESTAMP sentAt
    TIMESTAMP viewedAt
    TIMESTAMP paidAt
    TEXT notes
    TEXT terms
    VARCHAR(500) pdfUrl
    UUID createdBy FK
    TIMESTAMP createdAt
    TIMESTAMP updatedAt
}

InvoiceItem {
    UUID id PK
    UUID invoiceId FK
    INT lineNumber
    VARCHAR(500) description
```

```
    DECIMAL(10_2) quantity
    DECIMAL(19_4) unitPrice
    DECIMAL(5_2) taxRate
    DECIMAL(19_4) lineTotal
    UUID accountId FK "nullable"
    TIMESTAMP createdAt
}

Expense {
    UUID id PK
    UUID organizationId FK
    UUID vendorId FK "nullable"
    VARCHAR(50) expenseNumber UK
    DATE expenseDate
    CHAR(3) currencyCode
    DECIMAL(12_6) exchangeRate
    DECIMAL(19_4) amount
    DECIMAL(19_4) baseAmount
    DECIMAL(19_4) taxAmount
    VARCHAR(100) category
    VARCHAR(50) paymentMethod
    UUID accountId FK "nullable"
    TEXT description
    VARCHAR(500) receiptUrl
    ENUM status "pending|approved|paid|rejected"
    UUID approvedBy FK "nullable"
    TIMESTAMP approvedAt
    TIMESTAMP paidAt
    UUID createdBy FK
    TIMESTAMP createdAt
    TIMESTAMP updatedAt
}

Transaction {
    UUID id PK
    UUID organizationId FK
    DATE transactionDate
    VARCHAR(255) description
    UUID debitAccountId FK
    UUID creditAccountId FK
```

```
    DECIMAL(19_4) amount
    CHAR(3) currencyCode
    DECIMAL(12_6) exchangeRate
    DECIMAL(19_4) baseAmount
    VARCHAR(50) referenceType "invoice|expense|payment|manual"
    UUID referenceId "nullable"
    BOOLEAN locked "default: false"
    TIMESTAMP lockedAt
    BOOLEAN reconciled "default: false"
    TIMESTAMP reconciledAt
    TEXT notes
    UUID createdBy FK "nullable"
    TIMESTAMP createdAt
}

BankAccount {
    UUID id PK
    UUID organizationId FK
    UUID accountId FK
    VARCHAR(255) bankName
    VARCHAR(50) accountNumber
    VARCHAR(50) iban
    CHAR(3) currencyCode
    DECIMAL(19_4) currentBalance
    BOOLEAN isActive
    TIMESTAMP createdAt
    TIMESTAMP updatedAt
}

BankTransaction {
    UUID id PK
    UUID bankAccountId FK
    DATE transactionDate
    DECIMAL(19_4) amount
    VARCHAR(500) description
    VARCHAR(255) reference
    BOOLEAN reconciled
    UUID matchedTransactionId "nullable"
    TIMESTAMP createdAt
}
```

```
Currency {
    CHAR(3) code PK
    VARCHAR(100) name
    VARCHAR(10) symbol
    SMALLINT decimalPlaces "default: 2"
    BOOLEAN isActive
    TIMESTAMP createdAt
}

ExchangeRate {
    UUID id PK
    CHAR(3) baseCurrency FK
    CHAR(3) targetCurrency FK
    DECIMAL(12_6) rate
    DATE effectiveDate
    VARCHAR(50) source
    TIMESTAMP lastUpdated
}

LoggedAction {
    BIGINT eventId PK "autoincrement"
    TEXT schemaName
    TEXT tableName
    UUID userId FK "nullable"
    TIMESTAMP actionTimestamp
    ENUM action "INSERT|UPDATE|DELETE"
    JSONB rowData "full row snapshot"
    JSONB changedFields "diff for UPDATE"
    TEXT queryText
    INET clientIp
    TEXT applicationName "default: bilko-api"
}

SchemaVersion {
    VARCHAR(20) version PK
    TIMESTAMP appliedAt
    TEXT description
}
```

```
    Organization ||--o{ User : has
    Organization ||--o{ Account : owns
    Organization ||--o{ Contact : owns
    Organization ||--o{ Invoice : owns
    Organization ||--o{ Expense : owns
    Organization ||--o{ Transaction : owns
    Organization ||--o{ BankAccount : owns
    AccountType ||--o{ Account : classifies
    Account ||--o{ Account : "parent-child"
    Contact ||--o{ Invoice : "billed to"
    Contact ||--o{ Expense : "billed from"
    Invoice ||--o{ InvoiceItem : contains
    Account ||--o{ InvoiceItem : "revenue account"
    Account ||--o{ Expense : "expense account"
    Account ||--o{ BankAccount : links
    Account ||--o{ Transaction : "debit side"
    Account ||--o{ Transaction : "credit side"
    BankAccount ||--o{ BankTransaction : holds
    Currency ||--o{ ExchangeRate : "base"
    Currency ||--o{ ExchangeRate : "target"
    User ||--o{ LoggedAction : audits
```

2.2 Key Indexes

Table	Index	Columns	Purpose
`users`	`idx_users_organization`	`organizationId`	User lookup by org
`users`	`idx_users_email`	`email`	Login lookup
`accounts`	`idx_accounts_organization`	`organizationId`	List accounts by org
`accounts`	Unique	`organizationId, code`	Prevent duplicate account codes
`invoices`	`idx_invoices_organization`	`organizationId`	List invoices by org
`invoices`	`idx_invoices_status`	`status`	Filter by status
`invoices`	`idx_invoices_due_date`	`dueDate`	Overdue detection
`invoices`	`idx_invoices_org_status_date`	`organizationId, status, invoiceDate`	Complex report queries
`transactions`	`idx_transactions_org_date`	`organizationId, transactionDate`	Date range queries
`transactions`	`idx_transactions_reference`	`referenceType, referenceId`	Find transactions for an invoice/expense

Table	Index	Columns	Purpose
`exchange_rates`	`idx_exchange_rates_pair`	`baseCurrency, targetCurrency`	Currency pair lookup
`exchange_rates`	Unique	`baseCurrency, targetCurrency, effectiveDate`	One rate per pair per day
`logged_actions`	`idx_logged_actions_timestamp`	`actionTimestamp`	Audit log queries by time

---

3. Service Layer Design

All services follow the same pattern:

- Constructor receives `PrismaClient` (or use singleton `prisma` from `lib/prisma.ts`)
- All methods receive `organizationId` as first parameter
- Return plain objects (not Prisma model instances) for clean API layer separation
- Use Prisma transactions (`prisma.$transaction()`) for multi-step operations
- Throw errors from `utils/errors.ts` for consistent HTTP responses

3.1 `InvoiceService`

File: `apps/api/src/services/invoice.service.ts`

Method	Description
`listInvoices(orgId, params)`	Paginated list with filters
`getInvoice(orgId, id)`	Single invoice with items
`createInvoice(orgId, userId, data)`	Create draft, calculate amounts, lock exchange rate
`updateInvoice(orgId, id, data)`	Update draft only, recalculate if items changed
`changeInvoiceStatus(orgId, id, data)`	Dispatches to `sendInvoice()`, `markInvoicePaid()`, `cancelInvoice()`
`deleteInvoice(orgId, id)`	Delete draft only
`generateInvoiceNumber(orgId)`	`INV-YYYY-NNN` sequential
`getExchangeRate(from, to, date)`	DB lookup, falls back to 1.0 with warning
`sendInvoice(invoice)`	Prisma tx: create DR Receivable/CR Revenue + update status
`markInvoicePaid(invoice, paidAt)`	Prisma tx: create DR Bank/CR Receivable + update status

3.2 `ExpenseService`

File: `apps/api/src/services/expense.service.ts`

Method	Description
`listExpenses(orgId, params)`	Paginated list with filters
`getExpense(orgId, id)`	Single expense
`createExpense(orgId, userId, data)`	Create pending, lock exchange rate
`updateExpense(orgId, id, data)`	Update pending only
`approveExpense(orgId, id, userId)`	Prisma tx: create DR Expense/CR Payable + update status
`payExpense(orgId, id)`	Prisma tx: create DR Payable/CR Bank + update status
`deleteExpense(orgId, id)`	Delete pending only

3.3 `ContactService`

File: `apps/api/src/services/contact.service.ts`

Method	Description
`listContacts(orgId, params)`	Paginated list with type filter
`getContact(orgId, id)`	Single contact
`createContact(orgId, data)`	Create contact
`updateContact(orgId, id, data)`	Update contact
`deleteContact(orgId, id)`	Soft delete (`isActive = false`)

3.4 `AccountService`

File: `apps/api/src/services/account.service.ts`

Method	Description
`listAccounts(orgId, params)`	List with optional balance calculation
`createAccount(orgId, data)`	Create account (checks code uniqueness)
`updateAccount(orgId, id, data)`	Update account metadata

3.5 `ReportService`

File: `apps/api/src/services/report.service.ts`

Method	Description

`getDashboard(orgId)`	Aggregate MTD metrics
`getProfitLoss(orgId, query)`	Revenue vs expense by account, net profit
`getBalanceSheet(orgId, query)`	Assets, liabilities, equity as of date
`getCashFlow(orgId, query)`	Operating/investing/financing cash flows
`getVATReport(orgId, query)`	Output VAT (invoices) vs input VAT (expenses), net
`getTrialBalance(orgId, query)`	All accounts with debit/credit totals, balanced check
`getGeneralLedger(orgId, query)`	Per-account transaction history

3.6 `BankingService`

File: `apps/api/src/services/banking.service.ts`

Method	Description
`listBankAccounts(orgId)`	List all active bank accounts
`getBankAccount(orgId, id)`	Single account with recent transactions
`createBankAccount(orgId, data)`	Create bank account linked to GL account
`listBankTransactions(orgId, bankAccountId, params)`	Paginated bank transactions
`importBankStatement(orgId, bankAccountId, csvContent)`	Parse CSV, detect duplicates, insert
`reconcileTransaction(orgId, bankAccountId, body)`	Match bank transaction to GL transaction

3.7 `SettingsService`

File: `apps/api/src/services/settings.service.ts`

Method	Description
`getOrganization(orgId)`	Organization details
`updateOrganization(orgId, data)`	Update organization metadata
`listUsers(orgId, params)`	List users in org
`inviteUser(orgId, data)`	Create user with temporary password
`changeUserRole(orgId, userId, requesterId, data)`	Change role (cannot demote self)
`deleteUser(orgId, userId, requesterId)`	Remove user (cannot delete self)
`listCurrencies()`	All active currencies
`getExchangeRate(params)`	Get rate for currency pair on date
`getTaxRates(orgId)`	Get org tax rate config

Method	Description
`updateTaxRates(orgId, data)`	Update tax rate config

---

4. Middleware Stack

File: `apps/api/src/app.ts`

Order is critical. Each middleware passes control to `next()` or sends error response.

```
Request
   │
   ▼
1. helmet()             — Sets security headers (CSP, HSTS, X-Frame-Options: deny, noSniff)
   │
   ▼
2. cors()               — Validates Origin header against whitelist [bilko.io,
localhost:3000]
   │                      credentials: true (allows cookies)
   ▼
3. express.json()       — Parses request body as JSON (limit: 10mb)
   │
   ▼
4. express.urlencoded() — Parses URL-encoded bodies (limit: 10mb)
   │
   ▼
5. cookieParser()       — Parses cookie header, makes cookies accessible via req.cookies
   │
   ▼
6. apiLimiter           — Rate limit: 100 req per 15 min per IP (applied to /api/v1/*)
   │   authLimiter      — Stricter rate limit on /auth/login and /auth/register
   ▼
7. routes               — Mounts all route modules at /api/v1
   │
   ├── authGuard()      — Verifies JWT Bearer token, attaches req.user
   │                      Source: apps/api/src/middleware/auth.ts
   │
   ├── organizationScope() — Validates req.user.organizationId (currently no-op, used as
anchor)
```

```
    │                          Source: apps/api/src/middleware/org-scope.ts
    │
    ├── validate(schema)  — Validates req.body or req.query against Zod schema
    │                       Source: apps/api/src/middleware/validate.ts
    │
    └── routeHandler      — Business logic (calls service layer)
    │
    ▼
8. errorHandler()         — Centralized error handler (MUST be last)
                            Source: apps/api/src/middleware/error-handler.ts
                            Translates AppError → HTTP status + JSON
```

Error response format:

```
{
  "error": "Invoice not found",
  "code": "NOT_FOUND",
  "details": {}
}
```

HTTP status codes used:

- `400` — Validation error, bad request
- `401` — Missing token, expired token, invalid credentials
- `403` — Insufficient permissions (role check)
- `404` — Resource not found
- `409` — Duplicate (unique constraint)
- `422` — Unprocessable entity (e.g., same debit/credit account)
- `429` — Rate limit exceeded
- `500` — Unhandled server error

---

5. Double-Entry Bookkeeping Implementation

Core library: `packages/core/src/accounting/index.ts` **Prisma model:** `Transaction` in `packages/database/prisma/schema.prisma`

5.1 Fundamental Rule

Every financial event creates exactly one `Transaction` record with:

- `debitAccountId` — account to debit
- `creditAccountId` — account to credit
- `amount` — must be equal for both sides (enforced by model design, not DB constraint)
- `currencyCode` + `exchangeRate` + `baseAmount` — for multi-currency

5.2 `validateDoubleEntry()` (core engine)

```
// packages/core/src/accounting/index.ts
export function validateDoubleEntry(lines: JournalEntryLine[]): boolean {
  // Returns false if: < 2 lines, negative amounts, unbalanced
  let totalDebits = new Decimal(0)
  let totalCredits = new Decimal(0)
  for (const line of lines) {
    const amount = new Decimal(line.amount)
    if (amount.lte(0)) return false
    if (line.side === 'debit') totalDebits = totalDebits.plus(amount)
    else totalCredits = totalCredits.plus(amount)
  }
  return totalDebits.eq(totalCredits)
}
```

5.3 Transaction Creation Patterns

Invoice sent (DR Receivable / CR Revenue):

```
DR  Accounts Receivable (code: 12x)   +120,000 RSD
    CR  Revenue (code: 6xx)                       +120,000 RSD
```

Payment received (DR Bank / CR Receivable):

```
DR  Bank Account (code: 10x)          +120,000 RSD
    CR  Accounts Receivable (code: 12x)            +120,000 RSD
```

Expense approved (DR Expense / CR Payable):

```
DR  Expense Account (code: 5xx)       +5,000 RSD
    CR  Accounts Payable (code: 22x)               +5,000 RSD
```

Expense paid (DR Payable / CR Bank):

```
DR  Accounts Payable (code: 22x)      +5,000 RSD
    CR  Bank Account (code: 10x)                    +5,000 RSD
```

5.4 Account Lookup Strategy

Services find accounts by account type ID + code prefix:

- `accountTypeId: 1` = Asset
- `accountTypeId: 2` = Liability
- `accountTypeId: 3` = Equity
- `accountTypeId: 4` = Revenue
- `accountTypeId: 5` = Expense

Code prefixes (Balkan chart of accounts):

- `10x` = Bank/Cash accounts
- `12x` = Accounts Receivable
- `22x` = Accounts Payable
- `5xx` = Expense accounts
- `6xx` = Revenue accounts

5.5 Trial Balance

```
// packages/core/src/accounting/index.ts
export function calculateTrialBalance(transactions: JournalEntry[]): TrialBalance {
  // Groups by accountNumber, sums debits and credits
  // Returns: { rows[], totalDebits, totalCredits, isBalanced }
  // isBalanced = totalDebits.eq(totalCredits)
}
```

---

6. Tax Calculation Logic Per Country

Core module: `packages/core/src/tax/index.ts` **Country modules:** `packages/country-{rs|ba|hr}/src/tax/index.ts`

6.1 Serbia (RS)

File: `packages/country-rs/src/tax/index.ts`

Rate	Value	Applies To

Standard	20%	Most taxable supplies
Reduced	10%	Basic food, medicine, newspapers, public transport, utilities
Zero/Exempt	0%	Exports, international transport, financial services

```
export const serbianVATRates = {
  standard: '20',
  reduced: '10',
  zero: '0',
  exempt: '0',
}

// VAT registration: mandatory above 8M RSD annual revenue
export const SERBIAN_VAT_THRESHOLD = '8000000'
// Pausal (simplified) regime: below 6M RSD
export const SERBIAN_PAUSAL_THRESHOLD = '6000000'
// Corporate income tax
export const SERBIAN_CIT_RATE = '15' // flat 15%
```

Key function:

```
export function calculateSerbianPDV(amount: MonetaryAmount, rate = 'standard'): string {
  // Returns: net.times(rateDecimal).dividedBy(100).toFixed(2)
}
```

6.2 Bosnia & Herzegovina (BA)

File: `packages/country-ba/src/tax/index.ts`

Rate	Value	Applies To
Standard	17%	All taxable supplies (single rate, no reduced)
Zero	0%	Exports

```
export const bosnianVATRates = { standard: '17', zero: '0' }
// Registration threshold: 100,000 BAM
export const BIH_VAT_THRESHOLD = '100000';
// CIT: 10% for both FBiH and RS entities
export const BIH_CIT_RATES = { fbih: '10', rs: '10' }
```

```
// WHT: FBiH dividends 5%, RS dividends 10%
export const BIH_WHT_RATES = { fbih: { dividends: '5', interest: '10' }, rs: { dividends:
'10', ... } }
```

6.3 Croatia (HR)

File: `packages/country-hr/src/tax/index.ts`

Rate	Value	Applies To
Standard	25%	Most taxable supplies
Reduced	13%	Food products, accommodation, utilities
Super-reduced	5%	Books, medicines, newspapers
Zero	0%	Intra-EU transport, international transport

```
export const croatianVATRates = { standard: '25', reduced: '13', superReduced: '5', zero: '0'
}
// Registration threshold: 60,000 EUR (aligned with EU 2025)
export const CROATIAN_VAT_THRESHOLD = '60000'
// CIT: progressive — 10% if revenue < 1M EUR, 18% if >= 1M EUR
export const CROATIAN_CIT_RATES = { small: '10', standard: '18', threshold: '1000000' }
```

6.4 Generic VAT Calculation (Core Engine)

```
// packages/core/src/tax/index.ts
export function calculateVAT(amount: MonetaryAmount, rate: MonetaryAmount): VATResult {
  const base = new Decimal(amount);       // net amount
  const vatRate = new Decimal(rate);
  const tax = base.times(vatRate).dividedBy(100);
  const total = base.plus(tax);
  return {
    base: new Decimal(base.toFixed(4)),
    tax: new Decimal(tax.toFixed(4)),
    total: new Decimal(total.toFixed(4)),
  };
}

export function calculateNetFromGross(grossAmount: MonetaryAmount, vatRate: MonetaryAmount):
```

```
VATResult {
  // Reverse VAT: gross / (1 + rate/100)
  const divisor = new Decimal(100).plus(new Decimal(vatRate)).dividedBy(100);
  const base = new Decimal(grossAmount).dividedBy(divisor);
  ...
}
```

---

7. Invoice Lifecycle

7.1 Status Machine

```
draft ──[send]──► sent ──[mark-paid]──► paid
  │                 │
  │            [cancel]
  │                 │
  └────[cancel]──► cancelled
                sent ──[overdue cron]──► overdue ──[mark-paid]──► paid
                viewed ──[mark-paid]──► paid
```

7.2 Numbering

Invoice numbers are generated sequentially per organization per year: `INV-YYYY-NNN` (e.g., `INV-2026-001`). The service queries the last invoice number with the current year prefix and increments.

```
private async generateInvoiceNumber(organizationId: string): Promise<string> {
  const year = new Date().getFullYear();
  const prefix = `INV-${year}-`;
  const lastInvoice = await prisma.invoice.findFirst({
    where: { organizationId, invoiceNumber: { startsWith: prefix } },
    orderBy: { invoiceNumber: 'desc' },
  });
  const nextNumber = lastInvoice ? parseInt(lastInvoice.invoiceNumber.split('-')[2]) + 1 : 1;
  return `${prefix}${String(nextNumber).padStart(3, '0')}`;
}
```

7.3 Amount Calculation

On create/update:

1. For each line item: `lineTotal = quantity × unitPrice`
2. `taxAmount` per line: `lineTotal × taxRate / 100`
3. `subtotal = Σ lineTotals`
4. `taxAmount = Σ lineTax amounts`
5. `totalAmount = subtotal + taxAmount`
6. `baseAmount = totalAmount × exchangeRate`

All using `Decimal.js` — never JavaScript `number`.

---

8. Bank Import Flow

Source: `packages/core/src/bank-import/index.ts`

8.1 CSV Format

```
Date,Amount,Currency,Direction,Counterparty,Reference,Description
2026-02-01,5000.00,RSD,inbound,Acme Client,INV-2026-001,Invoice payment
```

Supported date formats: `YYYY-MM-DD`, `DD.MM.YYYY`, `DD/MM/YYYY`

8.2 Import Process

```
POST /api/v1/bank-accounts/:id/import
  │
  ├── parseCSV(csvContent) → BankTransaction[]
  │     - Split by newline, skip header
  │     - Parse each field: date, amount, currency, direction, reference
  │     - Generate deterministic ID for dedup: hash(date|amount|currency|reference|lineIndex)
  │
  ├── detectDuplicates(existingTxs, importedTxs)
  │     - Fingerprint: YYYY-MM-DD|amount|currency|reference
  │     - Returns list of duplicate transactions
  │
  ├── Filter out duplicates
  │
  └── Insert new BankTransactions into database
        Returns: { imported: N, duplicates: M, errors: K }
```

8.3 Reconciliation

Manual reconciliation links a `BankTransaction` to a `Transaction` (GL entry):

```
POST /api/v1/bank-accounts/:id/reconcile
  body: { bankTransactionId, transactionId }
  │
  ├── Verify both belong to organization
  ├── Set BankTransaction.reconciled = true
  ├── Set BankTransaction.matchedTransactionId = transactionId
  └── Set Transaction.reconciled = true
```

---

9. Core Engine Modules

Package: `@bilko/core` (`packages/core/src/`)

Module	File	Purpose
`accounting`	`src/accounting/index.ts`	`validateDoubleEntry`, `createJournalEntry`, `calculateTrialBalance`
`tax`	`src/tax/index.ts`	`calculateVAT`, `calculateNetFromGross`, `getVATRates`, `calculateCIT`
`multi-currency`	`src/multi-currency/index.ts`	`convertCurrency`, `lockExchangeRate`, `calculateForexGainLoss`
`bank-import`	`src/bank-import/index.ts`	`parseCSV`, `detectDuplicates`
`invoicing`	`src/invoicing/index.ts`	Invoice computation helpers
`chart-of-accounts`	`src/chart-of-accounts/index.ts`	Chart structure definitions
`reporting`	`src/reporting/index.ts`	Report calculation utilities

Key constraint: `MonetaryAmount = string | Decimal` — JavaScript `number` is never used for monetary values anywhere in the core engine.

---

10. Cross-Reference Notes (Schema Verification 2026-02-23)

This document was cross-checked against the actual codebase on 2026-02-23. Notes:

10.1 Implementation Status

Layer	Status	Notes
`packages/database/prisma/schema.prisma`	**Implemented** — verified	15 models, all types and indexes match this LLD
`apps/api/`	**Not yet implemented**	Sections 1–3 are target design specs, not live code
`apps/web/`	**Implemented** with mock data	All pages exist; data sourced from `lib/mock-data.ts` until API is ready
`packages/core/`	**Partially implemented**	Module structure matches Section 9; some stubs may be incomplete

10.2 Schema Discrepancies Found

Location	LLD Says	Actual Schema	Action Required
`LoggedAction.applicationName`	Default: `"bilko-api"`	Default: `"fiken-clone-api"` (legacy project name)	**Fix before first deploy** — update default in schema migration
`Invoice.invoiceNumber`	Described as globally `UNIQUE`	`@@unique([organizationId, invoiceNumber])` composite	No fix needed — composite is correct; LLD description imprecise
`Expense.expenseNumber`	Described as globally `UNIQUE`	`@@unique([organizationId, expenseNumber])` composite	No fix needed — composite is correct
`BankTransaction`	No `currencyCode` field noted	No `currencyCode` in schema	Verify if currency tracking needed for bank transactions; may need migration

10.3 Cross-References

Topic	Document
Hosting decision (Vercel + Railway, not AWS)	`docs/architecture/ADR.md` — ADR-010
Database schema detailed spec	`docs/architecture/DATABASE-SCHEMA-DOCUMENT.md`
API endpoint specification (OpenAPI)	`docs/architecture/API-SPECIFICATION.md`
`@bilko/core` module design	`docs/architecture/MODULE-DESIGN.md`
Data flow & retention policy	`docs/architecture/DATA-FLOW.md`
External integrations (SEF, eRačun, SendGrid)	`docs/architecture/INTEGRATION-DESIGN.md`

Revision #9
Created 2026-02-23 16:21:50 UTC by John
Updated 2026-07-26 20:02:22 UTC by John