Bank Account Management Implementation

Overview

Comprehensive bank account management system has been implemented in the My Profile page, allowing users to add, edit, and manage multiple bank accounts with full international banking support.

Schema Changes

Added BankAccountUsage Enum

```
enum BankAccountUsage {
  SALARY
  GROSS
  EXPENSES
  OTHER
}
```

Updated Bank Model

All banking fields are now optional to support varying international requirements:

Account Identification

- `accountName` - Account name/label (optional)
- `accountNumber` - IBAN or local account number (optional)
- `accountHolder` - Account holder name (optional)

Bank Information

- `bankName` - Bank name (optional)
- `swiftCode` - SWIFT/BIC code (optional)
- `intermediarySwiftCode` - Intermediary SWIFT code for international transfers (optional)
- `routingNumber` - Routing number (US) (optional)
- `sortCode` - Sort code (UK) (optional)
- `branchCode` - Branch code (optional)
- `iban` - IBAN (optional)

Bank Address

- `bankAddress` - Street address (optional)
- `bankCity` - City (optional)
- `country` - Country (optional)
- `state` - State/County (optional)
- `postCode` - Post code/ZIP code (optional)

Account Details

- `currency` - Account currency (optional)
- `usage` - Account usage/purpose (BankAccountUsage enum, optional)

User Association

- `userId` - Associates bank account with user (null for company banks)

Flags

- `isPrimary` - Marks account as primary (default: false)
- `isActive` - Marks account as active (default: true)

Database Migration

IMPORTANT: After pulling these changes, you must run:

```
npx prisma migrate dev --name add_comprehensive_bank_account_fields
```

This will create the necessary database migrations for the new fields and enum.

API Endpoints

Bank Router (`server/api/routers/bank.ts`)

All endpoints enforce RBAC permissions with `bank.*.global` and `bank.*.own` scopes.

`getMyBankAccounts` Query

- **Permission**: `bank.list.own`
- **Description**: Get all bank accounts for the current user
- **Returns**: Array of bank accounts, ordered by primary status and creation date

`create` Mutation

- **Permission**: `bank.create.global` or `bank.create.own`
- **Description**: Create a new bank account
- **Input**: All bank fields (all optional)
- **Returns**: Created bank account

`update` Mutation

- **Permission**: `bank.update.global` or `bank.update.own`
- **Description**: Update existing bank account
- **Input**: Bank ID + fields to update (all optional)
- **Returns**: Updated bank account

`delete` Mutation

- **Permission**: `bank.delete.global` or `bank.delete.own`
- **Description**: Delete bank account
- **Input**: Bank ID
- **Returns**: Success status

Permission Checks

The router checks both `userId` and `createdBy` fields to determine ownership:

```
existing.userId === user.id || existing.createdBy === user.id
```

UI Components

Component Structure

```
components/profile/banks/
├── BankAccountList.tsx      - Main container, handles data fetching
├── BankAccountCard.tsx      - Individual account display
├── BankAccountForm.tsx      - Form with all fields
└── BankAccountDialog.tsx    - Modal wrapper for form
```

BankAccountList

Main component that:
- Fetches user's bank accounts using `api.bank.getMyBankAccounts`
- Displays accounts in a responsive grid (1/2/3 columns)
- Shows empty state when no accounts exist
- Handles create/update/delete operations with confirmation dialogs
- Shows loading and error states

BankAccountCard

Displays individual bank account with:
- Primary badge (star icon) if `isPrimary` is true
- Usage type badge with color coding:
- SALARY: Green
- GROSS: Blue
- EXPENSES: Orange
- OTHER: Gray
- Account holder name
- Bank name
- Masked account number/IBAN (shows first 4 and last 4 characters)
- SWIFT code
- Currency and country
- Edit and Delete action buttons

BankAccountForm

Comprehensive form with sections:
1. **Account Identification**: Account name, holder, usage, currency
2. **Account Numbers**: Account number, IBAN, routing number, sort code, branch code
3. **Bank Information**: Bank name, SWIFT codes
4. **Bank Address**: Full address fields
5. **Primary Account**: Checkbox to set as primary

All fields are optional with helpful placeholders.

BankAccountDialog

Modal dialog that wraps the form with:
- Responsive design (max-width: 4xl)
- Scrollable content for long forms
- Context-aware title (Add/Edit)
- Loading states during submission

Helper Functions

`/lib/constants/bank-usage.ts`

`BANK_USAGE_LABELS`

Human-readable labels for usage types.

`BANK_USAGE_DESCRIPTIONS`

Descriptions for each usage type.

`getBankAccountsByUsage(accounts, usage)`

Filter accounts by usage type.

```
const salaryAccounts = getBankAccountsByUsage(accounts, "SALARY");
```

`getPrimaryBankAccount(accounts)`

Get primary account or first account if no primary is set.

```
const primary = getPrimaryBankAccount(accounts);
```

`getPrimaryBankAccountByUsage(accounts, usage)`

Get primary account for specific usage type.

```
const primarySalaryAccount = getPrimaryBankAccountByUsage(accounts, "SALARY");
```

Usage in Workflows

Payroll Workflow Integration

```
import { getBankAccountsByUsage, getPrimaryBankAccountByUsage } from "@/lib/constants/
bank-usage";

// Get user's bank accounts
const accounts = await prisma.bank.findMany({
  where: { userId, tenantId },
});

// Get salary account for payroll
const salaryAccount = getPrimaryBankAccountByUsage(accounts, "SALARY");

if (salaryAccount) {
  // Use salaryAccount for payroll transfer
  await processPayroll({
    accountNumber: salaryAccount.accountNumber,
    iban: salaryAccount.iban,
    swiftCode: salaryAccount.swiftCode,
    bankName: salaryAccount.bankName,
    // ... other fields
  });
}
```

Invoice Workflow Integration

```
// Get expense account for reimbursements
const expenseAccount = getPrimaryBankAccountByUsage(accounts, "EXPENSES");

// Get gross account for gross payments
const grossAccount = getPrimaryBankAccountByUsage(accounts, "GROSS");

// Use appropriate account based on payment type
const targetAccount = paymentType === "GROSS" ? grossAccount : salaryAccount;
```

Querying by Usage

You can query banks by usage type directly in Prisma:

```
// Get all salary accounts for a user
const salaryAccounts = await prisma.bank.findMany({
  where: {
    userId,
    tenantId,
    usage: "SALARY",
    isActive: true,
  },
  orderBy: { isPrimary: "desc" },
});
```

Features

✅ Multiple Bank Accounts

- Users can add unlimited bank accounts
- Each account can have a different usage type
- Support for multiple accounts of the same type (e.g., two SALARY accounts)

✅ International Banking Support

- All fields are optional (requirements vary by country)
- Support for IBAN (Europe)
- Support for routing numbers (US)
- Support for sort codes (UK)
- Support for branch codes (various countries)
- Intermediary SWIFT codes for complex transfers

✅ Primary Account Selection

- One account can be marked as primary
- Primary accounts appear first in lists
- Primary badge with star icon in UI

✅ Usage Type System

- SALARY: For receiving salary payments
- GROSS: For receiving gross amount payments
- EXPENSES: For expense reimbursements
- OTHER: For other payment purposes

✅ RBAC Compliance

- All endpoints enforce proper permissions
- Users can only manage their own bank accounts (with `.own` permissions)
- Admins can manage all accounts (with `.global` permissions)

✅ User Experience

- Responsive design (mobile, tablet, desktop)
- Empty state with call-to-action
- Loading states during operations
- Error handling with toast notifications
- Delete confirmation dialogs
- Masked account numbers for security
- Color-coded usage type badges
- Primary account indication

Testing

Test CRUD Operations

1. **Create Account**:
   - Navigate to My Profile → Bank Accounts
   - Click “Add Bank Account”
   - Fill in any fields (all optional)
   - Select usage type
   - Submit

2. **View Accounts**:
   - Accounts display in cards
   - Primary accounts show star badge
   - Usage type shows colored badge

3. **Edit Account**:
   - Click “Edit” on any account
   - Modify fields
   - Submit

4. **Delete Account**:
   - Click “Delete” on any account
   - Confirm deletion in dialog

5. **Set Primary**:
   - Edit account
   - Check “Set as primary account”
   - Save

Test Permissions

1. Login as contractor (`.own` permissions)
2. Verify can only see own accounts
3. Login as admin (`.global` permissions)
4. Verify can see all accounts

Migration Notes

Backward Compatibility

The implementation maintains backward compatibility with legacy fields:
- `name` field (deprecated, use `bankName` or `accountName`)
- `address` field (deprecated, use `bankAddress`)

Existing data remains intact. The router checks these fields as fallbacks:

```
entityName: bank.accountName || bank.bankName || bank.name || "Bank Account"
```

Data Migration (Optional)

If you want to migrate existing data to new fields:

```
-- Migrate name to bankName
UPDATE banks SET "bankName" = name WHERE "bankName" IS NULL AND name IS NOT NULL;

-- Migrate address to bankAddress
UPDATE banks SET "bankAddress" = address WHERE "bankAddress" IS NULL AND address IS NO
T NULL;
```

Future Enhancements

Potential Improvements

- [ ] Bank logo integration
- [ ] Account verification system
- [ ] Direct bank API integration
- [ ] Real-time balance checking
- [ ] Transaction history
- [ ] Multi-currency support enhancements
- [ ] Automatic IBAN validation
- [ ] SWIFT code lookup
- [ ] Bank branch finder

Summary

The comprehensive bank account management system provides:
- ✅ Full international banking support
- ✅ Multiple accounts per user
- ✅ Usage type classification for workflows
- ✅ Primary account selection
- ✅ Complete RBAC compliance
- ✅ Intuitive UI with all CRUD operations
- ✅ Helper functions for workflow integration
- ✅ All fields optional (flexibility for different countries)
- ✅ Backward compatible with existing data

Users can now manage their bank accounts directly in My Profile, and workflows can query accounts by usage type to select the appropriate account for different payment scenarios.