InstaBudget API Specification

This document lists all the functions and methods we implemented in our codebase. It's organized by feature/module to make it easier to find things.

Table of Contents

Core Utilities

Date Utilities (`lib/date-utils.ts`)

We created these functions to handle dates in local timezone because we were having issues with UTC dates showing the wrong day.

`parseLocalDate(dateString: string): Date`

- **What it does:** Takes a date string like "2025-12-03" and converts it to a Date object in local time (not UTC)
- **Parameters:**
  - `dateString` - date in YYYY-MM-DD format
- **Returns:** Date object
- **Side effects:** None

`formatLocalDate(date: Date): string`

- **What it does:** Takes a Date object and formats it as YYYY-MM-DD string in local timezone
- **Parameters:**
  - `date` - the Date object to format
- **Returns:** string in YYYY-MM-DD format
- **Side effects:** None

`getTodayLocal(): string`

- **What it does:** Gets today's date as a string in local timezone

- **Parameters:** None
- **Returns:** Today's date as YYYY-MM-DD string
- **Side effects:** None

`getTodayUTC(): string`

- **What it does:** Gets today's date in UTC (we use this to compare with edge function dates)
- **Parameters:** None
- **Returns:** Today's date in UTC as string
- **Side effects:** None

`parseLocalDateIfPossible(dateString: string): Date`

- **What it does:** Tries to parse as local date if it's YYYY-MM-DD, otherwise parses normally. Used for formatting functions that might get different date formats.
- **Parameters:**
  - `dateString` - can be YYYY-MM-DD or a full datetime string
- **Returns:** Date object
- **Side effects:** None

Authentication & User Management

Auth Provider (`lib/auth-provider.tsx`)

`AuthProvider({ children })`

- **What it does:** React context provider that manages user authentication state and profile data
- **Parameters:**
  - `children` - the child components that need auth
- **Returns:** JSX component
- **Side effects:**
  - Fetches user session from Supabase
  - Loads user profile from database when user logs in
  - Updates state when auth changes

`useAuth()`

- **What it does:** Hook to get auth data from the context
- **Parameters:** None
- **Returns:** Object with:
  - `user` - current user object (or null)
  - `profile` - user profile data (or null)
  - `loading` - whether we're still loading
  - `signIn(email)` - function to sign in with email
  - `signOut()` - function to sign out

  - `updateProfile(updates)` - function to update profile in database
- **Side effects:** None (it's just reading from context)

User Data Provider (`lib/user-data-provider.tsx`)

`UserDataProvider({ children })`

- **What it does:** Context provider for all user data - transactions, budget categories, receipts, etc.
- **Parameters:**
  - `children` - child components
- **Returns:** JSX component
- **Side effects:**
  - Fetches transactions from database
  - Fetches budget categories
  - Fetches receipts
  - Manages AI insights

`useUserData()`

- **What it does:** Hook to access all the user data
- **Parameters:** None
- **Returns:** Object with:
  - `transactions` - array of transactions
  - `budgetCategories` - array of budget categories
  - `receipts` - array of receipts
  - `aiInsights` - AI insights object
  - `loading` - loading state
  - `addTransaction(transaction)` - adds transaction to local state
  - `updateTransaction(id, updates)` - updates transaction
  - `deleteTransaction(id)` - deletes transaction
  - `addBudgetCategory(category)` - adds budget category (also saves to DB)
  - `updateBudgetCategory(id, updates)` - updates category (also saves to DB)
  - `deleteBudgetCategory(id)` - deletes category (also saves to DB)
  - `refreshTransactions()` - refetches from database
  - `refreshBudgetCategories()` - refetches from database
- **Side effects:** None (read-only hook, but the functions it returns have side effects)

User Data Hooks

`useTransactions({ user })`

- **File:** `lib/user-data-provider/hooks/use-transactions.ts`
- **What it does:** Hook for managing transactions - fetching, adding, updating, deleting
- **Parameters:**
  - `user` - the current user object (or null)

- **Returns:** Object with:
  - `transactions` - list of transactions
  - `fetchTransactions()` - fetches from database
  - `refreshTransactions()` - similar to fetchTransactions just serves to "refresh" to fetch new data that may have been added, updated, or deleted
  - `addTransaction(transaction)` - adds to local state only
  - `updateTransaction(id, updates)` - updates in local state only
  - `deleteTransaction(id)` - removes from local state only
- **Side effects:**
  - `fetchTransactions()` queries the `transactions` table in Supabase
  - Updates local React state

`useBudgetCategories({ user, profile, transactions })`

- **File:** `lib/user-data-provider/hooks/use-budget-categories.ts`
- **What it does:** Hook for managing budget categories. Also calculates how much was spent in each category.
- **Parameters:**
  - `user` - current user
  - `profile` - user profile (needed for budget cycle dates)
  - `transactions` - list of transactions (to calculate spent amounts)
- **Returns:** Object with:
  - `budgetCategories` - list of categories with spent amounts
  - `fetchBudgetCategories()` - fetches from DB and calculates spent in each category
  - `refreshBudgetCategories()` - serves as a "refresh" fetch on new changed data
  - `addBudgetCategory(category)` - adds new category to DB
  - `updateBudgetCategory(id, updates)` - updates category in DB
  - `deleteBudgetCategory(id)` - deletes from DB
- **Side effects:**
  - `fetchBudgetCategories()` queries `budget_categories` table, calculates spent amounts, then updates the `spent_amount` field in the database
  - `addBudgetCategory()` inserts into `budget_categories` table, then updates the `spent_amount` field
  - `updateBudgetCategory()` updates `budget_categories` table, recalculates spent, then updates `spent_amount` again
  - `deleteBudgetCategory()` deletes from `budget_categories` table
  - There's also an internal `updateSpentAmountsInDB()` function that updates the `spent_amount` field in the database

`useReceipts({ user })`

- **File:** `lib/user-data-provider/hooks/use-receipts.ts`

- **What it does:** Hook for fetching receipts
- **Parameters:**
  - `user` - current user
- **Returns:** Object with:
  - `receipts` - list of receipts
  - `fetchReceipts()` - fetches from database
  - `refreshReceipts()` - same thing
- **Side effects:**
  - `fetchReceipts()` queries the `receipts` table

`useAIInsights({ userId })`

- **File:** `lib/user-data-provider/hooks/use-ai-insights.ts`
- **What it does:** Hook for getting AI insights about spending
- **Parameters:**
  - `userId` - user ID string
- **Returns:** Object with:
  - `insights` - AI insight object (or null)
  - `loading` - loading state
  - `fetchInsights()` - fetches insights from edge function
- **Side effects:**
  - `fetchInsights()` calls the `ai_insight` edge function

Category Spending Calculator (`lib/user-data-provider/utils/calculate-category-spent.ts`)

`calculateCategorySpent(categoryName, transactions, cycleStart, cycleEnd)`

- **What it does:** Calculates how much was spent in a specific category, optionally within a date range
- **Parameters:**
  - `categoryName` - name of the category to calculate
  - `transactions` - array of all transactions
  - `cycleStart` - optional start date (YYYY-MM-DD) for filtering
  - `cycleEnd` - optional end date (YYYY-MM-DD) for filtering
- **Returns:** number - total amount spent
- **Side effects:** None

Transaction Management

Transaction Utilities (`app/(app)/transactions/utils.ts`)

`formatCurrency(amount: number, currency: string): string`

- **What it does:** Formats a number as currency with the right symbol

- **Parameters:**
  - `amount` - the amount to format
  - `currency` - currency code or symbol (like "USD" or "$")
- **Returns:** formatted currency string
- **Side effects:** None

`convertTransactionToFormData(transaction: Transaction): TransactionFormData`

- **What it does:** Converts a transaction from the database format to the form data format we use in the UI
- **Parameters:**
  - `transaction` - transaction object from database
- **Returns:** transaction in form data format
- **Side effects:** None

Transaction Hooks

`useTransactionsFilters({ transactions })`

- **File:** `app/(app)/transactions/hooks/use-transactions-filters.ts`
- **What it does:** Hook for filtering and sorting the transaction list
- **Parameters:**
  - `transactions` - array of all transactions
- **Returns:** Object with:
  - `searchQuery` - current search text
  - `setSearchQuery(query)` - update search
  - `sortBy` - current sort field
  - `setSortBy(field)` - change sort field
  - `filterCategory` - selected category filter (or null)
  - `setFilterCategory(category)` - set category filter
  - `filteredAndSortedTransactions` - the filtered/sorted results
  - `clearFilters()` - clears all filters
  - `hasActiveFilters` - boolean if any filters are set
- **Side effects:** None (just local state)

`useTransactionEdit({ updateTransaction, refreshTransactions })`

- **File:** `app/(app)/transactions/hooks/use-transaction-edit.ts`
- **What it does:** Hook for editing transactions
- **Parameters:**
  - `updateTransaction` - function to update transaction in context
  - `refreshTransactions` - function to refresh the list
- **Returns:** Object with:
  - `isSaving` - whether we're currently saving
  - `handleSaveTransaction(transaction, formData)` - saves the edited transaction

- **Side effects:**
  - `handleSaveTransaction()` updates the `transactions` table in Supabase

`useTransactionsStats({ filteredTransactions })`

- **File:** `app/(app)/transactions/hooks/use-transactions-stats.ts`
- **What it does:** Calculates statistics from the transaction list
- **Parameters:**
  - `filteredTransactions` - the filtered transaction list
- **Returns:** Object with:
  - `totalSpent` - total amount spent
  - `transactionCount` - number of transactions
  - `averageTransaction` - average transaction amount
  - `categoryBreakdown` - object with spending by category
- **Side effects:** None

Budget Management

Budget Utilities (`app/(app)/budget/utils.ts`)

`calculateEndDate(startDate: string): string`

- **What it does:** Calculates the end date as start date + 30 days (for monthly budget cycles)
- **Parameters:**
  - `startDate` - start date in YYYY-MM-DD format
- **Returns:** end date as YYYY-MM-DD string
- **Side effects:** None

`getTodayISO(): string`

- **What it does:** Gets today's date in ISO format (YYYY-MM-DD) in local timezone
- **Parameters:** None
- **Returns:** today's date as string
- **Side effects:** None

Budget Hooks

`useBudgetState({ profile })`

- **File:** `app/(app)/budget/hooks/use-budget-state.ts`
- **What it does:** Manages the budget configuration state (cycle dates, currency, auto-renew)
- **Parameters:**
  - `profile` - user profile (to get current budget settings)
- **Returns:** Object with:
  - `profileBudget` - budget configuration object

    - `handleProfileBudgetChange(update)` - function to update budget config
- **Side effects:** None (just local state)

`useBudgetCategories({ budgetCategories, addBudgetCategory, updateBudgetCategory, deleteBudgetCategory, refreshBudgetCategories, userId })`

- **File:** `app/(app)/budget/hooks/use-budget-categories.ts`
- **What it does:** Manages budget categories in the budget page. Keeps local state and syncs with database.
- **Parameters:**
    - `budgetCategories` - list from context
    - `addBudgetCategory` - function to add category
    - `updateBudgetCategory` - function to update category
    - `deleteBudgetCategory` - function to delete category
    - `refreshBudgetCategories` - function to refresh
    - `userId` - user ID
- **Returns:** Object with:
    - `localCategories` - local state of categories
    - `handleCategoriesChange(updates)` - update local state
    - `saveCategories()` - save all changes to database
- **Side effects:**
    - `saveCategories()` updates the `budget_categories` table

`useBudgetCalculations({ localCategories })`

- **File:** `app/(app)/budget/hooks/use-budget-calculations.ts`
- **What it does:** Calculates budget progress and totals
- **Parameters:**
    - `localCategories` - list of categories
- **Returns:** Object with:
    - `categoryProgress` - array with progress data for each category
    - `totalLimit` - total budget limit across all categories
    - `totalSpent` - total amount spent
    - `totalCategoryLimit` - sum of all category limits
- **Side effects:** None

`useBudgetSave({ user, profile, profileBudget, updateProfile, saveCategories })`

- **File:** `app/(app)/budget/hooks/use-budget-save.ts`
- **What it does:** Handles saving the entire budget configuration (profile settings + categories)
- **Parameters:**
    - `user` - current user
    - `profile` - user profile

  - `profileBudget` - budget configuration
  - `updateProfile` - function to update profile
  - `saveCategories` - function to save categories
- **Returns:** Object with:
  - `isSaving` - saving state
  - `handleSaveAll()` - saves everything
- **Side effects:**
  - `handleSaveAll()` updates the `profiles` table with budget settings, then saves categories

---

Receipt Management

Receipt Utilities (`app/(app)/receipts/utils.ts`)

`formatDate(dateString: string): string`

- **What it does:** Formats date to readable string
- **Parameters:**
  - `dateString` - date string
- **Returns:** formatted date
- **Side effects:** None

`formatCurrency(amount: number, currency: string | null): string`

- **What it does:** Formats currency amount
- **Parameters:**
  - `amount` - the amount
  - `currency` - currency symbol or code (can be null)
- **Returns:** formatted currency string
- **Side effects:** None

`formatCategory(category: string | null): string`

- **What it does:** Capitalizes the first letter of category name
- **Parameters:**
  - `category` - category name (can be null)
- **Returns:** formatted category string
- **Side effects:** None

Receipt Hooks

`useReceipts({ user })`

- **File:** `app/(app)/receipts/hooks/use-receipts.ts`
- **What it does:** Hook for fetching receipts
- **Parameters:**
  - `user` - current user

- **Returns:** Object with:
  - `receipts` - list of receipts
  - `loading` - loading state
  - `refreshReceipts()` - refetches from database
- **Side effects:**
  - `refreshReceipts()` queries the `receipts` table

Scan & AI Processing

Scan Utilities (`app/(app)/scan/utils.ts`)

`transformTransactionToFormData(transaction)`

- **What it does:** Converts the transaction object we get from the edge function into the format our form uses. Also handles timezone conversion if the edge function sent UTC today.
- **Parameters:**
  - `transaction` - transaction object from edge function response
- **Returns:** TransactionFormData object or null
- **Side effects:** None

`buildTransactionPayload(formData, userId, currency, receiptId?)`

- **What it does:** Builds the payload object we send to Supabase when saving a transaction
- **Parameters:**
  - `formData` - form data from the UI
  - `userId` - user ID
  - `currency` - currency code
  - `receiptId` - optional receipt ID if it came from a receipt scan
- **Returns:** payload object for database
- **Side effects:** None

`validateTransactionForm(formData)`

- **What it does:** Validates the transaction form before saving
- **Parameters:**
  - `formData` - form data to validate
- **Returns:** error message string or null if valid
- **Side effects:** None

Scan Hooks

`useEdgeFunction({ onAutoSaveSuccess, onPreviewSuccess, onError })`

- **File:** `app/(app)/scan/hooks/use-edge-function.ts`
- **What it does:** Hook for calling the receipt scanning edge function

- **Parameters:**
  - `onAutoSaveSuccess` - callback when transaction is auto-saved
  - `onPreviewSuccess` - callback when we get preview data
  - `onError` - callback when there's an error
- **Returns:** Object with:
  - `callEdgeFunction(file, userId, autoSave)` - function to call the edge function
- **Side effects:**
  - `callEdgeFunction()` calls the `LLM_receipt` edge function
  - If autoSave is true, the edge function inserts into `transactions` and `receipts` tables

`useVoiceRecording({ onAutoSaveSuccess, onPreviewSuccess, onError })`

- **File:** `app/(app)/scan/hooks/use-voice-recording.ts`
- **What it does:** Hook for voice recording and processing
- **Parameters:**
  - `onAutoSaveSuccess` - callback for auto-save
  - `onPreviewSuccess` - callback for preview
  - `onError` - error callback
- **Returns:** Object with:
  - `isRecording` - whether we're currently recording
  - `recordingTime` - how many seconds we've been recording
  - `startRecording()` - starts the recording
  - `stopRecording()` - stops recording
  - `sendRecording(userId, autoSave)` - sends audio to edge function for processing
- **Side effects:**
  - `sendRecording()` calls the `ai_audio_scan` edge function
  - If autoSave is true, edge function inserts into `transactions` table

`useTransactionSave({ user, profile })`

- **File:** `app/(app)/scan/hooks/use-transaction-save.ts`
- **What it does:** Hook for saving transactions manually
- **Parameters:**
  - `user` - current user
  - `profile` - user profile (for currency)
- **Returns:** Object with:
  - `saveTransaction(formData, userId, currency, receiptId?)` - saves transaction to database
- **Side effects:**
  - `saveTransaction()` inserts into `transactions` table

useScanState()

- **File:** `app/(app)/scan/hooks/use-scan-state.ts`
- **What it does:** Manages all the state for the scan page (which tab, extracted data, etc.)
- **Parameters:** None
- **Returns:** Object with all the state and setters
- **Side effects:** None (just local state)

UI Components & Hooks

Global Hooks

useAuthGuard(options?)

- **File:** `hooks/use-auth-guard.ts`
- **What it does:** Protects routes - redirects to login if user isn't authenticated
- **Parameters:**
  - `options` - optional configuration
- **Returns:** Object with user, profile, and loading state
- **Side effects:** May redirect user if not logged in

useBudgetLimitToasts({ categories, totalLimit, totalSpent })

- **File:** `hooks/use-budget-limit-toasts.ts`
- **What it does:** Shows toast notifications when user exceeds budget limits
- **Parameters:**
  - `categories` - array of category progress data
  - `totalLimit` - total budget limit
  - `totalSpent` - total amount spent
- **Returns:** Nothing
- **Side effects:** Shows toast notifications

useToast()

- **File:** `hooks/use-toast.ts`
- **What it does:** Hook for showing toast notifications
- **Parameters:** None
- **Returns:** Object with `toast()` function
- **Side effects:** Displays toast notifications

Profile Hooks

useProfileForm({ profile, userEmail })

- **File:** `app/(app)/profile/hooks/use-profile-form.ts`
- **What it does:** Manages the profile form state and submission
- **Parameters:**

  - `profile` - current profile data
  - `userEmail` - user's email
- **Returns:** Object with form state and handlers
- **Side effects:**
  - `handleSubmit()` updates the `profiles` table in Supabase

Supabase Edge Functions

These are serverless functions that run on Supabase. They handle AI processing and some database operations.

Receipt Processing (`supabase-backend-edge-fn/LLM_receipt.ts`)

Edge Function Handler

- **What it does:** Takes a receipt image, uses Google Gemini API to extract transaction data from it
- **HTTP Method:** POST
- **Parameters (FormData):**
  - `receipt_file` - the image file
  - `user_id` - user ID string
  - `isAuto` - "true" or "false" for whether to auto-save
- **Returns:** JSON with:
  - `success` - boolean
  - `mode` - "auto" or "preview"
  - `transaction` - transaction object (if auto) or payload (if preview)
  - `receipt_id` - receipt ID if it was saved
- **Side effects:**
  - If `isAuto === "true"`: inserts into `transactions` table and `receipts` table
  - Uploads image file to Supabase Storage
  - Calls Google Gemini API for OCR and data extraction

Voice Processing (`supabase-backend-edge-fn/ai_audio_scan.ts`)

Edge Function Handler

- **What it does:** Takes audio recording, transcribes it with OpenAI Whisper, then extracts transaction data with Google Gemini
- **HTTP Method:** POST
- **Parameters (FormData):**
  - `audio_file` - the audio file
  - `user_id` - user ID string
  - `isAuto` - "true" or "false" for auto-save
- **Returns:** JSON with:

  - `success` - boolean
  - `mode` - "auto" or "preview"
  - `transaction` - transaction object or payload
- **Side effects:**
  - If `isAuto === "true"`: inserts into `transactions` table
  - Calls OpenAI Whisper API for transcription
  - Calls Google Gemini API for data extraction

AI Insights (`supabase-backend-edge-fn/ai_insight.ts`)

Edge Function Handler

- **What it does:** Generates AI insights about user's spending patterns
- **HTTP Method:** POST
- **Parameters (JSON):**
  - `user_id` - user ID string
- **Returns:** JSON with:
  - `success` - boolean
  - `insights` - object with `summary` and `tip` strings
- **Side effects:**
  - Queries `transactions` table to get spending data
  - Queries `budget_categories` table
  - Calls Google Gemini API to generate insights

Budget Categories (`supabase-backend-edge-fn/budget-categories.ts`)

Edge Function Handler

- **What it does:** Handles budget category operations
- **HTTP Method:** POST
- **Parameters:** Varies depending on operation
- **Returns:** JSON response
- **Side effects:**
  - CRUD operations on `budget_categories` table

Profile Management (`supabase-backend-edge-fn/upsert-profile.ts`)

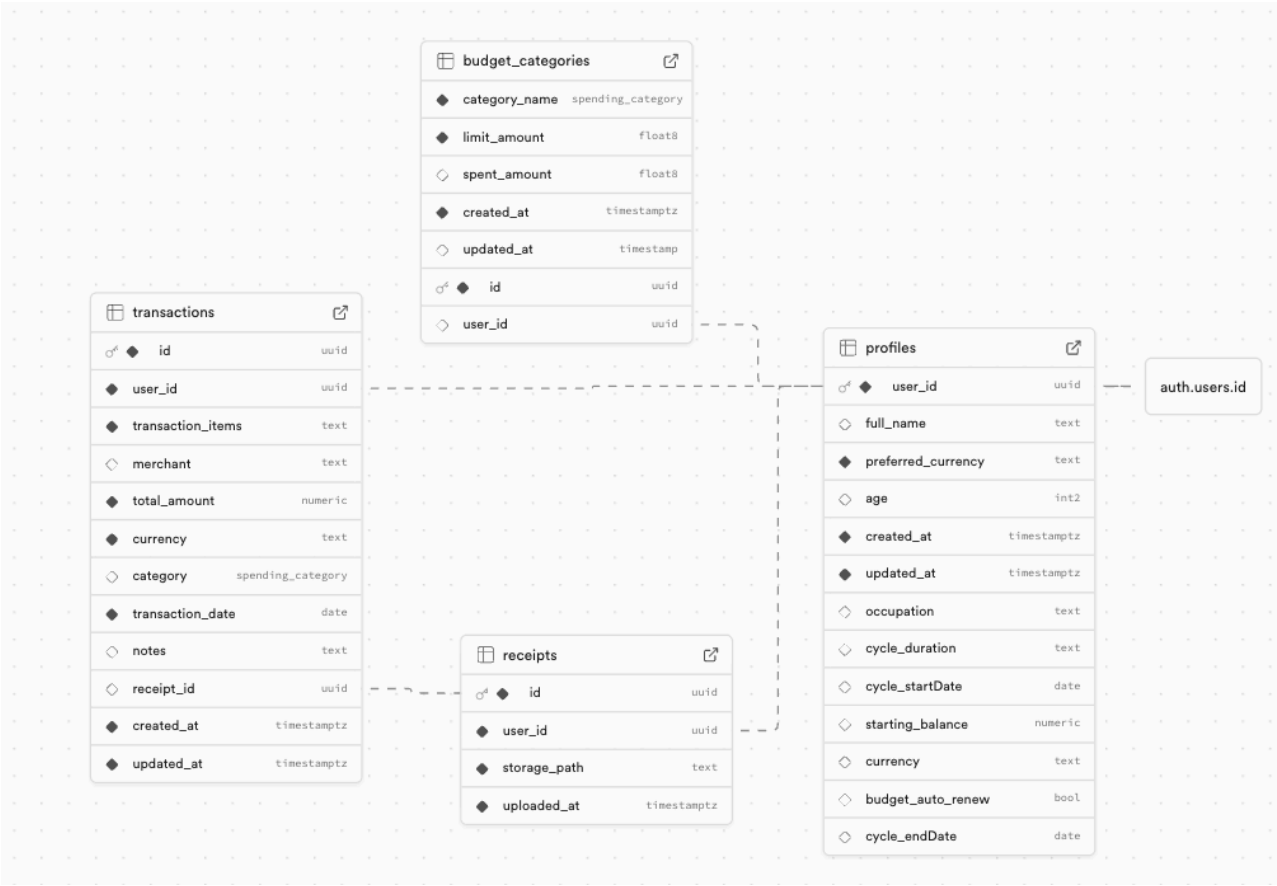
Edge Function Handler

- **What it does:** Creates or updates user profile
- **HTTP Method:** POST
- **Parameters (JSON):**
  - `user_id` - user ID
  - `profile_data` - profile data object
- **Returns:** JSON with profile data
- **Side effects:**

- Inserts or updates `profiles` table

Database Operations Summary

Here's which functions modify which database tables:



`profiles` table

- Updated by: `updateProfile()` in auth provider, `useBudgetSave.handleSaveAll()`, `upsert-profile` edge function
- Fields updated: `cycle_startDate`, `cycle_endDate`, `preferred_currency`, `budget_auto_renew`, `full_name`, etc.

`transactions` table

- Inserted by: `saveTransaction()`, `useTransactionEdit.handleSaveTransaction()`, `LLM_receipt` edge function (if auto-save), `ai_audio_scan` edge function (if auto-save)
- Updated by: `useTransactionEdit.handleSaveTransaction()`
- Queried by: `useTransactions.fetchTransactions()`, `useBudgetCategories` (to calculate spent), `calculateCategorySpent()`, `ai_insight` edge function

`budget_categories` table

- Inserted by: `useBudgetCategories.addBudgetCategory()`
- Updated by: `useBudgetCategories.updateBudgetCategory()`, `useBudgetCategories.updateSpentAmountsInDB()`, `useBudgetCategories.addBudgetCategory()` (updates spent_amount after insert)
- Deleted by: `useBudgetCategories.deleteBudgetCategory()`
- Queried by: `useBudgetCategories.fetchBudgetCategories()`

`receipts` table

- Inserted by: `LLM_receipt` edge function
- Queried by: `useReceipts.fetchReceipts()`

Supabase Storage

- Files uploaded by: `LLM_receipt` edge function (receipt images are stored here)

Notes

- The Supabase Edge Functions and database setup were implemented by us directly within the Supabase platform. The edge function files included in the code repository are provided for reference only, since the functions are deployed, managed, and executed through Supabase rather than run directly from the application code.
- All date operations use local timezone to avoid UTC conversion issues (we had bugs with this earlier)
- Transaction dates are stored as YYYY-MM-DD strings
- All database operations use Supabase client with Row Level Security (RLS) enabled
- Edge functions require authentication via Supabase session token in the Authorization header
- AI processing (Gemini, Whisper) happens server-side in edge functions
- All hooks follow React hooks rules and use proper dependency arrays
- We use React Context for global state management (auth and user data)