CS 341 Database Systems - Course Project Report

FinanceHub: Multi-Organization Financial Management System

Course: CS 341 Database Systems

Semester: Fall 2025

Instructor: Abeera Tariq

Team Members:

Adeena Arif – 29025

Eesha Ali – 29025

December 10, 2025

Contents

Executive Summary

FinanceHub is a comprehensive multi-organization financial management system designed to streamline financial operations for small to medium-sized businesses. Built on PostgreSQL via Supabase, the system employs a three-tier architecture, featuring a TypeScript/Node.js backend and a React TypeScript frontend. Github Repo Link: https://github.com/6d2nr6npdh-dev/finance-management-system

Key Features

- Multi-tenant architecture supporting multiple independent organizations
- Role-based access control (Admin, Manager, Employee, Viewer)
- Real-time account balance tracking with automated triggers
- Comprehensive transaction management (Income, Expense, Transfer)
- Invoice generation and payment tracking
- Budget management with automated spending alerts

Technology Stack

- **Frontend:** React with TypeScript, TailwindCSS
- **Backend:** Node.js with TypeScript, Express.js
- **Database:** PostgreSQL (Supabase)
- **Authentication:** Supabase Auth

1 Business Scenario

1.1 Domain Selection

Financial Management and Accounting domain targeting small to medium-sized businesses requiring comprehensive financial tracking without the complexity of enterprise ERP systems.

1.2 Real-World User Interview

Interviewee: Syed Arshad Ali, Finance Manager
Date: October 5
Duration: 30 minutes

Key Pain Points Identified

- Data Isolation: Complete separation needed between client accounts
- Manual Balance Tracking: Error-prone reconciliation after each transaction
- Invoice Management: Tracking payment status requires multiple spreadsheets
- Budget Monitoring: No automated way to track spending against budgets
- Recurring Transactions: Manual monthly entries lead to missed payments
- Category Flexibility: Need for custom categorization per department

1.3 Existing Application Analysis

Applications Analyzed: QuickBooks Online, Wave Accounting, FreshBooks

Common Limitations

- No true multi-organization support (separate subscriptions required)
- Expensive pricing models ($30–$200/month)
- Limited customization for specific business needs
- Limited automated recurring transaction support
- Weak audit capabilities

1.4 Problem Statement

- Lack of True Multi-Tenancy
- Manual Data Entry
- No Real-Time Updates
- Missing Budget Controls

1.5 Our Solution: FinanceHub

- Multi-organization support with complete data isolation
- Automated balance management via database triggers
- Recurring transaction automation
- Comprehensive audit trail for every change
- Flexible budget system with threshold alerts

- Role-based security per organization
- Modern TypeScript stack for type safety
- Cost-effective open-source foundation

2 Business Rules

2.1 Organization and User Management

- BR-1: Each organization has exactly one owner with full administrative privileges
- BR-2: Users can belong to multiple organizations with different roles
- BR-3: Complete data isolation—users only access their organization's data
- BR-4: Valid roles: Admin, Manager, Employee, Viewer (descending privileges)

2.2 Account Management

- BR-5: Each organization must have at least one active account
- BR-6: Account types: checking, savings, credit card, loan, investment
- BR-7: Account balances automatically calculated from transactions
- BR-8: Accounts can be soft-deleted but never permanently deleted if transactions exist

2.3 Transaction Rules

- BR-9: All transaction amounts must be positive numbers (> 0)
- BR-10: Transaction types: income, expense, transfer
- BR-11: Income increases balance; expense decreases balance
- BR-12: Transfers require both source and destination accounts
- BR-13: Every transaction must have a category matching its type
- BR-14: Only completed transactions affect balances and budgets

2.4 Invoice and Payment Rules

- BR-15: Invoice numbers must be unique system-wide
- BR-16: Invoice total = subtotal + tax - discount (auto-calculated)
- BR-17: Invoice status workflow: draft → sent → viewed → partially_paid → paid/overdue

- BR-18: Payments cannot exceed the outstanding amount due
- BR-19: Each invoice must have at least one line item

2.5 Budget Rules

- BR-20: Budget periods: monthly, quarterly, or annual
- BR-21: Budget spent amount auto-calculated from completed expense transactions
- BR-22: Alert threshold (default 80%) triggers notification when reached
- BR-23: Multiple budgets can exist for same category in different periods (no overlap)

2.6 Audit and Compliance

- BR-24: All transaction operations logged in audit_logs
- BR-25: Audit logs are immutable, cannot be updated or deleted
- BR-26: Logs capture: user, action, timestamp, table, record ID, old/new values

3 Entities, Attributes, and Relationships

3.1 Core Entities

Organizations

Independent business entities using the platform.
Attributes:

- id (UUID, PK)
- name (TEXT)
- slug (TEXT, UNIQUE)
- currency (TEXT, DEFAULT 'USD')
- fiscal_year_start (INTEGER)
- created_by (UUID, FK)

User Profiles

Extended user profile information.
Attributes:

- id (UUID, PK)

- full_name (TEXT)
- avatar_url (TEXT)
- phone (TEXT)

Organization Members

Links users to organizations with defined roles.
Attributes:

- id (UUID, PK)
- organization_id (UUID, FK)
- user_id (UUID, FK)
- role (ENUM: admin, manager, employee, viewer)
- UNIQUE(organization_id, user_id)

Accounts

Financial accounts belonging to organizations.
Attributes:

- id (UUID, PK)
- organization_id (UUID, FK)
- name (TEXT)
- type (ENUM)
- current_balance (NUMERIC)
- is_active (BOOLEAN)

Categories

Hierarchical classification structure for financial records.
Attributes:

- id (UUID, PK)
- organization_id (UUID, FK)
- name (TEXT)

- type (ENUM)
- parent_id (UUID, FK, self-reference)

Transactions

Represents income, expense, or transfer operations.
Attributes:

- id (UUID, PK)
- organization_id (UUID, FK)
- type (ENUM)
- status (ENUM)
- account_id (UUID, FK)
- to_account_id (UUID, FK, nullable)
- category_id (UUID, FK)
- amount (NUMERIC)
- date (DATE)

Invoices

Represents billing for customers or vendors.
Attributes:

- id (UUID, PK)
- invoice_number (TEXT, UNIQUE)
- payee_id (UUID, FK)
- status (ENUM)
- total (NUMERIC)
- amount_paid (NUMERIC)
- amount_due (NUMERIC)

Budgets

Represents planned spending for categories.
Attributes:

- id (UUID, PK)
- category_id (UUID, FK)
- amount (NUMERIC)
- spent (NUMERIC)
- period (ENUM)
- alert_threshold (INTEGER)

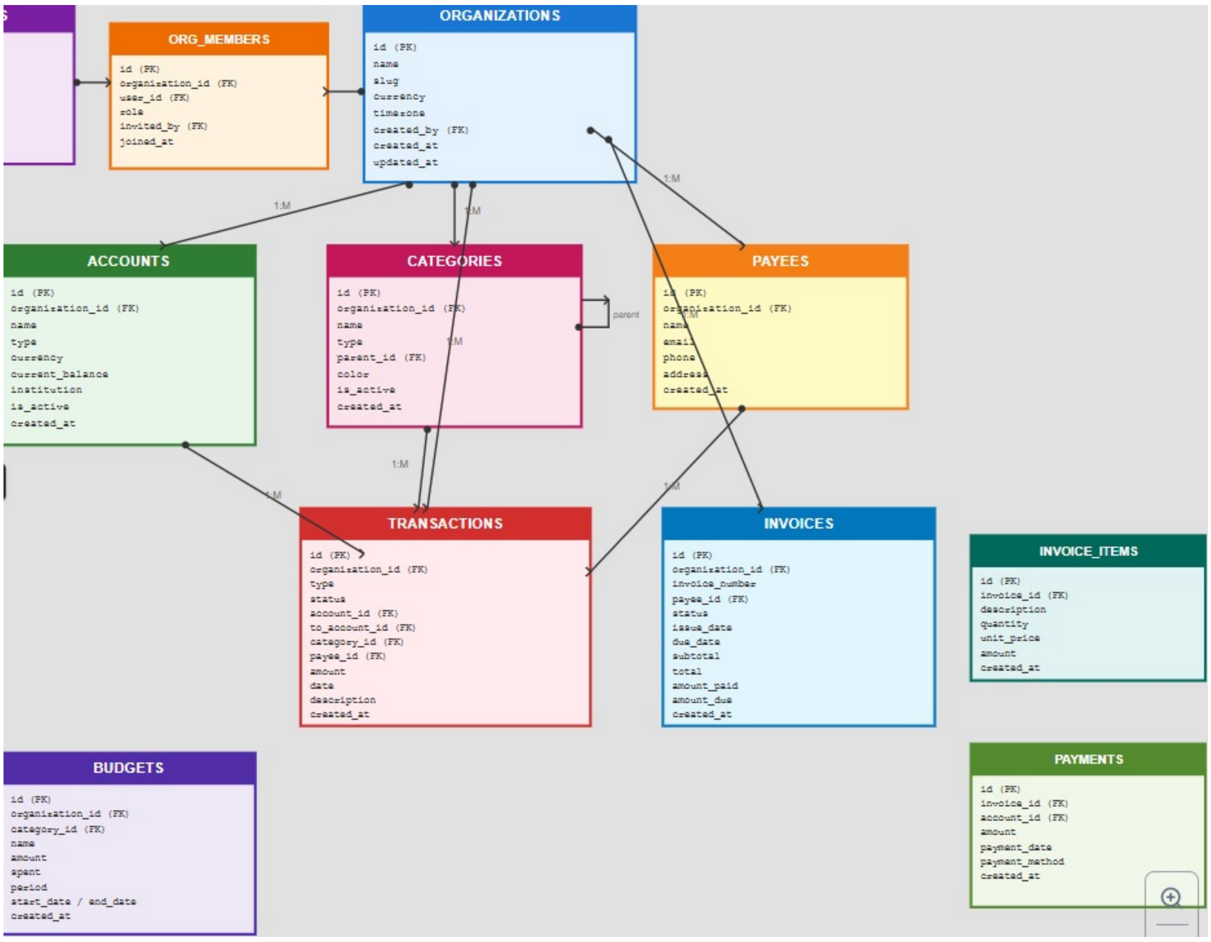


Figure 1: ER Diagram

4 Relational Schema

4.1 Database Schema Diagram

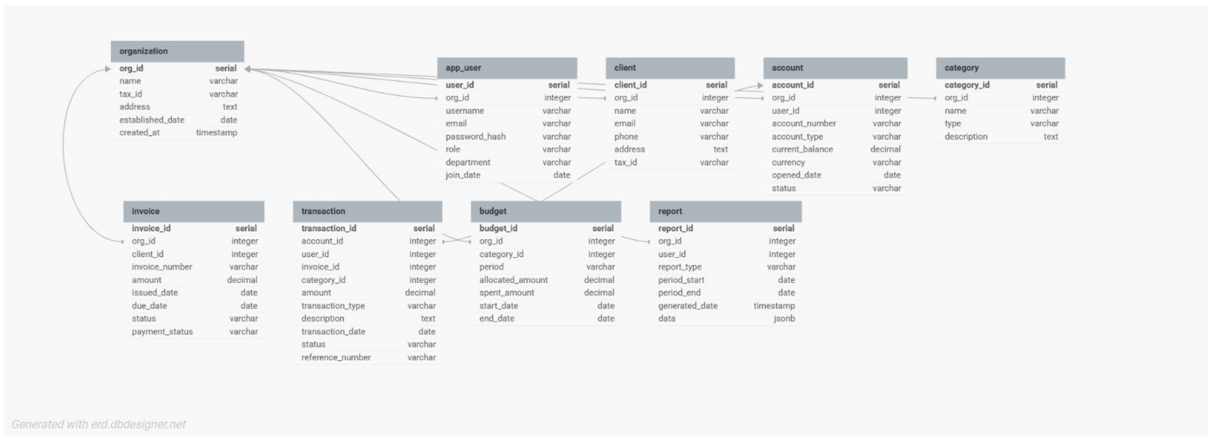


Figure 2: Database Schema

4.2 Normalization Validation

The database schema has been systematically normalized to Third Normal Form (3NF) to eliminate redundancy, prevent update anomalies, and ensure data integrity. The normalization process proceeded through each normal form as follows:

4.2.1 First Normal Form (1NF)

Definition: A relation is in 1NF if all attributes contain only atomic (indivisible) values, there are no repeating groups, and each table has a defined primary key.

Validation: All entities satisfy 1NF requirements:

- Every attribute contains single, indivisible values
- Each table has a primary key (UUID)
- No repeating groups within entities
- Each row is unique and identifiable

Example - Transactions Table:

Violates 1NF:

transaction_id	**amounts**	**categories**
1	100, 200	Food, Rent

Satisfies 1NF (our design):

transaction_id	**amount**	**category_id**
1	100	cat-123
2	200	cat-456

4.2.2 Second Normal Form (2NF)

Definition: A relation is in 2NF if it is in 1NF AND all non-key attributes are fully functionally dependent on the entire primary key (no partial dependencies).

Validation: All entities satisfy 2NF requirements:

- All tables use single-column primary keys (UUID)
- No composite primary keys exist, therefore no partial dependencies are possible
- All non-key attributes depend on the entire primary key
- Each attribute is functionally dependent on the primary key alone

Example - Invoice Items Table:

Correct (2NF):

```
invoice_items (id, invoice_id, description, quantity, unit_price, amount)
```

All attributes (description, quantity, unit_price, amount) depend on the primary key id, with no partial dependency on invoice_id alone.

Would violate 2NF:

```
invoice_items (invoice_id, item_number, description, invoice_date)
```

Here, invoice_date depends only on invoice_id (partial dependency), not on the complete composite key (invoice_id, item_number). This attribute should be in the invoices table instead.

4.2.3 Third Normal Form (3NF)

Definition: A relation is in 3NF if it is in 2NF AND has no transitive dependencies (non-key attributes must not depend on other non-key attributes).

Validation: All entities satisfy 3NF with strategic denormalizations:

Transitive Dependencies Eliminated:

1. **Accounts Table:**

  - current_balance could be derived from transactions but is stored for performance
  - Maintained by triggers to ensure consistency
  - This is an acceptable denormalization for read optimization

2. **Invoices Table:**
  - subtotal, tax_amount, total, amount_due are derived values
  - Calculated and maintained by triggers
  - Prevents expensive JOIN calculations on every query

3. **Budgets Table:**
  - spent is derived from transactions
  - Automatically updated by trigger when transactions are created
  - Improves query performance for budget monitoring

Example of 3NF Compliance:

Correct (3NF):

```
transactions (id, org_id, account_id, category_id, payee_id, amount)

accounts (id, name, type, current_balance)

categories (id, name, type)

payees (id, name, email)
```

- Transaction attributes do not depend on each other
- Related data stored in separate normalized tables
- Joined via foreign keys when needed

Would violate 3NF:

```
transactions (id, org_id, account_id, account_name, account_type, category_id,
category_name, amount)
```

- account_name depends on account_id (transitive dependency)
- category_name depends on category_id (transitive dependency)
- These create update anomalies and redundancy

Conclusion:

The database is normalized to 3NF with strategic denormalizations for performance that are maintained by database triggers to ensure consistency. These denormalizations optimize read operations (which occur more frequently than writes) while the triggers ensure that derived values remain accurate and synchronized with source data.

5 DDL Script Highlights

5.1 Constraints Applied to Tables

Primary Key Constraints

Every table uses a UUID primary key generated via `uuid_generate_v4()` to ensure globally unique identifiers.

```sql
CREATE TABLE public.organizations (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  name TEXT NOT NULL,
  slug TEXT UNIQUE NOT NULL
  -- other columns...
);
```

Listing 1: Primary Key Example

Foreign Key Constraints

Delete behaviors reflect business logic.

```sql
CREATE TABLE public.transactions (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  organization_id UUID NOT NULL REFERENCES organizations(id)
    ON DELETE CASCADE,   -- Remove transactions with organization
  account_id UUID NOT NULL REFERENCES accounts(id)
    ON DELETE RESTRICT,  -- Prevent deletion if referenced
  category_id UUID REFERENCES categories(id)
    ON DELETE SET NULL,  -- Preserve transactions, null category
  payee_id UUID REFERENCES payees(id)
    ON DELETE SET NULL
);
```

Listing 2: Foreign Key Constraints with Delete Behaviors

Foreign Key Delete Strategies

- **ON DELETE CASCADE:** For organization_id to remove related data when an organization is deleted.
- **ON DELETE RESTRICT:** For account_id to prevent accidental account deletion.

- **ON DELETE SET NULL:** For category_id and payee_id to preserve transaction history.

Unique Constraints

```sql
-- Organization slug must be unique across system
CREATE TABLE public.organizations (
  slug TEXT UNIQUE NOT NULL
  -- other columns...
);

-- User can only be member of organization once
CREATE TABLE public.organization_members (
  organization_id UUID NOT NULL,
  user_id UUID NOT NULL,
  UNIQUE (organization_id, user_id)
);

-- Invoice numbers must be globally unique
CREATE TABLE public.invoices (
  invoice_number TEXT UNIQUE NOT NULL
  -- other columns...
);
```

Listing 3: Unique Constraints

Check Constraints

```sql
-- Transaction amounts must be positive
CREATE TABLE public.transactions (
  amount DECIMAL(15,2) NOT NULL,
  CONSTRAINT positive_amount CHECK (amount > 0)
);

-- Transfer transactions must have destination account
ALTER TABLE transactions ADD CONSTRAINT transfer_requires_to_account
  CHECK (type != 'transfer' OR to_account_id IS NOT NULL);
```

Listing 4: Check Constraints for Business Logic

NOT NULL Constraints

```sql
CREATE TABLE public.transactions (
  organization_id UUID NOT NULL,
  type transaction_type NOT NULL,
  account_id UUID NOT NULL,
  amount DECIMAL(15,2) NOT NULL,
  date DATE NOT NULL,
  created_at TIMESTAMPTZ DEFAULT NOW()
);
```

Listing 5: NOT NULL Constraints

Default Value Constraints

```sql
CREATE TABLE public.accounts (
  currency TEXT DEFAULT 'USD',
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  current_balance DECIMAL(15,2) DEFAULT 0
);

CREATE TABLE public.budgets (
  alert_threshold INTEGER DEFAULT 80,  -- 80% default alert
  alert_enabled BOOLEAN DEFAULT true,
  spent DECIMAL(15,2) DEFAULT 0
);
```

Listing 6: Default Value Constraints

ENUM Type Constraints

```sql
CREATE TYPE account_type AS ENUM (
  'checking', 'savings', 'credit_card', 'cash', 'investment', 'loan', 'other'
);

CREATE TYPE transaction_type AS ENUM ('income', 'expense', 'transfer');

CREATE TYPE transaction_status AS ENUM ('pending', 'cleared', 'reconciled', 'void');

CREATE TYPE member_role AS ENUM ('owner', 'admin', 'manager', 'employee', 'viewer');

CREATE TYPE invoice_status AS ENUM (
  'draft', 'sent', 'viewed', 'partial', 'paid', 'overdue', 'cancelled'
);

CREATE TYPE budget_period AS ENUM ('monthly', 'quarterly', 'yearly', 'custom');
```

Listing 7: ENUM Type Definitions

5.2 Triggers, Stored Procedures, and Views

Database Triggers

Trigger 1: Update Account Balance

Automatically updates account balances for cleared/reconciled transactions.

```sql
CREATE OR REPLACE FUNCTION update_account_balance()
RETURNS TRIGGER AS $$
DECLARE
  v_should_affect_old BOOLEAN;
  v_should_affect_new BOOLEAN;
BEGIN
  v_should_affect_old := (OLD.status IN ('cleared', 'reconciled'));
  v_should_affect_new := (NEW.status IN ('cleared', 'reconciled'));

  IF TG_OP = 'INSERT' THEN
    IF v_should_affect_new THEN
      IF NEW.type IN ('expense','transfer') THEN
        UPDATE accounts
        SET current_balance = current_balance - NEW.amount
        WHERE id = NEW.account_id;
      ELSIF NEW.type = 'income' THEN
        UPDATE accounts
        SET current_balance = current_balance + NEW.amount
        WHERE id = NEW.account_id;
      END IF;

      IF NEW.type = 'transfer' AND NEW.to_account_id IS NOT NULL
         THEN
        UPDATE accounts
        SET current_balance = current_balance + NEW.amount
        WHERE id = NEW.to_account_id;
      END IF;
    END IF;

  ELSIF TG_OP = 'UPDATE' THEN
    -- Reverse old if needed
    IF v_should_affect_old THEN
      IF OLD.type IN ('expense','transfer') THEN
        UPDATE accounts
        SET current_balance = current_balance + OLD.amount
        WHERE id = OLD.account_id;
      ELSIF OLD.type = 'income' THEN
        UPDATE accounts
        SET current_balance = current_balance - OLD.amount
        WHERE id = OLD.account_id;
      END IF;
      IF OLD.type = 'transfer' AND OLD.to_account_id IS NOT NULL
         THEN
        UPDATE accounts
        SET current_balance = current_balance - OLD.amount
        WHERE id = OLD.to_account_id;
      END IF;
    END IF;

    -- Apply new if needed
    IF v_should_affect_new THEN
      IF NEW.type IN ('expense','transfer') THEN
        UPDATE accounts
        SET current_balance = current_balance - NEW.amount
        WHERE id = NEW.account_id;
      ELSIF NEW.type = 'income' THEN
        UPDATE accounts
        SET current_balance = current_balance + NEW.amount
        WHERE id = NEW.account_id;
      END IF;
      IF NEW.type = 'transfer' AND NEW.to_account_id IS NOT NULL
         THEN
        UPDATE accounts
        SET current_balance = current_balance + NEW.amount
        WHERE id = NEW.to_account_id;
      END IF;
    END IF;

  ELSIF TG_OP = 'DELETE' THEN
    IF v_should_affect_old THEN
      IF OLD.type IN ('expense','transfer') THEN
        UPDATE accounts
        SET current_balance = current_balance + OLD.amount
        WHERE id = OLD.account_id;
      ELSIF OLD.type = 'income' THEN
        UPDATE accounts
        SET current_balance = current_balance - OLD.amount
        WHERE id = OLD.account_id;
      END IF;
      IF OLD.type = 'transfer' AND OLD.to_account_id IS NOT NULL
         THEN
        UPDATE accounts
        SET current_balance = current_balance - OLD.amount
        WHERE id = OLD.to_account_id;
      END IF;
    END IF;
  END IF;

  RETURN COALESCE(NEW, OLD);
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_account_balance_trigger
AFTER INSERT OR UPDATE OR DELETE ON transactions
FOR EACH ROW EXECUTE FUNCTION update_account_balance();
```

Listing 8: Account Balance Update Trigger

Trigger 2: Update Invoice Totals

Recalculates invoice totals when line items change.

```sql
CREATE OR REPLACE FUNCTION update_invoice_totals()
RETURNS TRIGGER AS $$
DECLARE
  v_invoice_id UUID;
  v_subtotal DECIMAL(15,2);
BEGIN
  v_invoice_id := COALESCE(NEW.invoice_id, OLD.invoice_id);

  SELECT COALESCE(SUM(amount), 0) INTO v_subtotal
  FROM invoice_items
  WHERE invoice_id = v_invoice_id;

  UPDATE invoices
  SET
    subtotal      = v_subtotal,
    tax_amount    = v_subtotal * tax_rate / 100,
    total         = v_subtotal + (v_subtotal * tax_rate / 100) -
       discount_amount,
    amount_due    = (v_subtotal + (v_subtotal * tax_rate / 100) -
       discount_amount) - amount_paid
  WHERE id = v_invoice_id;

  RETURN COALESCE(NEW, OLD);
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_invoice_totals_trigger
AFTER INSERT OR UPDATE OR DELETE ON invoice_items
FOR EACH ROW EXECUTE FUNCTION update_invoice_totals();
```

Listing 9: Invoice Totals Auto-Calculation Trigger

Trigger 3: Auto-Update Budgets on Transaction Changes

Updates budget spending on cleared/reconciled expenses.

```sql
CREATE OR REPLACE FUNCTION auto_update_budgets_on_transaction()
RETURNS TRIGGER AS $$
DECLARE
  v_budget RECORD;
BEGIN
  IF TG_OP IN ('INSERT','UPDATE') THEN
    IF NEW.status IN ('cleared','reconciled') AND NEW.type = '
       expense' THEN
      IF NEW.category_id IS NOT NULL THEN
        FOR v_budget IN
```

```sql
          SELECT id FROM budgets
          WHERE organization_id = NEW.organization_id
            AND category_id = NEW.category_id
            AND is_active = true
            AND NEW.date BETWEEN start_date AND end_date
        LOOP
          PERFORM calculate_budget_spending(v_budget.id);
        END LOOP;
      END IF;

      FOR v_budget IN
        SELECT id FROM budgets
        WHERE organization_id = NEW.organization_id
          AND account_id = NEW.account_id
          AND is_active = true
          AND NEW.date BETWEEN start_date AND end_date
      LOOP
        PERFORM calculate_budget_spending(v_budget.id);
      END LOOP;
    END IF;
  ELSIF TG_OP = 'DELETE' THEN
    IF OLD.status IN ('cleared','reconciled') AND OLD.type = '
       expense' THEN
      -- Recompute budgets affected by deletion
      IF OLD.category_id IS NOT NULL THEN
        FOR v_budget IN
          SELECT id FROM budgets
          WHERE organization_id = OLD.organization_id
            AND category_id = OLD.category_id
            AND is_active = true
            AND OLD.date BETWEEN start_date AND end_date
        LOOP
          PERFORM calculate_budget_spending(v_budget.id);
        END LOOP;
      END IF;

      FOR v_budget IN
        SELECT id FROM budgets
        WHERE organization_id = OLD.organization_id
          AND account_id = OLD.account_id
          AND is_active = true
          AND OLD.date BETWEEN start_date AND end_date
      LOOP
        PERFORM calculate_budget_spending(v_budget.id);
      END LOOP;
    END IF;
  END IF;

  RETURN COALESCE(NEW, OLD);
END;
$$ LANGUAGE plpgsql;
```

```

CREATE TRIGGER auto_update_budgets_trigger
AFTER INSERT OR UPDATE OR DELETE ON transactions
FOR EACH ROW EXECUTE FUNCTION auto_update_budgets_on_transaction();
```

Listing 10: Budget Auto-Update Trigger

Trigger 4: Update Timestamp Trigger

Automatically sets `updated_at` on modification.

```
CREATE OR REPLACE FUNCTION update_updated_at()
RETURNS TRIGGER AS $$
BEGIN
  NEW.updated_at = NOW();
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_transactions_updated_at
BEFORE UPDATE ON transactions
FOR EACH ROW EXECUTE FUNCTION update_updated_at();

CREATE TRIGGER update_accounts_updated_at
BEFORE UPDATE ON accounts
FOR EACH ROW EXECUTE FUNCTION update_updated_at();
```

Listing 11: Automatic Timestamp Update

Stored Procedures (Functions)

Function 1: Create Transaction with Payee Auto-Creation

```
CREATE OR REPLACE FUNCTION public.create_transaction(
  p_organization_id UUID,
  p_type transaction_type,
  p_account_id UUID,
  p_to_account_id UUID,
  p_category_id UUID,
  p_payee_name TEXT,
  p_amount DECIMAL(15,2),
  p_date DATE,
  p_description TEXT,
  p_reference_number TEXT,
  p_notes TEXT,
  p_tags TEXT[],
  p_status transaction_status
)
RETURNS UUID
SECURITY DEFINER
SET search_path = public
LANGUAGE plpgsql
```

```sql
AS $$
DECLARE
  v_transaction_id UUID;
  v_payee_id UUID;
  v_user_role TEXT;
BEGIN
  SELECT role::TEXT INTO v_user_role
  FROM organization_members
  WHERE organization_id = p_organization_id
    AND user_id = auth.uid();

  IF v_user_role NOT IN ('owner','admin','manager','employee') THEN
    RAISE EXCEPTION 'Insufficient permissions';
  END IF;

  IF p_type = 'transfer' AND p_to_account_id IS NULL THEN
    RAISE EXCEPTION 'Transfer requires destination account';
  END IF;

  IF p_payee_name IS NOT NULL AND p_payee_name <> '' THEN
    SELECT id INTO v_payee_id
    FROM payees
    WHERE organization_id = p_organization_id
      AND LOWER(name) = LOWER(p_payee_name)
    LIMIT 1;

    IF v_payee_id IS NULL THEN
      INSERT INTO payees (organization_id, name)
      VALUES (p_organization_id, p_payee_name)
      RETURNING id INTO v_payee_id;
    END IF;
  END IF;

  INSERT INTO transactions (
    organization_id, type, status, account_id, to_account_id,
    category_id, payee_id, amount, date, description,
    reference_number, notes, tags, created_by
  )
  VALUES (
    p_organization_id, p_type, COALESCE(p_status, 'pending'),
    p_account_id, p_to_account_id, p_category_id, v_payee_id,
    p_amount, p_date, p_description, p_reference_number,
    p_notes, p_tags, auth.uid()
  )
  RETURNING id INTO v_transaction_id;

  RETURN v_transaction_id;
END;
$$;
```

Listing 12: Transaction Creation with Auto-Payee

Function 2: Calculate Budget Spending

```sql
CREATE OR REPLACE FUNCTION public.calculate_budget_spending(
  p_budget_id UUID
)
RETURNS DECIMAL(15,2)
SECURITY DEFINER
SET search_path = public
LANGUAGE plpgsql
AS $$
DECLARE
  v_budget RECORD;
  v_spent DECIMAL(15,2);
BEGIN
  SELECT * INTO v_budget
  FROM budgets
  WHERE id = p_budget_id;

  IF NOT FOUND THEN
    RAISE EXCEPTION 'Budget not found';
  END IF;

  IF v_budget.category_id IS NOT NULL THEN
    SELECT COALESCE(SUM(amount),0)
    INTO v_spent
    FROM transactions
    WHERE organization_id = v_budget.organization_id
      AND category_id = v_budget.category_id
      AND type = 'expense'
      AND status IN ('cleared','reconciled')
      AND date BETWEEN v_budget.start_date AND v_budget.end_date;
  ELSIF v_budget.account_id IS NOT NULL THEN
    SELECT COALESCE(SUM(amount),0)
    INTO v_spent
    FROM transactions
    WHERE organization_id = v_budget.organization_id
      AND account_id = v_budget.account_id
      AND type = 'expense'
      AND status IN ('cleared','reconciled')
      AND date BETWEEN v_budget.start_date AND v_budget.end_date;
  END IF;

  UPDATE budgets
  SET spent = v_spent, updated_at = NOW()
  WHERE id = p_budget_id;

  RETURN v_spent;
END;
$$;
```

Listing 13: Budget Spending Calculation

Function 3: Get Dashboard Statistics

```sql
CREATE OR REPLACE FUNCTION public.get_dashboard_stats(
  p_organization_id UUID
)
RETURNS TABLE (
  total_balance DECIMAL(15,2),
  monthly_income DECIMAL(15,2),
  monthly_expenses DECIMAL(15,2),
  outstanding_invoices DECIMAL(15,2),
  pending_invoice_count BIGINT,
  income_trend DECIMAL(5,2),
  expense_trend DECIMAL(5,2)
)
SECURITY DEFINER
SET search_path = public
LANGUAGE plpgsql
AS $$
BEGIN
  IF NOT EXISTS (
    SELECT 1 FROM organization_members
    WHERE organization_id = p_organization_id
      AND user_id = auth.uid()
  ) THEN
    RAISE EXCEPTION 'Not authorized';
  END IF;

  RETURN QUERY
  WITH current_month AS (
    SELECT
      COALESCE(SUM(CASE WHEN type = 'income'  THEN amount ELSE 0 END
         ),0) AS income,
      COALESCE(SUM(CASE WHEN type = 'expense' THEN amount ELSE 0 END
         ),0) AS expenses
    FROM transactions
    WHERE organization_id = p_organization_id
      AND date >= DATE_TRUNC('month', CURRENT_DATE)
      AND status IN ('cleared','reconciled')
  ),
  last_month AS (
    SELECT
      COALESCE(SUM(CASE WHEN type = 'income'  THEN amount ELSE 0 END
         ),0) AS income,
      COALESCE(SUM(CASE WHEN type = 'expense' THEN amount ELSE 0 END
         ),0) AS expenses
    FROM transactions
    WHERE organization_id = p_organization_id
      AND date >= DATE_TRUNC('month', CURRENT_DATE - INTERVAL '1
         month')
      AND date <  DATE_TRUNC('month', CURRENT_DATE)
      AND status IN ('cleared','reconciled')
  ),
  account_totals AS (
    SELECT COALESCE(SUM(current_balance),0) AS total
    FROM accounts
    WHERE organization_id = p_organization_id
      AND is_active = true
  ),
  invoice_totals AS (
    SELECT
      COALESCE(SUM(amount_due),0) AS outstanding,
      COUNT(*) AS count
    FROM invoices
    WHERE organization_id = p_organization_id
      AND status NOT IN ('paid','cancelled')
  )
  SELECT
    at.total,
    cm.income,
    cm.expenses,
    it.outstanding,
    it.count,
    CASE WHEN lm.income   > 0 THEN ROUND((((cm.income   - lm.income
       ) / lm.income  * 100)::numeric, 2) ELSE 0 END,
    CASE WHEN lm.expenses > 0 THEN ROUND((((cm.expenses - lm.expenses
       ) / lm.expenses* 100)::numeric, 2) ELSE 0 END
  FROM current_month cm, last_month lm, account_totals at,
     invoice_totals it;
END;
$$;
```

Listing 14: Dashboard Statistics Function

Function 4: Get Monthly Summary for Reports

```
CREATE OR REPLACE FUNCTION public.get_monthly_summary(
  p_organization_id UUID,
  p_year INTEGER
)
RETURNS TABLE (
  month TEXT,
  month_number INTEGER,
  total_income DECIMAL(15,2),
  total_expenses DECIMAL(15,2),
  net_income DECIMAL(15,2),
  transaction_count BIGINT
)
SECURITY DEFINER
SET search_path = public
LANGUAGE plpgsql
AS $$
BEGIN
  RETURN QUERY
  SELECT
```

```sql
    TO_CHAR(DATE_TRUNC('month', t.date), 'Mon') AS month,
    EXTRACT(MONTH FROM t.date)::INTEGER AS month_number,
    COALESCE(SUM(CASE WHEN t.type = 'income'  THEN t.amount ELSE 0
       END),0),
    COALESCE(SUM(CASE WHEN t.type = 'expense' THEN t.amount ELSE 0
       END),0),
    COALESCE(SUM(CASE WHEN t.type = 'income'  THEN t.amount ELSE -t.
       amount END),0),
    COUNT(*)::BIGINT
  FROM transactions t
  WHERE t.organization_id = p_organization_id
    AND EXTRACT(YEAR FROM t.date) = p_year
    AND t.status IN ('cleared','reconciled')
  GROUP BY DATE_TRUNC('month', t.date), EXTRACT(MONTH FROM t.date)
  ORDER BY month_number;
END;
$$;
```

Listing 15: Monthly Financial Summary

Database Views

View 1: Transaction Details

```sql
CREATE OR REPLACE VIEW transaction_details AS
SELECT
  t.id,
  t.organization_id,
  o.name AS organization_name,
  t.type,
  t.status,
  t.amount,
  t.date,
  t.description,
  a.name AS account_name,
  a.type AS account_type,
  ta.name AS to_account_name,
  c.name AS category_name,
  c.type AS category_type,
  p.name AS payee_name,
  t.created_at,
  t.updated_at
FROM transactions t
JOIN organizations o ON t.organization_id = o.id
JOIN accounts a      ON t.account_id      = a.id
LEFT JOIN accounts ta ON t.to_account_id  = ta.id
LEFT JOIN categories c ON t.category_id   = c.id
LEFT JOIN payees p     ON t.payee_id      = p.id;
```

Listing 16: Transaction Details View

View 2: Budget Tracking

```sql
CREATE OR REPLACE VIEW budget_tracking AS
SELECT
  b.id,
  b.organization_id,
  o.name AS organization_name,
  b.name AS budget_name,
  c.name AS category_name,
  a.name AS account_name,
  b.amount AS budget_amount,
  b.spent,
  b.amount - b.spent AS remaining,
  ROUND((b.spent / NULLIF(b.amount,0) * 100), 2) AS percent_used,
  b.period,
  b.start_date,
  b.end_date,
  b.alert_threshold,
  CASE
    WHEN (b.spent / NULLIF(b.amount,0) * 100) >= b.alert_threshold
       THEN true
    ELSE false
  END AS alert_triggered,
  b.is_active
FROM budgets b
JOIN organizations o ON b.organization_id = o.id
LEFT JOIN categories c ON b.category_id   = c.id
LEFT JOIN accounts   a ON b.account_id    = a.id;
```

Listing 17: Budget Tracking View

View 3: Invoice Summary

```sql
CREATE OR REPLACE VIEW invoice_summary AS
SELECT
  i.id,
  i.organization_id,
  o.name AS organization_name,
  i.invoice_number,
  p.name AS payee_name,
  i.status,
  i.issue_date,
  i.due_date,
  i.total,
  i.amount_paid,
  i.amount_due,
  i.currency,
  CASE
    WHEN i.due_date < CURRENT_DATE AND i.amount_due > 0 THEN true
    ELSE false
  END AS is_overdue,
  CURRENT_DATE - i.due_date AS days_overdue
FROM invoices i
```

```sql
JOIN organizations o ON i.organization_id = o.id
JOIN payees p        ON i.payee_id        = p.id;
```

Listing 18: Invoice Summary View

View 4: Monthly Summary

```sql
CREATE OR REPLACE VIEW monthly_summary AS
SELECT
  organization_id,
  DATE_TRUNC('month', date) AS month,
  SUM(CASE WHEN type = 'income'  THEN amount ELSE 0 END)  AS
     total_income,
  SUM(CASE WHEN type = 'expense' THEN amount ELSE 0 END)  AS
     total_expenses,
  SUM(CASE WHEN type = 'income'  THEN amount ELSE -amount END) AS
     net_income,
  COUNT(*) AS transaction_count
FROM transactions
WHERE status IN ('cleared','reconciled')
GROUP BY organization_id, DATE_TRUNC('month', date)
ORDER BY month DESC;
```

Listing 19: Monthly Summary View

Summary of Database Objects

- **Triggers:** 4 active triggers
- **Stored Procedures:** 15+ functions (transactions, budgets, invoices, reporting, organization management)
- **Views:** 5 materialized/regular views (transaction_details, budget_tracking, invoice_summary, monthly_summary, account_balances)

5.3 Seed Data

Overview. This section provides a comprehensive seed script to bootstrap a demo organization with realistic categories, accounts, payees, transactions, budgets, and invoices. It is designed to support dashboards, reports, and trigger-driven calculations out of the box.

Categories

Defines standardized income and expense categories with icons and colors for consistent UI presentation across reports and transaction entry forms. Categories cover common business operations (e.g., revenue streams, payroll, SaaS, marketing, rent), enabling granular analysis and budget assignment.

```sql
-- Set the organization ID
DO $$
DECLARE
    v_org_id UUID := '1129c18e-695f-43ae-b2e0-dba3f7b5eabe';
BEGIN


-- 1. CATEGORIES

-- Income categories represent revenue sources; expense categories track operational costs.
-- Icons and colors are for frontend tagging and quick visual identification.

-- Income Categories
INSERT INTO categories (organization_id, name, type, icon, color, is_system) VALUES
(v_org_id, 'Sales Revenue', 'income', '     ', 'bg-green-500', false),
(v_org_id, 'Consulting Services', 'income', '     ', 'bg-blue-500', false),
(v_org_id, 'Product Sales', 'income', '       ', 'bg-purple-500', false),
(v_org_id, 'License Fees', 'income', '     ', 'bg-indigo-500', false),
(v_org_id, 'Interest Income', 'income', '     ', 'bg-green-600', false);

-- Expense Categories
INSERT INTO categories (organization_id, name, type, icon, color, is_system) VALUES
(v_org_id, 'Salaries & Wages', 'expense', '     ', 'bg-red-500', false),
(v_org_id, 'Office Rent', 'expense', '     ', 'bg-orange-500', false),
(v_org_id, 'Marketing & Advertising', 'expense', '     ', 'bg-pink-500', false),
(v_org_id, 'Software & Subscriptions', 'expense', '     ', 'bg-blue-600', false),
(v_org_id, 'Office Supplies', 'expense', '     ', 'bg-yellow-500', false),
(v_org_id, 'Utilities', 'expense', '  ', 'bg-orange-600', false),
(v_org_id, 'Travel & Entertainment', 'expense', '     ', 'bg-purple-600', false),
(v_org_id, 'Professional Services', 'expense', '     ', 'bg-indigo-600', false),
(v_org_id, 'Equipment & Machinery', 'expense', '     ', 'bg-gray-600', false),
(v_org_id, 'Insurance', 'expense', '       ', 'bg-blue-700', false);
```

Listing 20: Seed Data: Categories

Accounts

Creates a realistic mix of accounts (checking, savings, credit card, cash) with initial balances and metadata (institution, account number, color). This supports transfers, reconciliation, and balance snapshots for the dashboard and reports.

```sql

-- 2. ACCOUNTS

-- Accounts include liquidity (checking), reserves (savings), liabilities (credit card),
-- and petty cash for incidental expenses. Currency and institution fields help UI and audits.

INSERT INTO accounts (organization_id, name, type, currency, initial_balance, current_balance, account_number, institution, color) VALUES
(v_org_id, 'Business Checking', 'checking', 'EUR', 50000.00, 50000.00, '1234', 'Deutsche Bank', 'bg-blue-500'),
(v_org_id, 'Business Savings', 'savings', 'EUR', 100000.00, 100000.00, '5678', 'Deutsche Bank', 'bg-green-500'),
(v_org_id, 'Corporate Credit Card', 'credit_card', 'EUR', 0.00, 0.00, '9012', 'Visa Corporate', 'bg-purple-500'),
(v_org_id, 'Petty Cash', 'cash', 'EUR', 2000.00, 2000.00, NULL, NULL, 'bg-orange-500');
```

Listing 21: Seed Data: Accounts

Payees

Seeds both clients (income sources) and vendors (expense targets) with basic contact data. This enables invoice generation, transaction attribution, and vendor/client analytics.

```sql

-- 3. PAYEES (Clients & Vendors)

-- Clients drive income categories; vendors are associated with expense categories.
-- Contact fields are optional and helpful for invoicing workflows.

-- Clients (for income)
INSERT INTO payees (organization_id, name, email, phone, address, notes) VALUES
(v_org_id, 'TechVision GmbH', 'contact@techvision.de', '+49 30 12345678', 'Alexanderplatz 1, 10178 Berlin', 'Major client - Tech consulting'),
(v_org_id, 'Global Solutions AG', 'info@globalsolutions.ch', '+41 44 1234567', 'Bahnhofstrasse 50, 8001 Z rich', 'Swiss enterprise client'),
(v_org_id, 'Innovate Ltd', 'sales@innovate.co.uk', '+44 20 7123 4567', '10 Downing Street, London SW1A', 'UK-based software client'),
(v_org_id, 'EuroTech Industries', 'contact@eurotech.fr', '+33 1 23 45 67 89', '15 Avenue des Champs- lyses , Paris', 'French manufacturing client'),
(v_org_id, 'Nordic Enterprises', 'info@nordic.se', '+46 8 123 456', 'Drottninggatan 20, Stockholm', 'Scandinavian partner');

-- Vendors (for expenses)
INSERT INTO payees (organization_id, name, email, phone, address, notes) VALUES
(v_org_id, 'AWS Europe', 'billing@aws-emea.com', '+1 206 266 1000', 'Amazon Web Services EMEA', 'Cloud hosting provider'),
(v_org_id, 'Microsoft 365', 'billing@microsoft.com', NULL, NULL, 'Software subscriptions'),
(v_org_id, 'WeWork Berlin', 'berlin@wework.com', '+49 30 98765432', 'Friedrichstra e 76, Berlin', 'Co-working space'),
(v_org_id, 'Vodafone Business', 'business@vodafone.de', '+49 800 172 1212', NULL, 'Telecommunications'),
(v_org_id, 'Metro Office Supplies', 'orders@metro-office.de', '+49 30 55554444', NULL, 'Office supplies vendor'),
(v_org_id, 'LinkedIn Ads', 'billing@linkedin.com', NULL, NULL, 'Marketing platform'),
(v_org_id, 'Google Workspace', 'billing@google. com', NULL, NULL, 'Email and productivity'),
(v_org_id, 'Allianz Insurance', 'business@allianz.de', '+49 89 38000', NULL, 'Business insurance provider');

END $$;
```

Listing 22: Seed Data: Payees

Transactions

Populates a full-year ledger with realistic monthly income and expense activity across accounts and categories. Cleared transactions update balances via triggers, while pending entries demonstrate workflow status without affecting budgets or balances.

```sql

-- 4. TRANSACTIONS (Full Year - 2024)

-- Includes monthly income/expense activity; uses cleared vs pending statuses.
-- Demonstrates category coverage and multi-account posting for analytics.

DO $$
```

```
DECLARE
    v_org_id UUID := '1129c18e-695f-43ae-b2e0-dba3f7b5eabe';
    -- (variables omitted for brevity; see full script above)
BEGIN
    -- (full transaction inserts for  J a n Dec  as provided)
END $$;
```

Listing 23: Seed Data: Transactions (Full Year - 2024)

Budgets

Adds monthly, quarterly, and yearly budgets tied to categories and validates trigger-driven spending updates. This shows alert thresholds and real usage percentages for dashboard visualizations.

```

-- 5. BUDGETS

-- Demonstrates period variability (monthly, quarterly, yearly) and
   auto-spent recalculation.

DO $$
DECLARE
    v_org_id UUID := '1129c18e-695f-43ae-b2e0-dba3f7b5eabe';
    -- (variables omitted; see full script above)
BEGIN
    -- (budget inserts and calculate_budget_spending calls)
END $$;
```

Listing 24: Seed Data: Budgets

Invoices

Creates paid, sent, viewed, and draft invoices with line items and tax/discount calculations. This enables invoice status workflows, overdue detection (via views), and invoice-to-transaction relationships.

```

-- 6. INVOICES

-- Includes multiple statuses and detailed line items for reporting
   and totals triggers.

DO $$
DECLARE
    v_org_id UUID := '1129c18e-695f-43ae-b2e0-dba3f7b5eabe';
    -- (variables omitted; see full script above)
BEGIN
    -- (invoice inserts and invoice_items inserts)
```

```sql
END $$;

-- Update account balances based on all cleared transactions
UPDATE accounts
SET current_balance = initial_balance + (
    SELECT COALESCE(SUM(
        CASE
            WHEN t.type = 'income' AND t.account_id = accounts.id
               THEN t.amount
            WHEN t.type = 'expense' AND t.account_id = accounts.id
               THEN -t.amount
            WHEN t.type = 'transfer' AND t.account_id = accounts.id
               THEN -t.amount
            WHEN t.type = 'transfer' AND t.to_account_id = accounts.
               id THEN t.amount
            ELSE 0
        END
    ), 0)
    FROM transactions t
    WHERE (t.account_id = accounts.id OR t.to_account_id = accounts.
       id)
      AND t.status IN ('cleared', 'reconciled')
)
WHERE organization_id = '1129c18e-695f-43ae-b2e0-dba3f7b5eabe';
```

Listing 25: Seed Data: Invoices

6 Application Flow

6.1 User Flow Diagram

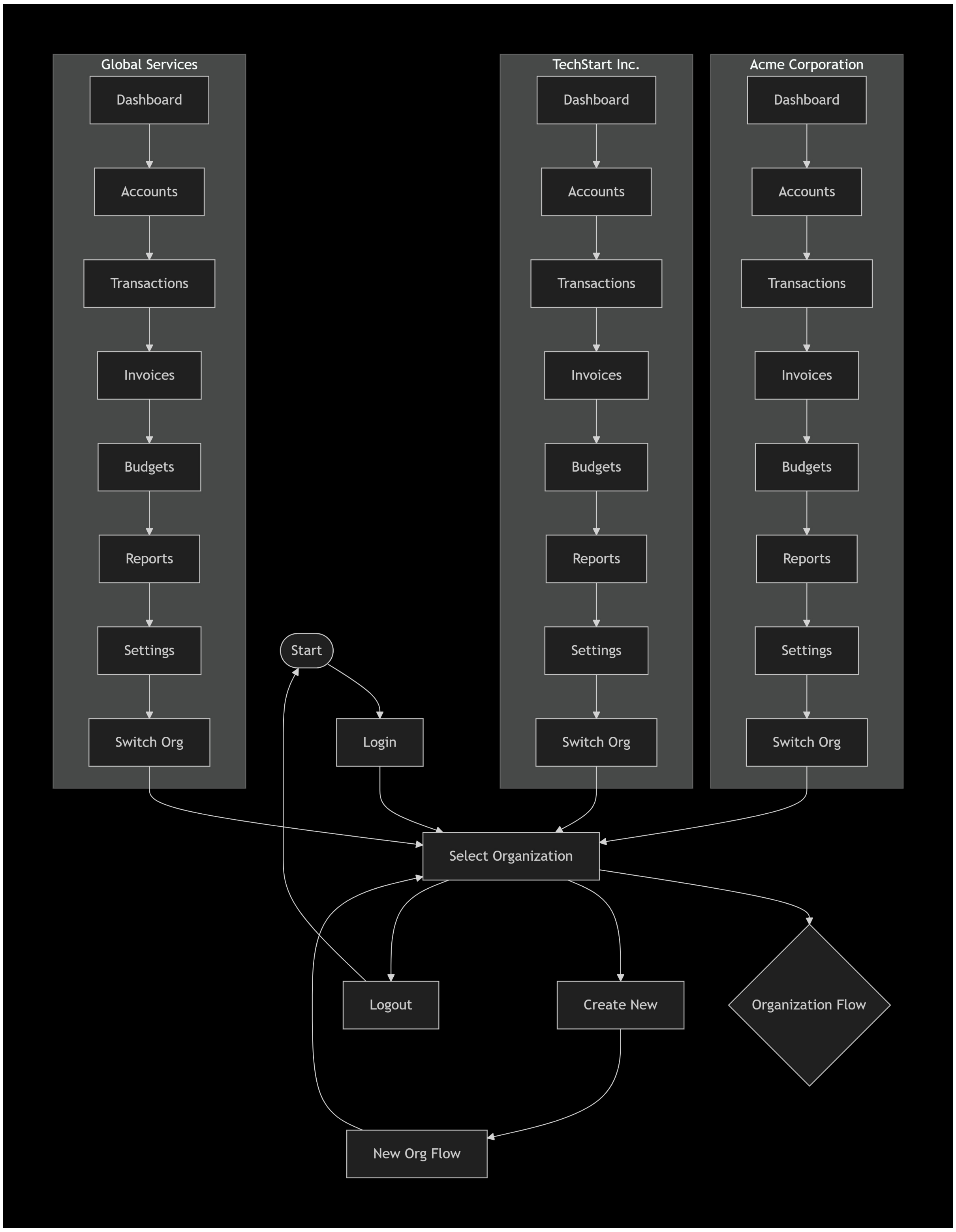


Figure 3: User Flow Diagram

6.2 Wireframes

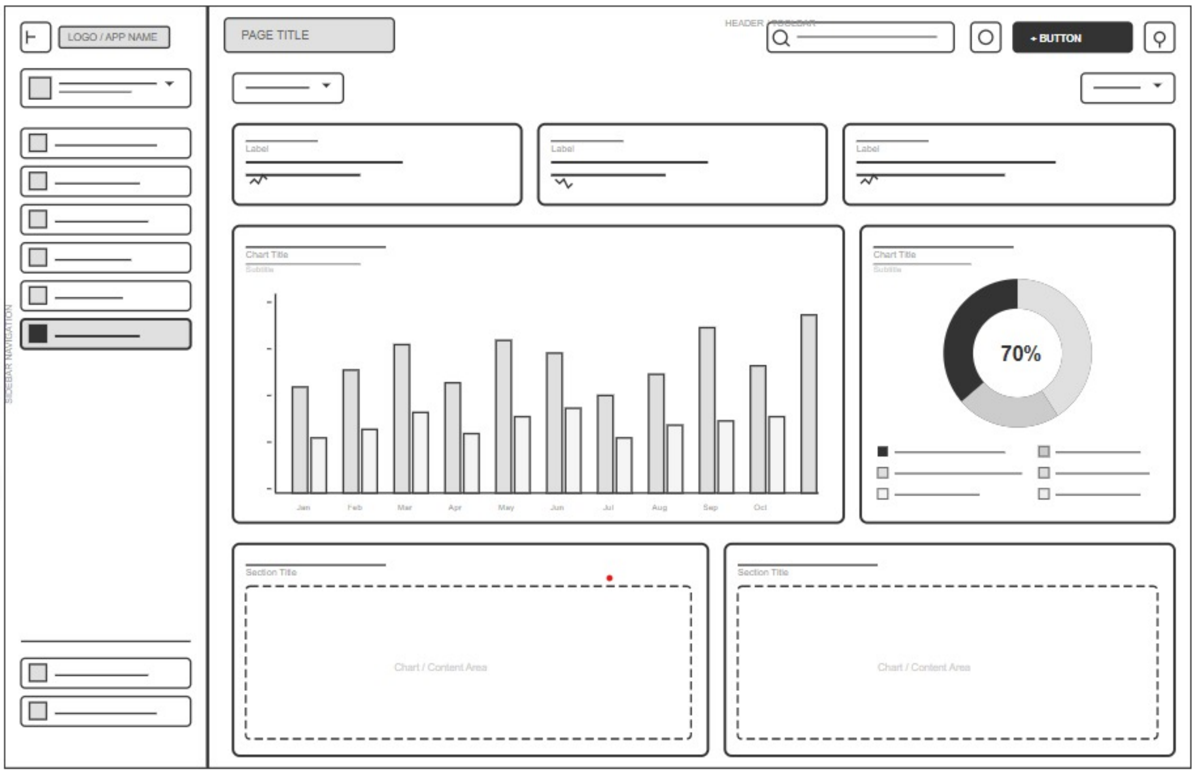


Figure 4: Wireframe

orts

7 Page-by-Page Navigation and SQL Queries

This section documents each application screen, its purpose, how users interact with it, and the SQL queries used to fetch or process data for that page.

7.1 Dashboard

Purpose: The Dashboard provides a high-level overview of the organization's financial health. It summarizes account balances, recent transactions, budget utilization, and statistical indicators.

```sql
SELECT * FROM get_organization_summary('org-uuid');
```

Listing 26: Organization Summary

Explanation: Fetches total revenue, expenses, net position, upcoming invoices, and trends.

```sql
SELECT id, name, type, current_balance, currency
FROM accounts
WHERE organization_id = $1 AND is_active = TRUE
ORDER BY current_balance DESC;
```

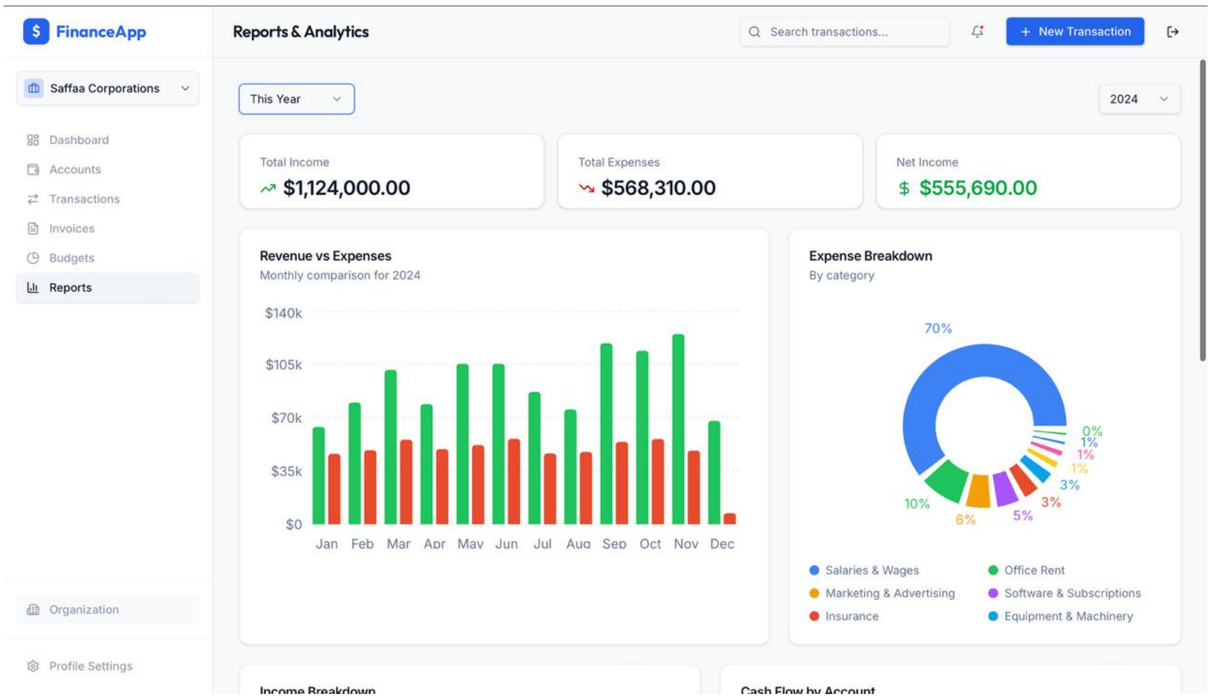
Listing 27: Account Balances

Explanation: Returns a list of active accounts sorted by balance.

```sql
SELECT t.*, a.name AS account_name, c.name AS category_name
FROM transactions t
LEFT JOIN accounts a ON t.account_id = a.id
LEFT JOIN categories c ON t.category_id = c.id
WHERE t.organization_id = $1 AND t.status IN ('cleared','reconciled')
ORDER BY t.date DESC, t.created_at DESC
LIMIT 10;
```

Listing 28: Recent Transactions

Explanation: Shows recent finalized transactions.



Total Income $1,124,000.00
Total Expenses $568,310.00
Net Income $ $555,690.00

Figure 5: Dashboard

7.2 Accounts

Purpose: The Accounts screen allows organizations to manage multiple financial accounts such as checking, savings, cash, credit card, or digital wallets. Users can:

- View all accounts and their balances - Create new accounts - Edit or deactivate existing accounts - Filter by account type (asset, liability, equity, etc.)

```sql
SELECT id, name, type, current_balance, currency, is_active
FROM accounts
WHERE organization_id = $1
ORDER BY type, name;
```

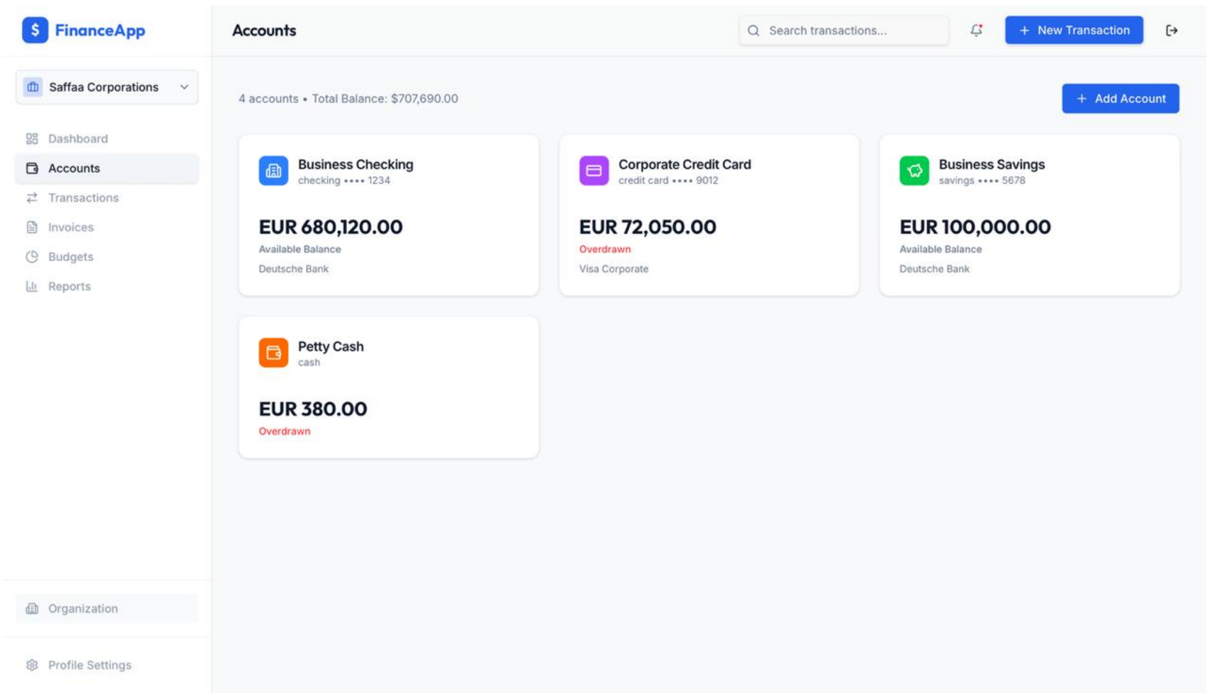
Listing 29: Accounts List

Explanation: Fetches every account created by the organization, grouped by type and alphabetically for easier navigation.

```sql
SELECT a.*,
       COALESCE(SUM(t.amount),0) AS total_transactions
FROM accounts a
LEFT JOIN transactions t ON t.account_id = a.id
WHERE a.organization_id = $1 AND a.id = $2
GROUP BY a.id;
```

Listing 30: Single Account Details

Explanation: Loads account details such as name, currency, type, and aggregates total transaction amount for quick reference.



4 accounts • Total Balance: $707,690.00

Business Checking
checking •••• 1234
EUR 680,120.00
Available Balance
Deutsche Bank

Corporate Credit Card
credit card •••• 9012
EUR 72,050.00
Overdrawn
Visa Corporate

Business Savings
savings •••• 5678
EUR 100,000.00
Available Balance
Deutsche Bank

Petty Cash
cash
EUR 380.00
Overdrawn

Figure 6: Accounts Page

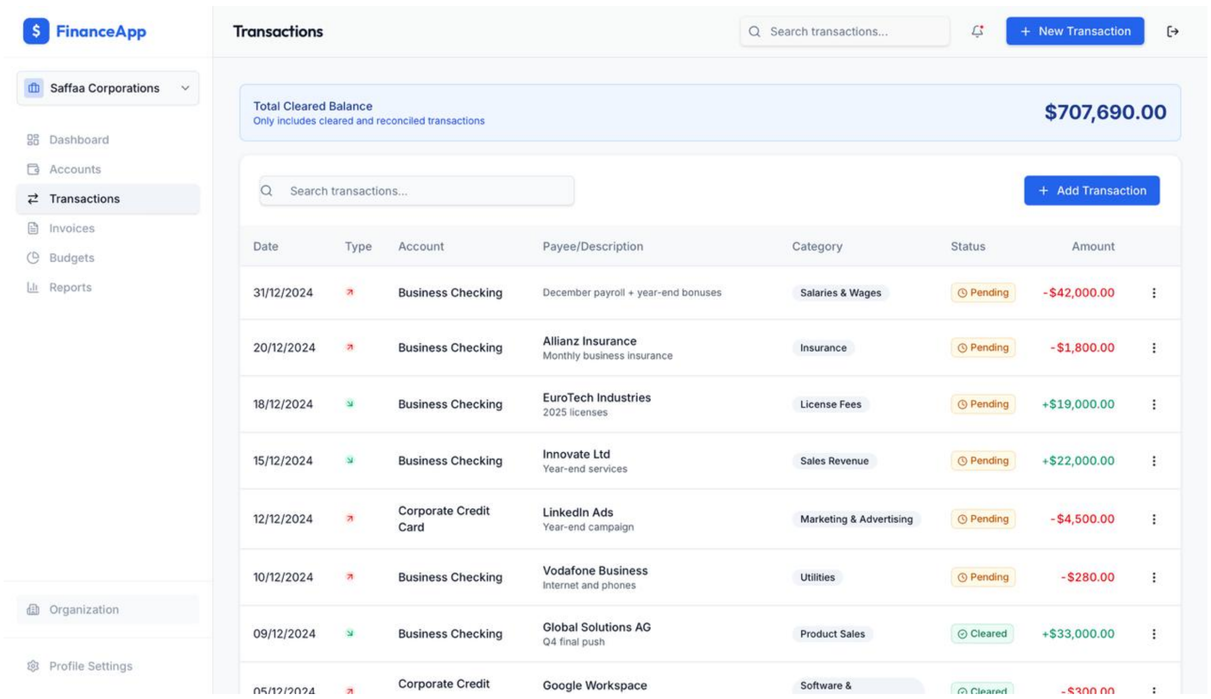
7.3 Transactions List

Purpose: Displays all transactions with filters (account, type, date range). Allows searching, pagination, and navigation to transaction details.

```sql
SELECT t.*, a.name AS account_name, c.name AS category_name
FROM transactions t
LEFT JOIN accounts a ON t.account_id = a.id
LEFT JOIN categories c ON t.category_id = c.id
WHERE t.organization_id = $1
  AND ($2::uuid IS NULL OR t.account_id = $2)
  AND ($3::transaction_type IS NULL OR t.type = $3)
  AND ($4::date IS NULL OR t.date >= $4)
  AND ($5::date IS NULL OR t.date <= $5)
ORDER BY t.date DESC, t.created_at DESC
LIMIT $6 OFFSET $7;
```

Listing 31: Transactions List Query

Explanation: Supports filtering and pagination for efficient browsing.



FinanceApp

Saffaa Corporations

- Dashboard
- Accounts
- Transactions
- Invoices
- Budgets
- Reports

Organization

Profile Settings

Transactions

Search transactions...

+ New Transaction

Total Cleared Balance
Only includes cleared and reconciled transactions

$707,690.00

Search transactions...

+ Add Transaction

Date	Type	Account	Payee/Description	Category	Status	Amount	
31/12/2024	↗	Business Checking	December payroll + year-end bonuses	Salaries & Wages	Pending	-$42,000.00	⋮
20/12/2024	↗	Business Checking	Allianz Insurance Monthly business insurance	Insurance	Pending	-$1,800.00	⋮
18/12/2024	↘	Business Checking	EuroTech Industries 2025 licenses	License Fees	Pending	+$19,000.00	⋮
15/12/2024	↘	Business Checking	Innovate Ltd Year-end services	Sales Revenue	Pending	+$22,000.00	⋮
12/12/2024	↗	Corporate Credit Card	LinkedIn Ads Year-end campaign	Marketing & Advertising	Pending	-$4,500.00	⋮
10/12/2024	↗	Business Checking	Vodafone Business Internet and phones	Utilities	Pending	-$280.00	⋮
09/12/2024	↘	Business Checking	Global Solutions AG Q4 final push	Product Sales	Cleared	+$33,000.00	⋮
05/12/2024	↗	Corporate Credit	Google Workspace	Software &	Cleared	-$300.00	⋮

Figure 7: Transaction Page

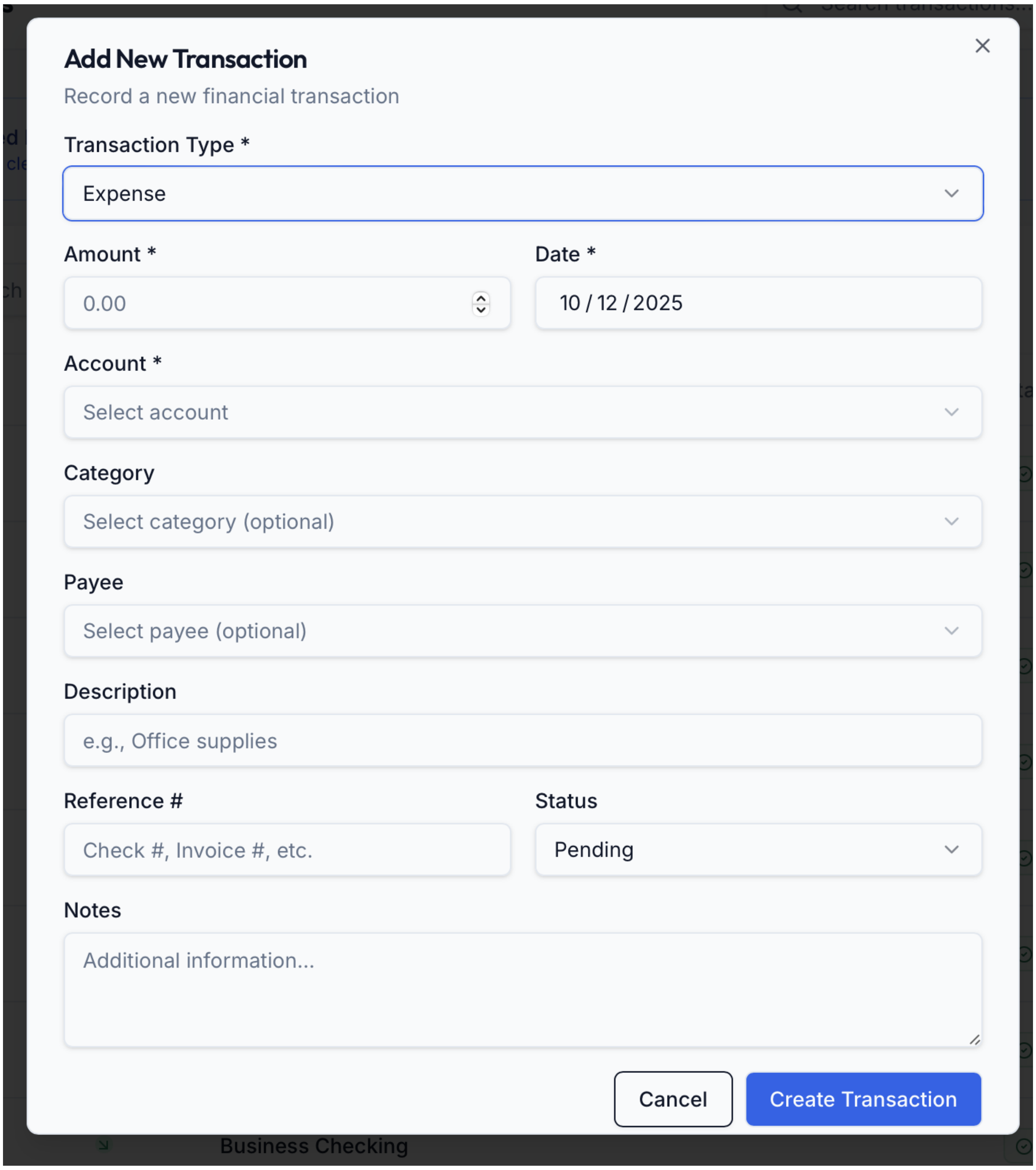
7.4 Transaction Form (Create/Edit)

Purpose: Allows users to create or edit transactions by selecting accounts, categories, and entering details.

```sql
SELECT id, name, type, current_balance
FROM accounts
WHERE organization_id = $1 AND is_active = TRUE
ORDER BY name;
```

Listing 32: Transaction Form Accounts

Explanation: Loads accounts for the form dropdown.


Add New Transaction

Record a new financial transaction

Transaction Type *

Expense

Amount *

0.00

Date *

10 / 12 / 2025

Account *

Select account

Category

Select category (optional)

Payee

Select payee (optional)

Description

e.g., Office supplies

Reference #

Check #, Invoice #, etc.

Status

Pending

Notes

Additional information...

Figure 8: Transaction Form

7.5 Invoices

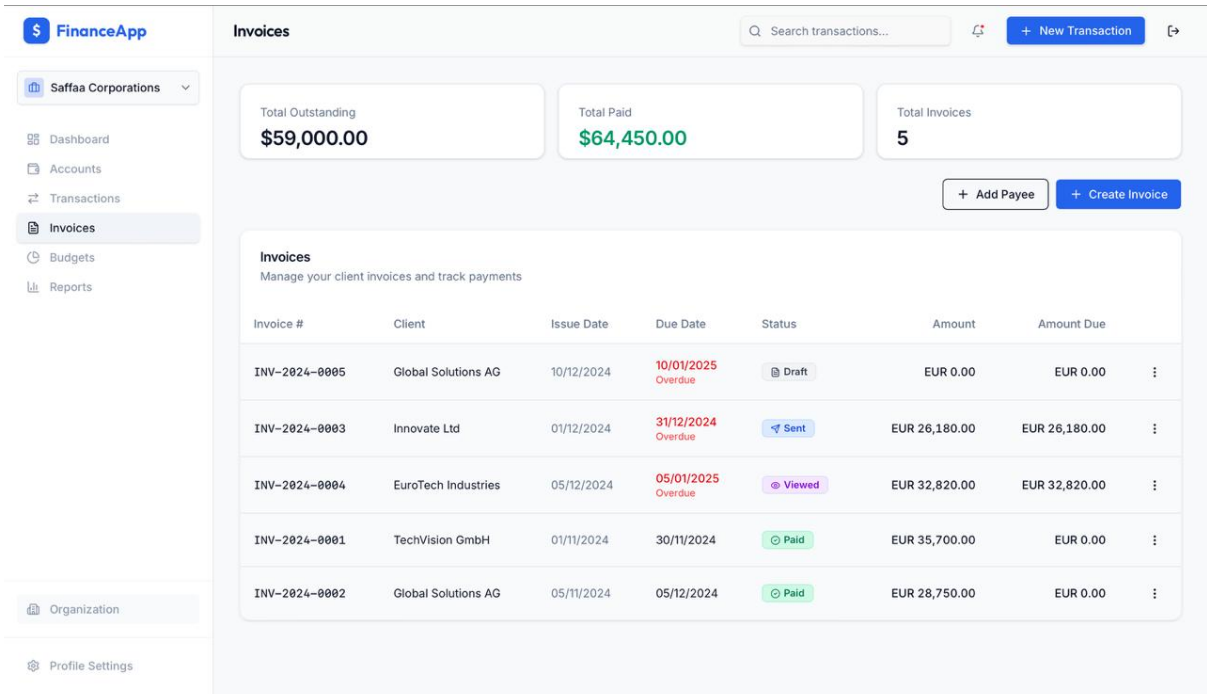
Purpose: The Invoices screen allows management of billing, outstanding payments, and invoice details.

```sql
SELECT i.*, p.name AS payee_name, COUNT(ii.id) AS item_count
FROM invoices i
```

```sql
LEFT JOIN payees p        ON i.payee_id  = p.id
LEFT JOIN invoice_items ii ON ii.invoice_id = i.id
WHERE i.organization_id = $1
GROUP BY i.id, p.name
ORDER BY i.issue_date DESC;
```

Listing 33: Invoices List

Explanation: Shows invoices with payee and item counts.



Total Outstanding
$59,000.00

Total Paid
$64,450.00

Total Invoices
5

Invoices
Manage your client invoices and track payments

Invoice #	Client	Issue Date	Due Date	Status	Amount	Amount Due
INV-2024-0005	Global Solutions AG	10/12/2024	10/01/2025 Overdue	Draft	EUR 0.00	EUR 0.00
INV-2024-0003	Innovate Ltd	01/12/2024	31/12/2024 Overdue	Sent	EUR 26,180.00	EUR 26,180.00
INV-2024-0004	EuroTech Industries	05/12/2024	05/01/2025 Overdue	Viewed	EUR 32,820.00	EUR 32,820.00
INV-2024-0001	TechVision GmbH	01/11/2024	30/11/2024	Paid	EUR 35,700.00	EUR 0.00
INV-2024-0002	Global Solutions AG	05/11/2024	05/12/2024	Paid	EUR 28,750.00	EUR 0.00

Figure 9: Invoices Page

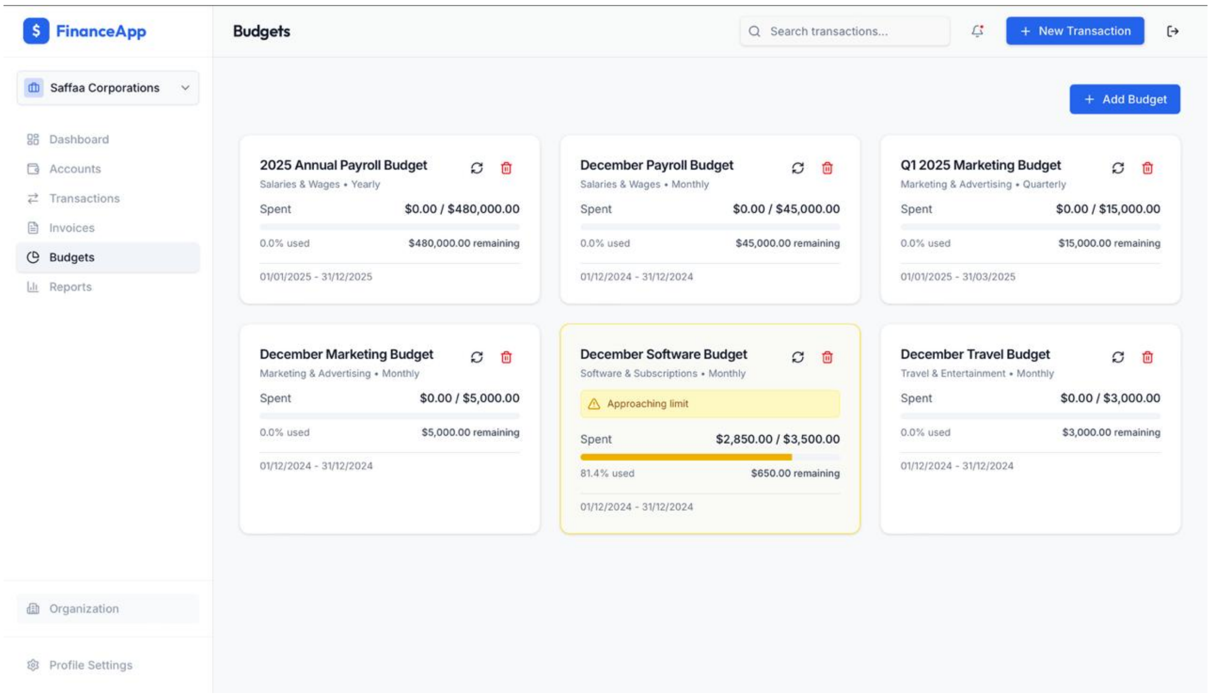
7.6 Budgets

Purpose: Tracks spending against predefined budgets and calculates usage.

```sql
SELECT b.*, c.name AS category_name,
       (b.spent / NULLIF(b.amount,0) * 100) AS percent_used,
       (b.amount - b.spent) AS remaining
FROM budgets b
JOIN categories c ON b.category_id = c.id
WHERE b.organization_id = $1;
```

Listing 34: Budgets Overview

Explanation: Shows percent spent and remaining budget.



2025 Annual Payroll Budget
Salaries & Wages • Yearly
Spent $0.00 / $480,000.00
0.0% used $480,000.00 remaining
01/01/2025 - 31/12/2025

December Payroll Budget
Salaries & Wages • Monthly
Spent $0.00 / $45,000.00
0.0% used $45,000.00 remaining
01/12/2024 - 31/12/2024

Q1 2025 Marketing Budget
Marketing & Advertising • Quarterly
Spent $0.00 / $15,000.00
0.0% used $15,000.00 remaining
01/01/2025 - 31/03/2025

December Marketing Budget
Marketing & Advertising • Monthly
Spent $0.00 / $5,000.00
0.0% used $5,000.00 remaining
01/12/2024 - 31/12/2024

December Software Budget
Software & Subscriptions • Monthly
Approaching limit
Spent $2,850.00 / $3,500.00
81.4% used $650.00 remaining
01/12/2024 - 31/12/2024

December Travel Budget
Travel & Entertainment • Monthly
Spent $0.00 / $3,000.00
0.0% used $3,000.00 remaining
01/12/2024 - 31/12/2024

Figure 10: Budgets Page

7.7 Reports & Analytics

Purpose: Displays charts and insights on financial performance over time.

```sql
SELECT DATE_TRUNC('month', date) AS month,
       SUM(CASE WHEN type = 'income'  THEN amount ELSE 0 END) AS income,
       SUM(CASE WHEN type = 'expense' THEN amount ELSE 0 END) AS expenses
FROM transactions
WHERE organization_id = $1
GROUP BY DATE_TRUNC('month', date)
ORDER BY month DESC;
```

Listing 35: Monthly Income and Expenses

Explanation: Aggregates month-wise financial trends.

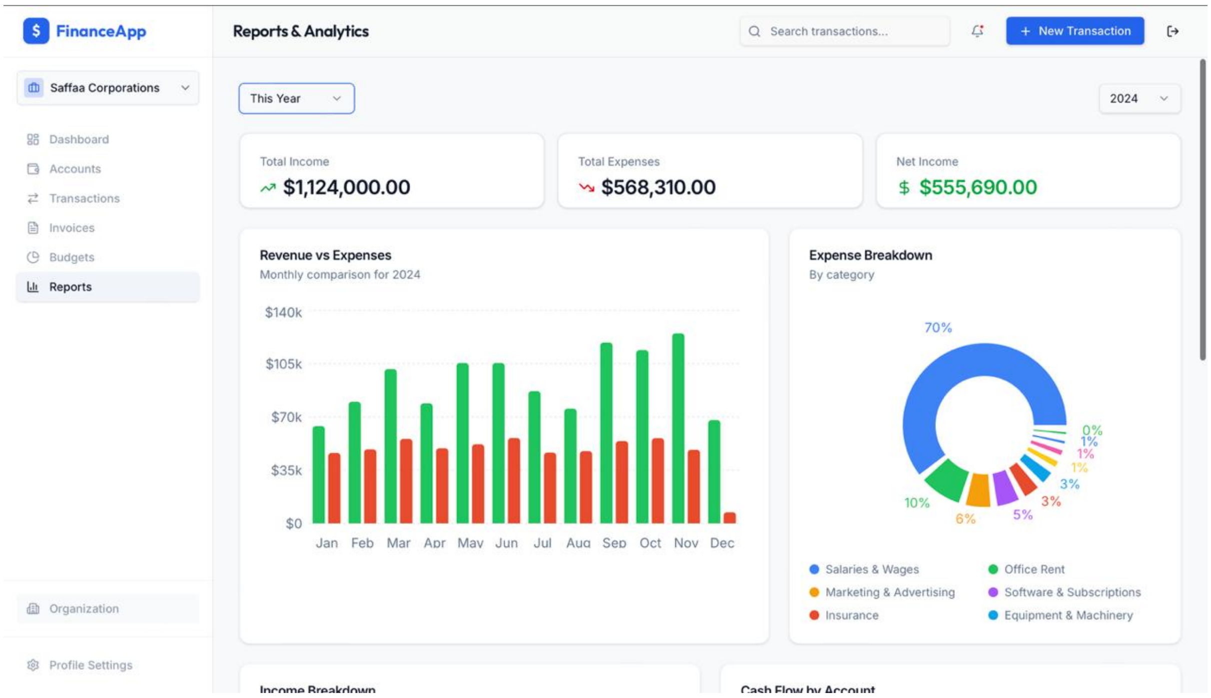


Figure 11: Reports Page

7.8 Organization Settings

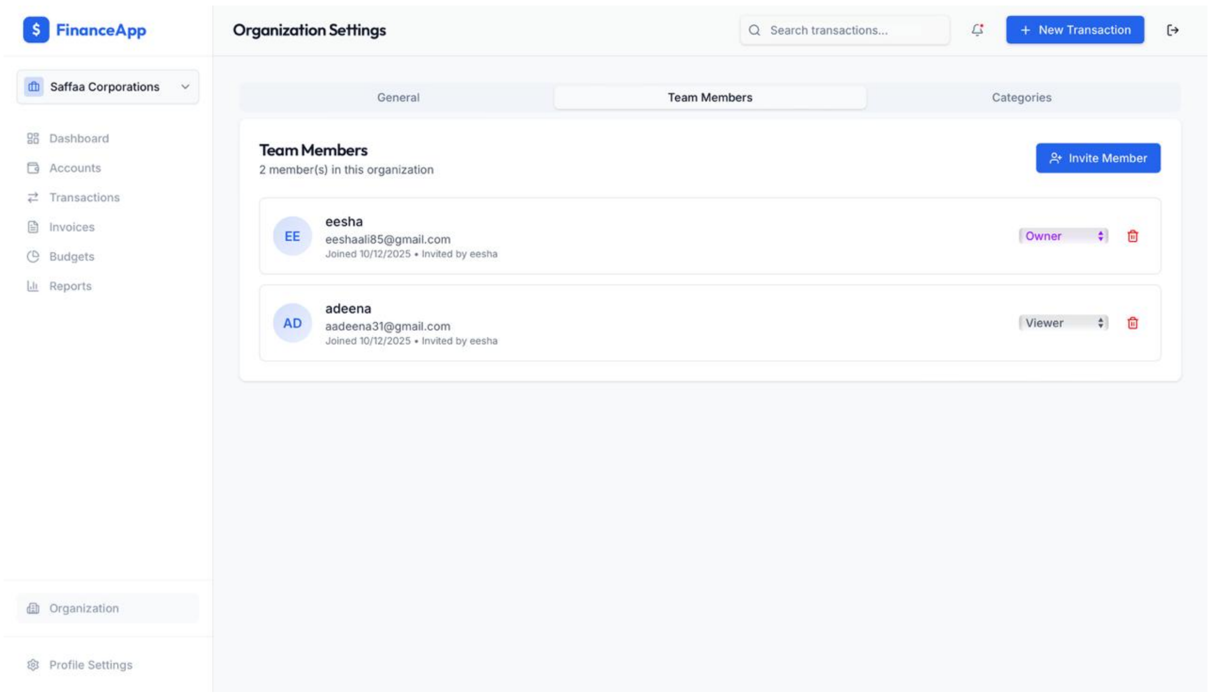
Purpose: Manage organization-wide configuration such as name, slug, currency, fiscal year start, member roles, and security settings. Supports editing metadata and viewing current members with roles for access control.

```sql
-- Fetch core organization configuration
SELECT id, name, slug, currency, fiscal_year_start, created_by
FROM organizations
WHERE id = $1;

-- List members and roles for the organization
SELECT om.id, u.full_name, om.user_id, om.role, om.organization_id
FROM organization_members om
JOIN user_profiles u ON u.id = om.user_id
WHERE om.organization_id = $1
ORDER BY om.role DESC, u.full_name ASC;

-- Example: Update organization metadata
UPDATE organizations
SET name = $2,
    slug = $3,
    currency = $4,
    fiscal_year_start = $5
WHERE id = $1
```

```sql
RETURNING id, name, slug, currency, fiscal_year_start;
```

Listing 36: Get Organization Settings

Explanation: Provides read and update operations for organization metadata and role-based membership. The first query loads settings for display; the second lists members for role management; the third demonstrates updating the organization configuration safely.

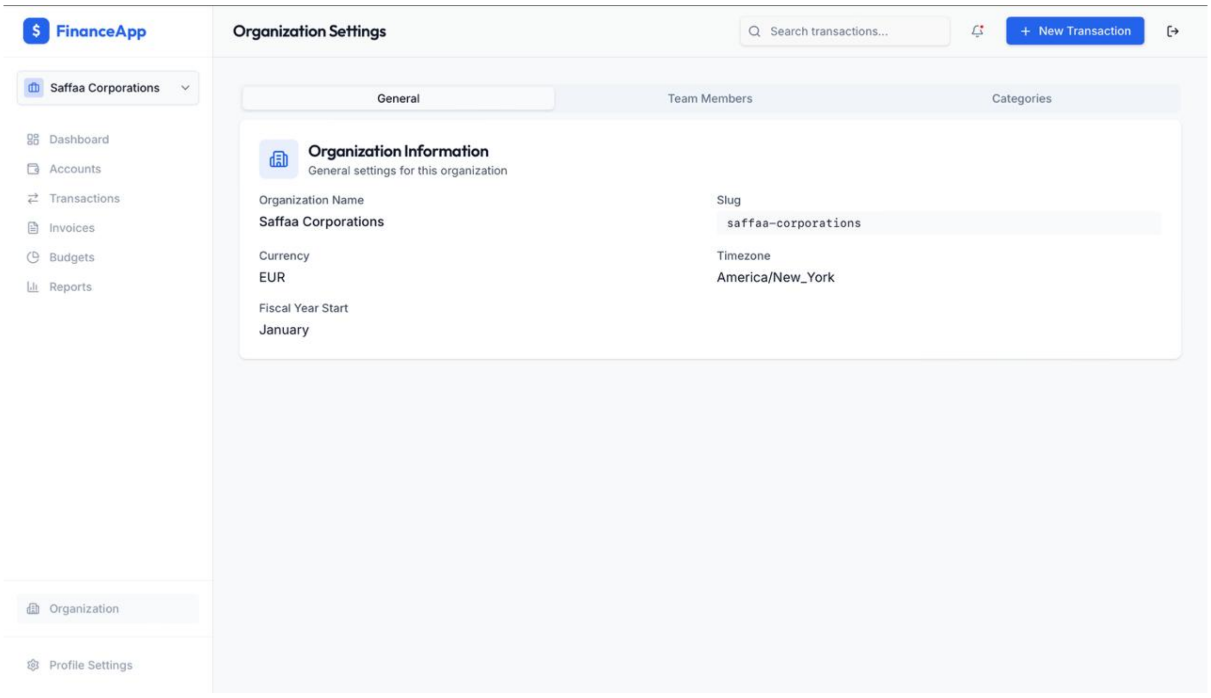


Team Members
2 member(s) in this organization

EE eesha
eeshaali85@gmail.com
Joined 10/12/2025 • Invited by eesha
Owner

AD adeena
aadeena31@gmail.com
Joined 10/12/2025 • Invited by eesha
Viewer

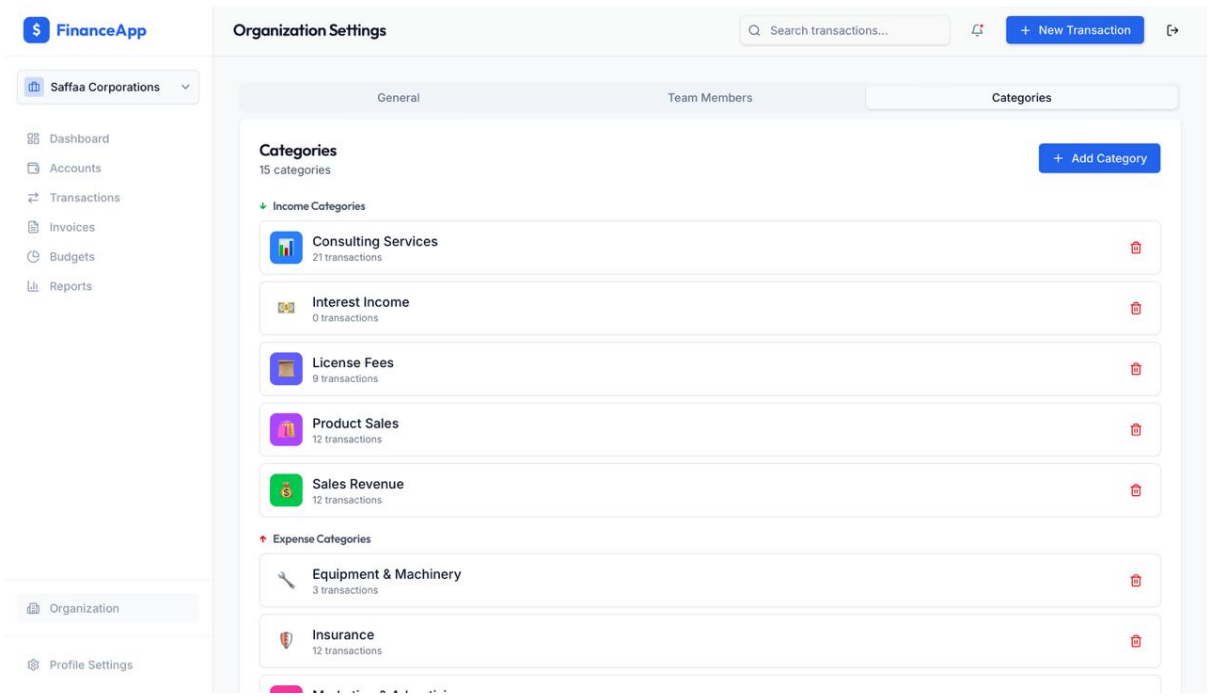
Figure 12: Organization Settings



Organization Information
General settings for this organization

Organization Name: Saffaa Corporations
Slug: `saffaa-corporations`
Currency: EUR
Timezone: America/New_York
Fiscal Year Start: January

Figure 13: Organization Settings



Categories
15 categories

Income Categories

- Consulting Services — 21 transactions
- Interest Income — 0 transactions
- License Fees — 9 transactions
- Product Sales — 12 transactions
- Sales Revenue — 12 transactions

Expense Categories

- Equipment & Machinery — 3 transactions
- Insurance — 12 transactions

Figure 14: Organization Settings

8 Work Contribution

- Adeena Arif: Backend implementation, triggers, stored procedures, authentication, database queries
- Eesha Ali: Frontend design, dashboard, transactions and invoices pages, wireframes, report, database queries