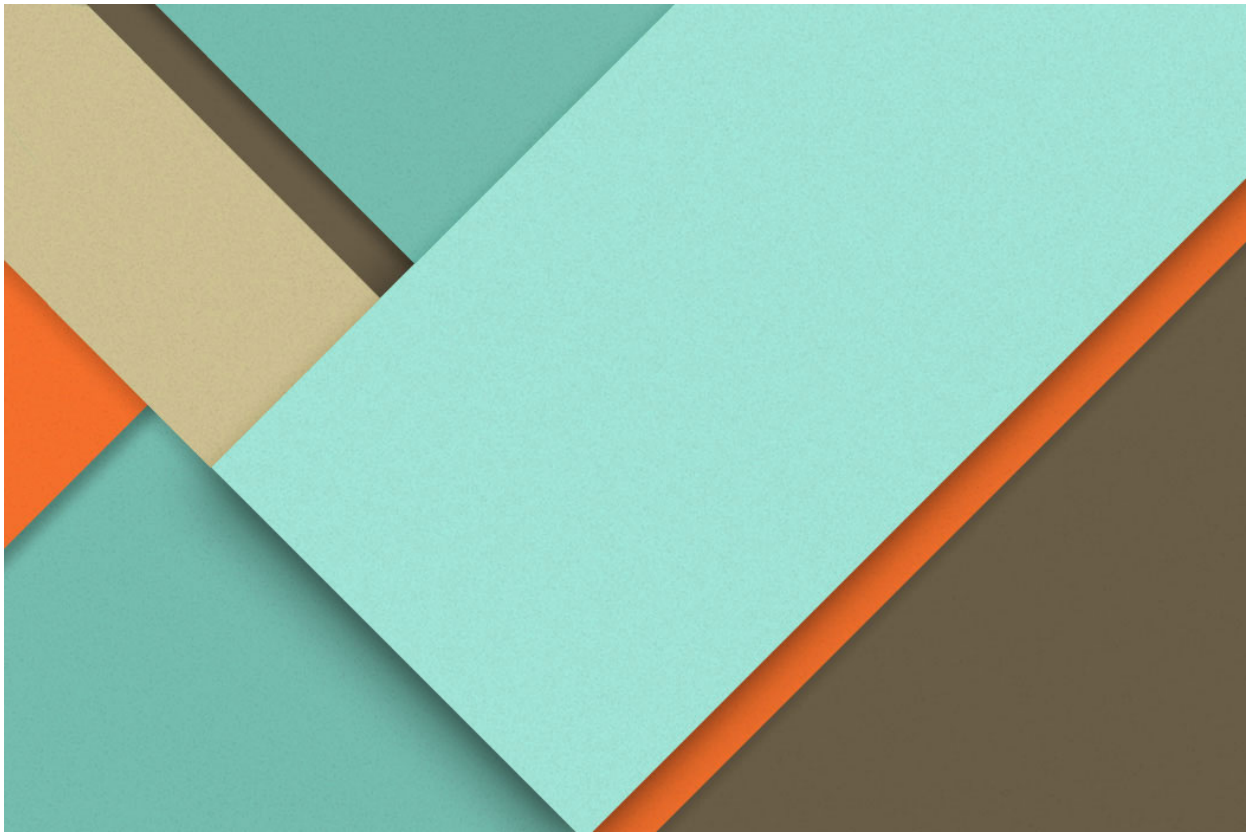


Custom ERP System for Multi Integrated Service Firm.

08.25.2025

Overview

To build an integrated operational backbone that unifies the different functions of a modern service-based firm (Consultancy services, Contractors, Marketers and HR.) into a simple ERP system. The primary objectives are toautomate complex financial calculations (commissions/fees), provide real-time business insights, and scale efficiently with the company's growth.

Table of Contents

1.0 Executive Summary

1.1 Project Vision & Objectives:

To build an integrated operational system that unifies the different functions of a modern service based firm that has the following functionalities: **consultancy services, contractors, marketers and HR.**

1.2 Core Value Proposition:

1. **Automation**: Reduce manual effort in payroll, commission calculation and document generation.

2. **Accuracy:** Ensure 100% accuracy in financial disbursements and reporting.
3. **Show Analytics:** Show leadership a real time view of company performance, project profitability and sales pipeline health.
4. **Control:** Enforce role-based security and maintain an audit trail for all financial and client interactions.

2.0 System Overview & Architecture

2.1 High-Level Functional Modules

The system is decomposed into seven core functional modules, each representing a context within the business domain. I advise that we go with a monolith structure for now so that we can maintain the simplicity of development, testing and a single deployment unit initially in the early phase of the project.

So we will have:

1. **Authentication & Authorization Module:**

   This will be the gatekeeper of the system. Manage the Identity of the user, authenticate(login/logout) and also do Authorization that is Role Based Access depending on the modules a specific user can access.

2. **Personnel Management Module (HR):**

   This module will centralize and manage all data related to people within the organization. The system should be capable of distinguishing between different categories of personnel, including internal employees, contractors and marketers.

   Key Entities:

   - User
   - Employee
   - Contractor
   - Marketer
   - Role
   - Department
   - Attendance
   - LeaveRequest

Key Considerations:

- Integration with the payroll system to ensure accurate compensation management.
- Role-based access control for data security and compliance.
- Scalability to handle organizational growth and diverse workforce types.

3. **Project & Service Delivery Module:**

This module will serve as the core of the system, managing the entire lifecycle of client projects from initiation to completion. It will ensure effective coordination, tracking, and delivery of services.

Key Entities:

- **Client**
- **Project**
- **ProjectPhase**
- **Task**
- **Timelog**
- **Deliverable**

Key Considerations:

- Assigning contractors and employees to specific tasks.
- Tracking time spent on tasks and overall project phases.
- Monitoring project budgets against actual expenses.
- Ensuring adherence to project timelines and milestones.
- Providing visibility into project status for clients and managers.

4. Commission & Incentive Module

This module will serve as a key differentiator in the system. It is a rules-based engine designed to automatically calculate commissions, bonuses and fees for marketers and contractors based on predefined business rules.

Key Entities:

- **CommissionRule**
- **Payout**

- **Transaction** *(audit log of payments)*

Key Considerations:

- Flexible rule definitions (percentage-based, flat rates).
- High performance to handle large-scale commission calculations in batch runs.
- Accuracy and consistency in financial computations.
- Immutability of calculated transactions to support compliance and auditing.

5. Customer Relationship Management (CRM) Module:

This module manages the organization's interactions with current and potential clients, with a strong focus on supporting the sales and marketing pipeline. It enables tracking of leads, nurturing client relationships and ensuring seamless transition from sales to project execution.

Key Entities:

- **Lead**
- **Contact**
  - **Company**
  - **Opportunity**
  - **SalesPipelineStage**
  - **CommunicationLog**

Key Considerations:

- Lead scoring and qualification to prioritize opportunities.
- Conversion tracking from lead to closed deal.
- Pipeline analytics and reporting for informed decision-making.

6. Finance & Accounting Module

This module manages all financial transactions across the system. It ensures proper handling of revenues, expenses and payments, while maintaining compliance and auditability.

Key Entities:

- Invoice
- Expense
- ChartOfAccounts
- JournalEntry
- Payment
- TaxRule

Key Considerations:

- Ensuring financial data integrity and ACID compliance.
- Maintaining detailed audit trails for regulatory and internal review.
- Supporting reconciliation of accounts and financial statements.
  - Integration with the Commission & Incentive Engine for payouts.
  - Integration with the Project & Service Delivery Module for invoicing and expense tracking.

7. **Document Management and Shared Services.**

These services provide foundational capabilities used across all modules, ensuring consistency, reusability, and scalability in the system.

Key Services:

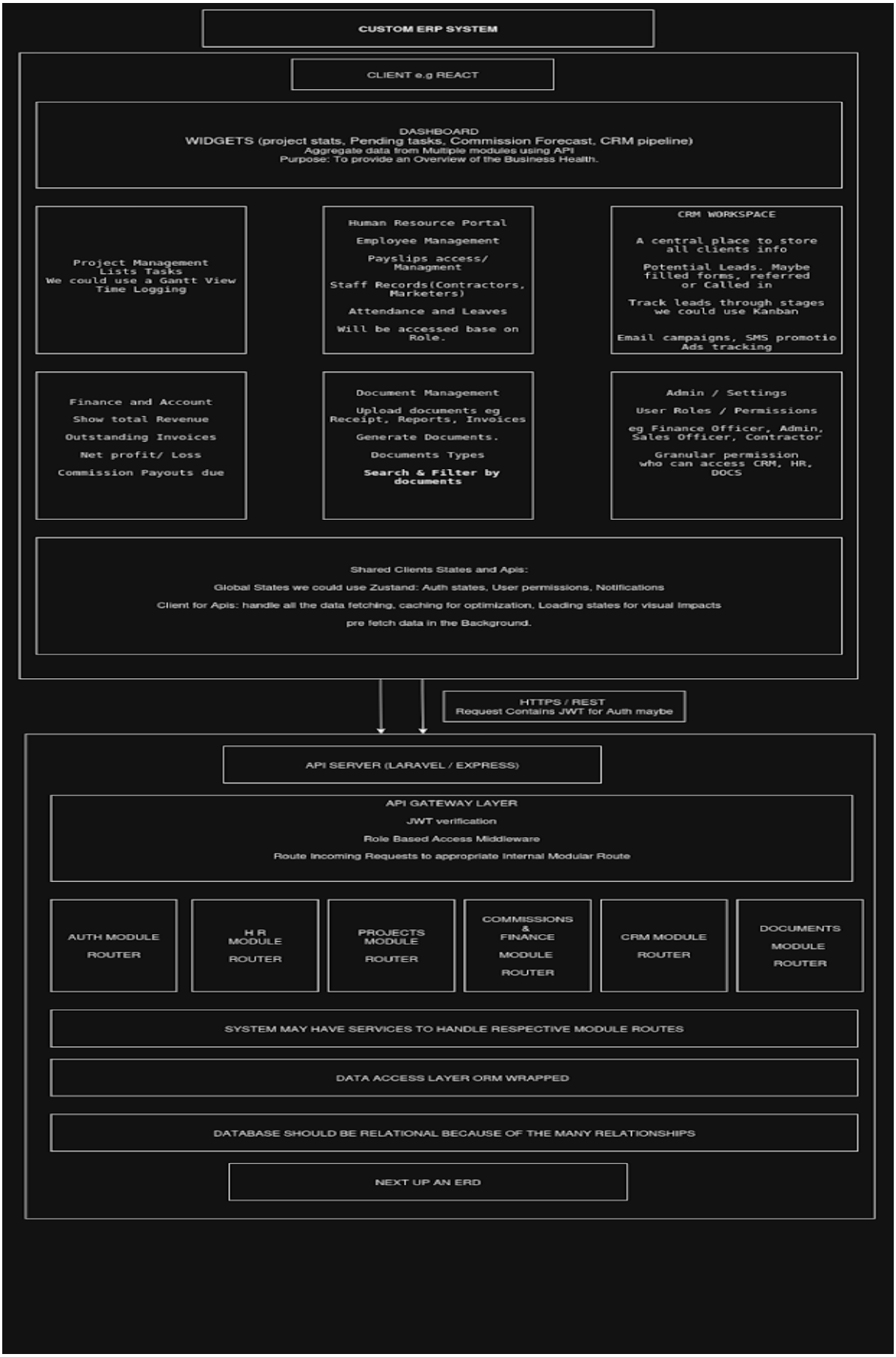
- Document Service: Handles file uploads, secure storage, versioning, and automated document generation (e.g., PDF invoices, contracts).
- Notification Service: Manages internal alerts and external communications (email, SMS, push notifications) via a centralized messaging queue.
- Reporting Service: Delivers aggregated data, dashboards, and analytics across all modules through a standardized API.

Key Considerations:

- Security and access control for shared resources.
- Scalability to handle large volumes of documents, notifications, and reporting requests.
- Consistency in API standards to simplify integration across modules.
- Auditability and traceability of shared services usage.

2.2 Core Architectural Diagram:

For this project, a monolith architecture is proposed. All modules are developed within a single codebase and deployed as a single unit, but are strictly separated by clear internal boundaries. This is the optimal starting point as it reduces operational complexity while enforcing a clean structure that can be later split into microservices if necessary.


CUSTOM ERP SYSTEM
CLIENT e.g REACT
DASHBOARD
WIDGETS (project stats, Pending tasks, Commission Forecast, CRM pipeline)
Aggregate data from Multiple modules using API
Purpose: To provide an Overview of the Business Health.
Project Management
Lists Tasks
We could use a Gantt View
Time Logging
Human Resource Portal
Employee Management
Payslips access/ Managment
Staff Records(Contractors, Marketers)
Attendance and Leaves
Will be accessed base on Role.
CRM WORKSPACE
A central place to store all clients info
Potential Leads. Maybe filled forms, referred or Called in
Track leads through stages we could use Kanban
Email campaigns, SMS promotio
Ads tracking
Finance and Account
Show total Revenue
Outstanding Invoices
Net profit/ Loss
Commission Payouts due
Document Management
Upload documents eg Receipt, Reports, Invoices
Generate Documents.
Documents Types
Search & Filter by documents
Admin / Settings
User Roles / Permissions
eg Finance Officer, Admin, Sales Officer, Contractor
Granular permission who can access CRM, HR, DOCS
Shared Clients States and Apis:
Global States we could use Zustand: Auth states, User permissions, Notifications
Client for Apis: handle all the data fetching, caching for optimization, Loading states for visual Impacts
pre fetch data in the Background.
HTTPS / REST
Request Contains JWT for Auth maybe
API SERVER (LARAVEL / EXPRESS)
API GATEWAY LAYER
JWT verification
Role Based Access Middleware
Route Incoming Requests to appropriate Internal Modular Route
AUTH MODULE ROUTER
H R MODULE ROUTER
PROJECTS MODULE ROUTER
COMMISSIONS & FINANCE MODULE ROUTER
CRM MODULE ROUTER
DOCUMENTS MODULE ROUTER
SYSTEM MAY HAVE SERVICES TO HANDLE RESPECTIVE MODULE ROUTES
DATA ACCESS LAYER ORM WRAPPED
DATABASE SHOULD BE RELATIONAL BECAUSE OF THE MANY RELATIONSHIPS
NEXT UP AN ERD

○

2.3 Technology Stack Proposal

1. Frontend : React

Why React:

Component-based architecture → ERP dashboards and modules (Inventory, CRM, Finance, HR, etc.) can be built as reusable, modular components.

Rich ecosystem → Availability of libraries for charts, tables, maps, forms, and state management (Zustand, Redux, etc.).

Performance with Virtual DOM → Handles frequent UI updates efficiently (important for ERP dashboards with live data).

Strong community & long-term support → Widely adopted by enterprises in Kenya (banks, fintechs, SaaS startups) → easy to hire developers.

2. Backend Technology

When designing the ERP system, two backend frameworks were considered: **Laravel (PHP)** and **Express (Node.js)**.

Option 1: Laravel

Pros:

Mature ecosystem with built-in features like authentication, queues, mail, and caching.

Strong ORM (Eloquent) for database interactions.

Robust MVC structure, good for large-scale enterprise apps.

Excellent community and long-term support.

Cons:

Heavier compared to lightweight Node.js frameworks.
Requires PHP environment setup (different runtime from frontend stack).

Option 2: Express (Node.js)

Pros:
Lightweight and flexible — minimal boilerplate.
Same language (JavaScript/TypeScript) as the frontend → smoother development workflow.
Excellent for building RESTful APIs quickly.
Large ecosystem of NPM packages.

Cons:
Lacks built-in features → must integrate third-party libraries.
Requires careful structuring for large-scale apps (not opinionated).

2.4 Data Flow & Integration

The ERP system is designed around the principle of single source of truth (SSOT), this is needed to ensure that all modules will operate consistently. Each module eg CRM, Finance or HR will connect through APIs and shared db models to eliminate duplication.

Key Principles:

1. Single Source of Truth: All modules will lead from a centralized database.
2. Data entered once eg CRM module is instantly available to all other modules
3. Event-driven integrations: For example when you add an Expense it should automatically reflect in other modules.
4. API-First design: Every module must expose a REST APIs
5. Security and Access Controll. Role-based permissions ensure that users only access the data relevant to their role.
6. Data Consistency & Validation: Input is validated at both frontend and backend.

Example Data Flows

HR Flow:
New Employee record in HR → Payroll module picks salary info → Finance tracks salary as expense → Document Management stores signed contract.

Inventory Flow:
Product stock updated in Inventory → CRM pulls availability info → Finance auto-adjusts cost of goods sold → Admin notified if stock drops below threshold.

3.0 Detailed Module Specifications

3.1 Human Resource Management (HRM) / Personnel Management

Purpose & Scope

The HRM module centralizes all people related records and processes for employees, contractors and marketers

- Identity & profile management
- Employment & contract lifecycle (onboarding → active → offboarding)
- Org structures (departments, positions)
- Leave management (policies, requests, approvals, balances)
- Performance & compliance documentation
- Integration points: Payroll/Finance, Projects/Tasks, Document Service, Notifications, Audit

Actors & Permissions (RBAC)

- **HR Admin**: Full HRM read/write; approve/reject leave; manage org data; view all profiles.
- **Department Manager**: View team profiles; approve leave for their reports; view team timesheets; comment on performance.
- **Employee**: View/edit own profile (limited); submit leave; submit timesheets; view balances and documents linked to self.
- **Contractor/Marketer**: Similar to Employee but limited to assignment & timesheets; no access to internal HR data.
- **Auditor**: Read-only access to audit logs and compliance docs.

Assumptions & Constraints

- Each person has a unique **User ID** (from Auth module), mapped to a **Person/Employee ID** in HRM.

- Tenancy: single-tenant per company (multi-tenant support via `tenant_id` on all tables if required later).
- Kenyan context supported: **NHIF**, **NSSF**, **KRA PIN** fields in identity; public holidays loaded by country.

Core Workflows

A) Onboarding

1. HR creates **Person** → minimal fields (name, email, phone, KRA PIN if available).
2. HR assigns **EmploymentContract** (type: employee/contractor/marketer;start date; probation; compensation terms).
3. HR associates **Department** & **Position**; sets **Manager** (report-to).
4. System provisions **User** (Auth) + baseline **RBAC role(s)**.
5. Required **Documents** checklist generated (ID copy, contract, NDA, compliance forms). Notifications sent.
6. Status transitions: `invited` → `pending-docs` → `active` (with audit trail).

B) Offboarding

1. HR initiates offboarding (reason, last working date).
2. System triggers tasks: asset return, final timesheet cutoff, revoke access, document archive.
3. Notify Finance for **final payout** (salary, reimbursements, accrued leave per policy).
4. Status transitions: `active` → `offboarding` → `inactive`. Records retained per retention policy.

C) Time & Attendance / Timesheets

- **Schedules** (standard/shift) define expected hours.
- Users submit **TimesheetEntries** (project/task-linked for contractors/marketers if applicable).
- Manager reviews/approves; approvals lock entries (immutable except via adjustment path).

- Approved hours feed **Project billing** and **Finance payroll** (if applicable).

D) Leave Management

- **LeaveTypes** (Annual, Sick, Unpaid, Maternity, etc.) with accrual rules (monthly/annually), carry-over, max caps, proration.
- Employee submits **LeaveRequest** (dates, type, reason, attachments) → routed to manager (and HR if policy requires).
- Balance checks at submit-time; conflicts (overlaps, blackout periods) flagged pre-submit.
- On approval: blocks schedule; triggers **calendar events**; notifies stakeholders.

E) Performance & Documentation

- Periodic cycles (quarterly/annual) → **PerformanceReview** with goals, ratings, manager comments.
- **Document links**: contracts, performance letters, warnings, certifications stored via Document Service with versioning.

Data Model (Relational, simplified)

person

- id: uuid PK
- user_id: uuid FK -> auth.users (nullable pre-provision)
- first_name, last_name, other_names
- email (unique), phone
- national_id, kra_pin, nhif_no, nssf_no
- date_of_birth, address, city, country
- employment_status ENUM('applicant','invited','pending_docs','active','offboarding','inactive')
- person_type ENUM('employee','contractor','marketer')
- manager_person_id: uuid FK -> person.id (nullable)
- created_at, updated_at, tenant_id

department

- id: uuid PK
- name (unique within tenant)
- code, description
- parent_department_id: uuid FK -> department.id (nullable)
- created_at, updated_at, tenant_id

position

- id: uuid PK
- title
- department_id: uuid FK
- grade, is_manager BOOLEAN
- created_at, updated_at, tenant_id

employment_contract

- id: uuid PK
- person_id: uuid FK
- contract_type ENUM('permanent','fixed_term','contractor','marketer')
- start_date, end_date (nullable)
- probation_end (nullable)
- base_salary_amount NUMERIC(18,2) NULL (employees)
- rate_amount NUMERIC(18,4) NULL (hourly/day/retainer)
- rate_unit ENUM('hour','day','month','retainer')
- currency_code CHAR(3)
- allowances JSONB (transport, airtime, etc.)
- deductions JSONB
- status ENUM('draft','active','suspended','terminated')
- created_at, updated_at, tenant_id

work_schedule

- id: uuid PK
- person_id: uuid FK

- type ENUM('standard','shift')
- tz VARCHAR
- weekly_template JSONB (Mon–Sun expected hours)
- effective_from, effective_to
- created_at, updated_at, tenant_id

timesheet_entry

- id: uuid PK
- person_id: uuid FK
- date, hours NUMERIC(5,2)
- project_id: uuid FK -> project.project
- task_id: uuid FK -> project.task
- notes TEXT
- status ENUM('draft','submitted','approved','rejected','locked')
- approved_by: uuid FK -> person.id (nullable), approved_at
- created_at, updated_at, tenant_id

leave_type

- id: uuid PK, name, code
- requires_attachment BOOLEAN
- accrual_rule JSONB (rate, period, caps)
- carry_over_rule JSONB
- created_at, updated_at, tenant_id

leave_balance

- id: uuid PK, person_id: uuid FK, leave_type_id: uuid FK
- period_start, period_end
- opening NUMERIC(6,2), accrued NUMERIC(6,2), taken NUMERIC(6,2), closing NUMERIC(6,2)
- created_at, updated_at, tenant_id

leave_request

- id: uuid PK
- person_id: uuid FK, leave_type_id: uuid FK
- start_date, end_date, days NUMERIC(5,2)
- reason TEXT, attachment_doc_id: uuid (nullable)
- status ENUM('draft','submitted','manager_approved','hr_approved','rejected','cancelled')
- approver_chain JSONB
- created_at, updated_at, tenant_id

performance_review

- id: uuid PK, person_id: uuid FK
- period_start, period_end
- goals JSONB, ratings JSONB, manager_comments TEXT, overall_rating NUMERIC(3,2)
- status ENUM('draft','in_review','final')
- created_at, updated_at, tenant_id

person_document

- id: uuid PK, person_id: uuid FK, doc_service_id: uuid (from Document Service)
- type ENUM('contract','id','kra','nhif','nssf','warning','certificate','other')
- version, status ENUM('draft','final','archived')
- created_at, updated_at, tenant_id

audit_log *(shared service but HRM-sourced events tagged)*

- id: uuid PK, actor_user_id, entity, entity_id, action, before JSONB, after JSONB, at TIMESTAMP, tenant_id

Business Rules

- **Uniqueness**: email, national_id and kra_pin must be unique within tenant when provided.

- **Manager scope**: A manager can only approve leave/timesheets for direct/indirect reports (hierarchical traversal).
- **Leave accrual**: Run monthly batch to add accrual; proration by start date; cap closing <= max_cap.
- **Overlapping leave**: Disallow overlapping approved requests (same type), allow different types only with HR override.
- **Timesheet locking**: Approved entries become locked when payroll cycle closes; changes only via adjustment request.
- **Contract changes**: Salary/rate changes create **new contract version** effective from date; old version auto-expires.

API Contracts (REST, JSON)

Base: /api/hrm or /api/personnel

People

- POST /people → create person
- GET /people?search=&dept=&type=&status=&page=&size= → list/filter
- GET /people/{id} → details (scoped by RBAC)
- PATCH /people/{id} → update allowed fields

Contracts

- POST /people/{id}/contracts → create
- PATCH /contracts/{id} → update status/fields
- GET /contracts/{id}

Timesheets

- POST /timesheets (bulk create entries)
- GET /timesheets?person=&date_from=&date_to=&status=
- POST /timesheets/{id}/submit
- POST /timesheets/{id}/approve
- POST /timesheets/{id}/reject

Leave

- POST /leave/requests
- GET /leave/requests?person=&status=&type=&from=&to=
- POST /leave/requests/{id}/submit
- POST /leave/requests/{id}/approve (auto-detect approver level)
- POST /leave/requests/{id}/reject
- GET /leave/balances?person=&type=&period=

Performance

- POST /performance/reviews
- GET /performance/reviews?person=&period=
- PATCH /performance/reviews/{id} (status transitions)

Events to other modules (webhooks/queue)

- hr.person.activated, hr.person.offboarded, hr.timesheet.approved, hr.leave.approved

Sample: POST /leave/requests

```
{
  "personId": "uuid",
  "leaveTypeId": "uuid",
  "startDate": "2025-09-01",
  "endDate": "2025-09-05",
  "reason": "Family event",
  "attachmentDocId": null
}
```

Response

```
{
  "id": "uuid",
  "status": "submitted",
  "approverChain": [
    {"role": "Manager", "personId": "uuid"},
    {"role": "HR", "personId": "uuid"}
  ]
}
```

Validation & Error Codes

- HRM_001: Duplicate KRA PIN
- HRM_002: Manager scope violation
- HRM_003: Insufficient leave balance
- HRM_004: Overlapping leave window
- HRM_005: Timesheet period locked
- HRM_006: Contract overlap or invalid effective dates

UI/UX Requirements (Developer Notes)

- People Directory with faceted search (dept, status, type)

- Profile page tabs: **Overview**, **Employment**, **Timesheets**, **Leave**, **Docs**, **Performance**
- Manager dashboard: Pending approvals (leave/timesheets), team KPIs
- HR admin dashboard: Headcount, joining/leaving this month, leave liability, approval SLAs

Non-Functional Requirements

- **Performance**: People list ≤ 300 ms at p95 (cached); write ops ≤ 800 ms p95
- **Scalability**: Handle 10k people, 1M timesheet entries/year
- **Reliability**: 99.9% API uptime; graceful retries on queue/webhook failures
- **Security**: Row-level scoping by tenant; PII encryption at rest where required; audit all admin actions
- **Compliance**: Retain HR records 7 years (configurable); export on DSAR (data subject access request)

3.2 Commission & Fee Engine Module

Purpose & Scope

A **rules-based engine** to calculate commissions, bonuses, referral fees and service charges for **marketers, contractors partners or sales agents**. Supports real-time or batch calculations, immutable financial results and auditability.

Concepts & Definitions

- **Earning Event**: A business event eligible for commission (e.g., *Invoice Paid*, *Project Milestone Complete*, *Subscription Renewal*).
- **Commission Rule**: A declarative definition of *who* earns *what*, *when* and *how much*.
- **Calculation Run**: A deterministic execution that evaluates eligible events against rules to produce **Earning Lines**.

- **Payout**: A grouping of approved earnings by payee and period, leading to disbursement (Finance posts journal entries and executes payment).

- **Adjustment/Clawback**: Positive/negative corrections (e.g., refund, chargeback, dispute resolution).

Rule Model (Types & Conditions)

Supported rule actions

- **Percentage** of base amount (e.g., 5% of net revenue)
- **Flat** amount per event (e.g., KES 500 per qualified sale)
- **Tiered percentage** (e.g., 3% for first KES 1M, 5% thereafter in a period)
- **Tiered by count** (e.g., KES 1,000 per sale for first 10, KES 1,500 for next 20)
- **Stair-step bonuses** (lump sums upon thresholds)
- **Time-bounded promos** (effective_from/to)
- **Caps/Floors** (per event, per period, per payee)
- **Splits** (multi-recipient split by percentage or absolute)

Conditions (any combination via AND/OR):

- Product/Service, Category, Project, Client Segment, Geography
- Opportunity Stage (e.g., only Closed-Won), Payment Status (only when **paid**), Revenue Type (new vs renewal)
- Min/Max deal size, Margin threshold, Contract term length
- Payee Attributes (role, seniority, region)

Precedence & conflicts

- Rules have priority (lower = earlier) and stop_processing flags.
- Optionally **most specific match wins** (longest condition match) when conflicts arise.

3.2.4 Data Model (Relational, simplified)

commission_rule

- id: uuid PK
- name, description

- status ENUM('draft','active','paused','archived')
- priority INT, stop_processing BOOLEAN default false
- `action_type ENUM('pe

-

3.4 Customer Relationship Management (CRM) Module

Purpose & Scope

The CRM module manages the entire lifecycle of client acquisition and engagement. It enables tracking of leads, opportunities, communications and client accounts, with a focus on **sales pipeline visibility**, **conversion tracking** and **seamless handoff to project execution** upon deal closure.

Core Entities

- **Lead**: Potential client with initial interest.
- **Contact**: Individual client representatives.
- **Company**: Client organization or corporate entity.
- **Opportunity**: A qualified deal linked to a lead or contact.
- **Pipeline Stage**: Predefined sales process stages (Prospect → Qualified → Proposal → Negotiation → Closed).
- **Communication Log**: Emails, calls, meetings, and notes.
- **Campaign**: Marketing efforts linked to lead generation.

Key Features

- Lead capture (manual, import, API integration with website forms).
- Lead scoring and qualification rules.
- Multi-stage sales pipeline with visual kanban board.
- Activity tracking (calls, meetings, tasks).
- Automated reminders for follow-ups.
- Email/SMS integration for outbound campaigns.
- Opportunity forecasting (expected close date, probability).
- Automatic creation of **Client record** in Project module when deal is won.

Workflows

1. **Lead Capture → Qualification → Conversion**
    - Lead enters system (manual entry, website, campaign).
    - Qualification based on predefined scoring rules.
    - Convert qualified lead into **Opportunity**.
2. **Opportunity Pipeline Management**
    - Sales user moves opportunity across pipeline stages.
    - Notes, communications, and documents are attached.
3. **Deal Closure → Handoff to Project Management**
    - Closed deal triggers:
        - Auto-creation of **Client record** in Project module.
        - Generation of first Invoice draft in Finance module.
        - Notification to assigned Project Manager.

Integration Touchpoints

- **Finance**: For invoicing once opportunity is closed.
- **HRM**: Salesperson/Account Manager assignment.
- **Projects**: Client & opportunity handoff.
- **Docs**: Attach proposals, contracts.

Compliance & Security

- GDPR-style consent for client contact.
- Role-based access (sales see only their leads).
- Audit trail for all pipeline movement.

Reporting

- Pipeline velocity reports.
- Conversion rate analytics.
- Top sources of leads.
- Forecast vs. actual closed sales.

UI Considerations

- Kanban pipeline board.
- Lead & contact detail pages with full history.
- Dashboard KPIs: New leads this month, conversion %, open opportunities.

3.5 Finance & Accounting Module

Purpose & Scope

The Finance module ensures all transactions are tracked in line with double-entry principles. It supports **invoicing, expense tracking, reconciliation, commission payouts**, and ensures financial transparency for audits.

Core Entities

- **Invoice**: Issued to clients.
- **Expense**: Company cost record.
- **Chart of Accounts**: Accounting ledger setup.
- **Journal Entry**: Debit/credit postings.
- **Payment**: Incoming/outgoing transaction.
- **Tax Rule**: VAT, withholding, or local tax rules.

Key Features

- Invoice generation (manual & auto from projects).
- Recurring billing support.
- Expense categorization (CapEx, OpEx).
- Journal entries for every transaction.
- Payment gateway integration (M-Pesa, bank).
- Commission payout integration with **Fee Engine**.
- Automatic financial statements: Balance Sheet, P&L, Cash Flow.
- Multi-currency support with exchange rate logs.

Workflows

1. **Invoice Lifecycle**
    - Created from Project deliverables or CRM deal.
    - Sent via email with PDF attachment.
    - Payment recorded → Status updated.

2. **Expense Recording**

    - Expense submitted (manual or receipt upload).
    - Linked to Chart of Accounts.
    - Approved → Journal entry posted.
3. **Month-End Close**
    - Reconciliation of bank statement.
    - Generation of P&L, Balance Sheet.

Integration Touchpoints

- **CRM**: For invoicing upon deal closure.
- **Projects**: For billing based on milestones.
- **Commission Engine**: For partner/contractor payout.
- **Docs**: Auto-generate invoices, receipts.

Compliance & Security

- ACID compliance for all financial data.
- Audit trail for every journal entry.
- Access control (only Finance/Admin can approve).
- Local tax regulations compliance.

Reporting

- Monthly P&L statement.
- Expense breakdown by category.
- Aged receivables/payables.
- Cash flow forecasting.

UI Considerations

- Dashboard with cash position, outstanding invoices, and top expenses.
- Drill-down reports with filters.
- Invoice templates customizable with branding.

3.6 Document Management System (DMS)

Purpose & Scope

Centralized management of all business documents with versioning, controlled access, and auto-generation of business artifacts (e.g., invoices, NDAs, contracts).

Core Entities

- **Document**: Any uploaded file (contract, receipt, report).
- **Version**: Document revision history.
- **Template**: For auto-generation (invoice, NDA, proposal).
- **Access Control**: Who can view/edit/download.

Key Features

- Upload & store files (PDF, DOCX, XLSX).
- Auto-generate from templates (invoices, contracts).
- Document tagging & search.
- Version control with rollback.
- Expiry/reminder system (contract renewals).
- Integration with Notification service for alerts.

Workflows

1. **Document Upload**
    - User uploads contract/receipt.
    - System stores, tags, and versions file.
2. **Document Generation**
    - Finance generates Invoice.
    - HRM generates Offer Letter.
    - Project module generates Delivery Note.
3. **Document Sharing**
    - Internal (restricted by roles).
    - External via secure link.

Integration Touchpoints

- **Finance**: Invoice/receipt auto-generation.
- **HRM**: Offer letters, contracts.
- **Projects**: Reports, deliverables.
- **CRM**: Proposals.

Compliance & Security

- Role-based access.
- Encryption at rest & in transit.
- Document history log.

Reporting

- Documents generated per module.
- Contract expiry report.
- Storage utilization.

UI Considerations

- Document library with filters (by type, owner, module).

- Version comparison UI.
- One-click generate & send options.

3.7 Admin, Security, & Reporting Module

Purpose & Scope

Provides **system-wide governance** including user roles, permissions, audit trails, module configuration, and central reporting for management.

Core Entities

- **User**: System account holder.
- **Role**: Defines access scope (Admin, Finance, Sales, Project Manager).
- **Permission**: CRUD access for each module.
- **System Setting**: Configurations (currency, tax rules, branding).
- **Audit Log**: Historical record of user actions.

Key Features

- Role-based access control (RBAC).
- Module configuration (enable/disable features).
- Centralized audit trail.
- Organization-wide settings (logo, fiscal year start, tax %).
- Super-admin management of other admins.

- Reporting engine that aggregates across modules.

Workflows

1. **User Onboarding**
    - Admin creates user → assigns role.
    - User gets login credentials via email.
2. **Permission Enforcement**
    - System enforces permissions per role.
    - Logs unauthorized access attempts.
3. **Audit & Reporting**
    - All user actions logged.
    - Reports generated across HRM, CRM, Finance, etc.

Integration Touchpoints

- **All modules** for user permissions & logging.
- **Notification service** for security alerts.

Compliance & Security

- Encryption for sensitive data.

- Password policies & 2FA.
- Audit logs immutable.

Reporting

- User activity report.
- Role-permission matrix.
- Module usage analytics.

UI Considerations

- Admin console dashboard (users, roles, recent activity).
- Audit log viewer with filters.
- Settings page for org configuration.

4.0 Data Model & Database Design

This section will define the relational data model, key tables, integrity constraints, and validation rules for the ERP.

- UUID primary keys (uuid_generate_v4()), snake_case identifiers, singular table names.
- **Multi-tenancy** via tenant_id on all domain tables (UUID referencing an admin.tenant record if multi-tenant; for single-tenant deployments, retain the column for portability).

- Timestamps as timestamptz (UTC) + explicit business dates (date). Default created_at/updated_at audit columns.
- Monetary values as numeric(18,2) or numeric(18,4) for rates. Currencies as ISO-4217 char(3).
- Controlled enumerations via either **Postgres ENUM types** (for stable, low-change enums) or **lookup tables** (for tenant-configurable labels). In this spec we mix both and call out which is which.
- **JSONB** columns where flexibility is needed (attributes, tier configs) while keeping primary facts normalized.
- **Row-Level Security (RLS)** policies scoped by tenant_id and user role claims.

Indexing, partitioning, and performance notes are embedded under the examples most affected by scale.

4.1 Entity-Relationship Diagram (ERD)

4.1.1 Conceptual ERD (Modules & Key Relations)

Admin/Security

auth.user (1) —< user_role >— (1) admin.role —< role_permission >— (1) admin.permission

This describes a chain of 4 tables:

auth.user (1) —< user_role

What it means: The user_role table is a **junction table** (or linking table) that sits between auth.user and admin.role. Its purpose is to break up a "many-to-many" relationship.

In the Database: The user_role table will have at least two columns: user_id (foreign key to auth.user.id) and role_id (foreign key to admin.role.id).

user_role >— (1) admin.role

Many User Roles are linked to one Role.

This completes the first half of the many-to-many relationship. Many records in the user_role table can point to the same single record in the admin.role table.

Complete Picture (so far): This establishes that **"One User can be associated with Many Roles and conversely, One Role can be assigned to Many Users."**

admin.role —< role_permission

One Role can have many Role Permissions.

What it means: Just like before, role_permission is another junction table. It sits between admin.role and admin.permission. It will have columns for role_id and permission_id.

role_permission >— (1) admin.permission

Many Role Permissions are linked to one Permission.

What it means: Many records in the role_permission table can point to the same single record in the admin.permission table.

Complete Picture: This establishes that One Role can be associated with Many Permissions, and One Permission can be granted to Many Roles.

Basically , a single User can be assigned multiple Roles. Each of those Roles can be granted multiple Permissions.

HRM

- hrm.person (1)—< hrm.employment_contract

  One Person can have many Employment Contracts.

  This is designed to track a person's employment history over time. A single person e.g contractor can have multiple contracts. For example:

  They might start with a **3-month probation** contract.

Then get a **1-year fixed-term** contract.

Then finally be offered a **permanent** contract.

The hrm.employment_contract table will have a person_id column that points back to the hrm.person table. One person_id will appear on many different contract records

- hrm.department (1)—< hrm.position (1)—< hrm.person

  One Department has many Positions. One Position is held by one Person.

  This is about job roles and organizational structure.

  A **Department** "Marketing" contains many job **Positions** (like "Senior Designer".

  A **Position** is a specific role in the company. At any given time, a Position is held by one, and only one, Person. This models the company's "headcount" or org chart.

  Why it's useful: If the person in the "Senior Designer" position leaves, the position itself remains. You can then assign a new person to that same position, maintaining the company structure.

- hrm.person (1)—< hrm.timesheet_entry >—(1) project.task

  One Person logs many Timesheet Entries. One Timesheet Entry is for one specific Task.

  What it means: This is the core of time tracking.

  A **Person** (employee/contractor) logs their worked hours by creating many **Timesheet Entries** (e.g., one for each day, or even multiple per day for different tasks).

  Each **Timesheet Entry** must be linked to one specific **Task** from a project. This answers the question, "What specific work did you spend time on?"

The Big Picture: This is a critical link between **HR** (who is working) and **Projects** (what they are working on). It's how you know who to pay and which client to bill for the time

- hrm.person (1)—< hrm.leave_request (>) hrm.leave_type

  One Person submits many Leave Requests. One Leave Request is for one specific Leave Type."

  What it means: This manages time-off requests.

  A **Person** can submit many **Leave Requests** throughout the year e.g., for vacation, sick days, parental leave.

  Each **Leave Request** must be for a specific **Leave Type** (e.g., "Annual Leave", "Sick Leave", "Unpaid Leave"). The hrm.leave_type table defines the rules for each type (e.g., how many days are allocated, whether a medical certificate is required).

  Why it's useful: It automates leave management by enforcing rules based on the type of leave being requested.

- hrm.person (1)—< hrm.person_document —(1) dms.document

  One Person has many Personal Documents. Each Personal Document is one file in the Document Management System.

  What it means: This handles all the paperwork associated with a person.

  A **Person** will have many **Documents** associated with them (e.g., signed contract, CV, ID copy, certificate copies, performance reviews.

  The hrm.person_document table likely acts as a link table that connects a Person to a file stored in the central **Document Management System (DMS)**. It might also store metadata about that document (e.g., "Document Type", "Expiry Date").

  The Big Picture: This creates a single, organized repository for all employee-related documents, making it easy to find, manage, and control access to sensitive files

CRM

- crm.company (1)—< crm.contact
- crm.company (1)—< crm.opportunity (> crm.pipeline_stage)
- crm.lead (0..1) —> crm.opportunity (conversion)
- crm.communication_log (> contact|opportunity)

Projects

- project.project (1)—< project.project_phase (1)—< project.task

  One Project is broken down into many Phases. One Phase contains many Tasks.

  It's how we will take a large, complex project and break it down into manageable pieces.

  A **Project** is the overall engagement "Build a new website for Client X"

  A **Phase** is a major stage within that project "Phase 1: Discovery & Planning", "Phase 2: Design", "Phase 3: Development", "Phase 4: Launch"

  A **Task** is a specific, actionable unit of work within a phase "Create sitemap", "Design homepage mockups", "Develop user login functionality").

  This hierarchy allows for better planning, assignment, and tracking. You can see progress at the task, phase, and overall project level

- project.task (1)—< project.timelog (> hrm.person)

  One Task has many Timelog entries. One Timelog entry is logged by one Person.

  What it means: This is the connection between **planning** and **execution**.

  A **Task** ("Develop user login functionality") is what needs to be done.

A **Timelog** is a record of time spent doing that work (e.g., "8 hours worked on May 27th").

Each timelog is created by a specific **Person** (an employee or contractor), answering the question "Who did the work?"

The Big Picture: This is the most critical link in the entire ERP. It connects:

Projects (What work was done?)

HR (Who did the work and needs to be paid?)

- project.project (1)—< project.deliverable (> dms.document)

  One Project has many Deliverables. One Deliverable is one file (or document) in the Document Management System.

  What it means: This tracks the actual **outputs** or **assets** produced by the project.

- A **Deliverable** is a tangible item promised to the client (e.g., "Final Design Report", "Source Code", "User Training Manuals").
- Each deliverable is a **Document** (a PDF, a ZIP file, a video) stored in the central DMS.

  Why it's useful: It ensures that everything that was promised is actually produced, approved, and stored in an organized way. It's your proof of work.

- project.project (1)—< project.change_request

  One Project has many Change Requests.

  What it means: This handles the inevitable **"scope creep"** in a controlled way.

  - A **Change Request** is a formal proposal to alter the original project plan "The client wants to add a new feature", "We discovered a technical constraint that requires a different approach"
  - Each request would typically be evaluated for its impact on the project's budget, timeline and resources before being approved or rejected.

It prevents unauthorized changes from derailing a project. It ensures that any change to the original agreement is properly documented, priced, and approved, protecting profitability.

Finance

- fin.invoice (1)—< fin.invoice_line (> project|crm)
- fin.payment (>) fin.invoice (many payments to invoice)
- fin.journal_entry (1)—< fin.journal_line (> fin.chart_of_accounts)

1. The "Invoicing & Billing" Feature

You create a bill for a client. The bill has a list of items and a total.

How it works in the database:

fin.invoice table: Stores the main bill (Invoice Number, Client, Date, Total Amount).

fin.invoice_line table: Stores every single item *on* that bill e.g., "50 hours of development," "Project Management fee").

The Magic Link (> project|crm): Each of those line items can be connected to the actual **work that was done**.

Did you log time in the Projects module? The system can automatically generate an invoice line for it.

Is it a fixed fee? It can be linked directly to the project.

This is the **integration** that makes the system powerful.

2. The "Expense Tracking" & "Profit/Loss & Cash Flow" Features

What you see: You get reports showing your income, your costs, and how much money is in the bank.

How it works in the database:

fin.journal_entry table: This is the system's official diary for **every single money-related event**.

fin.journal_line table: For every event, the system writes two diary entries to keep everything in balance (this is called **double-entry bookkeeping**).

Example: You pay an electricity bill of KES 200.

The system writes in its diary: "**Decreased** Bank Account by kes 200"

And simultaneously: "**Increased** Electricity Expenses by kes 200"

Your **Profit/Loss** report is just a summary of all the "Expense" and "Income" entries in this diary.

Your **Cash Flow** report is a summary of all the "Bank" and "Cash" entries.

3. The "Integration with the Commission Engine" Feature

When an invoice is paid, the salesperson's commission is calculated automatically.

How it works in the database:

- When an invoice in the fin.invoice table is marked as **PAID**, it sends a message (an event) to the Commission module.
- The commission engine reads its rules and calculates the commission owed.
- It then creates a payout record for the salesperson.
- **Crucially**, it also makes an entry in the financial diary (journal_entry) to record this new business cost:
    - "**Increased** Commission Expenses"
    - "**Increased** Commissions Payable (we now owe this money to the salesperson)"

4. The "Audit Logs" Feature

What you see: A log that shows who created, edited, or deleted any invoice or financial record.

How it works in the database:

- A separate audit_log table watches the invoice, payment, and journal_entry tables.
- Every time someone changes anything, it automatically records:

    - **Who** did it
    - **What** they changed
    - **When** they did it
- This log cannot be easily changed, ensuring everything is traceable and secure.

Simply:

The invoice and payment tables handle dealing with clients.
The journal_entry system is the secret engine that keeps all the money perfectly accounted for.
And everything is connected to the other modules (Projects, CRM, HR) to automate processes and give you a complete picture of your business health.

Commission

- com.rule (1)—< com.earning_line (> com.event)

  One Commission Rule can generate many Earning Lines. One Earning Line is triggered by one Event.

  his is the core of the "engine." You define the logic for how people get paid.

  A **Rule** is the "if-then" statement. (e.g., "IF a project invoice is paid, THEN pay the marketer 10% of the value" or "IF a contractor completes a task, THEN pay a Kes 5000 bonus").

  An **Event** is the "IF" trigger. It's something that happens in the system (e.g., invoice.paid, task.completed, milestone.achieved).

  An **Earning Line** is the "THEN" result. It's a record that says, "Because of this event, this person is owed this amount of money according to that rule."

  It's incredibly flexible. You can create any commission structure without needing a programmer to change the code. When an event occurs, the system checks all the rules and automatically creates the earnings.

  com.calculation_run (1)—< com.earning_line

One Calculation Run creates many Earning Lines."

This is for **auditing and control**. Sometimes you don't want to process events one-by-one. Instead, you want to run a batch process (e.g., "Calculate all commissions for the month of October").

- A **Calculation Run** is like pushing a "Calculate" button. It's a record of when the commission engine was run and for what period.
- All the **Earning Lines** created during that specific run are linked to it.

If there's ever a question about the numbers, you can go back and see exactly which calculation run created them. You can also compare different runs or re-run a period if a mistake was found.

The Complete Commission Story in Simple Steps:

1. **Something Happens:** An invoice is paid (event).
2. **Rule Checks:** The system checks all its rules. "Is there a rule for 'invoice.paid'? Yes!"
3. **Earning Created:** It follows the rule and creates an earning_line record: "Jane gets kes 1000." This earning might be part of a specific calculation_run.
4. **Adjustment (Optional):** A manager sees the earning and adds a kes 200 bonus adjustment. The total owed is now kes 1200.
5. **Payout Time:** At the end of the month, someone creates a payout_batch. The system gathers all approved earnings and creates a payout_line for Jane for kes 1200.
6. **Payment:** The payout_batch is sent to the Finance module, which then pays Jane.

DMS

- dms.document (1)—< dms.document_version ;

  One Document can have many Versions.

  This is version control for your files, just like Google Docs or GitHub.

  - A **Document** is the main file (e.g., "Client X Contract").

    - A **Document Version** is a record of every single time that file was updated. So you have "Client X Contract v1", "v2", "v3", etc.

You never lose history. You can always see who changed what and when. If someone makes a mistake, you can revert to a previous version. This is critical for legal and compliance reasons.

Template documents are used to automatically create contracts, invoices, and reports."

When a marketer wins a deal, the system can automatically generate a new contract from the dms.template, pulling the client's name from the crm module and the project value from the project module. The same goes for invoices, proposals, and reports.

It saves massive amounts of time, ensures consistency, and reduces errors in your paperwork.

Cross-Module Anchors

- crm.company —(1) project.project —(>) fin.invoice (Deal → Project → Invoice)

  One Company can have many Projects. One Project results in one Invoice.

  This is the entire **core business process**.

1. **CRM:** You start with a company in your CRM (a lead or client).
2. **Projects:** You win their business and create a project for them. This project is linked to the company.
3. **Finance:** You do the work and then create an invoice for that specific project.

This connection lets you run reports like "How much revenue did we generate from Client X?" or "What's the profitability of Project Y?" You can trace every shilling back to its source.

- fin.invoice (paid) —> com.event (drives earnings)

  When an Invoice is paid, it triggers an Event that tells the Commission Engine to calculate earnings.

This is the critical link between **Revenue** and **Commission**.

The system doesn't pay commissions when you *invoice* a client. It pays commissions when the client *actually pays* the invoice.

This event (invoice.paid) is the trigger that kicks off the entire commission calculation process for the salesperson or marketer.

It ensures you only pay out commissions on money you have actually received. This protects your company's cash flow.

- hrm.person —> project.timelog —> fin.invoice_line (T&M)

  One Person records many Timelog entries. Those Timelog entries become Line Items on an Invoice."

  This is the **"Time & Materials" (T&M)** billing process.

1. **HR:** A person (employee/contractor) does work.
2. **Projects:** They log their time in a timelog against a specific task.
3. **Finance:** When it's time to bill the client, the system automatically generates an invoice_line for every hour logged, ready to be put on an invoice.

It fully automates the billing process for time-based work. It ensures that every hour of work is captured, billed, and can be connected to the person who did it for payroll purposes. This is how you ensure you don't leave money on the table