Guía Completa de Páginas y Flujo - AhorraAI Frontend

ACTUALIZACIÓN IMPORTANTE - 1 de Noviembre de 2025

Redefinición Completa de la Arquitectura de Ahorros

Se ha implementado una nueva arquitectura de 3 tipos de metas de ahorro con lógicas completamente distintas:

Tipo	Cálculo	Depósitos	Uso
Monthly	Automático (Ingresos - Gastos)	NO requiere	Meta mensual con reset
Global	Automático (Acumulativo)	NO requiere	Meta de largo plazo
Custom	Manual por usuario	Requiere depósitos	Objetivos específicos con cuenta fantasma

Cambios de Base de Datos:

- ☑ Migración 016: Agregadas columnas `is_virtual_account` y `virtual_account_id`

Cambios de Frontend:

- ☑ SavingsPage: Rediseñada con 3 pestañas y formularios responsive
- ☑ Tipos: Actualizados para soportar cuentas virtuales
- ☑ Servicios: Nuevos métodos para transferencias desde cuentas fantasma
- ☑ Diseño: Completamente optimizado para mobile/tablet/desktop

---

Índice

Resumen del Proyecto

AhorraAI es una aplicación de finanzas personales diseñada específicamente para estudiantes y jóvenes. Su objetivo es ayudar a los usuarios a tomar control de sus finanzas, establecer metas de ahorro y visualizar su progreso financiero.

Tecnologías Principales

- **Framework**: React 19 con TypeScript
- **Bundler**: Vite
- **Estilos**: Tailwind CSS con sistema de diseño Shadcn/ui
- **Routing**: React Router DOM v7
- **Temas**: Next-themes (Día/Noche)
- **Backend**: API REST con Supabase o servicio similar
- **Tipos**: TypeScript con definiciones claras

Estructura de Componentes

- `src/components/ui/` - Componentes base de Shadcn/ui
- `src/components/auth/` - Componentes específicos de autenticación
- `src/pages/` - Componentes de páginas principales
- `src/context/` - Contextos globales (AuthContext, CurrencyContext, DashboardContext)
- `src/services/` - Lógica de comunicación con APIs
- `src/hooks/` - Hooks personalizados
- `src/types/` - Definiciones de tipos TypeScript

Arquitectura de Rutas

Rutas Públicas

- `/` → LandingPage
- `/auth` → AuthPage (login/registro)

Rutas Protegidas

- `/onboarding` → OnboardingPage
- `/dashboard` → DashboardPage
- `/income` → IncomePage
- `/expenses` → ExpensePage
- `/categories` → CategoryPage
- `/savings` → SavingsPage
- `/accounts` → AccountPage
- `/settings` → SettingsPage

Sistema de Protección

- `ProtectedRoute`: Componente wrapper que redirige si el usuario no está autenticado
- `AuthContext`: Provee estado de autenticación global
- `MainLayout`: Layout común para rutas protegidas con sidebar

Flujo Principal del Usuario

```
1. Usuario visita "/" (LandingPage)
   ↓
2. Click en "Registrarse" → "/auth?mode=register"
   ↓
3. Completa formulario de registro → AuthContext.register()
   ↓
4. Redirige a "/onboarding"
   ↓
5. Completa configuración inicial (3 pasos) → OnboardingPage
   ↓
6. Redirige a "/dashboard"
   ↓
7. Usa la aplicación navegando entre rutas protegidas
   ↓
8. Click en logout → Limpia sesión y redirige a "/"
```

Detalle Completo de Cada Página

LandingPage

Ruta: `/` **Tipo**: Pública

Propósito: Presentar la aplicación a usuarios no autenticados y motivarlos a registrarse.

Componentes:

1. **Header**: Logo, navegación a auth
2. **Hero Section**: Título principal, descripción, CTAs
3. **Features Grid**: 6 características principales con iconos
4. **CTA Final**: Card con llamada a acción
5. **Footer**: Información general

Flujo:

- Usuario no registrado puede navegar a `/auth` para registrarse o iniciar sesión
- Botón "Comenzar Gratis" → `/auth?mode=register`
- Botón "Ya tengo cuenta" → `/auth`

Interacciones:

- Navegación a AuthPage para login/register
- No requiere API calls ni autenticación

Efectos:

- Componente estático sin useEffect
- Muestra información general de la aplicación

AuthPage

Ruta: `/auth` **Tipo**: Pública (con redirección si autenticado)

Propósito: Autenticar usuarios (login y registro).

Componentes:

1. **Logo y Título**: Marca de la aplicación
2. **Formulario**: LoginForm o RegisterForm según query param
3. **Toggle**: Entre login y registro

Flujo:

1. Si usuario autenticado → redirige a `/dashboard`
2. Lee query param `?mode=register` para mostrar formulario correspondiente
3. Toggle cambia el formulario y actualiza query param
4. Login exitoso → `/dashboard`
5. Registro exitoso → `/onboarding`

Interacciones:

- `useSearchParams` para manejar modo (login/register)
- `useAuth` para verificar autenticación
- Navegación a Dashboard o Onboarding según acción

Efectos:

- Redirect si ya está autenticado
- Toggle entre formulario según query param

OnboardingPage

Ruta: `/onboarding` **Tipo**: Protegida

Propósito: Configuración inicial para nuevos usuarios en 3 pasos.

Componentes:

1. **Progress Bar**: Indicador de 3 pasos
2. **Step 1**: Configuración de ingresos (salario, meta de ahorro)
3. **Step 2**: Creación de cuentas (bancarias, efectivo)
4. **Step 3**: Creación de metas de ahorro (mensuales, globales, personalizadas)

Flujo Completo:

1. **Paso 1 - Configuración Financiera**:
    - Ingreso de salario mensual
    - Establecimiento de meta de ahorro mensual (opcional)
    - API: `financialSettingService.create()`
    - Siguiente → Paso 2
2. **Paso 2 - Creación de Cuentas**:

    - Nombres, tipos, balances iniciales
    - API: `accountService.create()` para cada cuenta
    - Puedes agregar múltiples cuentas
    - Siguiente → Paso 3

3. **Paso 3 - Metas de Ahorro**:

    - Nombres, montos, tipos de meta
    - API: `savingsGoalService.create()` para cada meta
    - Puedes crear múltiples metas
    - Finalizar → Redirige a `/dashboard`

Interacciones:

- `financialSettingService` - para guardar configuración
- `accountService` - para crear cuentas
- `savingsGoalService` - para crear metas de ahorro
- `currencyService.getAll()` - para mostrar monedas en creación de cuentas
- Navegación con `useNavigate` entre pasos y al finalizar

Efectos:

- Fetch de monedas para uso en cuentas
- Manejo de estado local para cada paso
- Validación de formularios
- Navegación controlada por pasos

DashboardPage

Ruta: `/dashboard` **Tipo**: Protegida

Propósito: Vista general de las finanzas del usuario.

Componentes:

1. **Saludo Personalizado**: Con nombre del usuario
2. **Stats Cards**: Balance total, ingresos del mes, gastos del mes, ahorros
3. **Quick Actions**: Botones para añadir ingreso/gasto/ahorro
4. **Monthly Summary**: Comparación ingresos vs gastos con barras de progreso
5. **Onboarding Notice**: Sugerencia para completar perfil

Flujo:

1. Fetch de datos del mes actual desde servicios API
2. Calcula totales para mostrar en cards
3. Formatea montos con `formatCurrency`
4. Muestra gráficos de progreso visual

Interacciones:

- `useDashboard` hook - para obtener stats de la API
- `useAuth` - para obtener información del usuario

- Navegación a IncomePage, ExpensePage, SavingsPage para acciones rápidas

Efectos:

- `useEffect` que fetch datos del mes actual
- Cálculos de totales y porcentajes
- Actualización automática de estadísticas
- Manejo de loading states

IncomePage

Ruta: `/income` **Tipo**: Protegida

Propósito: Gestión completa de ingresos puntuales y salarios recurrentes (fuentes de salario fijo y promedio).

Arquitectura de Pestañas: La página tiene dos pestañas principales:

Pestaña 1: Ingresos Puntuales

Componentes:

1. **Stats Summary** (2 tarjetas):
    - Total de Ingresos (cantidad)
    - Monto Total de Ingresos confirmados
2. **Panel de Filtros**:
    - Toggle: Mes/Año vs Rango Personalizado
    - **Si Mes/Año**:
        - Selector de Mes (enero-diciembre)
        - Selector de Año (últimos 5 años)
    - **Si Rango Personalizado**:
        - Selector de Fecha Inicio
        - Selector de Fecha Fin
    - **Ordenamiento**:
        - Orden Por: Fecha / Monto / Categoría
        - Dirección: Ascendente / Descendente
    - Contador: "Mostrando X de Y gastos"
3. **Botones de Acción**:
    - "Nuevo Ingreso"
    - "Generar de Salarios" (genera ingresos pendientes desde fuentes de salario fijo)
4. **Formulario de Ingreso Puntual** (colapsible):
    - Nombre (ej: "Ingreso extra", "Regalo")
    - Monto
    - Moneda (selector)
    - Fecha específica (opcional)
    - Cuenta (opcional)
    - Descripción (opcional)
5. **Data Table**: Lista de ingresos con acciones (editar, eliminar)

Flujo de Ingresos Puntuales:

1. Usuario hace click en "Nuevo Ingreso"
2. Completa formulario con datos del ingreso (todo se marca como tipo 'extra' y confirmado por defecto)
3. Al guardar, se crea el ingreso y se actualiza la tabla
4. Usuario puede editar o eliminar ingresos existentes
5. Botón "Generar de Salarios" llama al endpoint que crea ingresos pendientes desde fuentes de salario fijo programadas

Pestaña 2: Salarios Recurrentes (Fuentes de Salario)

Componentes:

1. **Stats Summary** (4 tarjetas):
   - Total de Salarios (cantidad)
   - Salarios Activos (cantidad)
   - Monto Total de Salarios (suma)
   - Próximos Pagos (cantidad)
2. **Botón "Nueva Fuente de Salario"**: Abre formulario
3. **Formulario de Fuente de Salario**:
   - Nombre de la Fuente (ej: "Trabajo en X", "Freelance habitual")
   - Tipo de Ingreso: Solo "Fijo" actualmente (esto controla si se generan confirmaciones automáticas)
   - Monto
   - Moneda
   - Frecuencia: `weekly` o `monthly`
   - Día de Pago: día de la semana o del mes según frecuencia
   - Fecha de Inicio: para calcular próximos pagos
   - Cuenta Destino (opcional)
   - Descripción (opcional)
4. **Data Table**: Lista de fuentes de salario con acciones (editar, eliminar)

Flujo de Salarios Recurrentes:

1. Usuario agrega una nueva fuente de salario con monto, frecuencia, día de pago y fecha de inicio
2. Al guardar, se almacena la fuente como `SalarySchedule` con todos los parámetros
3. El sistema puede generar ingresos pendientes automáticamente (llamando al endpoint `/income-service/generate/salary-incomes`)
4. Las fuentes de salario no crean ingresos directamente, sino que planean pagos futuros
5. Estadísticas se actualizan mostrando información sobre fuentes activas y próximos pagos

Formularios y Campos Especiales:

- El formulario de salarios muestra campos específicos para programación de pagos (frecuencia, día, fecha inicio)
- No hay distinción entre "fijo" y "promedio" en el código actual de IncomePage (esto parece haber cambiado desde la documentación original)

Interacciones:

- `incomeService.getAll()` - para listar ingresos
- `salaryScheduleService.getAll()` - para listar fuentes de salario

- `salaryScheduleService.create()` - para crear fuente de salario
- `salaryScheduleService.update()` - para editar fuente de salario
- `salaryScheduleService.delete()` - para eliminar fuente de salario
- `incomeService.create()` - para crear ingreso puntual
- `incomeService.update()` - para editar ingreso
- `incomeService.delete()` - para eliminar ingreso
- `incomeService.generateSalaryIncomes()` - para generar ingresos pendientes desde fuentes de salario
- `accountService.getAll()` - para mostrar cuentas disponibles
- `currencyService.getAll()` - para mostrar monedas

Efectos:

- Fetch inicial de ingresos, fuentes de salario, cuentas y monedas
- Separación de datos entre ingresos puntuales y fuentes de salario
- Cálculo dinámico de estadísticas según pestaña activa
- Toggle entre pestañas de ingresos y salarios

ExpensePage

Ruta: `/expenses` **Tipo**: Protegida

Propósito: Registro, gestión y análisis completo de gastos con filtros y ordenamiento avanzados.

Componentes:

1. **Stats Summary** (3 tarjetas):
   - Total Gastos (monto)
   - Necesarios (monto)
   - Innecesarios (monto)
2. **Controles de Acción**:
   - Botón "Nuevo Gasto"
   - Selector de Tipo de Gasto (Todos/Necesarios/Innecesarios)
3. **Panel de Filtros Completo** (colapsible):
   - Toggle: Mes/Año vs Rango Personalizado
   - **Si Mes/Año**:
     - Selector de Mes (enero-diciembre)
     - Selector de Año (últimos 5 años)
   - **Si Rango Personalizado**:
     - Selector de Fecha Inicio
     - Selector de Fecha Fin
   - **Ordenamiento**:
     - Orden Por: Fecha / Monto / Categoría
     - Dirección: Ascendente / Descendente
   - Contador: "Mostrando X de Y gastos"
4. **Formulario de Gasto** (colapsible):
   - Descripción
   - Tipo: `fixed` o `variable`
   - Categoría (selector)

    - Monto
    - Moneda
    - Fecha
    - Cuenta (opcional)
5. **Data Table**: Lista de gastos filtrados y ordenados

Flujo Completo:

Filtración:

1. Usuario selecciona tipo de gasto (necesario/innecesario/todos)
2. Elige entre filtro Mes/Año o Rango Personalizado
3. Si Mes/Año: selecciona mes y año específicos
4. Si Rango: establece fecha inicio y fin
5. Tabla se actualiza automáticamente mostrando solo gastos coincidentes

Ordenamiento:

1. Usuario selecciona campo de ordenamiento:
    - **Fecha**: Ordena por `expense_date`
    - **Monto**: Ordena por cantidad gastada
    - **Categoría**: Ordena alfabéticamente por categoría
2. Usuario elige dirección (ascendente/descendente)
3. Tabla se reorganiza inmediatamente (sin nuevo request)

CRUD de Gastos:

1. **Creación**: Click en "Nuevo Gasto", completa formulario
2. **Edición**: Click en botón editar en fila, recarga formulario
3. **Eliminación**: Click en botón eliminar, confirmación, remueve y actualiza tabla
4. **Visualización**: Tabla muestra descripción, categoría, tipo, monto, acciones

Interacciones:

- `expenseService.getAll()` - para listar gastos
- `expenseService.create()` - para crear gasto
- `expenseService.update()` - para editar gasto
- `expenseService.delete()` - para eliminar gasto
- `categoryService.getAll()` - para mostrar categorías
- `accountService.getAll()` - para mostrar cuentas
- `currencyService.getAll()` - para mostrar monedas

Efectos:

- Fetch inicial de gastos, categorías, cuentas y monedas
- Filtrado en cliente por tipo de categoría, rango de fechas y ordenamiento
- Cálculos de totales actualizados según filtros aplicados
- Contador dinámico de gastos mostrados vs totales

SavingsPage

Ruta: `/savings` **Tipo**: Protegida

Propósito: Gestión de metas de ahorro con 3 tipos distintos: Metas Mensuales, Metas Globales y Metas Personalizadas. Ahora con control total desde la misma página.

Arquitectura de Pestañas: La página tiene tres pestañas principales para organizar las metas:

Pestaña 1: Metas Personalizadas (Custom)

Componentes:

1. **Stats Summary** (4 tarjetas):
    - Meta Mensual: Muestra el progreso de ahorro mensual calculado automáticamente
    - Meta Global: Muestra el progreso de ahorro global acumulado
    - Metas Personalizadas: Cantidad de metas custom creadas
    - Progreso Total: Porcentaje completado de todas las metas personalizadas
2. **Botón "Nueva Meta Personalizada"**: Abre formulario para crear meta
3. **Formulario de Meta Personalizada** (colapsible):
    - Nombre (ej: "Auto", "Vacaciones", "Laptop")
    - Monto Objetivo
    - Moneda (deshabilitada, toma la moneda por defecto del usuario)
    - Fecha Límite (opcional, sin fecha límite predeterminada)
    - Descripción (opcional)
4. **Data Table**: Lista de metas personalizadas con columnas:
    - Meta: Nombre con descripción
    - Objetivo: Monto total deseado
    - Ahorrado: Dinero actual en la meta (en su cuenta fantasma)
    - Progreso: Barra visual + porcentaje
    - Fecha Límite: Fecha objetivo o "-" si no hay
    - Estado: Badge con "active" o "completed"
    - Acciones:
        - **Depositar** (Plus icon): Abre formulario para meter dinero a la meta FROM una cuenta real
        - **Transferir** (Send icon): Transfiere dinero desde la cuenta fantasma a una cuenta real
        - **Eliminar** (Trash icon): Borra la meta y su cuenta fantasma
5. **Formulario de Depósito a Cuenta Fantasma** (TRANSFERENCIA DE DINERO REAL):
    - **De Cuenta (Origen)**: Selector de cuentas reales del usuario con saldo disponible
    - Monto a Depositar: Input numérico (validado contra saldo disponible)
    - Descripción: Campo opcional
    - **Efecto Crítico - EL DINERO SE MUEVE**:
        - ☑ RESTA el monto de la cuenta origen (real) → Balance disminuye
        - ☑ SUMA el monto a la cuenta fantasma (virtual) → Balance aumenta
        - ☑ El dinero YA NO está disponible en la cuenta real
        - ☑ El balance total del usuario NO cambia (solo se redistribuye)

        - 💡 Ejemplo: Tienes $1,000 en banco → Apartas $500 a meta → Banco: $500, Meta: $500
    - **Validación**: No puedes depositar más dinero del que tienes disponible en la cuenta origen

6. **Formulario de Transferencia desde Cuenta Fantasma** (LIBERAR DINERO):
    - Muestra disponible: Cantidad actual en la cuenta fantasma
    - Monto a Transferir: Input con validación de máximo
    - Cuenta Destino: Selector de cuentas reales (excluye cuentas virtuales)
    - Botones: Cancelar / Transferir
    - **Efecto**: Disminuye el saldo de la cuenta fantasma, aumenta la cuenta real destino

Flujo de Metas Personalizadas:

1. Usuario crea meta personalizada con nombre, monto y fecha opcional
2. Sistema automáticamente crea una cuenta fantasma/virtual para esa meta
3. Usuario **deposita dinero a la meta** (nuevo flujo):
    - Click en botón "Depositar" (Plus)
    - **Selecciona de qué cuenta REAL quiere transferir** (cuenta origen)
    - Ingresa monto
    - Sistema automáticamente:
        - Resta del saldo de la cuenta origen (real)
        - Suma a la cuenta fantasma (virtual)
    - El usuario ve: "Tenía $1,000, ahora tengo $900 disponibles. Los $100 están en la meta X"
4. Dinero en cuenta fantasma está "bloqueado" (no afecta balance real)
5. Si necesita el dinero, transfiere desde la cuenta fantasma a una cuenta real:
    - Click en botón "Transferir" (Send)
    - Selecciona cuenta destino
    - Sistema crea una transacción de transferencia
    - Disminuye balance de cuenta fantasma
    - Aumenta balance de cuenta real
6. Progreso se actualiza automáticamente según el saldo de la cuenta fantasma

Pestaña 2: Meta Mensual

Comportamiento:

- Si NO existe meta mensual: Muestra botón "Crear Meta Mensual" + formulario colapsible
- Si EXISTE: Muestra progreso actual + card con estadísticas

Componentes (cuando existe):

1. Card principal con:
    - Progreso visual: Barra de progreso + monto actual
    - Meta objetivo mostrada debajo
    - Porcentaje completado
    - Falta por ahorrar (cálculo: target - current)
    - Info box: Explica la fórmula de cálculo

Flujo:

1. Usuario abre pestaña y ve "Crear Meta Mensual"
2. Click en botón abre formulario colapsible
3. Ingresa monto objetivo (ej: $1,500)
4. Sistema crea meta mensual con `goal_type: 'monthly'`
5. Meta se calcula automáticamente: **Ingresos Totales - Gastos Totales = Ahorro Mensual**
6. No requiere depósitos manuales, se actualiza automáticamente

Lógica:

- La meta mensual NO es una cuenta fantasma
- Se calcula directamente desde la lógica financiera del mes
- Se puede actualizar el monto objetivo desde la misma pestaña

Pestaña 3: Meta Global

Comportamiento:

- Si NO existe meta global: Muestra botón "Crear Meta Global" + formulario colapsible
- Si EXISTE: Muestra progreso actual + card con estadísticas

Componentes (cuando existe):

1. Card principal con:
    - Progreso visual: Barra de progreso + monto actual
    - Meta objetivo mostrada debajo
    - Porcentaje completado
    - Falta por ahorrar (cálculo: target - current)
    - Info box: Explica que es el acumulado total

Flujo:

1. Usuario abre pestaña y ve "Crear Meta Global"
2. Click en botón abre formulario colapsible
3. Ingresa monto objetivo (ej: $9,000 = 6 meses × $1,500 mensuales)
4. Sistema crea meta global con `goal_type: 'global'`
5. Meta se calcula automáticamente: **Suma de todos los ahorros acumulados**
6. No requiere depósitos manuales, se actualiza con cada transacción

Lógica:

- La meta global NO es una cuenta fantasma
- Se calcula como: Suma de (Ingresos Confirmados - Gastos) desde inicio de usuario
- Representa el "colchón financiero" total del usuario
- Se puede actualizar el monto objetivo desde la misma pestaña

Interacciones:

- `savingsGoalService.getAll()` - para listar todas las metas
- `savingsGoalService.create()` - para crear metas personalizadas, monthly y global
- `savingsGoalService.update()` - para actualizar targets de monthly/global
- `savingsGoalService.delete()` - para eliminar meta personalizada

- `savingsGoalService.transferFromVirtualAccount()` - para transferir desde cuenta fantasma
- `savingsDepositService.create()` - para crear depósitos **CON source_account_id**
- `accountService.getAll()` - para mostrar cuentas reales en selectores

Efectos:

- Fetch inicial de metas (separadas por tipo) y cuentas reales
- Filtrado en cliente para separar custom/monthly/global
- Cálculo dinámico de estadísticas y progreso
- Tabs con contenido independiente por tipo de meta
- Validación de saldo suficiente en cuenta origen para depósitos
- Validación de monto máximo en transferencias desde virtual

Diseño Responsive:

- Grid de stats: 1 columna en mobile, 2 en tablet, 4 en desktop
- Tabs: Stack vertical en mobile, horizontal en desktop
- Formularios: Full width en mobile, columnas en desktop
- DataTable: Scroll horizontal en mobile si es necesario
- Selectores: Dropdown optimizado para mobile
- Deposit/Transfer forms: Diseño responsivo con 2 columnas en desktop, 1 en mobile

CategoryPage

Ruta: `/categories` **Tipo**: Protegida

Propósito: Gestión de categorías de gastos.

Componentes:

1. **Tabla de Categorías**: Nombre, tipo, descripción, acciones
2. **Formulario**: Crear/editar categorías
3. **Botones de Acción**: Añadir, editar, eliminar

Flujo Completo:

1. **Listado**: Fetch de categorías con paginación
2. **Creación**: Formulario con nombre, tipo (necesario/innecesario), descripción
3. **Edición**: Carga datos existentes en formulario
4. **Eliminación**: Confirmación con alerta

Formulario:

- **Nombre**: Campo obligatorio
- **Tipo**: Selector entre "Necesario" e "Innecesario"
- **Descripción**: Campo opcional

Interacciones:

- `categoryService.getAll()` - para listar categorías
- `categoryService.create()` - para crear categoría
- `categoryService.update()` - para editar categoría

- `categoryService.delete()` - para eliminar categoría

Efectos:

- Fetch inicial de categorías
- Manejo de formulario con validación
- Paginación de resultados
- Toggle entre formulario de creación/edición y tabla

AccountPage

Ruta: `/accounts` **Tipo**: Protegida

Propósito: Gestión de cuentas bancarias y efectivo.

Componentes:

1. **Stats Summary**: Total cuentas, balance total, cuentas activas
2. **Formulario**: Crear/editar cuentas
3. **Tabla de Cuentas**: Nombre, tipo, balance, descripción, acciones
4. **Botones de Acción**: Añadir, editar, eliminar

Formulario:

- **Nombre**: Campo obligatorio
- **Tipo**: Selector entre "Efectivo", "Banco", "Plataforma"
- **Moneda**: Selector de moneda (traído de currencyService)
- **Balance Inicial**: Campo numérico
- **Descripción**: Campo opcional

Flujo Completo:

1. **Listado**: Fetch de cuentas (excluyendo cuentas virtuales)
2. **Creación**: Completa formulario y guarda nueva cuenta
3. **Edición**: Carga datos existentes en formulario
4. **Eliminación**: Confirmación con alerta

Interacciones:

- `accountService.getAll()` - para listar cuentas
- `accountService.create()` - para crear cuenta
- `accountService.update()` - para editar cuenta
- `accountService.delete()` - para eliminar cuenta
- `currencyService.getAll()` - para mostrar monedas

Efectos:

- Fetch inicial de cuentas y monedas
- Cálculo de balance total (excluyendo cuentas virtuales)
- Manejo de formulario con validación
- Paginación de resultados

SettingsPage

Ruta: `/settings` **Tipo**: Protegida

Propósito: Configuración del perfil y preferencias financieras.

Componentes:

1. **Información de Perfil**:
   - Email (solo lectura)
   - Nombre completo (editable)
2. **Configuración Financiera**:
   - Moneda por defecto (seleccionable)
3. **Avatar**: Subida de imagen de perfil
4. **Zona de Peligro**: Eliminación de cuenta

Flujo Completo:

1. **Carga Inicial**: Fetch de datos del usuario y configuración financiera
2. **Edición de Perfil**: Actualización de nombre completo
3. **Configuración Financiera**: Actualización de moneda por defecto
4. **Subida de Avatar**: Selección y carga de imagen
5. **Eliminación de Cuenta**: Proceso de confirmación múltiple

Formularios:

- **Perfil**: Nombre completo
- **Financiera**: Moneda por defecto
- **Avatar**: Archivo de imagen

Interacciones:

- `authService.updateProfile()` - para actualizar datos del perfil
- `authService.uploadAvatar()` - para subir avatar
- `authService.deleteAccount()` - para eliminar cuenta (llama a logout)
- `financialSettingService.getCurrent()` - para cargar configuración financiera
- `financialSettingService.create()` - para crear configuración financiera
- `financialSettingService.update()` - para actualizar configuración financiera
- `useAuth.logout()` - para cerrar sesión tras eliminar cuenta

Efectos:

- Fetch inicial de datos del usuario y configuración
- Manejo de múltiples formularios (perfil, financiero, avatar)
- Proceso de confirmación para eliminación de cuenta
- Actualización del contexto de moneda por defecto

Eventos que Desencadena:

- Actualización de nombre del usuario en todas las páginas
- Cambio de moneda por defecto afecta a todo el sistema

- Eliminación de cuenta y logout completo

Interacciones Entre Páginas

Navegación Principal

- **Dashboard** → Acciones rápidas llevan a Income/Expense/Savings
- **Sidebar** → Navegación accesible desde cualquier página protegida
- **Header** → Logout disponible en todas las páginas protegidas

Flujo de Transacciones

- **IncomePage** → Crea ingresos que afectan Dashboard (stats)
- **ExpensePage** → Crea gastos que afectan Dashboard (stats) y Categorías
- **SavingsPage** → Crea metas y depósitos que afectan Dashboard (stats)

Flujo de Configuración

- **Onboarding** → Establece configuración base para Dashboard
- **Settings** → Permite actualizar configuración existente
- **CategoryPage** → Configura categorías usadas en ExpensePage
- **AccountPage** → Configura cuentas usadas en Income/Expense

Flujo de las 3 Metas de Ahorro

1. Meta Mensual (`monthly`)

Tipo: `monthly` (en el sistema)

Propósito: Meta de ahorro objetivo mensual que se calcula automáticamente basada en el flujo de efectivo.

Flujo Completo:

1. **Creación**: En SavingsPage o OnboardingStep3 con `goal_type: 'monthly'`
2. **Cálculo Automático**: **Ingresos Confirmados del Mes - Gastos del Mes = Ahorro Mensual**
3. **NO requiere depósitos**: Se calcula en tiempo real desde transacciones
4. **NO involucra cuentas fantasma**: Es solo un indicador de rendimiento mensual
5. **Seguimiento**: Se compara contra el monto objetivo establecido
6. **Visualización**: Progreso mostrado en Dashboard y SavingsPage/Pestaña Mensual
7. **Reinicio**: Se recalcula mensualmente para nuevo seguimiento
8. **Flexibilidad**: Usuario puede configurar diferentes métodos:
    - % de ingresos (ej: 10% de lo que gana)
    - Cantidad fija (ej: $1,500 mensuales)
    - Diferencia neta (Ingresos - Gastos) ← **Método actual implementado**

Ejemplo Real con Datos del Sistema:

- Usuario establece meta mensual de $1,000
- Noviembre 2025: Gana $1,499.99 (ingresos confirmados) - Gasta $100.00 = $1,399.99 ahorrados
- Progreso = $1,399.99 / $1,000 = 139.9% completado ☑

- Octubre 2025: Gana $1,999.99 - Gasta $600.00 = $1,399.99 ahorrados
- Progreso = $1,399.99 / $1,000 = 139.9% completado ☑

Conexión con Otras Páginas:

- **Dashboard**: Mostrada en "Ahorros" card como progreso mensual actual
- **IncomePage**: Los ingresos confirmados alimentan el cálculo de ahorro mensual
- **ExpensePage**: Los gastos se restan del ahorro mensual
- **Settings**: Se configura el monto objetivo de la meta mensual
- **Onboarding**: Primera meta sugerida durante configuración inicial

Diferencias con Meta Global:

- **Meta Mensual**: Se reinicia cada mes, mide rendimiento del mes actual
- **Meta Global**: Acumula todos los meses, mide patrimonio total creciente

2. Meta Global (`global`)

Tipo: `global` (en el sistema)

Propósito: Meta de ahorro de largo plazo que representa el "colchón financiero" total acumulado.

Flujo Completo:

1. **Creación**: En SavingsPage o OnboardingStep3 con `goal_type: 'global'`
2. **Configuración**: Nombre, monto objetivo total (ej: $150,000), descripción
3. **Cálculo Automático**: **Suma acumulada de todos los ahorros netos desde inicio del usuario**
4. **NO requiere depósitos**: Se calcula acumulativamente desde ingresos/gastos confirmados
5. **NO involucra cuentas fantasma**: Es un indicador de patrimonio neto
6. **Relación con Meta Mensual**: La meta global crece mes a mes según el ahorro mensual logrado
7. **Seguimiento**: Progreso mostrado comparando vs objetivo de largo plazo
8. **Compleción**: Meta se marca como completada al alcanzar objetivo

Ejemplo Completo: **Situación Inicial:**

- Usuario tiene balance inicial: $0
- Establece meta global: $100,000 (su objetivo de patrimonio)
- Establece meta mensual: $4,000/mes (para lograr la global)

Mes 1 (Octubre 2025):

- Ingresos: $1,999.99
- Gastos: $600.00
- Ahorro mensual: $1,399.99 ☑ (no alcanzó meta de $4,000)
- **Ahorro global acumulado**: $0 + $1,399.99 = $1,399.99

Mes 2 (Noviembre 2025):

- Ingresos: $1,499.99
- Gastos: $100.00
- Ahorro mensual: $1,399.99 ☑
- **Ahorro global acumulado**: $1,399.99 + $1,399.99 = $2,799.98

Progreso:

- Meta Global: $2,799.98 / $100,000 = 2.8% completado
- Faltan: $97,200.02
- A este ritmo (~$1,400/mes): ~69 meses para completar

Conexión con Otras Páginas:

- **Dashboard**: Contribuye al total de ahorros acumulados
- **SavingsPage/Pestaña Global**: Vista detallada con breakdown
- **IncomePage**: Cada ingreso confirmado incrementa el ahorro global
- **ExpensePage**: Cada gasto disminuye el ahorro global neto
- **Settings**: Puede influir en recomendaciones de ahorro

Nota Importante sobre Balance Real:

- El "ahorro global" NO es dinero físico guardado en ninguna cuenta
- Es un **indicador calculado**: Total Ingresos - Total Gastos desde inicio
- El dinero real está distribuido en las cuentas bancarias del usuario
- Para "guardar" dinero físicamente, usar **Metas Personalizadas** con cuentas fantasma

3. Meta Personalizada (`custom`)

Tipo: `custom` (en el sistema)

Propósito: Metas de ahorro específicas para objetivos particulares con "cuentas fantasma".

Flujo Completo:

1. **Creación**: En SavingsPage/Pestaña Custom con `goal_type: 'custom'`
2. **Configuración**: Nombre descriptivo, monto objetivo, fecha límite opcional, descripción
3. **Cuenta Fantasma Automática**: Sistema crea una cuenta virtual asociada
4. **Dinero Separado**: El usuario deposita dinero explícitamente en esta meta
5. **Cuenta Bloqueada**: El dinero en la cuenta fantasma no se cuenta como balance real
6. **Transferencia**: Para usar ese dinero, debe transferirlo a una cuenta real
7. **Seguimiento**: Progreso mostrado hasta alcanzar objetivo

Ejemplo Completo - Flujo Real de Dinero:

Situación Inicial:

- Usuario tiene $15,000 en Cuenta Bancaria Real
- Usuario tiene $0 en metas personalizadas

Paso 1: Crear Meta

- Crea meta: "Comprar Auto", objetivo $30,000
- Sistema automáticamente crea cuenta fantasma: "Virtual: Comprar Auto"
- Cuenta fantasma balance: $0
- Cuenta bancaria sigue: $15,000

Paso 2: Primera Transferencia (Apartar Dinero)

- Usuario deposita $5,000 en la meta DESDE su cuenta bancaria
- **Movimiento de dinero**:
  - Cuenta Bancaria: $15,000 → $10,000 (SE RESTA $5,000)
  - Cuenta Fantasma: $0 → $5,000 (SE SUMA $5,000)
- **Balance total**: Sigue siendo $15,000 (solo redistribuido)
- **Dinero disponible**: Solo $10,000 (los $5,000 están "bloqueados" en la meta)

Paso 3: Segunda Transferencia

- Usuario deposita otros $10,000 en la meta
- **Movimiento de dinero**:
  - Cuenta Bancaria: $10,000 → $0 (SE RESTA $10,000)
  - Cuenta Fantasma: $5,000 → $15,000 (SE SUMA $10,000)
- **Balance total**: Sigue siendo $15,000
- **Dinero disponible**: $0 (todo está en la meta)

Paso 4: Sacar Dinero de la Meta (cuando lo necesita)

- Usuario transfiere $5,000 desde cuenta fantasma a cuenta bancaria
- **Movimiento de dinero**:
  - Cuenta Fantasma: $15,000 → $10,000 (SE RESTA $5,000)
  - Cuenta Bancaria: $0 → $5,000 (SE SUMA $5,000)
- **Balance total**: Sigue siendo $15,000
- **Dinero disponible**: $5,000
- **Progreso de meta**: 33% ($10,000 de $30,000)

Características Clave:

- **Dinero Real en Movimiento**: El dinero SÍ se transfiere físicamente entre cuentas
- **No hay duplicación**: El balance total NUNCA cambia, solo se redistribuye
- **Protección Financiera**: Al apartar dinero, NO puedes gastarlo accidentalmente
- **Acceso Controlado**: Para usar ese dinero, debes transferirlo de vuelta explícitamente
- **Separación Visual**: Ves claramente cuánto tienes "disponible" vs "guardado"
- **Múltiples metas**: Puedes tener varias metas custom, cada una con su cuenta fantasma

Conexión con Otras Páginas:

- **Dashboard**: Incluida en cálculo total de ahorros (opcionalmente)
- **SavingsPage/Pestaña Personalizadas**: Vista detallada con todas las metas
- **Accounts**: Cuentas fantasma no aparecen en lista de cuentas reales
- **Presupuesto**: Puede usarse para planificación de gastos futuros

Flujo Compartido de las 3 Metas

Ciclo de Vida:

1. **Creación**:
   - Monthly/Global: Durante onboarding o settings
   - Custom: En SavingsPage cuando el usuario lo desee
2. **Seguimiento Automático**:

    - Monthly/Global: Se actualizan automáticamente desde ingresos/gastos
    - Custom: Se actualiza según depósitos/transferencias del usuario
3. **Visualización**:
    - Progreso mostrado en SavingsPage con tabs
    - Estadísticas en Dashboard
4. **Gestión**:
    - Edición limitada (solo eliminación o cambio de estado)
    - Para monthly/global: cambia en settings
    - Para custom: se puede eliminar desde SavingsPage
5. **Completitud**:
    - Cálculo automático de progreso y estado de cumplimiento

Integración con el Sistema:

- **Datos**: Almacenados en `savings_goals` table con diferentes `goal_type`
- **Cálculos**: Agregados automáticamente en triggers de BD
- **Visualización**: Servicios separados pero integrados
- **API**: `savingsGoalService` con métodos específicos por tipo
- **Cuentas Fantasma**: Almacenadas en tabla `accounts` con `is_virtual_account = true`

Impacto en la Experiencia del Usuario:

- **Motivación**: 3 formas de ver el progreso (mensual, global, específica)
- **Flexibilidad**: Elige cómo ahorrar según necesidad (automático o manual)
- **Control**: Con custom goals, control total sobre dinero separado
- **Seguimiento**: Progreso medible y visualizable en tiempo real
- **Libertad Financiera**: Combina automatización con control manual

---

📋 Historial de Cambios y Actualizaciones

Actualización del 30 de Octubre de 2025

IncomePage - Nueva Arquitectura de Fuentes de Salario

Nuevas Características:

1. ☑ **Sistema de Pestañas**: Separación clara entre Ingresos Puntuales y Fuentes de Salario
2. ☑ **Gestión de Fuentes de Salario**:
    - Fuentes con programación de pagos (frecuencia, día, fecha inicio)
    - Generación automática de ingresos pendientes mediante endpoint especial
    - Cálculo de próximos pagos programados

Cambios Técnicos:

- Se han eliminado las distinciones entre "fijo" y "promedio" en el código actual
- Las fuentes de salario ahora se manejan como `SalarySchedule` independientes
- El sistema de generación automática ahora es un endpoint separado

ExpensePage - Sistema de Filtración y Ordenamiento Completo

Nuevas Características:

1. ☑ **Filtración Avanzada**:
   - Toggle entre filtro Mes/Año vs Rango Personalizado
   - Selectores de mes y año con validación
   - Selectores de fecha inicio/fin para rango personalizado
2. ☑ **Ordenamiento Flexible**:
   - Por Fecha (más reciente primero por defecto)
   - Por Monto (mayor gasto primero)
   - Por Categoría (orden alfabético)
   - Dirección: Ascendente/Descendente
3. ☑ **UI Mejorada**: Panel de filtros colapsible en sección gris
4. ☑ **Indicadores**: Contador dinámico "Mostrando X de Y gastos"
5. ☑ **Filtración Mantenida**: El filtro por tipo (necesario/innecesario) se mantiene y combina con fecha

Cambios Técnicos:

- Lógica de filtración combinada (tipo + fecha + ordenamiento) ahora se ejecuta en cliente
- Ordenamiento en cliente para mejor UX (sin refetch del servidor)
- Almacenamiento de estado para cada filtro por separado
- Validación de rangos de fecha

Mejoras Compartidas

1. **Mejor Organización Visual**: Ambas páginas usan formularios colapsibles y secciones expandibles
2. **Stats Dinámicas**: Se actualizan según filtros y datos actuales
3. **UX Consistente**: Estilos y patrones similares en todas las páginas
4. **Performance**: Filtración en cliente para respuesta inmediata

---

Documentación actualizada el 31 de octubre de 2025

📊 Comparativa: Antes vs Después

Antes

```
SavingsPage
├── Metas (tabla única)
├── Depósitos (tabla única)
└── Un solo flujo para todos los tipos
```

Después ⭐

```
SavingsPage (3 Pestañas)
├── Pestaña 1: Personalizadas (Custom)
│   ├── Formulario para crear metas custom
│   ├── Tabla con metas y opciones de transferencia
```

```
│   └── Cuenta fantasma automática por meta
├── Pestaña 2: Mensual (Monthly)
│   ├── Solo lectura
│   └── Cálculo automático: Ingresos - Gastos
└── Pestaña 3: Global (Global)
    ├── Solo lectura
    └── Cálculo automático acumulativo
```

🔄 Flujo de Dinero en Custom Goals

```
Usuario Deposita Dinero
         ↓
Cuenta Fantasma (Virtual)
  - Balance: $5,000
  - NO afecta balance total
         ↓
Usuario Decide Transferir
         ↓
Cuenta Real (Bancaria)
  - Balance aumenta: +$5,000
  - Dinero disponible para usar
```

📱 Responsive Design Implementado

```
Mobile (< 640px)     Tablet (640-1024px)    Desktop (> 1024px)
├─ 1 column stats    ├─ 2 column stats      ├─ 4 column stats
├─ Stack vertical    ├─ Tabs horizontal     ├─ Tabs horizontal
├─ Full width forms  ├─ 2 col grids         ├─ 2 col grids
└─ Compact text      └─ Normal text         └─ Full text
```

📋 Cambios Principales en la Actualización del 31 de Octubre

Arquitectura de Ahorros - Redefinida

La lógica de las metas de ahorro ha sido completamente redefinida en 3 tipos distintos:

1. Metas Mensuales y Globales (Automáticas)

- **NO requieren depósitos manuales**
- Se calculan automáticamente: `Ingresos - Gastos = Ahorro`
- **Monthly**: Reinicia cada mes, compara contra objetivo mensual
- **Global**: Acumula todo el ahorro desde inicio del usuario
- Se actualizan automáticamente con cada transacción

2. Metas Personalizadas (Manuales con Cuenta Fantasma)

- **Requieren depósitos explícitos del usuario**
- Cada meta tiene una **cuenta fantasma/virtual** asociada automáticamente
- El dinero en la cuenta fantasma está "bloqueado" (no afecta balance real)
- Usuario puede transferir dinero de la cuenta fantasma a cuentas reales cuando necesite
- Perfectas para objetivos específicos (auto, vacaciones, etc.)

Cambios en la Base de Datos

- ☑ Migración `016_add_virtual_account_support`:
  - Agregada columna `is_virtual_account` a tabla `accounts`
  - Agregada columna `virtual_account_id` a tabla `savings_goals`
  - Nueva función SQL: `create_virtual_account_for_goal()`

Cambios en el Frontend

SavingsPage - Nueva Arquitectura

- ☑ Reemplazada por 3 pestañas (Personalizadas, Mensual, Global)
- ☑ Formularios más responsive: adaptables a mobile/tablet/desktop
- ☑ Nuevo formulario de transferencia desde cuentas fantasma
- ☑ Mejor separación visual entre tipos de metas
- ☑ Stats cards con colores y iconos diferenciados

Types Actualizados

- ☑ `Account`: Nuevo campo `is_virtual_account?: boolean`
- ☑ `SavingsGoal`: Nuevo campo `virtual_account_id?: string`

Servicios Actualizados

- ☑ `savingsGoalService`:
  - Nuevo método `getByType()` para filtrar por tipo
  - Nuevo método `transferFromVirtualAccount()` para transferencias
  - Endpoints optimizados para nueva arquitectura

Diseño Responsive

- ☑ Grid de stats: 1 col (mobile) → 2 (tablet) → 4 (desktop)
- ☑ Tabs: Stack vertical (mobile) → Horizontal (desktop)
- ☑ Formularios: Full width (mobile) → Columnas (desktop)
- ☑ Select dropdown: Agregado backdrop blur para mejor visibilidad
- ☑ Tablas: Scroll horizontal automático en mobile

Beneficios de la Nueva Arquitectura

1. **Flexibilidad**: 3 formas diferentes de ahorrar según necesidad
2. **Automatización**: Monthly/Global se actualizan automáticamente
3. **Control Manual**: Custom goals permiten guardar dinero específicamente
4. **Separación Virtual**: Dinero en custom goals no afecta balance real

5. **Facilidad de Transferencia**: Transferir desde cuenta fantasma cuando sea necesario

Cómo Funciona en Práctica

Ejemplo: Usuario queriendo ahorrar para auto, meta mensual y global

1. Sistema crea automáticamente: Meta Mensual ($1,500/mes) + Meta Global ($100,000)
2. Usuario crea Meta Custom: "Auto - $30,000"
3. Sistema crea automáticamente: Cuenta fantasma "Auto (Virtual)"
4. Cada mes:
    - Meta Mensual se calcula: Ingresos ($4,000) - Gastos ($1,000) = $3,000 ahorrados ☑
    - Meta Global se actualiza acumulativamente
    - Usuario decide transferir $1,000 a su meta Custom
5. Meta Custom ahora tiene $1,000 en su cuenta fantasma
6. Cuando necesita dinero, transfiere $500 desde la cuenta fantasma a su cuenta bancaria real

Próximos Pasos (Backend)

- Implementar endpoint `/savings-goals/[id]/transfer-from-virtual`
- Implementar triggers para crear cuentas fantasma automáticamente
- Implementar cálculos automáticos de monthly/global en BD
- Considerar reportes de ahorros por tipo y fecha

⚡ ACTUALIZACIÓN - Noviembre 2025: Correcciones y Mejoras Críticas

1. 🐛 Bug Crítico: Sistema de Generación de Salarios

Problema Identificado:

- El sistema de generación automática de salarios desde `salary_schedules` no funcionaba correctamente
- Los salarios se generaban correctamente (Octubre, Noviembre), pero `next_generation_date` estaba saltándose meses
- Ejemplo: Con salario mensual el día 1, después de generar Nov 1, `next_generation_date` mostraba Enero 1, 2026 en lugar de Diciembre 1, 2025

Causa Raíz: En `Backend/src/controllers/incomeController.js`, función `generateSalaryIncomes()`, líneas 323-329:

```
// ❌ CÓDIGO INCORRECTO (antes)
nextGenerationDate = new Date(now.getFullYear(), now.getMonth() + 1,
schedule.salary_day);
if (now.getDate() >= schedule.salary_day) {
    nextGenerationDate.setMonth(nextGenerationDate.getMonth() + 1); // Bug:
incremento doble
}
```

Solución Implementada:

```
// ☑ CÓDIGO CORRECTO (después)
nextGenerationDate = new Date(now.getFullYear(), now.getMonth(),
schedule.salary_day);
if (nextGenerationDate <= now) {
    nextGenerationDate = new Date(now.getFullYear(), now.getMonth() + 1,
schedule.salary_day);
}
```

Cambio de Lógica:

- **Antes**: Calculaba `mes_actual + 1` y luego agregaba otro mes si ya pasó el día de pago
- **Después**: Intenta mes actual primero, solo avanza al siguiente si esa fecha ya pasó

Correcciones en Base de Datos:

```
-- Se corrigió el registro existente:
UPDATE salary_schedules
SET next_generation_date = '2025-12-01'
WHERE id = 'a8ce3c65-dc33-4e44-be61-b9ff9f1817bd';
-- Antes: next_generation_date = '2026-01-01' ❌
-- Después: next_generation_date = '2025-12-01' ☑
```

Impacto:

- ☑ Generación de salarios mensuales ahora funciona correctamente
- ☑ No se saltan meses en el calendario
- ☑ Sistema respeta la frecuencia configurada (mensual/semanal)
- ☑ Compatible con zona horaria de Guatemala (America/Guatemala)

---

2. 🔢 Corrección: Precisión Decimal en Formularios

Problema Identificado:

- Al ingresar `4000` en formularios de Ingresos o Gastos, se guardaba como `3999.98`
- Error de precisión de punto flotante en JavaScript con `parseFloat()`
- Afectaba formularios en `ExpensePage`, `IncomePage` (salarios e ingresos regulares)

Solución Implementada: Creada nueva función utilitaria en `Frontend/src/lib/utils.ts`:

```
export function parseDecimalAmount(value: string | number): number {
  if (typeof value === "number") return value;
  const cleaned = value.replace(/[^\d.-]/g, "");
  const parsed = parseFloat(cleaned);
  if (isNaN(parsed)) return 0;
  return Math.round(parsed * 100) / 100; // ← Clave: redondeo seguro
}
```

Aplicación:

- ☑ `ExpensePage.tsx` línea 485: `amount: parseDecimalAmount(e.target.value)`
- ☑ `IncomePage.tsx` línea 609 (ingresos regulares): `amount: parseDecimalAmount(e.target.value)`
- ☑ `IncomePage.tsx` línea 844 (salarios): `amount: parseDecimalAmount(e.target.value)`

Resultado:

- ☑ Ahora `4000` se guarda exactamente como `4000.00`
- ☑ No más errores de precisión decimal en montos de dinero
- ☑ Solución aplicada consistentemente en todos los formularios

---

3. 📊 Mejora: Filtrado de Dashboard por Mes y Año

Problema Identificado:

- El Dashboard sumaba todos los ingresos sin filtrar por mes ni estado de confirmación
- Incluía ingresos del mes pasado, futuro y no confirmados
- Usuario veía totales inflados e incorrectos

Solución Implementada:

`DashboardContext.tsx`:

```
// ☑ Nuevo estado agregado
const [selectedMonth, setSelectedMonth] = useState<number>(new Date().getMonth() +
1);
const [selectedYear, setSelectedYear] = useState<number>(new
Date().getFullYear());

// ☑ Nueva lógica de filtrado
const filteredIncomes = incomes.filter((income: Income) => {
  if (!income.income_date) return false;
  const date = new Date(income.income_date);
  const isSelectedMonth =
    date.getFullYear() === targetYear &&
    date.getMonth() + 1 === targetMonth;
  const isConfirmed = income.is_confirmed === true;
  return isSelectedMonth && isConfirmed; // ← Solo confirmados del mes actual
});
```

`DashboardPage.tsx`:

- ☑ Agregado selector de mes/año con botones de navegación
- ☑ Función `handlePreviousMonth()` y `handleNextMonth()`
- ☑ Botón "Hoy" para volver al mes actual
- ☑ UI responsive: `w-32 sm:w-40`, `text-xs sm:text-sm`

Resultado:

- ☑ Dashboard ahora muestra solo ingresos **confirmados** del mes seleccionado
- ☑ Usuario puede navegar entre meses (anterior/siguiente)
- ☑ Totales correctos por periodo
- ☑ Interfaz intuitiva con selectors y botones de navegación

---

4. 🔍 Nueva Funcionalidad: Filtros de Fecha en IncomePage

Problema Identificado:

- `IncomePage` no tenía sistema de filtrado por fecha
- `ExpensePage` sí tenía filtros (mes/año, rango personalizado)
- Faltaba consistencia en experiencia de usuario

Solución Implementada:

Estados agregados en `IncomePage.tsx`:

```
const [filterMonth, setFilterMonth] = useState<number>(new Date().getMonth() + 1);
const [filterYear, setFilterYear] = useState<number>(new Date().getFullYear());
const [filterStartDate, setFilterStartDate] = useState<string>('');
const [filterEndDate, setFilterEndDate] = useState<string>('');
const [useCustomDateRange, setUseCustomDateRange] = useState(false);
```

Función de filtrado:

```
const getFilteredIncomes = () => {
  let filtered = allIncomes.filter(i => i.is_confirmed === true)

  if (useCustomDateRange && filterStartDate && filterEndDate) {
    filtered = filtered.filter(i => {
      const incomeDate = new Date(i.income_date)
      const start = new Date(filterStartDate)
      const end = new Date(filterEndDate)
      return incomeDate >= start && incomeDate <= end
    })
  } else {
    // Filter by month/year
    filtered = filtered.filter(i => {
      if (!i.income_date) return false
      const incomeDate = new Date(i.income_date)
      return incomeDate.getMonth() + 1 === filterMonth && incomeDate.getFullYear()
=== filterYear
    })
  }
```

```
    return filtered
  };
```

UI de Filtros:

- ☑ Card con título "Filtros"
- ☑ Selectores de Mes y Año
- ☑ Checkbox para activar rango personalizado
- ☑ DatePickers para fecha inicio/fin
- ☑ Diseño responsive adaptado a mobile

Resultado:

- ☑ IncomePage ahora tiene filtrado igual que ExpensePage
- ☑ Usuario puede filtrar por mes/año o rango personalizado
- ☑ Consistencia en UX entre páginas de transacciones
- ☑ Filtros afectan tanto ingresos regulares como salarios generados

---

5. 🧮 Mejoras Globales: Diseño Responsive

Problema Identificado:

- Múltiples páginas no se adaptaban bien a móviles
- Espaciado fijo, texto muy grande, overflow horizontal
- Experiencia móvil pobre

Solución Implementada:

`DashboardPage.tsx`:

```
// ☑ Espaciado responsive
space-y-4 sm:space-y-8

// ☑ Títulos escalables
text-2xl sm:text-3xl

// ☑ Selectores adaptables
w-32 sm:w-40

// ☑ Botones responsive
text-xs sm:text-sm
```

`ExpensePage.tsx`:

```
// ☑ Padding responsivo
px-2 sm:px-0
```

```
// ☑ Gaps adaptativos
gap-3 sm:gap-6

// ☑ Headers escalables
text-xl sm:text-2xl
```

`IncomePage.tsx`:

```
// ☑ Espaciado vertical
space-y-4 sm:space-y-6

// ☑ Padding horizontal
px-2 sm:px-0

// ☑ Tabs adaptables
px-2 sm:px-4
```

`DataTable.tsx` **(componente compartido):**

```
// ☑ Scroll horizontal en mobile
<div className="overflow-x-auto">

// ☑ Padding responsive
px-3 sm:px-4

// ☑ Texto adaptable
text-xs sm:text-sm

// ☑ Paginación flexible
flex-col sm:flex-row
```

Patrón Aplicado:

```
Mobile (< 640px)        →  Tablet/Desktop (≥ 640px)
├─ px-2                 →  px-0 / px-4
├─ text-xs / text-xl    →  text-sm / text-2xl
├─ space-y-4 / gap-3    →  space-y-6 / gap-6
├─ flex-col             →  flex-row
└─ w-full               →  w-32 / w-40
```

Resultado:

- ☑ Todas las páginas ahora responsive (320px - 1920px)
- ☑ Breakpoint consistente: `sm:` (640px)

- ☑ Experiencia móvil significativamente mejorada
- ☑ Sin overflow horizontal
- ☑ Texto legible en todos los dispositivos

📋 Resumen de Archivos Modificados

Backend:

- `Backend/src/controllers/incomeController.js` - **CRÍTICO**: Fix generación de salarios

Frontend:

- `Frontend/src/lib/utils.ts` - Nueva función `parseDecimalAmount()`
- `Frontend/src/pages/ExpensePage.tsx` - Decimal fix + responsive
- `Frontend/src/pages/IncomePage.tsx` - Decimal fix + filtros + responsive
- `Frontend/src/context/DashboardContext.tsx` - Filtrado por mes/año + confirmación
- `Frontend/src/pages/DashboardPage.tsx` - UI selector mes/año + responsive
- `Frontend/src/components/ui/DataTable.tsx` - Mejoras responsive

Base de Datos:

- Corrección manual: `salary_schedules.next_generation_date` → 2025-12-01
- Confirmación de ingreso de prueba para validar filtrado

☑ Estado Actual del Sistema

Funcionalidades Verificadas:

- ☑ Generación automática de salarios funcionando
- ☑ Precisión decimal correcta en todos los formularios
- ☑ Dashboard filtrando correctamente por mes y confirmación
- ☑ IncomePage con sistema de filtros completo
- ☑ Diseño responsive en todas las páginas principales
- ☑ Compatibilidad con zona horaria Guatemala (America/Guatemala)

Pruebas Pendientes:

- ⏳ Verificar generación de salario en Diciembre 1, 2025
- ⏳ Probar frecuencia semanal en salary_schedules
- ⏳ Testing completo de filtros de fecha en producción
- ⏳ Validación de responsive en dispositivos reales

Próximos Pasos:

1. Monitorear generación automática de salarios en Diciembre
2. Realizar testing de usuario en dispositivos móviles
3. Considerar agregar indicadores visuales para ingresos no confirmados
4. Evaluar agregar exportación de reportes filtrados

🔍 ACLARACIÓN CRÍTICA: Diferencias Entre Metas de Ahorro

Comparativa Directa: Monthly/Global vs Custom

Aspecto	Meta Mensual/Global	Meta Personalizada
Tipo de Dinero	💭 Virtual (calculado)	💰 Real (transferido)
Requiere Depósitos	❌ No	☑ Sí
Cuenta Fantasma	❌ No tiene	☑ Sí (automática)
Afecta Balance Real	❌ No	☑ Sí (redistribuye)
Cálculo	Automático (Ingresos - Gastos)	Manual (usuario transfiere)
Dinero Bloqueado	❌ No	☑ Sí (hasta transferir de vuelta)
Propósito	📊 Monitoreo de rendimiento	🎯 Guardar para objetivo específico
Ejemplo	"Este mes ahorré $1,400"	"Tengo $500 apartados para el auto"

¿Cuándo Usar Cada Una?

Usa Meta Mensual/Global si:

- ☑ Solo quieres monitorear tu capacidad de ahorro
- ☑ No necesitas "bloquear" dinero físicamente
- ☑ Quieres saber si estás gastando menos de lo que ganas
- ☑ Buscas un indicador de salud financiera
- 💡 **Ejemplo**: "Quiero saber si logro ahorrar $4,000 mensuales de mi salario"

Usa Meta Personalizada si:

- ☑ Necesitas apartar dinero para algo específico
- ☑ Quieres evitar gastar ese dinero accidentalmente
- ☑ Tienes un objetivo concreto (auto, vacaciones, emergencias)
- ☑ Quieres ver el dinero "bloqueado" separadamente
- 💡 **Ejemplo**: "Quiero guardar $35,000 para un auto y no tocar ese dinero"

Flujo Visual Comparativo

Meta Mensual (Virtual):

```
Mes 1: Gano $4,000 - Gasto $1,000 = Ahorré $3,000 ☑
       ↓
Dashboard muestra: "Ahorro mensual: $3,000 / $4,000 (75%)"
       ↓
El dinero ($3,000) sigue en tus cuentas bancarias normales
       ↓
Puedes gastarlo si quieres (no está bloqueado)
```

Meta Personalizada (Real):

```
Cuenta Bancaria: $10,000 disponibles
       ↓
Creo meta "Auto" con objetivo $35,000
       ↓
Transfiero $5,000 de banco a meta
       ↓
Cuenta Bancaria: $5,000 disponibles
Cuenta Fantasma "Auto": $5,000 bloqueados
       ↓
No puedo gastar los $5,000 de la meta sin transferirlos de vuelta
```

Interacción Entre Metas

Escenario Completo Real:

1. **Configuración Inicial:**
   - Meta Global: $100,000 (patrimonio objetivo)
   - Meta Mensual: $4,000/mes (para lograr la global)
   - Meta Custom "Auto": $35,000
2. **Mes 1:**
   - Ingresos: $8,000
   - Gastos: $3,000
   - **Ahorro Mensual**: $5,000 ☑ (superó meta de $4,000)
   - **Ahorro Global**: $0 → $5,000
   - Usuario decide apartar $2,000 a meta "Auto"
   - **Transferencia**: Banco ($8,000) → Banco ($6,000) + Auto ($2,000)
3. **Resultado Final Mes 1:**
   - Meta Mensual: 125% completada ($5,000 / $4,000)
   - Meta Global: 5% completada ($5,000 / $100,000)
   - Meta Custom "Auto": 5.7% completada ($2,000 / $35,000)
   - Dinero Real Disponible: $6,000 en banco
   - Dinero Real Bloqueado: $2,000 en meta Auto
   - Total Real: $8,000 (no cambió, solo redistribuyó)

🔑 **Punto Crítico:**

- Las metas Monthly/Global miden el FLUJO de dinero (ingresos vs gastos)
- Las metas Custom mueven el DINERO REAL entre cuentas (disponible vs bloqueado)
- Son complementarias, no mutuamente excluyentes
- Puedes tener las 3 activas simultáneamente sin conflicto

📊 Validación del Flujo con Datos Reales del Sistema

Usuario de Prueba Actual (Noviembre 2025):

Cuentas Reales:

- Efectivo: $0.00
- Banrural Ahorro: $2,899.98
- **Total Disponible**: $2,899.98

Cuenta Fantasma:

- Virtual: SSDPRUEBA: $200.00 ☑ (bloqueada en meta custom)

Balance Total Real: $2,899.98 + $200.00 = $3,099.98

Metas Configuradas:

- Meta Global: Objetivo $100,000 | Actual: $2,799.98 (2.8%)
- Meta Mensual: Objetivo $1,000 | Actual: $1,399.99 (139.9% ☑)

Summaries:

- Noviembre 2025: Ingresos $1,499.99 - Gastos $100.00 = $1,399.99 ahorrados
- Octubre 2025: Ingresos $1,999.99 - Gastos $600.00 = $1,399.99 ahorrados
- **Total Acumulado**: $2,799.98 ☑ (coincide con progreso de meta global)

Conclusión: ☑ El flujo está correctamente implementado en la BD

Verificación de Implementación Técnica

Backend - Triggers SQL: ☑ Migration 020: `transfer_funds_on_deposit()` trigger implementado correctamente

- Deduce dinero de `source_account_id` (cuenta real)
- Suma dinero a `virtual_account_id` (cuenta fantasma)
- Actualiza `current_amount` en `savings_goals`
- **Garantiza**: El dinero se mueve físicamente, no se duplica

Backend - Controller: ☑ `savingsDepositController.js` validaciones correctas:

- Requiere `source_account_id` obligatoriamente
- Verifica saldo suficiente en cuenta origen antes de depositar
- Valida que la cuenta fantasma pertenece al usuario
- Evita depósitos negativos o cero

Frontend - Services: ☑ `savingsDepositService.ts` documentado correctamente:

- Método `create()` requiere `source_account_id`
- Comentarios explican que dinero se resta de cuenta real
- Soporte para `virtual_account_id` en custom goals

☑ `savingsGoalService.ts` métodos completos:

- `transferFromVirtualAccount()` para liberar dinero bloqueado

- `getByType()` para filtrar monthly/global/custom

Frontend - UI: ☑ `SavingsPage.tsx` arquitectura de 3 pestañas:

- Pestaña Custom: Formulario de depósito con selector de cuenta origen
- Pestaña Monthly: Solo lectura, cálculo automático
- Pestaña Global: Solo lectura, cálculo acumulativo

Base de Datos - Estructura: ☑ Tabla `accounts`:

- Campo `is_virtual_account` para identificar cuentas fantasma
- Constraint: `balance >= 0` evita saldos negativos

☑ Tabla `savings_goals`:

- Campo `goal_type`: 'monthly', 'global', 'custom'
- Campo `virtual_account_id` para custom goals
- Campo `current_amount` se actualiza automáticamente

☑ Tabla `savings_deposits`:

- Campo `source_account_id`: De dónde sale el dinero
- Campo `virtual_account_id`: Hacia dónde va el dinero
- Campo `goal_id`: Meta asociada

Estado Final del Flujo: 100% Correcto ☑

El flujo descrito en este documento ahora coincide exactamente con la implementación real.

Los cambios realizados en esta actualización fueron:

1. ☑ Corregida la explicación del depósito (ahora indica que SÍ se resta de cuenta real)
2. ☑ Agregado ejemplo completo con números paso a paso
3. ☑ Aclarada diferencia entre metas virtual (monthly/global) vs real (custom)
4. ☑ Agregada tabla comparativa de las 3 metas
5. ☑ Validada implementación técnica contra código y BD
6. ☑ Confirmado que triggers SQL funcionan correctamente

No se requieren cambios en código, solo se actualizó la documentación para reflejar correctamente cómo funciona el sistema.