Product Requirements Document (PRD)

1. Executive Summary

What is this app?

Findash (branded in-app as **TS Personal Finance**) is a personal finance dashboard that aggregates account balances, transactions, budgets, and net worth from a Google Sheet (one per user), syncs them into Supabase, and presents them via a multi-page web app with charts, trend analysis, forecast evolution, and an AI assistant for natural-language queries. The app is **multi-tenant**: each user has isolated data and optionally their own Google Sheet.

Who is the primary user?

The primary user is an individual or household (e.g. family/trust) who maintains financial data in a Google Sheet and wants a single place to view net worth, budget vs actual, spending trends, cash runway, year-over-year net worth changes, and "why did my forecast change?"—with optional chat-based analysis. **Any** Google account can sign in; data is isolated per user via Row Level Security (RLS). New users connect their sheet in Settings before syncing.

---

2. Technical Architecture

Layer	Technology
Frontend	Next.js 16 (App Router), React 18, Tailwind CSS
Backend / Data	Supabase (PostgreSQL), server-side Supabase client via `@supabase/ssr`
Auth	Supabase Auth with Google OAuth; post-login redirect to `/insights`; auth callback creates/upserts `user_profiles` (id, email). No allowlist—any Google user can sign in; RLS isolates data per user.
Routing / Middleware	Next.js middleware (`proxy.ts`) enforces auth on all routes except `/login` and `/api/cron/*`; cron routes require `Authorization: Bearer <CRON_SECRET>`. Signed-in users visiting `/login` are redirected to `/insights`.

Key libraries

1. **AI SDK (`ai`, `@ai-sdk/react`, `@ai-sdk/google`)** – Streaming chat with tool use; model: Gemini 2.5 Flash. Powers the in-app "Financial Assistant" chat widget with tools for snapshots, spending, budget vs actual, and forecast evolution.
2. **Recharts** – All charts: net worth over time, income vs expenses, cumulative spend, annual cumulative spend, YoY net worth waterfall, forecast evolution (bridge) waterfall, forecast gap over time (line).
3. **googleapis** – Google Sheets API; used by `lib/sync-google-sheet.ts` to pull data into Supabase (account balances, transactions, budget targets, historical net worth, FX rates, trends, recurring payments, kids accounts, investment return, YoY net worth).
4. **Supabase (`@supabase/supabase-js`, `@supabase/ssr`)** – Database client, auth, and cookie-based session handling in server and client components.
5. **Zod** – Input validation and schema for AI tool parameters (chat API) and type-safe config.

Other notable deps: `lucide-react` (icons), `react-markdown` + `remark-gfm` (chat responses), `date-fns`, `sonner` (toasts), Radix UI primitives (dialog, checkbox, progress, etc.).

---

3. Data Model & Schema

Core entities are defined in `supabase/migrations/`. The app is **multi-tenant**: all user-specific tables include `user_id` (UUID, references `auth.users`). RLS policies restrict access to `user_id = current_user_id()`. FX tables (`fx_rates`, `fx_rate_current`) are global (no `user_id`).

User & config

Table	Purpose
user_profiles	One row per user: `id` (PK, references `auth.users`), `email`, `google_spreadsheet_id`, `display_name`, `default_currency`, `created_at`, `updated_at`. Created/updated on login; user sets `google_spreadsheet_id` and optional `display_name`/`default_currency` in Settings. `default_currency` (USD/GBP, default USD) is the display currency the app opens with; CurrencyProvider loads from localStorage then profile. Cron and manual sync use `google_spreadsheet_id` to know which sheet to sync per user.

Transactions & spending

Table	Purpose
transaction_log	Per-user transactions: `user_id`, `date`, `category`, `counterparty`, `amount_usd`, `amount_gbp`, `currency`, `counterparty_dedup`. Unique per user; RLS by `user_id`.
fx_rates	Historical FX (global): `date` (PK), `gbpusd_rate`, `eurusd_rate`.
fx_rate_current	Current GBP/USD (global): single row per date.

Budgets & forecast

Table	Purpose
budget_targets	Per-user, one row per category: `user_id`, `category`, `annual_budget_gbp/usd`, `tracking_est_gbp/usd`, `ytd_gbp/usd`. Unique on `(user_id, category)`.
budget_history	Per-user daily snapshots: `user_id`, `date`, `category`, `annual_budget`, `forecast_spend`, `actual_ytd`; unique on `(user_id, date, category)`.

Net worth & balances

Table	Purpose
account_balances	Per-user: `user_id`, `date_updated`, `institution`, `account_name`, `category`, `currency`, balance columns. Unique on `(user_id, institution, account_name, date_updated)`.
historical_net_worth	Per-user: `user_id`, `date`, `category`, `amount_usd`, `amount_gbp`; unique on `(user_id, date, category)`.

yoy_net_worth	Per-user: `user_id`, `category`, `amount_usd`, `amount_gbp`; unique on `(user_id, category)`.

Trends & derived

Table	Purpose
annual_trends	Per-user: `user_id`, `category`, `cur_yr_minus_4` … `cur_yr_est`, `cur_yr_est_vs_4yr_avg`; unique on `(user_id, category)`.
monthly_trends	Per-user: `user_id`, `category`, monthly columns; unique on `(user_id, category)`.
investment_return	Per-user: `user_id`, `income_source`, `amount_gbp`; unique on `(user_id, income_source)`.

Recurring & kids

Table	Purpose
recurring_payments	Per-user: `user_id`, `name`, annualized amounts, `needs_review`; unique on `(user_id, name)`.
recurring_preferences	Per-user: `user_id`, `counterparty_pattern`, `is_ignored`; unique on `(user_id, counterparty_pattern)`.
kids_accounts	Per-user: `user_id`, `child_name`, `account_type`, `balance_usd`, `date_updated`, `notes`, `purpose`; unique on `(user_id, child_name, account_type, date_updated, notes)`.

Sync metadata

Table	Purpose
sync_metadata	One row per user: `user_id` (unique), `last_sync_at` (timestamptz). Updated via `recordLastSync(supabase, userId)` after each successful sync. Header "Last Refresh" reads the current user's row (RLS scopes to `user_id`).

RPCs & RLS

Function	Purpose
get_cash_runway_net_burn(p_start, p_end)	Returns `gbp_net`, `usd_net`: net burn per currency for date range. RLS on `transaction_log` restricts to current user's rows. Used by `GET /api/cash-runway`.
current_user_id()	STABLE SECURITY DEFINER function returning `auth.uid()`; used in RLS policies so the planner evaluates it once per query (Performance Advisor–friendly).

---

4. Core Feature Specifications

4.0 Login (`/login`)

- **Purpose:** Unauthenticated users are redirected here by middleware; after sign-in, auth callback redirects to `/insights` (or back to `/login` with an error).
- **Flow:** Google sign-in button; redirects to Supabase OAuth then `/auth/callback`. Callback exchanges code for session, then upserts `user_profiles` (id, email). No allowlist—any Google user can sign in. If code exchange fails, redirect to `/login?error=auth_code_error`.
- **Already signed in:** If any signed-in user visits `/login`, middleware redirects to `/insights`.

4.1 Dashboard (`/`)

- **At a glance:** Executive summary (e.g. net worth, budget gap, key KPIs); mobile uses horizontal scroll carousel.
- **Net worth chart:** Line chart of historical net worth (with optional entity filters: Personal/Family/Trust); mobile: reduced ticks and compact Y-axis.
- **Income vs expenses chart:** Budget vs tracking vs YTD; optional investment return; mobile: toggles can be hidden.
- **Budget table:** Categories with annual budget, tracking (forecast), YTD actual; variance and over/under budget. Optional **Full table view** toggle on Expenses: opens the expense tables in a full-screen overlay scaled to fit. See `docs/COMPACT-DATA-GRID.md`.
- **Annual trends table:** Current year vs prior years by category (GBP/USD via FX). Optional **Full table view** toggle: opens the table in a full-screen overlay scaled to fit (no scrolling).
- **Monthly trends table:** Current month vs prior months, TTM avg, z-score, delta vs last 3 months. Optional **Full table view** toggle (same behavior as Annual).
- **Navigation:** In-page anchors (Net Worth, Budget Table, Annual Trends, Monthly Trends) and “Back to top.”

4.2 Key Insights (`/insights`)

- **Key Insights:** Combined view of budget targets, annual/monthly trends, historical net worth, and latest account balances. Includes charts (e.g. net worth over time, pie by category), progress indicators, and “at a glance” style cards. Currency toggle (GBP/USD) and mobile-friendly layout.

4.3 Accounts (`/accounts`)

- **Accounts overview:** List/cards of account balances (by institution, account name, category, currency). Latest balance per account; optional grouping. Mobile: card layout instead of table.

4.4 Kids Accounts (`/kids`)

- **Kids accounts overview:** Balances by child and account type (USD), with notes and purpose. Data sourced from Google Sheet and synced into `kids_accounts`.

4.5 Recurring (`/recurring`)

- **Recurring payments:** Table and cards of recurring items (name, annualized amount, needs review). Data from sheet `recurring_payments`.
- **Recurring preferences:** Support for marking counterparty patterns as ignored (not recurring).

4.6 Settings (`/settings`)

- **Connect your sheet:** User sets `google_spreadsheet_id` (and optional `display_name`) in `user_profiles`. Required before manual sync or cron can pull data for that user. Sidebar link to Settings.
- **Default currency:** User can set display currency (GBP or USD) in Settings; stored in `user_profiles.default_currency`. The app opens with this currency on login

(CurrencyProvider loads from localStorage, then user profile); new users default to USD. Changing the currency toggle elsewhere in the app persists the choice to `user_profiles` and localStorage.

- **Manual sync:** If `google_spreadsheet_id` is null, `POST /api/sync` returns 400 with "Connect your sheet first"; user is directed to Settings. Saving settings with a spreadsheet ID triggers a sync after save.

4.7 Analysis (`/analysis`)

- **Navigation:** In-page navigation (AnalysisNavigation) with anchor links to: Cash Runway, Transaction Analysis, Forecast Evolution, YTD Spend Over Time, Annual Cumulative Spend, YoY Net Worth Change, Monthly Trends by Category. Hash or `section` query param scrolls to the section (AnalysisHashScroll), e.g. `/analysis#forecast-evolution` or `/analysis?section=transaction-analysis` . Supports deep links from Dashboard (e.g. trends links with `section` , `period` , `year` , `month` , `category`). Transaction Analysis accepts optional URL params: `section` , `period` (YTD/MTD), `year` , `month` , `category` to pre-fill filters.
- **Cash runway:** Cards/metrics showing how long funds last at current burn (based on balances and spending). Net burn (last 3 full calendar months) from `GET /api/cash-runway` , which calls RPC `get_cash_runway_net_burn` (expenses + refunds by currency; category excludes Income, Excluded, Gift Money).
- **Transaction analysis:** Filter by period (YTD/month) and category; view transactions and category totals.
- **Forecast evolution:**
  - **Compare to:** Dropdown (Yesterday, Last Week, Last Month) to choose start date; end date defaults to today.
  - **Forecast bridge chart:** Waterfall (stacked bar "invisible spacer" technique): Start total → category-level drivers (forecast deltas) → End total. Green = forecast down (gap improved), red = forecast up (gap worsened). Data from `budget_history` via `/api/forecast-bridge` .
  - **Forecast gap over time:** Line chart of total budget gap (annual_budget – forecast_spend, expense categories only) over the selected date range. Data from `budget_history` via `GET /api/forecast-gap-over-time?startDate=&endDate=` .
  - **Logic:** For start date and end date, load snapshots from `budget_history` ; compute per-category `Spend_Delta = End_Forecast - Start_Forecast` ; sort by absolute delta; top drivers (e.g. top 5 + Other) drive the waterfall.
- **YTD spend over time:** Cumulative spend chart (e.g. by category) over the year.
- **Annual cumulative spend:** Multi-year cumulative spend vs budget (optional year toggles; mobile may show fewer lines by default).
- **YoY net worth change:** Start/end chart and YoY net worth waterfall (income, expenses, transfers, etc.) from `yoy_net_worth` .
- **Monthly Trends by Category:**
  - **Monthly Category Summary:** Summary cards showing latest month spending vs. L3M Avg, L12M Avg, and LY (Last Year) for the selected category, broken down by top transaction and other spending. Includes a category selector dropdown to choose which category to analyze.
  - **Monthly Trends Chart:** Stacked bar chart showing monthly spending trends for the selected category. Each bar represents a month and is split into two segments: the top transaction (counterparty with highest spending) for that month, and the rest of the category ("Other"). Highlights the latest month. Data from `transaction_log` ; shows last 13 months of trends. Category selection is controlled by the selector in the Monthly Category Summary section above.

4.8 Chat / AI Assistant

- **Entry:** Floating chat button (mobile: above bottom nav); opens modal "Financial Assistant." Auth timeout: `AuthTimeoutProvider` (inactivity 5 min or tab hidden 5 min) signs out and redirects to `/login`.
- **API:** `POST /api/chat` with AI SDK `streamText`, model Gemini 2.5 Flash; multi-step tool use (`maxSteps: 5`).
- **Tools:**
  1. **get_financial_snapshot** – Current or historical (`asOfDate`) net worth/balances; optional groupBy (currency, category, entity) and entity filter (Personal/Family/Trust). Uses `historical_net_worth` or `account_balances`.
  2. **analyze_spending** – Transactions over a date range; optional merchant, category, type (expenses/income/all), groupBy (category, merchant, month). Uses `transaction_log`; excludes non-expense categories unless requested.
  3. **get_budget_vs_actual** – Budget vs actual (YTD or annual) by category; over/under budget. Uses `budget_targets` and `transaction_log`.
  4. **analyze_forecast_evolution** – How forecasted annual spend (and thus budget gap) changed between two dates. Uses `budget_history` (with fallback: latest date ≤ endDate, then `budget_targets`). Computes per-category `Spend_Delta = End_Forecast − Start_Forecast`; sorts by |delta|; returns total forecast change, gap impact direction, and drivers (all in GBP); summary can be in GBP or USD via current FX.
- **System prompt:** Date context (today, "last month", "this year", etc.), capabilities (snapshots, spending, budget performance, forecast evolution), and rules (always use tools, format currency, no raw JSON).
- **UX:** Markdown responses, loading states, clear chat; errors surfaced in UI.

---

5. Integrations & External Services

5.1 Google Sheets

- **Role:** Per-user source of truth for balances, transactions, budgets, net worth, FX, trends, recurring, kids, investment return, YoY net worth. Each user's sheet ID is stored in `user_profiles.google_spreadsheet_id` (set in Settings). No direct user editing in the app; data is read from the sheet via Google Sheets API.
- **Config:** Google API credentials (service account) for the Sheets API. Spreadsheet ID is per-user from `user_profiles`, not from env.
- **Flow:**
  - **Sync:** `syncGoogleSheet(supabase, { spreadsheetId, userId })` reads the given sheet's ranges/tabs, maps rows to table columns, and upserts into Supabase with `user_id` on every row (except global `fx_rates`, `fx_rate_current`). After a successful sync, `recordLastSync(supabase, userId)` updates `sync_metadata` for that user; `snapshotBudgetHistory(date, supabase, userId)` snapshots that user's `budget_targets` into `budget_history`.
  - **Triggers:** (1) **Cron:** `GET|POST /api/cron/refresh` (06:00 UTC, `CRON_SECRET`) lists `user_profiles` where `google_spreadsheet_id IS NOT NULL`, then for each user runs sync, snapshot, and `recordLastSync(admin, user.id)`. (2) **Manual:** `POST /api/sync` (auth required) reads current user's `google_spreadsheet_id`; if null, returns 400 "Connect your sheet first". Otherwise runs sync, snapshot, and `recordLastSync(supabase, user.id)`.

- **Sheets → tables:** Same mapping as before (Account Balances → `account_balances`, etc.); all user tables receive `user_id` on each row.

5.2 Supabase

- **Auth:** Google OAuth; session in cookies; middleware requires authenticated user on protected routes; auth callback creates/upserts `user_profiles` and redirects to `/insights`. No allowlist.
- **Database:** All user-specific tables have RLS with `user_id = current_user_id()` (see migration 018/019). Cron and sync use service role (admin) client to write for any user; app reads use anon key with session so RLS scopes to current user.

5.3 Google AI (Gemini)

- **Role:** Chat model for the Financial Assistant (`@ai-sdk/google`, `google('gemini-2.5-flash')`).
- **Flow:** User message → `/api/chat` → `streamText` with tools → tool executions (Supabase reads) → model summarizes in natural language; response streamed to the client.

5.4 Vercel (or similar)

- **Cron:** `vercel.json` defines a daily cron for `/api/cron/refresh` at `0 6 * * *` (06:00 UTC). Caller must send `Authorization: Bearer <CRON_SECRET>`.

5.5 API routes summary

Route	Method	Purpose
`/api/chat`	POST	Financial Assistant; AI SDK `streamText`, tools, Gemini 2.5 Flash.
`/api/cron/refresh`	GET/POST	Cron-only; loops `user_profiles` with non-null `google_spreadsheet_id`, runs sync + snapshot + `recordLastSync` per user. Secured by `CRON_SECRET`.
`/api/sync`	POST	Manual refresh; reads current user's `google_spreadsheet_id` from `user_profiles`; if null returns 400. Otherwise runs sync, snapshot, `recordLastSync` for that user. Requires auth.
`/api/forecast-bridge`	GET	Forecast evolution waterfall data; query params for start/end dates.
`/api/forecast-gap-over-time`	GET	Forecast gap (annual_budget – forecast_spend) per date in range; `startDate`, `endDate`.
`/api/cash-runway`	GET	Net burn (GBP/USD) for last 3 full calendar months via RPC `get_cash_runway_net_burn`. Requires auth.

---

6. Unknowns / Areas for Improvement

- **Allowlist (optional):** `lib/allowed-emails.ts` reads `ALLOWED_EMAILS` (comma-separated) from env but is **not** used by the app today; any Google user can sign in and RLS isolates data. Optional allowlist behavior for a closed beta is described in `docs/SCALING-MULTI-USER.md`.
- **FX fallbacks:** Multiple places use a hardcoded GBP/USD fallback when `fx_rate_current` is missing or zero (e.g. `1.27` in chat route and currency context, `1.25` in some trend wrappers). Consider a single shared constant or config and document the source (e.g. "last known rate" or "default for display only").

- **Google Spreadsheet ID:** Per-user in `user_profiles.google_spreadsheet_id`; set in Settings. If unset, sync returns 400 and user is directed to Settings.
- **Cron secret:** If `CRON_SECRET` is not set, all cron requests are rejected (401). Document in README/deploy docs so Vercel (or other) cron is configured with the header.
- **Budget history coverage:** Forecast Evolution and chat "forecast evolution" depend on `budget_history` being populated (cron or manual refresh). If the sheet is never synced or history is sparse, comparisons may fail or return "no data"; consider empty-state messaging and prompting user to run Refresh Data.
- **sync_metadata:** Per-user; if no sync has run yet for that user, no row exists; header "Last Refresh" uses `.maybeSingle()` and should handle empty state (e.g. "Never" or hide the label).
- **Mobile layout:** Several charts and tables switch to card layout or hide toggles on small screens; regression testing on real devices is recommended.
- **Error handling:** Some API and sync paths return generic messages; consider structured error codes or user-facing messages for quota, auth, and "no data" cases.
- **Types:** Some Supabase responses are cast (e.g. `as BudgetTarget[]`); shared types or codegen from schema could reduce drift.
- **No automated tests referenced in repo:** Adding unit tests for sync mapping, forecast-bridge logic, and chat tool execution would help prevent regressions.

Document generated from codebase scan. Last updated: default_currency (user_profiles, migration 020), Settings default currency and post-save sync; Analysis in-page navigation (AnalysisNavigation), hash/query scroll (AnalysisHashScroll), and URL params for transaction analysis; allowlist noted as optional/unused.