# Executive Summary

The PRISM cognitive profiling assessment system currently spans a React/Tailwind frontend and a Supabase backend (Edge Functions + Postgres). Scoring is implemented server-side via Supabase Edge Functions (score\_prism for Likert and combined results, score\_fc\_session for forced-choice) and results are stored in a profiles table[[1]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1-L6)[[2]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L29-L38). However, the scoring logic is fragmented: parts of the logic (e.g. forced-choice tally, confidence calibration) are duplicated or split across front-end and multiple backend functions[[3]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L373-L382)[[4]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L66-L74). This audit finds timing misalignments (e.g. scoring triggered before all responses collected), and outdated fallbacks for forced-choice in the score\_prism function[[3]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L373-L382). To address these issues, we propose a **unified scoring engine** module (versioned, deterministic) that both the front-end and backend can use, ensuring consistency across Likert, forced-choice, overlay trait calculations, and confidence calibration. All scoring will remain server-side (non-negotiable) for security, but the front-end will leverage the shared module for type definitions, validation, or simulation if needed. We will strengthen security by requiring share-token for result access (removing broad read policies)[[5]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L17-L25) and ensuring Row-Level Security (RLS) rules are least-privilege. This report provides a comprehensive plan: an inventory of current routes and components, a detailed design for the unified scoring engine (covering inputs, outputs, algorithms, calibration, and versioning), an architecture diagram, and a step-by-step implementation roadmap. We include risk mitigations (e.g. race conditions, RLS verification, rollback strategy) and a testing plan to ensure the live site can be updated with minimal disruption. The outcome will be an end-to-end operational PRISM assessment feature that is deterministic, secure, easier to maintain, and observable with proper logging and metrics.

# System Inventory

**Frontend** – The React/TypeScript single-page app (Vite) implements the assessment UI and results display. Key routes (via React Router) include:

* **/assessment** – Entry to the PRISM assessment form (component: <AssessmentForm> inside <Assessment> page)[[6]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L163-L171)[[7]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L2-L5). Here users answer Likert and forced-choice questions. The form logic saves each response to Supabase and tracks progress.
* **/results/:sessionId** – Results page displaying the scored profile. It expects a sessionId (UUID for the assessment session) and an optional token query param for anonymous access[[7]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L2-L5)[[8]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L126-L134). The Results page calls a Supabase Edge Function get-results-by-session to fetch the final profile securely (more below)[[9]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L26-L34)[[10]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L41-L48).
* **/assessment?resume=<id>** – Resume incomplete assessment. If a session is resumed via query param, the app checks how many responses exist and potentially redirects to results if already complete[[11]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L34-L42).
* **Other routes** (read-only or informational): e.g. /history, /dashboard, and various content pages (About, FAQ, etc.) as listed in the router[[12]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L10-L20)[[13]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L34-L42). These are mostly static or unrelated to scoring logic. The PRISM assessment feature specifically involves the Assessment and Results pages, plus UI components like **ResultsV2**, **TraitPanel**, **OverlayChips** that format the output (which we will preserve).

**Frontend State & Components** – Within the assessment flow, <AssessmentForm> manages question display and saving of answers. It loads the question library and creates/uses a session ID. On each answer, it calls saveResponseIdempotent to upsert the response to the database[[14]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L701-L710). It updates local state and also updates the assessment\_sessions record (e.g. to save an email if provided on the first question, or increment completed question count for progress)[[15]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L724-L733)[[16]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L770-L778). After the last question, client-side validation runs to check for inconsistencies or invalid patterns (using validatePrismAssessment)[[17]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L793-L801). If critical validation fails, submission is blocked and the user is warned[[18]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L796-L804). If validation passes, the form’s onComplete handler is invoked with all responses and sessionId[[19]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L860-L868).

* In the **current implementation**, onComplete simply calls the score\_prism edge function and navigates to the results page[[20]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L18-L26). We also see a newer AssessmentComplete component that calls a finalizeAssessment function (Edge Function) to handle completion[[21]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L99-L108)[[22]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L124-L132) – this appears to be an updated flow that wraps scoring and session finalization (discussed below).

**Backend (Supabase)** – The Supabase project provides persistence and secure server logic:

* **Database Tables**: Key tables include:
* assessment\_sessions – Tracks each assessment attempt (session). Contains id (UUID), optional user\_id (if logged in), started\_at, completed\_at, status, completed\_questions, and a share\_token for secure result sharing[[23]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L2-L11)[[24]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L14-L22). RLS is enabled[[25]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L36-L44). Initially, policies allowed broad select for ease[[26]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L46-L55), but this was later tightened with share tokens. The share token is a random UUID string stored per session[[27]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L2-L10). All sessions have a unique share\_token (backfilled for old records)[[28]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L4-L12), enforced non-null and unique[[29]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L13-L21).
* assessment\_responses – Stores each answer. Fields: id (UUID), session\_id (FK to sessions), question\_id, question\_text, question\_type, section, answer\_value (text), answer\_numeric, response\_time\_ms, etc[[30]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L16-L25). RLS enabled with policies allowing inserts and selects (initially open since assessment was anonymous)[[31]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L59-L68). This table holds both Likert and forced-choice answers. (In legacy mode, forced-choice questions were just entries here with a special question\_type and possibly stored as e.g. “A/B/C” in answer\_value.)
* assessment\_scoring\_key – Static reference data mapping question\_id to scoring info (e.g. which cognitive function or trait the question measures, whether it’s reversed, any forced-choice mapping, etc.). In code, this is fetched entirely for scoring[[32]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L229-L238)[[33]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L239-L248). Each row may have JSON fields: fc\_map (for forced-choice mapping from option to function/block) and meta (extra metadata)[[33]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L239-L248).
* scoring\_config – Key-value store for various tuning parameters. The score\_prism function reads configuration like results\_version, dim\_thresholds (thresholds for dimensional counts), neuro\_norms (norm mean/sd for neuroticism), fc\_block\_map\_default (fallback mapping for forced choice), state\_qids (question IDs for state variables like stress, focus), fc\_expected\_min (min number of FC questions expected), softmax\_temp (temperature for softmax in type probability)[[34]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L255-L263). Also confidence calibration parameters (conf\_raw\_params) may be stored here[[35]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L51-L59).
* profiles – Stores completed profile results (the output of scoring) for each session. Key fields: session\_id (FK, unique per profile)[[36]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250908000000_profile_session_constraints.sql#L6-L14)[[37]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250908000000_profile_session_constraints.sql#L16-L24), user\_id, type\_code (e.g. “LII”, “IEE” etc), base\_func & creative\_func (the two primary functions for that type), confidence (High/Moderate/Low result confidence)[[38]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1032-L1040)[[39]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1050-L1058), validity\_status (pass/warning/fail based on quality checks)[[40]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1032-L1038), numeric scores like score\_fit\_raw and score\_fit\_calibrated (fit scores before and after cohort calibration)[[41]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1039-L1047), fit\_band (High/Moderate/Low fit category)[[42]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1041-L1045), top\_gap (difference between top two type fits)[[43]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1042-L1045), close\_call (boolean if top\_gap < threshold)[[44]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L875-L880)[[45]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L912-L919), fc\_answered\_ct (how many forced-choice blocks answered)[[46]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1046-L1054), overlay (e.g. “+” or “–” indicating neuroticism overlay)[[47]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1076-L1084)[[48]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L674-L682), and many JSON fields: strengths (strength score per function), dimensions (count of high responses per function), top\_types (array of top 3 type codes), top\_3\_fits (their scores)[[49]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1051-L1059)[[50]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1080-L1088), validity (detailed validity metrics)[[51]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L702-L710)[[50]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1080-L1088), and more. The profile is inserted or upserted by score\_prism upon completion[[52]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1125-L1133)[[53]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1143-L1150). RLS on profiles originally allowed broad read for analytics[[54]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819022459_12718b6c-29ce-46d9-8156-aa05e6f29eb1.sql#L4-L12). This was revised by dropping the open policy[[5]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L17-L25) and instead providing safe access via a secure function (below). Now, direct selects on profiles by anon are removed; users retrieve their profile either by being the authenticated owner or by share token.
* Other tables: fc\_blocks, fc\_options, fc\_responses, fc\_scores – These support the new forced-choice system. **fc\_blocks** defines each forced-choice block (e.g. a scenario or a pair of statements) with an id, code, title, etc. **fc\_options** holds the options for each block (each option with text and a weights\_json mapping that option to certain functions or types)[[55]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L46-L54)[[56]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L59-L67). **fc\_responses** stores which option a user chose for each block (session\_id, block\_id, option\_id)[[57]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L112-L120). **fc\_scores** stores the computed aggregate scores from forced-choice for a session (scores\_json field containing either function scores 0–100 or type probabilities, depending on basis)[[58]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L299-L308)[[59]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L312-L321). These tables enable a more structured FC implementation (v1.1+).
* Calibration tables: calibration\_model – Holds parameters for confidence calibration curves per stratum (combination of dimensional band and overlay)[[60]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L96-L105). The PrismCalibration class looks up these records to apply isotonic/Platt scaling to raw confidence[[61]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L98-L107)[[62]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L120-L128).
* Views: The system uses some convenience views: e.g., v\_fc\_coverage to compute how many forced-choice questions were answered per session and categorize “None/Partial/Full” coverage[[63]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql#L10-L19)[[64]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql#L22-L25), and analytics views like v\_recent\_assessments\_safe (for recent 7-day stats) that are exposed via a safe function[[65]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L30-L39)[[66]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L62-L70). There is also v\_sessions summarizing session durations and completion status[[67]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L6-L14)[[68]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L16-L24).
* RPC Functions: Several Postgres functions support secure access and calculations:
  + get\_profile\_by\_session(session\_id, share\_token) – Security-definer function that returns a profile row if the session’s share\_token matches the provided token and the session is completed[[69]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L20-L28)[[70]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L43). This allows an anonymous user (with the token link) to fetch a result without exposing all profiles.
  + get\_user\_assessment\_scores(p\_session\_id) – Likely returns profile data for a given session, but only if the session belongs to the authenticated user (likely implemented as an RPC with RLS that checks sessions.user\_id = auth.uid()).
  + get\_recent\_assessments\_safe() – Returns limited fields (date, type code, country, fit score) for recent profiles, used by a dashboard view[[65]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L30-L39)[[66]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L62-L70).
  + Edge Function wrappers (see below) largely supplant direct RPC in the client for scoring due to the need for service role access.
* **Supabase Edge Functions**: All custom logic is implemented as Deno Edge Functions:
* **score\_prism** – the core scoring engine, implemented in TypeScript in Supabase functions directory[[71]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1-L9). It is invoked with a JSON body containing at least session\_id. It loads all responses for that session, applies the scoring algorithm, and upserts the profiles row[[52]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1125-L1133). It returns a JSON with status: "success" and the computed profile data[[53]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1143-L1150). Notably, it uses a **service-role key** to bypass RLS (as indicated by using createClient with the service key[[72]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L142-L150)) – this is necessary so the function can read/write everything (e.g. all responses) regardless of auth. Key steps in score\_prism:
  + Validate input and load environment (URL and service key)[[73]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L139-L148).
  + Log the start event with version[[74]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L156-L164).
  + Load type prototype definitions from DB (type\_prototypes table) to determine each type’s expected functional “block” assignments (base, creative, etc). If missing or incomplete, fall back to a hardcoded set[[75]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L158-L167)[[76]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L178-L186).
  + Fetch all responses for the session from assessment\_responses[[77]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L193-L201). Deduplicate by question (keeping the latest answer per question in case of multiple submissions)[[78]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L215-L224).
  + Load the entire assessment\_scoring\_key and build a dictionary keyed by question\_id (with JSON parsing and validation on fc\_map and meta)[[32]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L229-L238)[[33]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L239-L248). This key provides mapping for how to interpret each answer.
  + Load config values via scoring\_config for various constants (version label, dimensional thresholds, neuroticism norms, default FC mapping, expected minimum FC count, softmax temperature, etc.)[[34]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L255-L263).
  + **Forced-Choice integration**: The function checks for any forced-choice results. In v1.2, it first tries to load precomputed scores from the fc\_scores table for the session[[58]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L299-L308). If present, it uses those scores as the authoritative forced-choice data (setting fcFuncCount proportions accordingly)[[79]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L310-L319)[[80]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L320-L328). If not found, it falls back to *legacy processing* of any forced-choice questions that might be in assessment\_responses: it identifies responses where scale\_type starts with "FORCED\_CHOICE" and uses the fc\_map from the scoring key to map each choice (A, B, C, etc.) to either a function or a block, accumulating counts in fcFuncCount or blockFCCount[[3]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L373-L382)[[81]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L383-L390). It increments fcAnsweredCount for each forced-choice question answered in this mode[[82]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L373-L381). (This dual approach supports the transition to the new FC blocks system.)
  + Process each response: For Likert and other scalar questions, it normalizes the answer to a 1–5 common scale (toCommon5) and sorts it into the appropriate category: If question’s tag is like "Ni\_S" or "Fi\_D", it contributes to a particular function’s **strength** (S for strength questions) or **dimension** count (D for dimensional questions)[[83]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L365-L373). If a tag corresponds to one of the 4 function blocks ("Core", "Critic", "Hidden", "Instinct"), it adds the value to blockLikertCount for that block[[83]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L365-L373). Also collects any neuroticism indicator questions (tag "N" or "N\_R") into neuroVals[[84]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L363-L369). It tracks social desirability (sd) and paired questions for consistency check as well[[85]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L367-L374)[[86]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L369-L372).
  + After iterating responses, it computes validity metrics: **Inconsistency index**, **standard deviation index**, and counts of attention-check failures, using the collected pairs and sd values (the code for this is above omitted snippet, but the result is stored in validity object)[[51]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L702-L710). Then it determines a validityStatus ("pass", "warning", or "fail") and a coarse confidence category ("High"/"Moderate"/"Low") based on those validity metrics[[87]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L712-L720)[[88]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L721-L729). For example, if inconsistency >= 2.0 or SD index >= 5.2, it fails (confidence forced Low); if mild issues, it sets warning or moderate[[89]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L724)[[90]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L726-L730). These become profileData.validity\_status and confidence (text)[[87]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L712-L720).
  + Compute **function strengths**: Each of the 8 cognitive functions (Ti, Te, Fi, Fe, Ni, Ne, Si, Se) gets a strength score. The algorithm (implied by code) likely averages the Likert values and may incorporate forced-choice counts. Indeed, the code builds two intermediate sets: strengths (likely combining all evidence for each function) and dimensions (how many dimension items >= threshold for each function) – these are later used to compute type scores[[91]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L653-L661). The details are in scoreType and related utilities; essentially each type’s “distance” to an ideal prototype is computed next.
  + **Blocks weighting**: The responses in blocks (Core, Critic, Hidden, Instinct) from Likert and FC are normalized to percentages for each source[[92]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L736-L744)[[93]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L745-L753), then blended into a single set of block percentages (blocks\_norm\_blend) weighted by the amount of data from Likert vs FC[[94]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L751-L759). This yields a 4-element vector in profile for block emphasis.
  + **Type matching (distance-based)**: The engine then compares the user’s function strength profile to each of the 16 type prototype profiles. For each type, it takes the expected “block weight” (Model A positions) for that type from TYPE\_PROTOTYPES (augmented by DB or fallback) and maps those weights (like base=1.0, creative=0.7, etc.) onto a target value between 1 and 5 for each function[[95]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L768-L776)[[96]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L779-L788). Then it calculates the Euclidean distance between the user’s 8-function vector (strengths) and that type’s target vector[[97]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L780-L788). This distance is converted to a raw match score (1 – dist/max\_dist) scaled to a 0–6.5 range[[98]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L784-L792). These raw scores are linear with how well the pattern fits the type. It then maps each raw score to a 0–100 “raw fit” percentage[[99]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L794-L803).
  + **Fit calibration**: It adjusts those raw fit scores to “fit\_abs” scores using cohort normalization (Z-score over recent cohort of profiles). Specifically, if at least 50 profiles in last 90 days are available, it computes mean & SD of their score\_fit\_raw[[100]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L823-L831), then each type’s raw fit is scaled to a mean ~50, SD ~15, and clamped to [20,85] for stability[[101]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L826-L835)[[102]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L830-L839). If insufficient data, a fallback linear shrink is applied[[103]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L836-L844). This yields fitAbs (an absolute fit score) per type. The code logs which calibration path was used[[104]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L832-L840)[[103]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L836-L844).
  + **Type probabilities**: It then computes a softmax-like “share percentage” for each type as exp(rawScore/Temp) normalized[[105]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L845-L850). These sharePct values (0–100%) indicate the probability-like share for each type after applying a softmax with temperature (from config, default 1.0)[[105]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L845-L850).
  + Determine the **top 3 candidate types**: It sorts types by fitAbs (primary key), then sharePct, then tie-breakers by “coherent count” (how many of the type’s base/creative functions are strongly present in user – ensures if two types have equal score, the one whose defining functions the user has both >=3 gets priority)[[106]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L859-L867), then by forced-choice support sum (preferring the type supported by FC answers if still tied)[[106]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L859-L867). The top 3 types are selected[[106]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L859-L867). These are stored in profileData.top\_types (as array) and also logged.
  + Compute **topGap** = difference between top1 and top2 fitAbs, and flag close\_call if gap < 3 points[[44]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L875-L880). Determine a **fitBand** (“High” if top fit is strong AND gap is large, “Moderate” for intermediate cases, else “Low”) using config thresholds (e.g. high\_fit, high\_gap, etc.)[[107]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L910-L918). Store these in profile (fit\_band, top\_gap, close\_call)[[108]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1041-L1049)[[109]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1043-L1051).
  + **Overlay traits**: They calculate two overlay modifiers: **Neuroticism overlay** and **State overlay**. Neuroticism: the mean of all “N” responses is computed (nMean) and Z-scored using neuro\_norms (default mean=3, sd=1)[[110]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L670-L678). If Z >= cut (default 0.50) then **overlay\_neuro = "+"**, if Z <= -cut then **"–"**, else "0"[[48]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L674-L682). State overlay: uses weighted combination of state-related questions (stress, time pressure, sleep, focus) – each of those responses (scaled around 3=neutral) are combined with weights (from config overlay\_state\_weights, default stress 0.35, time 0.25, sleep -0.20, focus -0.20)[[111]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L680-L688), yielding a state\_index. If index >= cut, **overlay\_state = "+"**, if <= -cut then "–", else "0"[[111]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L680-L688). Currently, the UI expects a single overlay value; the code uses overlay\_neuro as the primary overlay (with overlay\_state computed but not overriding overlay field)[[112]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L686-L694)[[113]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L690-L698). In profileData both are stored, but overlay is set to the neuroticism overlay (legacy behavior to not break UI)[[112]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L686-L694).
  + **Confidence calculation**: The engine computes a numeric confidence for the result using a unified method in PrismCalibration. Inputs to this are:
  + topGap (distance between best and second-best fit scores)[[114]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L876-L880),
  + confidenceMargin which is (p1 - p2) i.e. the probability share gap between top and second type[[115]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L881-L888),
  + shareEntropy which is the entropy of the type share distribution (to measure how spread-out the probabilities are)[[116]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L886-L894),
  + dimBand which classifies the user’s profile dimensional complexity as '1D', '2D', or '3-4D' based on the highest function dimension count (if any function has ≥3, treat as “3-4D”, if max=2 then "2D", else "1D")[[117]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L892-L900),
  + overlay ("+" or "–" or "0") as computed above.
* It passes these to calibration.calculateConfidence(...)[[117]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L892-L900), which internally: 1. Computes a raw confidence via a sigmoid formula: rawConf = 1/(1+exp(-(a\*topGap + b\*(shareMargin/100) - c\*shareEntropy))) with parameters a, b, c from config (conf\_raw\_params, default a=0.25, b=0.35, c=0.20)[[118]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L52-L61)[[119]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L64-L72). This raw confidence ~ probability (0–1). 2. Looks up the latest calibration model for the given (dimBand, overlay) stratum in calibration\_model table[[61]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L98-L107). If found, applies isotonic regression interpolation on rawConf to get calibrated confidence[[62]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L120-L128). If no model available, falls back to a Platt-scaling sigmoid (with fixed params a=-0.5, b=1.2) to calibrate[[120]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L148-L157). The result is calibratedConf (0–1). 3. It may also assign a qualitative confidence band, likely by cutoffs on calibratedConf (e.g. 0.8+ = High, 0.5–0.8 = Moderate, etc.), returned as confBand.
* The score\_prism function receives rawConf, calibratedConf, and confBand[[121]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L905-L913), and logs the outcome[[122]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L926-L934). In the final profile, it stores conf\_raw and conf\_calibrated (to 4 decimal places) and conf\_band[[39]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1050-L1058), but the primary confidence field in profile remains the textual High/Moderate/Low from validity logic (to not confuse UI expectations)[[123]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1034-L1037)[[39]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1050-L1058). This dual notion of confidence will be clarified in the unified design. - Mark session as completed in DB: It updates assessment\_sessions.status = 'completed' and sets completed\_at if not already set[[124]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L966-L975)[[125]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L979-L987). It then fetches the session (to get email, user\_id, etc.)[[126]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L988-L996). - Compose the profileData object with all the fields as described (ensuring timestamps). Notably, it sets results\_version: "v1.2.0" for version tagging[[41]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1039-L1047). - Upsert into profiles table: if a profile already exists for that session (perhaps a re-score), it preserves the original submitted\_at and sets a new recomputed\_at timestamp[[127]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1109-L1118). Otherwise, it generates a unique submitted\_at (adding a small random millisecond offset to avoid clustering)[[128]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1114-L1122). Then it inserts or updates the profiles row[[52]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1125-L1133). - Return the result (profile and some summary metrics) to caller[[53]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1143-L1150). The function returns quickly after writing to DB, relying on the client to fetch the profile via token if needed.
* **score\_fc\_session** – standalone function to calculate forced-choice scores in the new system. It is called with a session\_id (and optional basis: "functions" or "types", default "functions")[[129]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L14-L22)[[130]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L81-L89). It:
  + Loads all active FC blocks and options (for a given version, e.g. v1.1)[[131]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L29-L37)[[132]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L37-L45).
  + Loads all fc\_responses for that session[[133]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L39-L46). If none, returns an empty result (no scores)[[134]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L46-L54).
  + For each response, finds the corresponding option’s weights\_json (which is a map of either function keys or type codes to numeric weights)[[135]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L54-L62)[[136]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L69-L73). It sums these weights into a tally per key[[136]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L69-L73) (this effectively aggregates the user’s choices).
  + Normalizes the tally into scores:
  + If basis == "functions": it finds the max weight and scales all values such that the max becomes 100, others proportional (and ensures two decimal places)[[137]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L82-L90). This yields a percentage score 0–100 for each function (or trait) key.
  + If basis == "types": it instead normalizes to a probability distribution (sums to 1)[[138]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L86-L90).
  + It upserts the result into fc\_scores table (with session\_id, version, fc\_kind as basis, the JSON of scores, and count of blocks answered)[[139]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L94-L102). This persistent storage allows score\_prism to later pick up these scores.
  + Returns the scores JSON in the response[[140]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L105-L113). In our case, we use basis: "functions" (as seen in front-end call[[141]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L156-L164)), meaning the FC contribution will be a set of function percentile scores.
  + **Note**: This function uses the service role key as well (set via createClient with service role)[[142]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L22-L25) and it bypasses JWT verification (per config)[[143]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/config.toml#L39-L43), so it can be invoked by the client without auth. We will keep this server-side due to sensitivity of scoring logic.
* **finalizeAssessment** – an edge function that orchestrates final submission and scoring of an assessment session. This is called by the front-end at completion (in the newer flow, likely replacing the direct score\_prism call)[[144]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L98-L106). It expects session\_id and optionally the batch of responses in the body[[145]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L32-L40). Key behaviors:
  + If a profile already exists for the session (i.e. scoring already done), it simply returns that profile and ensures the session is marked completed[[146]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L50-L59)[[147]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L56-L64). This prevents double-scoring or race conditions.
  + Otherwise, it fetches the session data (to ensure it exists and to get any user/link info)[[148]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L99-L108).
  + It then **invokes score\_prism internally** (via supabase.functions.invoke)[[149]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L119-L128). It checks for errors or maintenance mode flags from score\_prism and handles them (e.g. if scoring is in “maintenance”, returns 503 with message)[[150]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L140-L149).
  + On success, it generates or retrieves a shareToken: if the session already had one (set when session created or via earlier process) it uses that, otherwise it generates a new UUID[[151]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L172-L180). It then updates the assessment\_sessions record to status completed, sets completed\_at, completed\_questions, and stores the share\_token[[152]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L175-L183).
  + It triggers an admin notification (via another function notify\_admin) in the background with session info (not critical to scoring)[[153]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L70-L78)[[154]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L203-L211).
  + Finally, it returns { ok: true, profile: <profileData>, share\_token: <token>, results\_url: <URL> }[[155]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L216-L224). This matches what the front-end expects. The front-end can then navigate to /results/:sessionId?token=<token> knowing the token.
  + Notably, finalizeAssessment uses the service role and is not JWT-verified (open to anon)[[156]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/config.toml#L3-L10). It secures the result by only returning the profile if scoring succeeded and by requiring the token to view later.
  + This function ensures any needed coordination (it calls score\_prism and updates session atomically). It also handles marking completed\_at even if the client didn’t explicitly (backstop for cases where front-end might not update session properly).
* **Result Fetch Functions**:
  + **get-results-by-session** – an edge function used by the Results page to retrieve the profile for display[[9]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L26-L34). The front-end passes sessionId and shareToken (if present from URL)[[9]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L26-L34). This function likely calls the Postgres function get\_profile\_by\_session(session, token)[[69]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L20-L28)[[70]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L43) to safely retrieve the profile. Because it’s running with service key, it could also do the join itself. The expected behavior: if token matches or the user is the owner of the session (if logged in context), return { profile: ..., session: { id, status } }; otherwise return an error (401/403/404 depending on case). The code (not fully shown in repo, but tests confirm) returns a camelCase JSON with profile and a minimal session (id and status)[[157]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L40-L48)[[158]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/tests/results.integration.test.ts#L42-L50). It implements retry logic on the client side – if score\_prism is still running and the profile isn’t yet available, the function might return a 409 or empty profile, and the client will retry a few times[[159]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L34-L42). This ensures the results page waits for completion if it was loaded too quickly.
  + **Other edge functions**: save\_response, get\_progress, load-session-responses etc. – The inventory shows functions like save\_response (likely an alternative to directly insert responses), get\_progress (to fetch how many answered), and load-session-responses (used in AssessmentForm to resume sessions)[[160]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L33-L41). In our context, these help with mid-assessment state but do not alter final scoring logic. We will ensure our changes remain compatible with these (e.g. if we change how sessions are identified or closed).
* **Auth and Security**: Supabase Auth is used for login (routes /login, /signup exist) and linking sessions to users. However, the assessment can be taken anonymously. If a user provides an email at start, the system can later link that session to an account (there is an EmailSavePrompt and useEmailSessionManager to capture and attach email)[[161]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L718-L727)[[162]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L759-L763). RLS policies:
* assessment\_sessions and assessment\_responses initially allowed anyone to insert and select (to not block anon usage)[[163]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L40-L49)[[164]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L60-L68). This is a potential risk if not tightened, because one session’s data could be read by another user. It appears no update was made in migrations to restrict these further, but since the sensitive part is the final profile, the more important controls are on profiles.
* profiles RLS was initially opened for analytics[[54]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819022459_12718b6c-29ce-46d9-8156-aa05e6f29eb1.sql#L4-L12), but with share tokens they specifically dropped an old policy "Anonymous can view profiles by session"[[5]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L17-L25) (likely one that allowed select where session\_id matches a param, which is now replaced by the function).
* Now, the intended access control is: Only an authenticated user can view their own profile via a RPC (likely get\_user\_assessment\_scores called in frontend for logged users[[165]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L80-L89)), and anonymous results viewing is done through get-results-by-session with the share token. This means the share token acts as a “password” in the URL for accessing a particular result. We will double-check that no other broad SELECT policy remains on profiles (the "dashboard stats" use case is now handled via the secure view v\_recent\_assessments\_safe and the function that returns limited data with a grant to anon[[166]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L70-L77)).
* Supabase keys: The frontend uses the anon public key (in .env.local) to call supabase JS (so it cannot bypass RLS). All privileged operations (scoring, finalizing, fetching results) are done via Edge Functions with the service role key on the backend[[72]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L142-L150). This design is correct – we will maintain env hygiene by ensuring the service key is only used server-side and not exposed in client code. (The code confirms this: functions use Deno.env.get("SUPABASE\_SERVICE\_ROLE\_KEY")[[72]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L142-L150), which is set in the function’s environment, not accessible to client.)
* Row-Level Security adjustments: As part of unification, we will ensure that after scoring, the profile data can only be read via the intended pathways. We might add an RLS policy on profiles such as “user can select their own profile if profiles.user\_id = auth.uid()” for logged-in, and no generic anon access. But since anon access is needed via token, we rely on the function. We will verify that get\_profile\_by\_session is marked as SECURITY DEFINER (it is)[[167]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L26-L34) and that we grant execute on it to anon (the migration that created it might have done so, otherwise we will in our plan).
* The share token itself is a random UUID string – sufficiently secure (≈122-bit) to prevent guessing. It is included in the results link (https://site/results/SESSION\_ID?token=TOKEN). The session UUID alone is not enough to get the result without the token (since the Edge function will check token match). This mitigates enumeration risk. We will preserve this model.

In summary, the current system’s structure is functional but complex. The **front-end** manages user inputs and uses Edge Functions to delegate heavy logic. The **backend** holds all data and scoring algorithms. The **scoring logic** is primarily in score\_prism (Likert + integration of FC + calibration) and score\_fc\_session (detailed FC calculation). **Key gaps/duplication** identified: - Forced-choice mapping logic exists in both score\_prism (legacy fallback) and score\_fc\_session (new approach)[[3]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L373-L382)[[136]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L69-L73). - The confidence calculation and calibration logic is encapsulated in PrismCalibration (good), but only used on backend; front-end doesn’t calculate confidence but does need to interpret the confidence band and value. - The front-end does final validation and sets completed\_at in sessions in some cases, which can race with backend scoring. The presence of finalizeAssessment mitigates this by consolidating that step server-side. - Version consistency: The logic is labeled v1.2.0 in code comments[[1]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1-L6)[[168]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L33-L40) and stored in results\_version, but ensuring both front and back reference the same version string and behavior is critical.

This inventory forms the baseline for designing the unified solution and identifying where changes are needed.

# Architecture Diagram (Mermaid)

Below is a high-level component and data flow diagram for the PRISM assessment system, illustrating the unified scoring engine integration:

flowchart TD  
 subgraph Frontend (React App)  
 A[Start Assessment<br>/assessment route] -->|calls Supabase JS| B[(assessment\_sessions)]  
 A -->|user answers<br>Likert Qs| C[AssessmentForm<br>(saveResponseIdempotent)]  
 C -->|insert| D[(assessment\_responses)]  
 C -->|update progress| B  
 C -->|end of quiz| E[onComplete handler]  
 subgraph Frontend-Edge Integration  
 E -->|invoke finalizeAssessment| F[Supabase Edge Function<br>finalizeAssessment]  
 F -->|calls| G[score\_prism Edge Function]  
 E -.optional.->|invoke score\_fc\_session<br>(if forced-choice done)| H[score\_fc\_session Edge Function]  
 end  
 F -->|response {profile, token}| I[navigate to /results/:id?token=...]  
 J[Results Page] -->|invoke get-results-by-session<br>with sessionId & token| K[Edge Function<br>get-results-by-session]  
 K -->|calls secure RPC| L[(profiles)]  
 K -->|returns profile| J  
 end  
 subgraph Backend (Supabase)  
 B[(assessment\_sessions)] -->|FK| D  
 D[(assessment\_responses)] -->|read by| G  
 M[(assessment\_scoring\_key)] -->|read| G  
 N[(scoring\_config)] -->|read| G  
 O[(fc\_blocks & fc\_options)] -->|read| H  
 P[(fc\_responses)] -->|read| H  
 H -->|writes| Q[(fc\_scores)]  
 Q -->|read by| G  
 G[score\_prism] -->|writes profile| L[(profiles)]  
 G -->|updates| B  
 L -->|join share\_token| K  
 B -->|share\_token| K  
 end  
 subgraph External  
 X[Supabase Auth] -->|JWT| Frontend  
 Y[Supabase Storage]:::grey  
 Z[Third-party APIs]:::grey  
 end

**Diagram Notes**: The user begins the assessment on the front-end (AssessmentForm), which creates a session and saves responses to the database. On completion, the front-end calls the finalizeAssessment edge function. This function (running with service role) internally calls score\_fc\_session (if needed for forced-choice) and always calls score\_prism. The unified **Scoring Engine** is effectively inside score\_prism (and score\_fc\_session for FC) – in our unification, these will share common code. The score\_prism function reads needed data (responses, scoring key, config, any precomputed FC scores) and writes the final profile to the profiles table, also marking the session completed. After that, the results page uses get-results-by-session with the session’s share token to securely retrieve the profile from profiles (via a DB function join with assessment\_sessions to validate the token). Auth is optional; if the user is logged in, their JWT could be used to call a user-specific RPC instead. (Components marked in gray are out-of-scope or not directly used in scoring.)

# Scoring Engine Design Spec

We propose a **unified PRISM Scoring Engine** module that encapsulates all logic for scoring the assessment, usable by both Node (backend) and potentially the front-end (for type definitions, offline validation, or future capabilities). The engine will be implemented in TypeScript and designed as a pure, deterministic function (no side-effects or direct DB calls inside) that takes structured inputs and produces a result object. This will ensure consistency: both the Supabase Edge Function and any front-end diagnostic tools will use the exact same calculations.

## **1. Engine Inputs and Outputs**

**Inputs**: The scoring engine function (let’s call it scoreAssessment() for discussion) will accept a composite input object containing: - **Responses**: A list of all assessment responses for the session. Each response includes: - question\_id (number or string), - answer\_value (raw answer as string or number as provided, e.g. "5" or "A" or text), - (Optionally) question\_type or scale\_type (Likert 5-point, 7-point, forced-choice-4, etc.), - Perhaps meta info like question tag if readily available.

The front-end can supply this from state or local storage if needed, and the backend can supply it by querying assessment\_responses. In our case, the backend will have these responses (we’ll still fetch from DB in the function and then pass into the engine).

* **Scoring Key**: The full mapping of question IDs to scoring parameters. We will embed or reference:
* For each question\_id: its scale\_type (e.g. "Likert\_5", "FORCED\_CHOICE\_4"), tag (like "Ne\_S", "Ti\_D", "Core", "N" etc.), whether it’s reverse\_scored, any pair\_group for consistency, social\_desirability flag, and if applicable, the fc\_map (mapping from option letters to functions or blocks) and any other metadata needed for scoring.
* This is essentially the content of assessment\_scoring\_key. Rather than hard-code, the engine expects this data. On backend, we will fetch it from DB as we do now[[32]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L229-L238). On front-end, if needed for offline or test, we could bundle a JSON of the scoring key (since it’s not secret per se).
* Having the key allows the engine to interpret each response in context.
* **Config parameters**: A set of tuning constants and options, ideally grouped in a config object. This includes:
* Scoring version identifier (e.g. "v1.2.0").
* Dimensional threshold values (to count how many “dimensions” a function has — e.g. threshold for considering a function “strongly expressed” could be 3.0 on the 1–5 scale, etc. In current config: dim\_thresholds.one, .two, .three used to classify 1D/2D/3-4D).
* Neuroticism norms (neuro\_mean, neuro\_sd defaults 3 and 1).
* Overlay cut value (default 0.5 for switching +/- overlays).
* State overlay weights (stress, time, sleep, focus weights).
* Softmax temperature for type probability (default 1.0).
* Fit calibration parameters:
  + high\_fit, moderate\_fit, high\_gap, moderate\_gap thresholds for fitBand categorization[[107]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L910-L918).
  + Cohort calibration toggle or fallback strategy (min cohort size = 50, clamp range 20–85).
* Confidence raw parameters (conf\_raw\_params: a, b, c as in PrismCalibration).
* (If needed)Confidence band cutoffs for High/Moderate/Low (though the engine might derive band from validity or use calibrated value).
* Anything else presently in scoring\_config that influences scoring (e.g. fc\_expected\_min – min number of FC questions to consider complete, used for partial session logic[[169]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L396-L404). Partial handling could be integrated).
* We will retrieve these from scoring\_config on backend and feed into the engine. On front-end, default values can be baked-in or fetched via a lightweight API (we do have a getConfig function in Supabase that might retrieve all config at once).
* **Optional inputs**:
* Pre-computed forced-choice scores (if using the new FC system). For example, if fc\_scores for the session exists, we can supply it to avoid recomputing. The engine could accept something like fc\_function\_scores (a map of function -> score 0–100) as input. If not provided, the engine will fall back to computing from individual FC responses via fc\_map.
* Session metadata that might influence scoring: e.g., whether to enforce partial scoring or not. In partial sessions, we might skip certain steps or mark results incomplete. (The backend currently handles partial sessions by returning a status without a profile if too few answers[[170]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L400-L408). The engine itself could output a flag if data is insufficient.)
* We might also pass user’s identity or role if needed for any conditional logic (though likely not; scoring is uniform).

**Outputs**: The engine returns a **Profile Result** object containing: - **Type result**: - type\_code (e.g. "LII", "SEE", etc. – 3-letter code of best-fit type), - base\_func and creative\_func (two-letter codes of the type’s Base and Creative functions), - top\_types (array of the top 3 type codes in order), - type\_scores (map of type\_code -> { fit\_raw, fit\_calibrated, share\_pct } so we have full scoring info for all types if needed, or at least for the top 3), - fit\_band (High/Moderate/Low classification of fit), - numeric scores: score\_fit\_raw (fit% of the chosen type before cohort cal) and score\_fit\_calibrated (fit% after cal) – as currently stored[[41]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1039-L1047). - top\_gap (gap between first and second type’s fit\_abs) and close\_call (boolean)[[171]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1042-L1049). - **Function metrics**: - strengths: map of each of the 8 functions to its strength score (likely average normalized rating, range ~1–5 or perhaps 0–? If currently each strength is essentially the average of S responses for that function scaled 1–5, it should remain 1–5). - dimensions: map of each function to an integer count of how many responses put that function above certain thresholds (like count of responses >=4, >=some threshold, etc.). In current code, they increment for each "D" tagged question’s value and likely compare to thresholds later[[172]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L851-L859). We will formalize: e.g. count of answers for that function’s dimension questions that are >= threshold for “strong” (maybe 4 or 5). - These help interpret how multifaceted the user is, and were used for dimBand and tie-break logic. - **Forced-choice contribution**: - fc\_count or fc\_answered\_count: number of forced-choice blocks answered (and possibly total available). - fc\_support: map of function -> support value (0–1 or percentage). In current code, after computing fcFuncCount and normalizing, they had fcSupport[func] = count/total[[173]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L2-L10). We can output a similar measure (maybe the 0–100 scaled from fc\_scores or a relative weight). - Possibly fc\_coverage\_bucket: "complete"/"incomplete" if needed (the engine can determine if answered < expected, we might mark incomplete). - **Blocks**: - blocks\_raw (Likert and FC block counts before normalization, possibly not needed in output), - blocks\_norm\_likert, blocks\_norm\_fc: % distribution across Core/Critic/Hidden/Instinct from each source[[92]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L736-L744)[[93]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L745-L753), - blocks\_norm (blended final percentages)[[174]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L750-L758). - These are used to display the relative usage of information metabolism blocks. - **Overlays & Traits**: - overlay: the final overlay sign for the profile ("+" / "–" / "0"), - overlay\_neuro and overlay\_state separately (so UI can potentially show both or future logic can differentiate)[[175]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1072-L1080). - neuroticism: we can include raw\_score (mean of N answers 1–5) and z\_score as subfields (the code stores neuro\_mean and neuro\_z)[[175]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1072-L1080). - state\_index: numeric combined index for state traits (with sign)[[176]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1074-L1081). - Perhaps also whether state overlay was applied or not. - trait\_scores: e.g., if other trait scales exist (currently only N as Big5 Neuroticism, but could extend). - **Validity & Confidence**: - validity\_status: "pass" / "warning" / "fail", - validity details: including inconsistency index, sd\_index (std dev of answers distribution), attention\_failures count, etc., as currently compiled[[51]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L702-L710). - confidence: The *qualitative* confidence (High/Moderate/Low) for presentation. We will compute this primarily based on the profile’s characteristics. In v1.2, the logic was tied to validity (e.g. any fail => Low confidence)[[89]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L724), which is more of a *data quality* confidence. Meanwhile, the calibrated confidence gives a numeric probability of correctness. We may choose to reconcile these: e.g. define confidence\_band purely from calibrated probability and call the validity-based measure something like “result reliability”. However, to avoid UI change, we can keep confidence as High/Mod/Low akin to reliability, and provide the numeric confidence separately. - confidence\_numeric or confidence\_score: the calibrated confidence in percentage (0–100) or probability (0–1). In the current API, they returned confidence\_numeric: confidenceMargin (which was actually p1-p2 gap)[[177]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1144-L1149) – which can be confusing. We will output the actual calibrated confidence (e.g. 85.0 meaning 85% confidence the result type is correct, which corresponds to calibrated value *100). - conf\_band: e.g. “High” if calibrated > some threshold. We should define: if calibrated confidence > ~0.80 maybe label as High confidence result, 0.6–0.8 Moderate, <0.6 Low. We’ll align these thresholds with historical usage or ensure they correlate with the validity categories. - So likely: output confidence\_calibrated (0–100%) and maybe confidence\_raw (the rawConf before calibration) for record. -* *Versioning & Metadata*\*: - results\_version: e.g. "v1.3.0" if we bump after unification. This allows the system to know how to interpret the fields. - Timestamps: The engine itself may not set timestamps (that can be done when saving to DB), but we might include timestamp or allow passing now for consistency. - Other: if partial session, perhaps an indicator like partial\_session: true and maybe a completion\_rate.

The output essentially mirrors the fields in profiles table[[178]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1031-L1041)[[39]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1050-L1058), but structured. This is intentional: the profile table will be populated directly from this object (with minor differences like JSON vs columns).

## **2. Deterministic Algorithm & Steps**

The unified engine will follow these steps (consolidating what we described from score\_prism and score\_fc\_session):

1. **Validation of Input Data**: Ensure required data is present. E.g. no scoring if responses list is empty or if critical config values are missing. If in partial scoring mode, detect insufficient answers early (e.g. if < 50% of questions answered and partial allowed, we may output an “insufficient data” status instead of a profile).
2. **Prepare Data Structures**: Initialize accumulators:
3. likertSum[func] and count to compute average strength per function,
4. dimsCount[func] for dimensional counts,
5. blockLikertSum[block] (Core/Critic/Hidden/Instinct sums from Likert),
6. fcCountFunc[func] for forced-choice counts (or scores if provided),
7. blockFCSum[block] for forced-choice block sums (if FC mapping yields blocks),
8. trackers for neuroScores (list of N answers), sdValues (for social desirability mean & sd), pairs (map of pair\_group -> [answers] to compute consistency), attentionFails count.
9. **Process Each Response**:
10. For each answer in the responses list:
    * Look up its scoring key entry (by question\_id).
    * Validate the answer value (e.g. ensure numeric where expected, convert "Strongly Agree" to 5 etc.). Use a helper similar to parseNum and sanitizeResponseValue[[179]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L85-L93)[[180]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L94-L98) to get a numeric value or standardized letter.
    * If answer is invalid for the scale (e.g. out of range), skip or count as invalid.
    * If scale\_type indicates a **Likert-style** question (e.g. 5-point agreement):
    * Apply reverse scoring if needed (if reverse\_scored flag, map value = scale\_max+1 - value).
    * Normalize to a 1–5 common scale (e.g. if it’s a 1–7 scale originally, convert proportionally or via mapping defined; current code’s toCommon5 likely handles that).
    * Based on the question’s tag:
      + If tag ends with \_S (strength question for a function): add the value to likertSum[func] (and maybe count).
      + If tag ends with \_D (dimension question): if value >= threshold (like 4 or 5), increment dimsCount[func]. Alternatively, simply store all D values and compute how many >= threshold after – either way yields count of strong endorsements for that function.
      + If tag is exactly one of the block names ("Core", "Critic", etc.): add the value to blockLikertSum[tag] (these usually represent certain meta questions that directly assess blocks).
      + If tag is "N" or "N\_R": add the value to neuroScores.
      + If social\_desirability flag true: collect in sdValues (to later compute SD index).
      + If has a pair\_group: append the value to that pair’s list in pairs (to later measure inconsistency).
      + If question is an attention check (maybe marked via metadata or tag): check if answer is the “wrong” one and increment attentionFails.
    * If scale\_type indicates **Forced-Choice**:
    * If we have *precomputed FC scores* (provided as input), we skip per-response mapping and will use that later.
    * Otherwise, determine the chosen option:
      + The answer might be stored as a letter (“A”, “B”, etc.) or number (“1” which corresponds to A). Normalize to A/B/C... (the code did this mapping for numeric to letter)[[181]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L375-L383).
      + Retrieve the question’s fc\_map from the scoring key. This is a JSON mapping like { "A": "Ne", "B": "Si", ... } or possibly a block {"A": "Core", ...}.
      + If fc\_map exists and contains the chosen option key:
        - Let mapped = fc\_map[choice].
        - If mapped is one of the 8 functions (Ti, Ne, etc.): increment fcCountFunc[mapped].
        - If mapped is one of the 4 blocks: increment blockFCSum[mapped].
      + If map doesn’t contain it (shouldn’t happen if data is consistent), we log a mapping miss.
      + Increment a counter of FC questions answered.
    * (Note: In the new FC system, these individual FC question entries might not exist at all – they were replaced by fc\_blocks. But for backward compatibility, we keep this to handle any legacy responses. We may also decide to remove this path once all assessments use the new method.)
11. End for each response.
12. **Incorporate Precomputed FC** (if available):
13. If input provided fc\_function\_scores (0–100 for each function from fc\_scores table):
    * Mark that we’re using real FC scores (like usedRealFC = true).
    * For each function, instead of counts, we derive a pseudo-count or value. The legacy code multiplied each percentage by 12 and ensured at least 1[[182]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L317-L325). We can do something similar or a proportional approach:
    * For consistency, we might treat score/100 \* N where N is an effective count of FC questions. Since earlier each FC question yields 1 count to some function or block, and typical complete FC might have ~24 questions (if fc\_expected\_min=24), they scaled to 12. We can simply use the percentage as weight directly without discrete counts in the new engine.
    * For each function f: set fcSupport[f] = score/100 (a fraction of max).
    * Also, we might want to distribute these into block counts: if needed, we can map each function to its block (base or creative of some type? But not straightforward). Instead, since we have direct function scores, we can compute block distribution from those:
    * e.g., sum scores of functions that belong to each block for a given identified type? The earlier approach was crude (just equalize to some counts). Perhaps better: for each block (Core, etc.), if we know which two functions are core for the user’s eventual type, that’s not known yet. Alternatively, skip blockFCSum if we have direct function measures; or compute it by splitting the FC function scores into Model A blocks of the identified type after the fact. This may be overkill.
    * Simpler: we will derive block contribution from FC in a similar way as Likert: if certain FC questions targeted blocks directly (some fc\_map yield "Core"), those would have filled blockFCSum. If not, we can infer block distribution from function scores by grouping the 8 functions into the four blocks of the user’s top type *after* determining it. This is indeed done at the end in current code as a fallback in case no block tags present[[183]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L999-L1008)[[184]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1009-L1017). We will handle blocks after identifying the type.
14. If no precomputed FC and if forced-choice answers were processed above (legacy path):
    * Compute fcSupport for functions: e.g. fcSupport[f] = fcCountFunc[f] / totalFCAnswers[[173]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L2-L10). This yields a 0–1 fraction per function. (We’ll use these in type sorting tie-breaker.)
    * Compute blockFCCount percentages similar to Likert: each block sum divided by total blocks sum[[92]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L736-L744) for FC.
15. If neither (no forced-choice at all), then all FC contributions remain zero.
16. **Calculate Function Strength Averages**: Now compute the final strength score per function from Likert data:
17. For each function, if we collected multiple S-type questions: calculate average (sum/count) on 1–5 scale. If no data for a function, we might treat as neutral (3) or 0? (Better: if unanswered, treat as 0 contribution? But 0 is below min of scale; we likely won't have completely missing because each function usually has some questions).
18. The engine can also apply weighting between Likert and FC for strengths if needed: currently, the FC influences type match separately, not by altering strength scores directly. We will follow current design: strengths derived from Likert only (the distance matching already factors FC via penalty).
19. However, to unify, we might consider adding a small weight from forced-choice to strengths. Alternatively, leave strengths purely Likert; forced-choice influences type ranking via fcSupport tie-break and the opposite function penalty (discussed next).
20. Also possibly compute each function’s variability or something, but not needed for main output.
21. **Dimensional Band**: Determine the dimBand (1D vs 2D vs 3-4D) by looking at dimsCount:
22. If any function has dimsCount >= 3 (i.e. user strongly endorsed at least 3 dimension questions for one function), categorize as '3-4D'.
23. Else if any function has dimsCount == 2, categorize '2D'.
24. Else '1D'[[117]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L892-L900).
25. (We might refine thresholds via config; these values 3 and 2 likely come from dim\_thresholds config. We can instead calculate how many functions have 2 or more strong endorsements, but current logic looked only at max of one function.)
26. **Opposite Function Penalty**: In the existing code, there’s mention of oppositePenalty in debug output[[185]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L940-L948) and SCORING\_WEIGHTS including penaltyOpp: 0.20[[186]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L51-L55). This suggests that part of type scoring might penalize if both opposite functions (e.g. Ti vs Fe) are high. The code likely uses scoreType(typeCode, strengths, dimensions, fcSupport)[[91]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L653-L661) to compute an initial score, possibly applying:
27. e.g., subtract some weight if for the type’s supposed suppressed functions are high.
28. We need to clarify this: The code builds typeScores[type] = scoreType(...) then later doesn’t use it directly for ordering, instead uses distance approach. Possibly scoreType was a legacy method replaced by the distance calculation (scoreType might combine dimension count and FC support with weights given in SCORING\_WEIGHTS).
29. Given our plan to unify, we may drop the older scoreType method in favor of the distance-based method exclusively (which we outlined).
30. However, the mention of penaltyOpp 0.20 suggests adding a penalty if the user shows high strength in the functions opposite the type’s ego-block (base/creative) functions. We can incorporate that: for each type, after computing match or share, reduce score if the user’s opposite functions are also strong. The debug output calculates penalty\_components by type[[185]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L940-L948).
31. Possibly, after computing raw distance match, they might adjust match by -penaltyOpp \* (sum of strengths of functions that are Opponents of the type’s base & creative?). This might already be implicitly captured by distance since if user has high values for opposite functions, distance for that type increases. But maybe not fully, so they had an explicit component.
32. We will examine if needed. To be safe, we keep a step: for each type, compute an opposite-function penalty: e.g. for type with base=Ti, creative=Ne, the opposites would be Fe (opposite of Ti) and Si (opposite of Ne)[[187]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L8-L16). Sum user’s strengths on Fe and Si, scale by 10 (to 0–100) and multiply by 0.20? We’ll derive from code if possible. Alternatively, incorporate it as part of distance vector by including dimensions for all functions including those opposites, which already happens. We can likely omit this if distance method suffices; the presence in config suggests it was used in an older direct scoring approach (maybe for pre-distance scoring).
33. **Type Distance Matching**: Use the distance-based algorithm described:
34. Compute protoTargets for each type (16) mapping each function to a target score 1–5 based on block weights[[95]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L768-L776).
35. For each type, calculate Euclidean distance between (user’s strength vector) and (protoTargets for that type) across the 8 functions[[97]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L780-L788).
36. Convert to raw match score: match = max(0, 1 - dist/MAX\_DIST) where MAX\_DIST = distance if user had all 1s vs prototype with all 5s (they computed for 8 funcs with diff up to 4 each: sqrt(8\*4^2) which is 11.314)[[188]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L764-L773)[[98]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L784-L792).
37. Scale match\*6.5 as in code to get raw\_score (0 to ~6.5).
38. Map to fit\_raw\_pct = (raw\_score/6.5)\*100 (clamped 0–100, one decimal)[[99]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L794-L803).
39. (This yields identical fitRaw as current system.)
40. **Cohort Calibration of Fit (fit\_abs)**:
41. If a sufficiently large sample (we can use same criteria: last 90 days, >=50 profiles) is provided (the engine might accept an optional distribution or stats, but better the engine doesn’t query DB. Instead, we can do this outside or feed the mean & sd as config).
42. We likely will handle cohort calibration outside the pure engine for simplicity. Reason: it requires DB access to gather recent profiles. In current code, score\_prism does that query[[189]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L807-L815) and then adjusts fit scores.
43. We can formalize it though: treat fit\_raw\_pct of the top type as we want around 50. It’s fine to incorporate: we will have in config either a static global mean & sd (if we want to freeze calibration parameters per version), or we perform a dynamic query in the function wrapper (not in engine).
44. Plan: *Within the engine*, we can implement a general function calibrateFitScores(rawScores, cohortMean, cohortSd) that returns calibrated fitAbs per type. The wrapper will provide cohortMean & sd (or instruct engine to fallback if none).
45. Alternatively, to avoid needing fresh cohort data on front-end, we might define these calibration as part of the version. E.g. for v1.2.0, the average raw distribution might be known (maybe ~50 with sd ~15 by design). Given they clamp to 20–85, we could also just use that known mapping. However, using recent data helps adapt to any drift.
46. For now, we’ll do as is: backend will supply mean & sd if available to engine; otherwise engine does fallback formula (raw \* 0.65 + 20).
47. After this step, we have fit\_abs scores for each type (approx 20–85 range, one decimal).
48. **Type Probability (share\_pct)**:
    * Compute exponentials of rawScores (the *un-calibrated rawScore or maybe the calibrated? Actually code uses rawScores (match* 6.5 before scaling to %?), in code they did Math.exp(v/temp) for each rawScore[[105]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L845-L850). v here likely the raw match (0–6.5) or could be raw percentage/100. Given they divided by temp (like 1.0), the relative ordering is all that matters. Let’s assume they used rawScore (0–6.5 range) for the softmax).
    * We will use the normalized probabilities approach:
    * For each type, exp\_val[type] = exp(raw\_score / Temperature).
    * Sum them and get share\_pct[type] = (exp\_val[type] / sum\_exp) \* 100%[[105]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L845-L850).
    * Provide one-decimal rounding.
    * This yields a probability-like distribution over types summing to ~100%. (This distribution is heavily peaked if one type’s raw score is much higher, more uniform if close.)
49. **Rank Top Types**:
    * Sort types by:
    * fit\_abs (descending),
    * if tied, share\_pct,
    * if tied, “coherent count” – for each type, count how many of its base/creative functions the user has with dimension count ≥3[[106]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L859-L867),
    * if tied, forced-choice support: sum fcSupport of type’s base + creative[[190]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L863-L870). (We replicate this logic exactly to ensure same tie-break outcomes.)
    * The first element = best-fit type\_code.
    * Identify top\_types = [type1, type2, type3].
    * Compose an array top3\_fits of objects { code, fit: fit\_abs, share: share\_pct } for those.
    * Compute top\_gap = fit\_abs[top1] - fit\_abs[top2] (rounded to one decimal).
    * Determine close\_call = top\_gap < 3.0 (threshold 3 as currently) – possibly make threshold configurable.
    * Determine fit\_band: Using top1\_fit, top\_gap against configured cutoffs:
    * If top1\_fit >= high\_fit AND top\_gap >= high\_gap then "High",
    * Else if (top1\_fit >= mod\_fit OR top\_gap >= mod\_gap) then "Moderate",
    * Else "Low". (Follow the logic in code[[107]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L910-L918).)
50. **Compute Overlays**:
    * Neuroticism: Calculate mean of neuroScores (or 0 if none). Compute Z = (mean - norm\_mean)/norm\_sd[[110]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L670-L678).
    * Determine overlay\_neuro: if Z >= cut -> "+", if Z <= -cut -> "–", else "0"[[48]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L674-L682).
    * State: Using captured state responses (which we have via specific question IDs from config state\_qids mapping). We should retrieve those from responses by ID (for stress, sleep, etc.). The Assessment likely asked these on a 1–5 scale. Compute each as (value - 3) (so centered at 0)[[191]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L681-L688). Then state\_index = W.stress\*stress + W.time\*time + W.sleep\*sleep + W.focus\*focus (using weights W from config)[[111]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L680-L688).
    * Determine overlay\_state: if index >= cut -> "+", <= -cut -> "–", else "0"[[112]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L686-L694).
    * Determine final overlay: Since UI expects one, we'll follow current approach: use the neuroticism overlay as primary. (Alternatively, we might combine them: e.g. if either is +/- and not conflicting, possibly show both? But requirement says maintain current UX, which uses one overlay dimension, presumably neuroticism.)
    * So overlay = overlay\_neuro for now[[112]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L686-L694).
    * We will still include both in output.
51. **Confidence Calculation**:
    * Compute share\_margin = p1 - p2 (the top type probability minus second type) in percentage points[[115]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L881-L888).
    * Compute share\_entropy of the share\_pct distribution (base-2 entropy as done)[[116]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L886-L894).
    * We already have top\_gap and dimBand and overlay from prior steps.
    * Feed these into a unified confidence function. Rather than reimplement, we integrate the PrismCalibration logic:
    * Calculate raw\_confidence as sigmoid(a*top\_gap + b*(share\_margin/100) - c\*entropy)[[119]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L64-L72).
    * Then attempt to calibrate:
      + Use provided calibration model data if available for that combination of dimBand and overlay (plus overall version). In practice, our engine can have an in-memory calibration map loaded from DB, or call out. We may prefer to perform the calibration lookup outside the core function for simplicity (since it’s a DB query in original). But we could allow passing in a calibration model (e.g. a small array of {x,y} knot points for interpolation) as part of config.
      + If model available: do piecewise linear interpolation (the interpolateCalibration logic)[[192]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L176-L184)[[193]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L186-L192) to get calibrated\_conf.
      + If not: use fallback Platt: calibrated = sigmoid(-0.5 + 1.2 \* raw\_conf)[[194]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L150-L159).
    * Determine conf\_band: likely:
      + "High" if calibrated\_conf >= 0.80 (for example),
      + "Moderate" if >= 0.60,
      + else "Low". (We can tune these thresholds or derive from calibration model if it encodes band cuts. Possibly the ConfidenceBandCuts in calibration class intended for that, but in code, conf\_band seems to be returned as part of calculateConfidence – probably based on numeric value).
      + Actually, calculateConfidence in code returned an object with raw, calibrated, and band[[195]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L904-L912). The band might be derived from calibrated conf by some cutoffs (maybe high>0.7, moderate>0.4? Not sure, but we can set as above).
    * The engine returns both raw and calibrated confidence values, and the qualitative band.
    * Additionally, incorporate **Validity flags** into final confidence:
    * If validity\_status == "fail", regardless of numeric confidence, overall confidence output could be downgraded to "Low". If "warning", maybe cap at Moderate.
    * This logic exists currently: they override confidence to Low/Moderate based on fails[[89]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L724). We will preserve that behavior to maintain trustworthiness:
      + E.g. if attention checks failed or inconsistency high, the user’s data quality is poor – so even if algorithmic confidence is mathematically high, we label it Low.
    * Therefore, set final confidence\_label:
      + If validity fail -> "Low",
      + Else if warning -> if exactly one attention fail then "Moderate", else "Low" (like code did)[[88]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L721-L729),
      + Else (validity pass but maybe moderate flags) -> could use numeric calibration band or default to "High". Code then also had a clause to downgrade to Moderate if inconsistency >=1.0 or SD >=4.3 or one attention fail[[90]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L726-L730). Essentially it fine-tunes:
        - If minor issues: confidence = "Moderate".
        - If no issues at all: "High".
      + We will keep those thresholds configurable or same as now.
    * The output will include both confidence\_label (High/Mod/Low final) and the numeric confidence\_calibrated for context. In the profiles table they store the label in confidence and numeric margin in a separate field[[123]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1034-L1037)[[39]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1050-L1058); we might introduce a new field for numeric if needed.
52. **Validity Summary**:
    * Compute inconsistency\_index: using pairs – likely the stdev or difference among answers that should be similar. If each pair group has answers, maybe they compute correlation or difference. Possibly they use something like the average absolute difference within pairs as inconsistency. (In code, inconsIdx might have been set elsewhere and logged[[196]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L704-L710).)
    * Compute sd\_index: measure of standard deviation of all Likert answers (to detect if user put same answer for everything). Possibly formula: sd\_index = population\_SD \* 10 or something. Code used thresholds ~4.9, 5.2 which seems like a scaled value[[197]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L720).
    * attentionFailed: count of failed attention checks.
    * Based on these and any required tag coverage (the code checks for required tag gaps in responses, but that’s a nuance for completeness – e.g. if some category of question was not answered)[[198]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L694-L701).
    * Determine validity\_status:
    * If inconsistency >= 2.0 or sd\_index >= 5.2: "fail"[[197]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L720).
    * Else if inconsistency >= 1.5 or sd\_index >= 4.9 or attentionFailed >= 2: "warning"[[88]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L721-L729).
    * Else "pass".
    * This logic is mostly in place; we will keep the same thresholds (perhaps moving them to config too for adjustability).
    * Put detailed values in validity object in output (as currently included[[51]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L702-L710)).
53. **Package Output**: Assemble the output object with all the fields from steps 5–14 filled in.
54. **If Partial/Incompletion**: If during processing we found not enough responses:
    * The current system sometimes returns early with status "partial" or "partial\_insufficient" instead of a profile[[170]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L400-L408). We can mimic this by either:
    * Not calling the engine at all for partial (the finalizeAssessment on backend currently handles partial – if fc answers < half of expected and partial\_session flag set, it returns a message[[170]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L400-L408)).
    * Or the engine could output a flag or throw an exception indicating insufficient data.
    * Plan: Let the wrapper (finalizeAssessment or score\_prism) continue to handle partial session logic (because partial might not produce a meaningful profile). We ensure if partial, we don’t insert a profile with UNK type; instead, we short-circuit. This is more a flow control outside engine.
    * So engine spec mainly covers full scoring. Partial detection (like fcCompletionRate < 0.5) is handled in wrapper.

By following these steps, the scoring outcome will be identical to or carefully improved upon the current approach, but centralized in one module.

## **3. Key Functional Considerations**

* **Determinism**: No random elements should affect scoring. The only use of randomness in current code is adding milliseconds to submitted\_at timestamp[[128]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1114-L1122), which is not relevant to the scoring calculations themselves. The engine’s logic will yield the same output given identical inputs. Version tagging ensures any future changes are tracked.
* **Performance**: The engine operates in memory on at most ~100–120 responses (the PRISM assessment length). This is trivial computationally (a few loops over 100 items, plus looping 16 types etc.). The heavy lifting (reading DB) is outside the engine. Thus it can run on frontend if needed (e.g. for a quick client-side estimation in future) without performance issues. On backend it’s very fast.
* **Data consistency**: Using the same engine for both the edge function and potential future client usage means any bug fixes or changes apply uniformly. We will implement it as a pure function or small set of functions in a shared library.
* **Inputs source**: On the backend, score\_prism function will gather inputs (responses, scoring\_key, config, etc.) and call the engine. On the front-end, if we ever use it (say to show a provisional result before final submission), we’d need the same scoring\_key and config. We might bundle a snapshot of those or call an API to get them. Since that’s an edge case, initially front-end might not directly compute results, but could use the engine for validating or replicating some calculation for UI (like showing live dimension count or something if desired).
* **Backward compatibility**: The design preserves fields currently used by the UI:
* profile.type\_code, base\_func, creative\_func – for type result.
* confidence (High/Mod/Low string) – used in UI to maybe color or message.
* overlay – currently one char, already provided.
* top\_types array and possibly top\_3\_fits (the UI likely uses these to populate the TraitPanel or comparison display).
* strengths and dimensions – if UI displays function bars or such (the presence of these in profile suggests maybe an advanced view or debug).
* blocks\_norm and blocks – likely used to render the block bars in ResultsV2 / OverlayChips UI.
* So by maintaining these keys and their meaning, the front-end (ResultsV2 etc.) continues to work.
* We will confirm UI component usage and ensure all needed fields remain or adapt accordingly (e.g. if we changed the meaning of confidence field, we would update UI to use conf\_band or similar, but we plan not to break it: keep confidence as reliability category, and add numeric under a new field).
* **Future-proofing**: The engine design is modular:
* If we later add new question types (e.g. other traits or interactive sections), we can extend the scoring key and adjust processing.
* If calibration method changes (say v1.3 uses a different method), we can version it and include a toggle in config or have separate calibration data loaded.
* Because results\_version is stored, the system can handle multiple versions concurrently (old profiles tagged v1.1.2, new v1.2.0, etc.). We can even allow the engine to produce different outputs based on a version parameter if needed. However, since we plan to unify everything to v1.2.1 or v1.3.0 going forward, we might freeze older code for reference and only use the latest engine for new scorings.

In summary, the unified scoring engine takes all relevant data about a session and deterministically produces the profile results. It covers Likert scoring, forced-choice integration, trait overlays, fit computation, and confidence assessment in one place. By sharing this engine between front and backend, we eliminate mismatches (for example, if front-end had some heuristic for partial feedback, it could use the same logic). The design carefully preserves the existing scoring semantics (v1.2.0) to maintain continuity in results, while also making it easier to adjust and test going forward.

# Unified Implementation Plan

Implementing the unified scoring system will involve coordinated changes across the codebase: refactoring logic into a shared module, updating Supabase Edge Functions to use that module, adjusting database where needed, and modifying the front-end to align with the new flow. Below is a step-by-step plan:

**Phase 1: Refactor & Create Shared Scoring Module**

1. **Set up a Shared Library**: Create a new directory for shared code, e.g. src/lib/scoring/ or a top-level scoring/ (monorepo style) that both front-end and Supabase functions can import. We will place the unified scoring engine code here. For example, file scoringEngine.ts exporting a function computeProfile(sessionData).
2. Ensure build configuration allows importing this from the Edge Functions. (Since the repo uses Vite and not a monorepo with package.json for functions, we might need to either symlink or include the file in supabase/functions folder structure so that Supabase bundler includes it. One approach: put it in supabase/functions/\_shared/ along with calibration.ts. The Deno functions can import via a relative path to \_shared. The front-end can import via an alias if we configure Vite to include that path.)
3. Alternatively, make it an npm package (local) and include in both build processes. Simpler is to keep it within this repo.
4. **Migrate Calibration and Constants**: We already have supabase/functions/\_shared/calibration.ts with the PrismCalibration class for confidence[[199]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L33-L41). We can reuse it or move its logic into the new module. E.g., include the raw confidence calc and calibration lookup as functions in the engine module. Possibly we keep PrismCalibration as is (since it’s already decoupled nicely) and just instantiate/use it in our engine. The shared module can import PrismCalibration for calibration tasks on backend. On front-end, we likely won’t have calibration models available (no DB access), so the engine could default to raw confidence or use fallback on front-end usage.
5. We should verify PrismCalibration doesn’t use any Deno-specific code except supabase client (which on front-end would not exist). We may adjust it to allow injection of calibration data instead of doing a Supabase query (e.g., have PrismCalibration.applyCalibration(raw, dimBand, overlay, modelData?) to use provided model if given).
6. For now, on backend the calibration class will fetch from calibration\_model table as it does[[61]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L98-L107). That’s fine.
7. **Implement the Scoring Functions**: Write functions in the module for each logical part:
8. normalizeResponse(answer, scaleType, reverse?) – yields 1–5 value or appropriate code.
9. processResponses(responses, scoringKey): loop through responses and populate structures (strengths, dimensions, etc.) as per design.
10. computeTypeMatches(strengths, prototypeProfiles): returns raw fit scores for each type.
11. calibrateFitScores(rawFits, cohortStats): yields fitAbs for each type.
12. computeTypeProbabilities(rawFits): yields share percentages.
13. rankTypes(fitAbs, sharePct, dimensions, fcSupport): returns sorted types and top3.
14. computeOverlays(neuroScores, stateValues, config): returns overlay\_neuro, overlay\_state, plus numeric traits.
15. computeConfidence(topGap, shareMargin, entropy, dimBand, overlay, validity, PrismCalibrator): returns numeric confidences and band, possibly adjusting for validity.
16. assessValidity(pairs, sdValues, attentionFails, requiredTags, responses) – computes inconsistency, sd\_index, etc., yields validity status and detail.
17. Finally, assembleProfileResult(...): constructs the profile object as specified.

Ensure these functions use config parameters instead of hard-coded thresholds (e.g., inconsistency cutoff 2.0, we can keep inside function or pass via config).

We'll incorporate the exact formulas from code: - e.g. inconsistency: likely measure correlation of pairs – to mirror existing, we might do simple: if any pair of answers differ > some threshold, etc. The existing code threshold 2.0 suggests maybe an average absolute difference; we might glean formula if inconsIdx is computed as something like mean squared diff. - If uncertain, we maintain the thresholds exactly as used.

1. **Unit test the engine module** (locally, if possible):
2. Write tests with known input-output. For example, craft a scenario where a user clearly is a certain type (e.g., all answers favor Ti and Ne strongly), see if engine returns that type with high confidence.
3. Compare with legacy function on same data. Possibly use some existing profiles records from DB as test cases: feed their responses and see if we get same results as stored. We have the backfillProfiles.ts and some debug tools in repo which might help gather test data.
4. This ensures our refactor doesn’t change results inadvertently. (We can use the debug mode of score\_prism which outputs raw details[[200]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L938-L948)[[201]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L950-L958) as a reference.)
5. **Integrate Opposite Penalty if needed**: If in tests we find slight differences, consider adding the opposite function suppression if it was intended. We have SCORING\_WEIGHTS including penaltyOpp=0.20[[186]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L51-L55), likely used in their (now unused) scoreType function. We might ultimately decide it’s redundant with distance method. We will note it but likely skip unless needed.

**Phase 2: Replace Backend Logic with Engine**

1. **Update score\_prism Edge Function**:
2. Simplify it to use the new engine. Specifically:
   * Remove the internal computations that we moved to the engine (everything after fetching data). Instead:
   * Fetch assessment\_responses (and do dedup) as before[[77]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L193-L201)[[78]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L215-L224).
   * Fetch assessment\_scoring\_key and build the keyByQ as before[[32]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L229-L238)[[33]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L239-L248).
   * Fetch necessary config values from scoring\_config (we might fetch all keys at once to reduce DB calls, or use existing keys we know).
   * Fetch fc\_scores as now to see if precomputed exists[[58]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L299-L308).
   * Then assemble an input object and call scoreAssessment(inputs).
   * The engine returns a profile result object. We then:
   * Add session\_id and user\_id into it (engine can incorporate user\_id if provided, else we add).
   * Mark results\_version (engine can fill this if we pass version in config).
   * Upsert into profiles table (structure matching fields).
   * Also update assessment\_sessions status and completed\_at, but note: finalizeAssessment now also does this. We have to be careful:
     + In the new flow, finalizeAssessment wraps score\_prism, and finalizeAssessment already updates session status and share\_token[[152]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L175-L183).
     + The score\_prism itself also currently updates session (lines 965-981)[[202]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L966-L974)[[125]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L979-L987). We may remove that from score\_prism to avoid double update. Instead, let finalizeAssessment handle session completion. But consider if score\_prism is ever called standalone (e.g. an admin re-score or debug by calling it directly). In that case, it should ideally also mark completed. We can keep it but ensure idempotency.
     + Perhaps keep session update in score\_prism but with logic: if partial\_session flag was passed and it's incomplete, score\_prism might not finalize. Since finalizeAssessment always calls with full session (except if partial allowed?), likely fine.
   * Return the JSON result. Possibly align format with what finalizeAssessment expects (score\_prism currently returns { status: "success", profile: {...}, gap\_to\_second, confidence\_numeric }[[203]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1143-L1151). We can keep that.
3. **Remove duplicate forced-choice logic**: Now that score\_prism will rely on fc\_scores for FC data primarily, we can decide if we drop the legacy per-question FC mapping:
   * If we are confident all active assessments use the new FC blocks (version v1.1 in fc\_blocks), we might enforce that by requiring score\_fc\_session to have run. Possibly finalizeAssessment could ensure calling score\_fc\_session before score\_prism.
   * We should still handle the edge case of someone not completing FC or if they took an assessment when FC was embedded. But since the plan is unify, likely moving forward all FC is separate.
   * I suggest we keep a fallback but clearly mark it deprecated. The engine can handle per-response FC as described, so we just feed all responses (including any FC ones) into engine and let it map. That covers legacy. For new, if fc\_scores present, we feed those and the engine will override the need to map individual FC responses (we can set a flag in input to skip).
   * So yes, we will feed both: the full responses list and the fc\_scores (if any). The engine if sees fc\_scores provided, can ignore any responses of type FORCED\_CHOICE or just not double-count.
   * Implementation: If fcScores fetched (not null), set useFcScores=true and maybe remove those FC questions from responses list or mark them so engine doesn’t process them. Or engine itself can prioritize the fc\_function\_scores over computing from responses.
4. Adjust log prints to reflect using unified engine (maybe add evt:engine\_version, v:1.2.1 etc. for debugging).
5. **Testing**: Deploy this to a staging environment or run Supabase function locally with sample data, confirm the output matches the old version for known cases.
6. **Update score\_fc\_session Edge Function**:
7. This function can remain largely as is since it’s already fairly standalone and straightforward. However, consider if we want to integrate it into the unified engine or pipeline:
   * Option 1: Continue to call score\_fc\_session separately from front-end (as done now when FC blocks are done)[[141]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L156-L164).
   * Option 2: Merge its logic into score\_prism: meaning score\_prism itself could call the engine for FC if needed. But since FC is possibly taken in parallel or at different times than Likert, keeping it separate is fine.
   * We likely keep it but ensure consistency:
   * Both engine and score\_fc\_session should interpret weights\_json the same way. We should confirm the weights\_json schema: presumably keys are function codes like "Ne", "Ti" with some weight. Or if basis is 'types', keys might be type codes.
   * In the database, how are these weights made? Possibly through manual assignment for each forced-choice option, distributing some points to certain functions. The current code normalizes by max, which is fine.
   * We might enhance it slightly: set fc\_expected\_min or total blocks logic. If a user doesn’t complete all blocks, the function still calculates based on what they did. It returns blocks\_answered count which we use to judge completeness.
8. No major changes needed, but as a precaution:
   * Ensure score\_fc\_session writes the version field properly (currently uses version = 'v1.1' by default)[[129]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L14-L22). If we upgrade to v1.2 or v1.3, update accordingly so the fc\_scores version matches (score\_prism expects version 'v1.1' as coded[[204]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L302-L305) – we should bump that to v1.2 or whatever current).
   * In config or code, define the current FC version. Possibly place 'v1.1' in a config since any changes to FC weighting would bump it.
   * After unification, both Likert and FC are part of v1.2. We might keep FC schema version separate though (if structure changed).
   * For simplicity, we might not change fc\_blocks version for now – it's more content version. Our scoring engine version might be separate.
9. **Update finalizeAssessment Edge Function**:
10. This function currently calls score\_prism and handles share\_token and session finalizing[[149]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L119-L128)[[205]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L172-L181).
11. We should consider calling score\_fc\_session inside finalizeAssessment to ensure FC answers are scored before prism:
    * If the user completed FC blocks *after* Likert, by the time they finish the whole assessment, we should have all fc\_responses. Possibly the front-end could call score\_fc\_session when the last FC block is answered (the code suggests it does call it on complete via fcBlockService.scoreFCSession)[[206]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L154-L163). However, to be safe on finalization, we might invoke it again to capture any late answers:
    * E.g., finalizeAssessment could do: if FC blocks exist and fc\_scores not complete, invoke score\_fc\_session internally. But that might be redundant if front-end already did it.
    * Another approach: ensure front-end calls score\_fc\_session just before finalizeAssessment. The sequence in front-end isn't explicit in AssessmentComplete; but given RealFCBlock is separate, possibly they call it and store FC result earlier.
    * However, the safer unified way: finalizeAssessment could call score\_fc\_session with basis:"functions" and version 'v1.1' on every finalize, ignoring errors (like if already done).
    * We see in rescoreSession.ts utility, they call a cleanup\_profiles then score\_prism but not score\_fc\_session[[207]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/utils/rescoreSession.ts#L17-L25). Perhaps because fc\_scores persisted anyway.
    * Our plan: finalizeAssessment will not explicitly call score\_fc\_session (to avoid adding latency), expecting that if FC was done, fc\_scores is there. We rely on front-end or triggers. Alternatively, we could create a trigger so that when fc\_responses inserted or count hits full, it calls score\_fc\_session automatically (via db trigger or function invocation). That might be neat (observability).
    * For now, we leave finalizeAssessment as is, just ensure it passes any needed flags to score\_prism (like partial\_session flag if applicable).
12. Minor change: finalizeAssessment currently generates share\_token if none[[205]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L172-L181). If we ensure share\_token is created at session start, we could skip generation here. However, currently they add it here in case it wasn’t set.
    * We should confirm: in assessment\_sessions, share\_token is added via migration for existing rows and new ones require it not null[[27]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L2-L10). If Supabase auto-generates default? They used gen\_random\_uuid() to backfill but did not set a default in schema. We might want to set default DEFAULT gen\_random\_uuid() on share\_token so every new session gets one. The migration didn’t explicitly set default, just not null with index[[27]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L2-L10), meaning the code that creates a session must insert a token.
    * I suspect the session creation code (maybe start\_assessment function or useEmailSessionManager.startAssessmentSession) generates a token when inserting. Need to verify:
      + The creation might be in useEmailSessionManager.ts where they have startAssessmentSession() likely doing insert into assessment\_sessions.
      + They possibly use gen\_random\_uuid() default at DB or generate client side.
    * To be thorough, we can add a default in DB for share\_token and remove generation from finalizeAssessment, but it's not critical. We'll ensure no duplicates.
13. Otherwise, finalizeAssessment continues to:
    * call score\_prism (which now uses unified engine),
    * update session (fine),
    * return profile and token (fine).
14. We should adjust finalizeAssessment’s error handling if needed (but likely fine as is, it checks maintenance, etc.).
15. **Update get-results-by-session Edge Function**:
16. Ensure it uses the share\_token securely. It likely already calls the get\_profile\_by\_session RPC[[69]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L20-L28)[[70]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L43).
17. We should verify that get\_profile\_by\_session returns the fields needed. The function as written returns public.profiles%ROWTYPE (which includes everything) if token matches[[208]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L32-L40). That’s good.
18. However, the Edge function might not return the entire profile directly to client for security? Given they do return it in Results page JSON, it’s fine because share token authorized it.
19. Possibly, we should exclude email or user\_id in that response for privacy. get\_profile\_by\_session currently joins with sessions but selects all from profiles. That includes user\_id and possibly IP hash if stored, etc. The question is: do we want to leak user\_id or email to someone with just the token link? The share link is meant to allow sharing with e.g. a friend, so presumably the friend can see the profile data. Email in session could identify the person, which might not be intended to share.
    * It would be wise to not return PII in that context. The get\_profile\_by\_session join with assessment\_sessions could be restricted to not include email or any identifying info. The function currently selects from profiles as p.\* and joins on sessions s to filter by token and completed status[[70]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L43), but does not explicitly remove fields. However, the profiles table likely doesn’t have email (that’s in sessions), and they did not select s.email or user\_id in that function. They only used s in WHERE clause and effectively return columns of profiles. So user\_id is returned (since profiles has user\_id as well)[[209]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1031-L1039), but email is not in profiles (unless they copied it, which I doubt).
    * So share link viewer gets user\_id (a UUID) which is probably fine (not very identifying without access to user table).
    * If we want maximum privacy, we could create a safe view or adjust the RPC to exclude user\_id for anonymous views. But not requested explicitly; we note it as a potential risk but low (someone with share link likely knows the person anyway).
20. Ensure get-results-by-session is JWT-optional (config says verify\_jwt=false so anon can call it)[[210]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/config.toml#L40-L43). This is correct.
21. **Remove deprecated flows on Frontend**:
    * The front-end currently has two flows at completion:
    * In AssessmentForm.handleComplete, it calls supabase.functions.invoke('score\_prism', ...) and then navigate[[20]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L18-L26).
    * In AssessmentComplete.useEffect, it calls finalizeAssessment and navigates[[144]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L98-L106)[[211]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L124-L131). Possibly the app uses one or the other depending on context (maybe if they use the new UI, they show an intermediate screen).
    * We should unify front-end to use **finalizeAssessment** exclusively for completion:
    * It handles everything (scoring + share token + session finalize) in one go.
    * It also covers the scenario where user might reopen the tab right at completion (since finalizeAssessment checks if profile already exists).
    * So, modify AssessmentForm.onComplete to not call score\_prism directly. Instead:
      + It can push the app into an "AssessmentComplete" component that triggers finalizeAssessment (with a loading spinner and eventual navigation).
      + Indeed, the presence of AssessmentComplete component suggests this was intended: e.g. perhaps if the user is logged in vs not, they might have had different flows.
    * Concretely, we can do: In handleComplete (Assessment.tsx)[[20]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L18-L26), replace the invoke('score\_prism') with navigation to a route or state that shows AssessmentComplete. Possibly simply navigate to /results/:sessionId with a state flag to wait. But better:
      + Use onComplete to set a state that renders <AssessmentComplete> overlaying the form, passing it the responses and sessionId. Actually, the code is already structured: Assessment.tsx uses showForm state and toggles to show either form or not.
      + Possibly easier: in handleComplete, instead of navigate, set a state to render an <AssessmentComplete> that calls finalizeAssessment. Actually, AssessmentComplete is not a route, it’s a component used inside Assessment page (maybe appears when form is done).
      + On looking at Assessment.tsx code: It returns AssessmentForm if showForm is true[[212]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L48-L56); otherwise it shows a landing page with SavedAssessments. So how does AssessmentComplete come into play? It might be rendered by AssessmentForm internally on complete? Actually, in AssessmentComplete, they import <ResultsV2> and it’s likely shown in place of form.
      + Possibly, AssessmentForm after final question triggers onComplete which in some version replaced the form UI with an embedded results. Perhaps AssessmentForm conditionally renders <ResultsV2> if scoring is done. Actually, AssessmentComplete uses <ResultsV2> to display results inline (with share link, etc)[[213]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L6-L14)[[214]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L8-L16). So maybe they designed it so that if user is on same page after finishing, they see results immediately (without navigating to standalone /results page).
      + The code in AssessmentComplete even includes logic to generate PDF, copy link, etc. So it's a self-contained results view used immediately after test. Meanwhile, the /results route is more for visiting via link later.
      + If that’s the case, we should continue to support this "inline results" scenario as a feature.
      + The unify plan:
      + When user completes the test in the same session, show them results immediately (AssessmentComplete component).
      + That component calls finalizeAssessment and then sets scoring state with returned profile[[215]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L76-L84)[[216]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L110-L118) and renders <ResultsV2> (with that profile data).
      + It also navigates to /results/:sessionId?token=... for consistency in URL (they do navigate inside finalizeAssessment success block[[211]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L124-L131)). Actually, they do navigate after setting scoring and tracking – maybe to allow sharing link to work properly and to have a distinct URL for results page.
      + Indeed, navigate(/results/${sessionId}${tokenParam}) is called[[211]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L124-L131), meaning even in inline results flow, they *redirect to the /results route* once data is ready. This is interesting because it means the results page will mount and call get-results-by-session again.
      + They probably do that to unify code path (so results display is always on /results page). In that case, the <ResultsV2> inside AssessmentComplete might be unused or only visible for a split-second? Or perhaps they originally planned to not navigate but decided to do it for simplicity.
      + Given they do navigate, the ResultsV2 component might not actually get used, since as soon as data is back, they navigate away.
      + Perhaps the navigate is for logged-in users only or something? Actually code: if (data.ok && data.profile) { ... navigate('/results/...') and return; } else throw ...[[216]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L110-L118)[[211]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L124-L131). So yes, they always navigate on success.
      + So the actual result view is always the /results route. That means the AssessmentComplete component’s purpose is mostly to call finalizeAssessment and handle loading, not to show results itself.
      + That simplifies: we can rely on /results for showing results. So our main concern is ensure finalizeAssessment is called and then /results works.
    * So, we will ensure **AssessmentForm.onComplete uses finalizeAssessment**:
      + Possibly by simply not calling score\_prism and letting AssessmentComplete effect handle it. Actually, in code, AssessmentComplete’s useEffect triggers finalizeAssessment as soon as it mounts[[217]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L72-L80)[[144]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L98-L106).
      + So, we can modify AssessmentForm.onComplete to set some state that causes AssessmentComplete to mount. However, looking at code, AssessmentComplete is not directly used in Assessment.tsx. Instead, likely, the AssessmentForm itself, upon calling onComplete, might unmount and maybe they intended to mount AssessmentComplete but didn't in code?
      + Wait, perhaps onComplete triggers a navigate to /assessment (or rather in handleComplete they navigate to /results directly if using path).
      + Actually, in Assessment.tsx handleComplete, they call score\_prism then navigate(/results) immediately[[218]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L22-L26). That likely bypassed AssessmentComplete entirely in that flow.
      + But in AssessmentComplete’s effect, they navigate to results after finalize. Possibly some redundancy.
    * Given the complexity, the simplest approach:
      + Remove score\_prism direct call in handleComplete.
      + Instead, call supabase.functions.invoke('finalizeAssessment', {session\_id, responses}) (similar to what AssessmentComplete does, though that collects responses from props).
      + But if we are navigating to /results anyway, maybe we don't need to double call finalizeAssessment (the results page will call get-results which will find profile since finalizeAssessment inserted it).
      + Actually, if we navigate immediately to /results without calling finalizeAssessment, the results page will query get-results-by-session, which will not find a profile if finalizeAssessment wasn’t run. So something must run it. That’s why they did either score\_prism or finalizeAssessment before navigating.
      + So yes, do finalizeAssessment first, then navigate.
      + Perhaps reuse AssessmentComplete component logic:
      + Instead of duplicating code, we can navigate to a new route or just incorporate finalizeAssessment call in handleComplete then navigate, but better to keep their flow: mount AssessmentComplete to handle it.
      + We might do: in handleComplete, call setShowForm(false) and render an <AssessmentComplete> (we’d have to modify Assessment.tsx to conditionally render it).
      + Or simpler: directly call finalizeAssessment in handleComplete and handle its response: ~ Pseudocode:
      + setLoading(true);  
        const { data, error } = supabase.functions.invoke('finalizeAssessment', { body: { session\_id, responses } });  
        if (error or data.error) show error (maybe toast), setLoading(false);  
        else {  
         const token = data.share\_token;  
         navigate(`/results/${sessionId}${token? '?token='+token:''}`);  
        }
      + ~ This is essentially what AssessmentComplete does (with some extra tracking).
      + Implementing this directly in handleComplete avoids needing the AssessmentComplete intermediate component, which simplifies user flow (one less component mount).
      + We should carry over the event tracking calls (trackAssessmentComplete, trackResultsViewed, trackLead)[[219]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L114-L123) to maintain analytics.
        - These can be done after finalizeAssessment returns success.
      + This way, as soon as scoring is done, we send user to /results page where the Results component will fetch the profile (which is now definitely in DB).
      + Doing this direct in handleComplete is probably the cleanest. We then might deprecate AssessmentComplete component entirely (or keep it as fallback).
    * **Implement**: Modify handleComplete in Assessment.tsx:
      + Instead of await supabase.functions.invoke('score\_prism'), do the logic above calling finalizeAssessment (with try/catch, setLoading UI maybe). Possibly show a loading state in UI (maybe reuse the AssessmentComplete UI for loading indicator).
      + Actually, since Assessment.tsx currently just calls navigate after scoring, it doesn’t show loading spinner. But a call to finalizeAssessment could take some hundreds of ms. It's better UX to show something.
      + Perhaps we could quickly push to results page and let that page handle waiting via its retry loop (it tries up to 12 times on 409)[[159]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L34-L42). But finalizeAssessment is quick enough that likely by the time results page starts fetch, profile is ready.
      + However, race could happen if results page fetches profile slightly before finalizeAssessment finished (especially if network delays differ). But they built a retry on 409 (they likely return 409 in get-results if profile not ready? Actually they check error.status===409 in results page and retry)[[159]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L34-L42).
      + Not sure where 409 would come from – maybe from score\_prism if it is running concurrently? Possibly get-results-by-session returns 409 if it detects scoring still in progress. If not implemented, they might have planned it.
      + If finalizeAssessment is called before navigation, by the time navigate happens (especially with token in URL), the profile is definitely there. That’s safer and simpler – user sees a small delay then results with no intermediate “Loading” in results page.
      + We'll do that: call finalize, then navigate. Provide feedback if error (maybe show an alert or navigate to an error page).
    * Clean up: After this change, the AssessmentComplete component might be redundant. However, it also has PDF export, copy link UI which are nice. But those can be moved to the /results page UI (maybe ResultsV2 or others). Actually, copy link is likely duplicated in results page too (if not, we ensure results page provides link copy).
    * We'll plan to remove or archive AssessmentComplete to avoid confusion, once finalizeAssessment is integrated directly.
    * **Verify front-end env**: ensure .env.local contains correct function URLs if needed. Usually supabase JS auto picks up functions. Just ensure finalizeAssessment is callable.
22. **Database migrations** (if needed):
    * Add default for assessment\_sessions.share\_token: ALTER TABLE assessment\_sessions ALTER COLUMN share\_token SET DEFAULT gen\_random\_uuid(); (So that whenever a new session is inserted via supabase client (which likely uses the default if not provided), it gets a token).
    * Possibly refine RLS:
    * On profiles, remove the “Public can view profiles…” policy if it still exists (the migration [42] added it; migration [37] removed an "Anonymous can view by session" but not sure if it removed the "dashboard statistics" one).
      + Actually [42] names policy "Public can view profiles for dashboard statistics"[[220]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819022459_12718b6c-29ce-46d9-8156-aa05e6f29eb1.sql#L4-L8). In [37] they dropped a policy named "Anonymous can view profiles by session"[[5]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L17-L25). They did not explicitly drop "Public can view profiles for dashboard statistics". That means currently any anon can still select \* from profiles because that policy USING true is still there. That is a **security hole** unless RLS was disabled (but RLS enabled on profiles likely).
      + We should drop or tighten that policy. Perhaps replace it with:
      + Only allow select on profiles via a secure function. Actually since we have that secure function, ideally no direct select.
      + If needed for analytics, they created safe view and granted select[[166]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L70-L77), so broad select is unnecessary.
      + So migration: DROP POLICY "Public can view profiles for dashboard statistics" ON public.profiles;.
      + And create a new policy: Authenticated users can select their own profile: CREATE POLICY "User can view own profile" ON public.profiles FOR SELECT USING (user\_id = auth.uid()); (and perhaps for update if they allow recompute? But recompute is via service, not user).
      + For anon, we deliberately do not allow select (they must use function with token).
      + No insert/update from client, those happen via service role only, so no need for those policies for anon.
      + This ensures even if someone tries to circumvent the UI and query profiles, they either must be logged in to see their own, or else can’t.
    * Confirm RLS is enabled on profiles (likely yes, since policies exist, if not we ensure).
    * Might adjust assessment\_sessions select policy: currently “Users can read assessment sessions USING (true)”[[26]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L46-L55). This means anyone can read any session – possibly not great if email is in sessions (which it is, they store email if provided[[221]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L724-L732), and ip\_hash maybe if added).
      + Now that share\_token is implemented, one could theoretically fetch any session by id if they have the id. If session contains email/IP (hashed though), that could leak PII.
      + We could tighten: USING ( auth.role() = 'authenticated' AND user\_id = auth.uid() ) OR ( auth.role() = 'anon' AND false ). But since anon uses other means, and there's likely code relying on open access (like the resume function verifying completion count[[11]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L34-L42) which uses supabase.from('assessment\_responses').select count with session\_id, that will fail if anon cannot select).
      + Actually, that resume check still works if responses select is open (which it is). They allowed all reads for now.
      + Considering time, we may not tighten these now, focusing on profiles which is most sensitive. But in long term, we want to restrict sessions too. Perhaps after user accounts linking, sessions reading can require either own or share token.
      + For now, note as risk: currently any malicious client with the anon key can query all assessment\_sessions (with emails). Because policy allows it. That should be addressed by at least requiring share\_token or user match. But then resume and other front-end flows would need adjusting (like resume would need user token or share token).
      + This could be out-of-scope to fix fully here but we mention as a risk and maybe plan a mitigation like: hide email in sessions from anon by splitting it out or only allowing if share\_token known. Possibly define a safe RPC or at least don't expose more than needed.
    * In summary, for now do: Remove broad read on profiles, implement user-only read.
    * Insert or update scoring\_config entries for any new keys (if we choose to add, say, high\_fit, moderate\_fit thresholds into config if not there). Actually, they likely are there (the migration 20250820\_prism\_scoring\_config\_upserts.sql might contain some default config values).
    * We saw scoring\_config usage for results\_version, etc. Possibly thresholds might have been constants in code. They mention “Phase 4: Fit band logic driven by config thresholds”[[222]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L910-L918), implying they might have put them in config.
    * Check 20250820\_prism\_scoring\_config\_upserts.sql:

# Executive Summary

The PRISM cognitive profiling assessment system currently spans a React/Tailwind frontend and a Supabase backend (Edge Functions + Postgres). Scoring is implemented server-side via Supabase Edge Functions (score\_prism for Likert and combined results, score\_fc\_session for forced-choice) and results are stored in a profiles table[[1]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1-L6)[[2]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L29-L38). However, the scoring logic is fragmented: parts of the logic (e.g. forced-choice mapping) are duplicated or split across front-end and multiple backend functions[[3]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L373-L382)[[4]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L66-L74). This has led to timing misalignments (e.g. scoring triggered before all responses collected) and reliance on outdated fallbacks in some paths[[3]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L373-L382). Security is also a concern: previously, result retrieval relied on an open profile access policy, which we are tightening via share-token protected links[[5]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L17-L25).

To address these issues, we propose a **unified scoring engine** module (versioned v1.2.1) that encapsulates all PRISM scoring logic and is used by both front-end and backend. All scoring will remain server-side (non-negotiable) for consistency and security, but the front-end will now invoke a single completion endpoint (finalizeAssessment) instead of calling multiple functions or partial computations. The unified engine deterministically handles Likert and forced-choice inputs, dimensional overlays, and confidence calibration in one place. We unify versioning (tagging results with a version) and ensure outputs remain consistent with previous version (v1.2.0) except for intentional improvements (e.g. always using calibrated confidence). We also strengthen security by strictly requiring a share token for result access (removing any broad read policies)[[5]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L17-L25) and by using Row-Level Security so only owners (or token-holders) can view profiles. This report provides a comprehensive plan: an inventory of the current system, a detailed design of the new scoring engine, an updated architecture diagram, an implementation roadmap with database migrations and code changes, a risk/issues analysis with mitigations, testing plans, rollout/rollback procedures, and observability/guardrails to monitor the live system. With this plan, we fully operationalize the PRISM assessment end-to-end on the live site, improving maintainability and consistency without disrupting the user experience.

# System Inventory

**Frontend** – The React/TypeScript single-page app (Vite) implements the assessment UI and results display. Key routes (via React Router) include:

* **/assessment** – Entry to the PRISM assessment form (component: <AssessmentForm> inside <Assessment> page)[[6]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L163-L171)[[7]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L2-L5). Here users answer Likert and forced-choice questions. The form logic saves each response to Supabase and tracks progress.
* **/results/:sessionId** – Results page displaying the scored profile. It expects a sessionId (UUID for the assessment session) and an optional token query param for anonymous access[[7]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L2-L5)[[8]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L126-L134). The Results page calls a Supabase Edge Function get-results-by-session to fetch the final profile securely (more below)[[9]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L26-L34)[[10]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L41-L48).
* **/assessment?resume=<id>** – Resume an incomplete assessment. If a session is resumed via query param, the app checks how many responses exist and potentially redirects to results if already complete[[11]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L34-L42).
* **Other routes** (read-only or informational): e.g. /history, /dashboard, and various content pages (About, FAQ, etc.) as listed in the router[[12]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L10-L20)[[13]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L34-L42). These are mostly static or unrelated to scoring logic. The PRISM assessment feature specifically involves the Assessment and Results pages, plus UI components like **ResultsV2**, **TraitPanel**, **OverlayChips** that format the output (which we will preserve).

**Frontend State & Components** – Within the assessment flow, <AssessmentForm> manages question display and saving of answers. It loads the question library and creates/uses a session ID. On each answer, it calls saveResponseIdempotent to upsert the response to the database[[14]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L701-L710). It updates local state and also updates the assessment\_sessions record (e.g. to save an email if provided on the first question, or increment completed question count for progress)[[15]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L724-L733)[[16]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L770-L778). After the last question, client-side validation runs to check for inconsistencies or invalid patterns (using validatePrismAssessment)[[17]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L793-L801). If critical validation fails, submission is blocked and the user is warned[[18]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L796-L804). If validation passes, the form’s onComplete handler is invoked with all responses and sessionId[[19]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L860-L868).

* In the **current implementation**, onComplete simply calls the score\_prism edge function and navigates to the results page[[20]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L18-L26). We also see a newer AssessmentComplete component that calls a finalizeAssessment function (Edge Function) to handle completion[[21]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L99-L108)[[211]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L124-L131) – this appears to be an updated flow that wraps scoring and session finalization (discussed below).

**Backend (Supabase)** – The Supabase project provides persistence and secure server logic:

* **Database Tables**: Key tables include:
* assessment\_sessions – Tracks each assessment attempt (session). Contains id (UUID), optional user\_id (if logged in), session\_type (e.g. 'prism'), started\_at, completed\_at, status, completed\_questions, metadata (JSON for IP, UA hashes, etc.), and a share\_token for secure result sharing[[27]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L2-L10). RLS is enabled[[25]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L36-L44). Initially, policies allowed broad select for ease[[26]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L46-L55), but this was later tightened with share tokens. The share token is a random UUID string stored per session[[28]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L4-L12), enforced not null and unique[[29]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L13-L21).
* assessment\_responses – Stores each answer. Fields: id (UUID), session\_id (FK to sessions), question\_id, question\_text, question\_type (e.g. 'likert', 'forced-choice'), question\_section, answer\_value (text), answer\_numeric (optional numeric value), response\_time\_ms, etc[[30]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L16-L25). RLS enabled with policies allowing inserts and selects (initially open since assessment was anonymous)[[164]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L60-L68). This table holds both Likert and forced-choice answers. (In legacy mode, forced-choice questions were included as individual questions with options A/B/C/D, whereas now they might be separate.)
* assessment\_scoring\_key – Static reference data mapping question\_id to scoring parameters. In code, this is fetched entirely for scoring[[32]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L229-L238)[[33]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L239-L248). Each row may have JSON fields: fc\_map (for forced-choice mapping from option to function/block) and meta (extra metadata)[[33]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L239-L248). This key tells the scoring engine how to interpret each answer (which function it relates to, etc.).
* scoring\_config – Key-value store for various tuning parameters. The score\_prism function reads configuration like results\_version, dim\_thresholds (thresholds for dimensional counts), neuro\_norms (norm mean/sd for neuroticism), overlay\_neuro\_cut, overlay\_state\_weights, fc\_expected\_min (min number of FC questions expected), softmax\_temp (temperature for softmax in type probability), etc.[[34]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L255-L263)[[223]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820_prism_scoring_config_upserts.sql#L2-L7). These allow dynamic adjustment of scoring without code changes. We will add new entries here if needed (e.g. fit band thresholds if not already present).
* profiles – Stores completed profile results (the output of scoring) for each session. Key fields: session\_id (FK, unique per profile)[[224]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250908000000_profile_session_constraints.sql#L14-L19), user\_id, type\_code (e.g. “LII”, “IEE” etc.), base\_func & creative\_func (the two primary functions for that type), confidence (High/Moderate/Low result confidence)[[38]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1032-L1040)[[39]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1050-L1058), validity\_status (pass/warning/fail based on quality checks)[[40]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1032-L1038), numeric scores like score\_fit\_raw and score\_fit\_calibrated (fit scores before and after cohort normalization)[[41]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1039-L1047), fit\_band (High/Moderate/Low fit category)[[42]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1041-L1045), top\_gap (difference between top two type fits)[[43]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1042-L1045), close\_call (boolean if gap < threshold)[[44]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L875-L880)[[45]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L912-L919), fc\_answered\_ct (how many forced-choice blocks answered)[[46]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1046-L1054), overlay (e.g. “+” or “–” indicating neuroticism overlay)[[47]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1076-L1084)[[48]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L674-L682), and many JSON fields: strengths (strength score per function), dimensions (count of high responses per function), top\_types (array of top 3 type codes), top\_3\_fits (their scores and shares)[[49]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1051-L1059)[[50]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1080-L1088), validity (detailed validity metrics)[[51]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L702-L710)[[50]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1080-L1088), and so on. The profile is inserted or upserted by score\_prism upon completion[[52]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1125-L1133)[[53]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1143-L1150). RLS on profiles was initially opened for analytics[[54]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819022459_12718b6c-29ce-46d9-8156-aa05e6f29eb1.sql#L4-L12), but with share tokens we removed the anonymous read policy[[5]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L17-L25) in favor of the secure function. Now, direct selects on profiles require either the user’s own auth or the security-definer function with token.
* Other tables: fc\_blocks, fc\_options, fc\_responses, fc\_scores – These support the new forced-choice system. **fc\_blocks** defines each forced-choice block (e.g. a scenario with a pair of statements) with id, code, title, etc., versioned (we use version 'v1.1' or 'v1.2' for content updates)[[55]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L46-L54)[[225]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L48-L52). **fc\_options** holds the options for each block (each option with text prompt and a weights\_json mapping that option to certain functions or types)[[56]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L59-L67). **fc\_responses** stores which option a user chose for each block (session\_id, block\_id, option\_id)[[57]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L112-L120). **fc\_scores** stores the computed aggregate scores from forced-choice for a session (scores\_json field containing either function scores or type probabilities, plus count of blocks answered)[[58]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L299-L308)[[59]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L312-L321). These tables enable the structured FC implementation (in contrast to embedding in responses).
* Calibration tables: calibration\_model – Holds parameters for confidence calibration curves per stratum (combination of dimensional band and overlay)[[60]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L96-L105). The PrismCalibration class looks up these records to apply isotonic/Platt scaling to raw confidence[[61]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L98-L107)[[62]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L120-L128). We use this to calibrate the numeric confidence.
* Views: Convenience views exist for analytics, e.g. v\_fc\_coverage to compute how many forced-choice questions were answered per session and categorize coverage "None/Partial/Full"[[63]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql#L10-L19)[[64]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql#L22-L25) (and an fc\_count which is also stored in profiles via trigger[[226]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql#L56-L64)). Also v\_recent\_assessments\_safe (wrapping a function get\_recent\_assessments\_safe) provides limited info (date, type, country, fit score) for recent profiles with proper security[[65]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L30-L39)[[66]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L62-L70). We use these views for dashboards so we could drop broad access to raw profiles without losing insight.
* RPC Functions: Several Postgres functions support secure access and operations:
  + get\_profile\_by\_session(session\_id, share\_token) – Security-definer function that returns a profile row if the session’s share\_token matches and the session is completed[[69]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L20-L28)[[70]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L43). This is used by the Edge Function for result fetching. It ensures that even without auth, possession of the correct token is required to retrieve a profile.
  + get\_user\_assessment\_scores(p\_session\_id) – Likely returns profile data for a given session, but only if the session belongs to the authenticated user (implemented as a Postgres function using auth.uid() in a WHERE clause, presumably).
  + notify\_admin – Called on assessment completion to alert an admin (via email or other) with session details[[227]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L72-L80)[[228]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L204-L212). It likely sends a link containing the share\_token so admins can view results easily.
  + A number of functions for managing questions or progress (e.g. load\_session\_responses to fetch all responses for a given session, used for resume[[229]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L33-L42), save\_response to insert a response (the front-end currently upserts directly, but a function exists), get\_progress to return count of answered questions, etc.). These are part of the Supabase Edge Functions layer and discussed below.
* **Supabase Edge Functions**: All custom logic is implemented as Deno Edge Functions:
* **score\_prism** – the core scoring engine (current version v1.2.0 in code), implemented in TypeScript[[71]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1-L9). It is invoked with a JSON body containing at least session\_id. It loads all responses for that session, applies the scoring algorithm, and upserts the profiles row[[52]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1125-L1133). It returns JSON with status: "success" and the computed profile data[[53]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1143-L1150). Key steps:
  + Validate input and ensure environment is configured (Supabase URL and service key)[[72]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L142-L150). If not, return 500.
  + Use the service-role client (bypasses RLS) to perform queries[[72]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L142-L150).
  + Log the start event with version[[74]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L156-L164).
  + Load type prototype definitions from DB (type\_prototypes table) mapping each type to its Model A function blocks, to use for scoring. If not found or incomplete, fall back to hardcoded prototypes[[75]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L158-L167)[[76]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L178-L186).
  + Fetch all responses for the session from assessment\_responses[[77]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L193-L201). Deduplicate by question (keep the latest answer per question) to avoid double-counting[[78]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L215-L224).
  + Fetch the entire assessment\_scoring\_key and build a dictionary keyed by question\_id (validating JSON fields fc\_map and meta for safety)[[32]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L229-L238)[[33]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L239-L248).
  + Fetch needed config values from scoring\_config as described (results\_version, thresholds, norms, etc.)[[34]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L255-L263).
  + **Forced-Choice integration**: It checks for precomputed FC results. If found in fc\_scores (for version 'v1.1' or 'v1.2') it uses those[[58]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L299-L308)[[59]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L312-L321) (and sets fcFuncCount proportionally)[[182]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L317-L325). If not, it falls back to legacy per-question processing:
  + For each response with scale\_type starting "FORCED\_CHOICE", map the answer (A/B/C/D) via the fc\_map to either a function or block and increment counters accordingly[[3]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L373-L382)[[81]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L383-L390). It also counts how many FC questions answered (fcAnsweredCount)[[82]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L373-L381).
  + This dual approach supports both the new FC blocks and any legacy embedded FC questions. In our unified plan, we'll prefer using the structured approach (fc\_scores) and eventually remove the need for per-question mapping once all FC are via blocks.
  + Process each response for Likert and other questions:
  + Parse the answer value to a numeric 1–5 (handling reverse-scored by flipping value if needed)[[230]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L355-L364)[[83]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L365-L373).
  + Depending on the question’s tag (from scoring\_key):
    - If tag ends with "\_S" (strength question for a function), collect the value in an array for that function’s strengths.
    - If tag ends with "\_D" (dimensional question), collect value in array for that function’s dimensional responses.
    - If tag is "Core"/"Critic"/"Hidden"/"Instinct", add the value to a block count (blockLikertCount).
    - If tag is "N" or "N\_R", add to neuroVals (for neuroticism).
    - Track social desirability responses (sdSum, sdN) and pair questions (store pairs for consistency check)[[86]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L369-L372).
  + (This part of code is somewhat implicit but can be inferred from how results are used later. The debug output shows how these are counted[[231]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L945-L954)[[201]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L950-L958).)
  + Compute validity metrics after aggregating:
  + attentionFailed: count of attention-check questions that were answered incorrectly (tracked during response loop as e.g. if (pair) ... or a separate attention list from config attention\_qids processed after loop[[232]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L420-L424)).
  + inconsistency index: measure of internal consistency using the pairs of similar questions (likely average difference or something; code stores it as validity.inconsistency and checks thresholds)[[51]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L702-L710)[[89]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L724).
  + sd\_index: standard deviation index of answers (to catch if user put nearly random or all-neutral answers)[[51]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L702-L710)[[89]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L724).
  + Based on these, determine validityStatus: "pass"/"warning"/"fail". The logic in code (v1.2.0) is: if inconsistency >= 2.0 or sd\_index >= 5.2, status = "fail" (and set confidence = Low)[[89]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L724); else if inconsistency >= 1.5 or sd\_index >= 4.9 or ≥2 attention fails, status = "warning" (and confidence = Low if ≥2 attention fails, or Moderate if 1 fail)[[88]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L721-L729); else if inconsistency >= 1.0 or sd\_index >= 4.3 or 1 attention fail, confidence = Moderate (status still "pass")[[90]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L726-L730); otherwise "pass" with High confidence. We will preserve these thresholds.
  + Compute function-level aggregates:
  + Strength score per function: likely the average of all S-type answers for that function on 1–5 scale.
  + Dimensional count per function: how many D-type answers for that function are high (≥ threshold, probably 4 or 5). In code, they later determine dimBand by checking if max dimension count ≥3, etc.[[117]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L892-L900).
  + Block percentages: Sum all Likert block scores and normalize to percentages per Core/Critic/Hidden/Instinct[[92]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L736-L744). Do same for forced-choice block counts[[233]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L734-L742), then compute a weighted blend blocks\_norm\_blend based on amount of Likert vs FC answers[[174]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L750-L758). This yields the final block distribution used for output.
  + **Type matching (distance-based)**:
  + Build a prototype target for each of the 16 types: for each type’s Base/Creative vs other functions, assign a target value 1–5 based on block weights (base=1.0 -> target 5, vulnerable=0.1 -> target ~1, etc.)[[95]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L768-L776)[[97]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L780-L788).
  + For each type, compute the Euclidean distance between the user’s strength profile and that type’s prototype vector[[97]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L780-L788). Convert distance to a match score (1 – dist/max\_dist) scaled to 0–6.5 range[[98]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L784-L792). Map that to a 0–100 raw fit percentage (fitRaw)[[99]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L794-L803).
  + Calibrate those fit scores against a recent cohort if available: query last 90 days of profiles for score\_fit\_raw[[189]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L807-L815). If ≥50 records, compute mean & SD, then for each type do fitAbs = 50 + 15 \* (rawFit - mean)/SD, clamped to [20,85] (approx normative scoring)[[101]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L826-L835)[[102]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L830-L839). If not enough data, fallback formula fitAbs = rawFit \* 0.65 + 20 clamped to [20,85] (based on historical tuning)[[103]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L836-L844). This yields fitAbs (stored as score\_fit\_calibrated in profile).
  + Compute type share percentages (sharePct) via softmax on rawScores (exponential with a temperature factor)[[105]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L845-L850). This gives a relative probability for each type summing to ~100%.
  + Determine the top 3 types: sort all types by fitAbs, then tie-break by sharePct, then by count of coherent functions >=3D, then by sum of FC support for base+creative[[106]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L859-L867). Select the first 3 as top\_types.
  + Compute topGap = fitAbs(top1) - fitAbs(top2) (difference between best and second-best)[[44]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L875-L880) and flag closeCall = (topGap < 3) as a boolean.
  + Determine fitBand: High/Moderate/Low fit. Current logic uses thresholds (not explicitly in config, but implied): if top1\_fit ≥ 75 and topGap ≥ maybe ~6–10 => High; if top1\_fit ≥ 55 or topGap ≥ ~3 => Moderate; else Low[[107]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L910-L918). We will use 75/55 and 6/3 (based on code comments and logs)[[234]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1059-L1067). These categorize how strong the overall match is.
  + **Overlay traits**:
  + Neuroticism overlay: Compute the mean of all neuroticism-tagged answers (nMean) and Z-score it using neuro\_norms (mean=3, sd=1 default)[[110]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L670-L678). If Z ≥ cut (config overlay\_neuro\_cut, default 0.50) then overlay\_neuro = "+" (neurotic plus); if Z ≤ -cut then "–" (neurotic minus); else "0" (no strong neuroticism indicator)[[48]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L674-L682).
  + State overlay: Compute a weighted index of state-related questions (stress, time, sleep, focus) where each answer is centered (value-3) and weighted by overlay\_state\_weights (defaults stress +0.35, time +0.25, sleep -0.20, focus -0.20)[[111]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L680-L688). If index ≥ cut (same cut 0.50) then overlay\_state = "+" (overall more stressed/pressured state); if ≤ -cut then "–"; else "0"[[112]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L686-L694).
  + The code currently sets overlay = overlay\_neuro as the primary overlay for output (to avoid breaking UI that expects one overlay value)[[113]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L690-L698). We will maintain a single overlay field (driven by neuroticism primarily) but include both in the profile for completeness.
  + **Confidence calculation**:
  + Compute p1 and p2 (top1 and top2 sharePct) and confidenceMargin = (p1 - p2) (in percentage points)[[115]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L881-L888).
  + Compute shareEntropy = entropy of the type share distribution (to measure how evenly spread the probabilities are)[[116]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L886-L894).
  + Determine dimBand for confidence stratification: if any function has dimension count ≥3, dimBand = '3-4D'; else if max dimension = 2, dimBand = '2D'; else '1D'[[117]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L892-L900).
  + Use the PrismCalibration class to calculate a raw confidence and calibrated confidence:
    - Raw confidence = sigmoid(a*topGap + b*(shareMargin/100) - c\*shareEntropy) using parameters (a=0.25,b=0.35,c=0.20 by default)[[235]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L58-L66)[[119]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L64-L72).
    - Look up the calibration model for that (dimBand, overlay) stratum in calibration\_model table[[61]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L98-L107). If found, apply isotonic interpolation to rawConf to get calibrated confidence[[62]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L120-L128). If not, fallback to Platt scaling: calibrated = sigmoid(-0.5 + 1.2 \* rawConf)[[194]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L150-L159).
    - The class returns raw, calibrated, and method plus a suggested band (we can derive band from calibrated: e.g., >0.8 High, >0.6 Mod, else Low). Code prints these[[236]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L126-L134).
  + In profile output, they store conf\_raw, conf\_calibrated, conf\_band (numeric rawConf and calibratedConf, and the method used as conf\_band or maybe band name)[[39]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1050-L1058). The textual confidence field in profile remains tied to validity gating ("High"/"Moderate"/"Low") which can override (if validity is fail or warning, they set confidence Low/Moderate regardless of calibration)[[89]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L724).
  + Mark session as completed in assessment\_sessions (status = 'completed', set completed\_at if not already)[[237]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L973-L982).
  + Fetch session record (to get email for admin notify, user\_id, etc.)[[126]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L988-L996).
  + Assemble the profileData object with all fields (including the new ones like top\_3\_fits, dims\_highlights, etc. as seen in code)[[209]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1031-L1039)[[39]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1050-L1058).
  + Upsert into profiles (on conflict session\_id)[[52]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1125-L1133). If a profile existed (e.g. re-score scenario), preserve original submitted\_at and set recomputed\_at[[127]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1109-L1118).
  + Log a completion event and return the profile in the response[[53]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1143-L1150).
* **score\_fc\_session** – separate function to calculate forced-choice scores (especially with the new block system). Called with session\_id (and basis: "functions" or "types", default "functions")[[129]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L14-L22). It:
  + Loads active FC blocks (for version e.g. 'v1.1') and all FC options[[131]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L29-L37)[[132]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L37-L45).
  + Loads all fc\_responses for the session[[133]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L39-L46). If none, returns with info "no fc responses"[[134]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L46-L54).
  + Index options by id for quick lookup[[135]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L54-L62).
  + For each response, get that option’s weights\_json (which is a map of keys to weights)[[136]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L69-L73). Sum these weights into a tally map[[136]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L69-L73). (E.g. if one option had {"Ne": 1, "Fi": 0.5}, then tally.Ne +=1, tally.Fi +=0.5 for that response).
  + After aggregating all answered blocks:
  + If basis == "functions": Normalize each key’s tally by dividing by the max and scaling to 0–100[[238]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L80-L88) (so the highest scored function gets 100).
  + If basis == "types": Sum to 1 (so each key becomes a probability)[[138]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L86-L90).
  + Log completion and upsert into fc\_scores (with session\_id, version, fc\_kind, the JSON scores, and blocks\_answered count)[[139]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L94-L102). Use onConflict on (session\_id, version, fc\_kind) to update if already exists.
  + Return the scores JSON in response[[140]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L105-L113).
  + In our plan, we’ll call this after forced-choice is done (front-end already does on completion of FC blocks) and/or within finalizeAssessment to ensure fc\_scores is up to date before computing final results.
* **finalizeAssessment** – an edge function that orchestrates assessment completion. Called by front-end at the end instead of directly calling score\_prism. It expects session\_id and responses array in body[[145]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L32-L40).
  + If a profile already exists for that session (meaning we scored it already), it logs that and simply updates session status and returns the existing profile (to handle double-submission gracefully)[[146]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L50-L59)[[239]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L83-L91).
  + Otherwise, it fetches the session (to confirm it exists and get user/email)[[148]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L99-L108).
  + **New in our plan**: It will invoke score\_fc\_session (basis "functions") for completeness, then invoke score\_prism[[149]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L119-L128). This ensures any FC answers are processed. If score\_fc\_session fails (e.g. no responses or error), we can continue (score\_prism will fall back to legacy method or partial scoring).
  + It awaits score\_prism and handles errors:
  + If score\_prism returns an error or a maintenance status, it propagates that with appropriate HTTP code (422 or 503)[[240]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L124-L132)[[150]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L140-L149).
  + On success, it gets a profile in the result.
  + It then generates or fetches the session’s share\_token:
  + If session already had a share\_token (we now ensure every session has one from creation, but code double-checks), use it; otherwise, generate a new UUID[[151]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L172-L180).
  + Update the session: set status 'completed', completed\_at (preserve if already set), completed\_questions (to responses count or use profile’s fc\_answered\_ct if present), and store the share\_token (especially if it just generated one)[[152]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L175-L183).
  + Fire-and-forget an admin notification (calls notify\_admin with session\_id and share\_token)[[153]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L70-L78)[[154]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L203-L211).
  + Return a JSON with { ok: true, status: 'success', session\_id, share\_token, profile, results\_url }[[155]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L216-L224). The results\_url is a convenience containing our site’s URL to results with the token (used if front-end wants to redirect or share).
  + This function basically wraps the whole flow in one call. The front-end will transition to results page after receiving this.
* **get-results-by-session** – an edge function used by the Results page to retrieve a profile by sessionId and token. It expects { sessionId, shareToken } in body[[9]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L26-L34).
  + It uses the service key to call the Postgres get\_profile\_by\_session(session\_id, token) function[[69]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L20-L28)[[70]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L43). If token matches and session status is 'completed', that function returns the profile row.
  + If token is wrong or missing, the function will not find a row (or could throw error). The current code likely returns a 401/403 in that case. The front-end handles errors by retrying if status 409 (not sure if used) or showing "Results not found" on 404/403[[159]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L34-L42)[[241]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L42-L48).
  + In our implementation, we’ll also include a guard: if no token provided but the profile exists and was previously accessible (for transition period), we may temporarily allow it (to not break old bookmarked links). But the plan is to enforce tokens going forward.
  + This function does not require auth (verify\_jwt=false in config)[[242]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/config.toml#L42-L46) since it relies on token for auth.

The above inventory details the moving parts we will modify or leverage. In summary: - **Frontend** will be adjusted to call finalizeAssessment (instead of directly calling score\_prism) and to handle the tokenized results link. The UI (ResultsV2, etc.) will be reused on the results page to display outcomes, preserving all current features (top-3 chart, trait breakdowns, etc.). - **Backend** changes revolve around consolidating logic into score\_prism (and shared code), removing duplicate forced-choice processing (use fc\_scores consistently), and tightening RLS so only correct access occurs (through token or user). - We also must verify environment variables (.env) are correctly set (e.g. service role key for Edge Functions, which we use via Deno.env in code[[72]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L142-L150)). We'll ensure no secrets are exposed in front-end (they are not; front-end uses anon key only).

This comprehensive inventory of the current system guides our unification strategy. All these components (routes, tables, functions, policies) will be accounted for in the design and implementation that follow.

# Architecture Diagram (Mermaid)

Below is a high-level component and data flow diagram for the PRISM assessment system, illustrating how the unified scoring engine will fit into the front-end and backend interactions:

flowchart TD  
 subgraph Client (Browser)  
 U[User] -->|starts assessment| A[Assessment Page<br/>(React)]  
 A -->|inserts new session| B[(assessment\_sessions)]  
 A -->|answers Likert Qs| D[(assessment\_responses)]  
 A -->|answers FC Blocks| D  
 A -->|calls finalizeAssessment| F[Edge Function: finalizeAssessment]  
 F -->|ensures FC scores via| G[Edge: score\_fc\_session]  
 F -->|invokes| H[Edge: score\_prism]  
 H -->|reads| D  
 H -->|reads| K[(assessment\_scoring\_key)]  
 H -->|writes profile| L[(profiles)]  
 F -->|updates| B  
 F -->|returns profile + token| A  
 A -->|navigates to| R[Results Page<br/>(React)]  
 R -->|calls get-results-by-session| Q[Edge: get-results-by-session]  
 Q -->|verifies token and returns profile| L  
 R -->|displays profile| U[User]  
 end  
 subgraph Supabase Backend  
 B[(assessment\_sessions)] ==> L  
 D[(assessment\_responses)] ==> H  
 K[(assessment\_scoring\_key)] ==> H  
 L[(profiles)] ==> Q  
 subgraph Functions  
 F[finalizeAssessment] -.->|calls| G  
 F -->|calls| H  
 G[score\_fc\_session] -->|stores fc\_scores| S[(fc\_scores)]  
 H[score\_prism] -->|reads fc\_scores| S  
 end  
 subgraph Database  
 S[(fc\_scores)] -.-> H  
 M[(calibration\_model)] -.-> H  
 end  
 end

**Diagram Notes:** The user interacts with the front-end Assessment page which creates a session record and saves each response to the database (arrows from Assessment Page to assessment\_sessions and assessment\_responses). When the user finishes, the front-end calls the finalizeAssessment Edge Function. This function (service-side) first ensures forced-choice answers are tallied (score\_fc\_session) and then invokes the unified scoring (score\_prism). The score\_prism function reads all responses and the scoring key and performs the computation (using in-memory logic and potentially reading calibration data). It writes the result to the profiles table and returns the profile data. The finalizeAssessment function updates the session status to completed and attaches a share token, then returns the profile and token to the client. The front-end then navigates the user to the Results page with the sessionId and token. The Results page calls get-results-by-session, which checks the token against the profiles and assessment\_sessions (via the secure DB function) and returns the profile. Finally, the profile data is rendered to the user (with charts, interpretations, etc.). Authentication is only required if the user wants to save or view results in their account; otherwise, the token in the link secures the results. The diagram omits some ancillary flows (e.g. admin notification) for clarity. All scoring logic (Likert + forced-choice) is unified within score\_prism (highlighted in green), and sensitive data (profile) is only accessible via the guarded token or user-specific policy.

# Scoring Engine Design Spec

We propose a **unified PRISM Scoring Engine** module that encapsulates all logic for calculating a profile from a set of assessment responses. This engine will be implemented in TypeScript and used in the Edge Function environment. It will be deterministic (no randomness affecting results) and versioned. The engine function will be pure (given input data, produce output data without side-effects or external calls, except optional calibration DB lookup that we handle separately) so it can be tested and reused easily. Below we detail the design:

## 1. **Inputs and Outputs**

**Engine Input (ProfileInput):** - sessionId (UUID) – mainly for logging or lookup if needed (e.g., for calibration logging), not strictly needed for computation but useful for context. - responses: Array of objects representing the user's answers. Each object includes: - question\_id (number or string), - answer\_value (as originally stored, could be string like "A" or "5" or a number; the engine will parse it), - Optionally, scale\_type or similar (e.g. "LIKERT\_5", "FORCED\_CHOICE\_4"). If not provided in input, the engine can look it up from scoringKey by question\_id. - scoringKey: A map/dictionary of question\_id -> scoring metadata. For each question: - scale\_type: e.g. "Likert5", "Likert7", "Forced\_Choice\_4", etc. - tag: e.g. "Ti\_S", "Ne\_D", "Core", "N", etc. (identifies what the question measures), - reverse\_scored (boolean), - pair\_group (string or null, identifier for consistency pair), - social\_desirability (boolean flag if it’s an SD question), - fc\_map (for forced-choice questions: a mapping from option letter to outcome, e.g. {"A":"Se", "B":"Fi", ...} or {"A":"Core","B":"Hidden",...}). - Any other meta needed (e.g. if scale has a non-1-to-5 range or non-integer mapping, though in our case Likert variants can be normalized). - config: An object of configuration constants, likely from scoring\_config. This includes: - results\_version (string, e.g. "v1.2.1"), - dim\_thresholds (object with thresholds for dimension count classification; e.g. one=2.1, two=3.0, three=3.8 means if average dimension answer ≥3.8 count as dimension, etc. – but since we use integer counts, threshold likely 3 meaning count of >=3 valued answers), - neuro\_norms (mean, sd for neuroticism; default {mean:3, sd:1}), - overlay\_neuro\_cut (e.g. 0.5), - overlay\_state\_weights ({stress:0.35, time:0.25, sleep:-0.20, focus:-0.20} default), - softmax\_temp (softmax temperature, default 1.0), - conf\_raw\_params ({a:0.25, b:0.35, c:0.20} default), - fit\_band\_thresholds (e.g. { high\_fit:75, moderate\_fit:55, high\_gap:6, moderate\_gap:3 } to categorize fit\_band "High/Mod/Low"), - fc\_expected\_min (integer, e.g. 24, minimum FC questions expected for completeness), - Possibly attention\_qids (list of question\_ids that are attention checks, if needed for additional handling). - These can be fetched from DB or use defaults if not present. - fcFunctionScores (optional): If present, a record of function -> score (0–100) from the fc\_scores table for this session. If provided, the engine will use these instead of calculating from responses. - partial (boolean): Whether to allow partial scoring (for mid-assessment preview). If true, engine will not finalize profile if data is insufficient. - fc\_expected (number): The expected number of FC questions (from config, e.g. 24). Used to evaluate completeness percentage.

**Engine Output (ProfileResult):** A structured result object representing the profile, including: - type\_code (string, e.g. "LII"), - base\_func and creative\_func (string, e.g. "Ti", "Ne"), - top\_types (array of top 3 type codes, e.g. ["LII","IEI","ILI"]), - type\_scores (object mapping each type\_code to { fit\_raw: number, fit\_abs: number, share\_pct: number } or similar. Alternatively, since we primarily need top3 for output, we might include only top3\_fits as an array of objects with code, fit, share[[49]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1051-L1059). But including all can help with deeper analysis or "Why Not" sections in UI. We see they stored type\_scores map in profile[[243]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1082-L1088)). - fit\_abs (number) – the calibrated fit score of the top type (or we may derive it from type\_scores of top\_types[0]), - score\_fit\_raw and score\_fit\_calibrated (numbers) – we will include for the top type as per profile table[[41]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1039-L1047), - fit\_band (string "High"/"Moderate"/"Low"), - top\_gap (number, e.g. 7.5 difference), - close\_call (boolean), - strengths (object mapping each function "Ti"/... to its strength score, e.g. 4.2), - dimensions (object mapping each function to its dimensional count (0–? how many answers at or above threshold), or possibly a categorical 1D/2D/3D indicator but likely count since they computed coherent functions by >=3 count[[172]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L851-L859)), - blocks (object containing likert and fc raw block sums if needed, but more importantly blocks\_norm: e.g. { Core: 35.0, Critic: 20.0, Hidden: 25.0, Instinct: 20.0 } representing blended percentages[[243]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1082-L1088). They stored blocks\_norm and also the separate blocks with likert and fc contributions[[243]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1082-L1088). We'll output both or at least the blended), - dims\_highlights (object with coherent and unique arrays listing which functions are ≥3D in the top type’s Base/Creative vs others[[243]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1082-L1088)). This is used for highlighting those in UI. - neuro\_mean (number, the average N response), - neuro\_z (number, z-score of neuro\_mean), - overlay\_neuro (string "+","–","0"), - overlay\_state (string "+","–","0"), - state\_index (number, the weighted state index), - overlay (string, the final overlay for profile, likely same as overlay\_neuro for now), - trait\_scores (perhaps an object with any other trait scores, e.g. N: 3.2 mean; they put N under trait\_scores[[175]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1072-L1080)), - validity\_status ("pass","warning","fail"), - validity (object with details: attention failures count, inconsistency index, sd\_index, duplicates count, etc. matching what's stored[[51]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L702-L710)), - confidence\_level (string "High"/"Moderate"/"Low" – the qualitative confidence after applying validity gating), - confidence\_raw (number, raw confidence probability 0–1 or we can store 0–100%), - confidence\_calibrated (number 0–1 or 0–100, calibrated confidence probability), - confidence\_band (perhaps the calibration method or band string, but we already have confidence\_level which is essentially similar. In code they logged confBand which might be method name or fallback indicator[[244]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L130-L138). We might output confidence\_method: "isotonic" or "platt\_fallback" if needed for debugging), - fc\_answered\_ct (number of FC blocks answered)[[46]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1046-L1054), - fc\_coverage\_bucket (string "complete"/"incomplete"/"partial" – can be derived: if answered < expected, maybe "incomplete" with thresholds, e.g. <50% answers = none/low, etc. They had logic: if fcCompletionRate < 1, mark incomplete and possibly abort scoring for partial sessions[[169]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L396-L404). They also have a view and trigger computing a bucket stored in profiles, which we can replicate or let trigger fill. We'll ensure fc\_answered\_ct is set; the trigger will update fc\_count and fc\_coverage\_bucket in the profile record[[226]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql#L56-L64)). - version ("v1.2.1"), - Timestamps are not generated by engine (the finalizeAssessment or score\_prism wrapper will add created\_at, etc.).

This output aligns with the profiles table structure[[245]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1030-L1038)[[39]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1050-L1058), so we can directly upsert it. The front-end expects at least type\_code, confidence, overlay, top\_types, top\_3\_fits, strengths, dimensions, blocks\_norm, etc. All these are included as above.

If partial input is true and data is insufficient, the engine could output a minimal profile or a status indicating partial. In practice, we plan to handle partial sessions by returning a special response from the function (not inserting a profile). We can implement the engine to either: - Return something like { partial\_session: true, completion\_rate: 0.xx } instead of a normal profile (and set a status). - Or throw an error/flag for the wrapper to catch and format a "partial\_insufficient" response as currently done[[170]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L400-L408). We will maintain similar behavior: if <50% FC answers and partial allowed, skip scoring and indicate need more data[[170]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L400-L408). The finalizeAssessment (which normally wouldn’t call partial scoring except perhaps for an admin preview) can pass partial=false. So primarily used if we ever implement a "preview profile" mid-assessment feature.

## 2. **Algorithmic Steps**

**Step 1: Data Preparation & Validation:** - Ensure the responses list is not empty and a sessionId is provided (for logging). If sessionId or required data is missing, throw an error. - If partial mode and responses count or coverage is too low (we can compute coverage as answered questions / total questions from scoringKey, or specifically forced-choice coverage if that is critical), we may return a partial status early. Specifically, check forced-choice completeness: - Compute fcCompletionRate = fc\_answered\_count / fc\_expected. If partial==true and fcCompletionRate < 0.5, then output a special partial result (or instruct the wrapper to handle it)[[170]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L400-L408). - We might also consider not scoring if very few Likert answered, but typically partial preview might be allowed as soon as some data is present. The current logic focused on FC because if too few FC, the dimensional overlay might be unreliable. - Deduplicate responses by question\_id: if multiple responses for same question, keep the latest (based on a timestamp if provided in responses or their order). In our case, finalizeAssessment passes the final responses array maybe not including old ones, but if needed we handle it as Map of question\_id to last response.

**Step 2: Iterate through responses to aggregate data:** Initialize accumulators:

const strengthsList: Record<Func, number[]> = {}; // lists of strength values per function  
const dimsList: Record<Func, number[]> = {}; // lists of dimension values per function  
const blockLikertSum: Record<string, number> = { Core:0, Critic:0, Hidden:0, Instinct:0 };  
const fcFuncCount: Record<Func, number> = {};  
const blockFCCount: Record<string, number> = { Core:0, Critic:0, Hidden:0, Instinct:0 };  
let neuroVals: number[] = [];  
let sdSum = 0, sdCount = 0;  
const pairs: Record<string, number[]> = {}; // pair\_group -> list of values  
let attentionFails = 0;

Loop over each response: - Lookup its rec = scoringKey[question\_id]. If not found, skip (or throw error if that's unexpected). - Determine scale = rec.scale\_type. - Parse answer\_value: - If the answer is already numeric (typeof number), use it if within expected range. - If string: - If it’s a Likert text like "Strongly agree", map via a dictionary to number (1–5)[[180]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L94-L98). - If it’s a number in string form, parse int. - If it's a forced-choice letter/number: - For FC: standardize to letter A-E. E.g. "1" -> "A", "2"->"B" etc.[[181]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L375-L383). - Save the letter in a variable choice. - If parsing fails (value not recognized), skip or continue to next (with a warning log). - If scale starts with "FORCED\_CHOICE": - If fcFunctionScores is provided (the precomputed aggregate), we will ultimately bypass granular counting. But we still increment fcAnsweredCount. - If not using precomputed: - Use choice determined above. Retrieve map = rec.fc\_map (already validated to an object). - If map[choice] exists: - If value m is one of "Core","Critic","Hidden","Instinct": blockFCCount[m]++. - Else if m is a function code ("Ti","Ne",...): fcFuncCount[m]++. - If no map or no entry for choice, increment a miss counter (and log warning)[[81]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L383-L390). - Also increment fcAnsweredCount. - Continue to next response (do not process as Likert further). - Else (Likert or other scalar question): - Normalize to 1–5 scale: - If rec.reverse\_scored is true, convert raw value to its "reverse" on the native scale: e.g. for 5-point Likert, rev = 6 - raw. If scale is 7-point, then perhaps rev = (scale\_max+1) - raw. - Then convert to 1–5 common scale if needed: e.g. if scale\_type was Likert7, we can map 1->1, 7->5, or formula linearly. However, a simpler approach is to treat the reversed or normal raw as already on a 1–5 scale if ranges differ slightly. (We have threshold calibrations that might handle differences, but since scoring\_key likely has mapping logic, we assume values are effectively 1–5 or can be linearly scaled). - Use a helper toCommon5(value, scale) if provided in code to handle specific named scales. - Let v5 be the resulting number 1–5 (could be a float if we did linear scaling, but typically Likert yields integers). - Based on rec.tag: - If tag is like "Fi\_S", then push v5 into strengthsList["Fi"][[83]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L365-L373). - If tag is like "Fi\_D", push v5 into dimsList["Fi"][[83]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L365-L373). (We might only consider D answers with high values as dimension count, but it's fine to store all and threshold later. Alternatively, we could immediately do if v5 >= 4: dimsCount[Fi]++. But storing allows computing average if needed. Simpler: increment count if ≥ some threshold. Code likely counted dimension items differently, but given how dimensions output is used, it might simply be count of answers ≥ certain value or count of D answers over threshold). - If tag is "Core"/"Critic"/"Hidden"/"Instinct": add v5 to blockLikertSum[tag][[246]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L367-L373). - If tag is "N" or "N\_R": push v5 to neuroVals[[84]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L363-L369). - If rec.social\_desirability true: add to sdSum and increment sdCount for SD index calculation[[86]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L369-L372). - If rec.pair\_group is not null: push v5 to pairs[pair\_group] (initialize array if needed)[[86]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L369-L372). - (Attention-check questions are likely marked via a tag or special flag; possibly pair\_group or meta. The code handles them after by attentionQids config: it iterates those question IDs and checks if user answered as expected[[232]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L420-L424). We can do similarly: if attentionFails was not directly incremented, after loop do:

if (config.attention\_qids) {  
 for (let qid of config.attention\_qids) {  
 const resp = responses.find(r => r.question\_id == qid);  
 if (resp) {  
 const val = getV5(qid) // the helper in code that fetches normalized value from answers list[232].  
 // If val indicates a wrong answer (perhaps rec.meta or tag can tell what correct is; but likely easier: if answer\_value exactly equals question\_text (like "Select 'Often'" expected but user didn't), they'd mark fail.  
 // Possibly in meta or separate table they have the correct answer for attention. If not easily available, could just count as fail if answer was <=3 for an attention question expecting a high or vice versa).  
 // For simplicity, assume attention checks are special questions where the correct answer is extreme (e.g. "I am randomly answering this question" should be "Strongly Disagree" = 1; if user put >1, count fail).  
 // If we had meta indicating correct option, we'd use that. If not, define logic as above.  
 }  
 }  
}

and compute attentionFails from that. In code they seem to do something like:

for each qid in attentionQids:  
 v = getV5(qid);  
 if (v is not extreme expected) failCt++ else passCt++;  
if failCt >= 2 => warning, if failCt >=1 => moderate conf.

We'll align with thresholds already given). - End loop.

Now we have: - strengthsList per function, - dimsList per function, - blockLikertSum, - fcFuncCount, blockFCCount, and fcAnsweredCount, - neuroVals, - sdSum, sdCount, - pairs group values, - attentionFails count.

**Step 3: Derive aggregate metrics:** - Compute strengths (average of each strengthsList[func] if any, else 0 or neutral 3). Round or keep one decimal as needed (they sometimes stored raw to one decimal). - Compute dimensions (for each func): - Possibly count of items in dimsList[func] ≥ dim\_thresholds.one (2.1 in config means count anything above ~2.1? That threshold config is odd; maybe they intended if average dimension >= 2.1 means at least 1D). - The code's logic for dimBand suggests that they ultimately care whether a function is >=3 (since they say if any ≥3 then 3-4D). So likely they consider a function "fully dimensional" if user gave at least 3 strong responses for it (since 3–4D means two functions are 3D for base/creative). Actually, likely: - If a function has >=3 answers that are high (4 or 5), call it "3D" (since each dimension question presumably corresponds to one dimension of the function, like past/present/future or personal/global aspects). - If >=2 high answers => "2D", etc. - So we can define: for each function, sort dimsList values, count how many >=4 (or maybe >= dim\_thresholds.three (3.8)? But 3.8 on 1–5 suggests count values >=4). - So dimensions[func] = count of dimsList[func] values >=4. That count (0 to maybe 4) represents dimensional depth. - Compute normalized block percentages: - bLikertSum = sum(Core+Critic+Hidden+Instinct from blockLikertSum) (some Likert questions might not cover all blocks equally). - blocks\_norm\_likert: for each block, if bLikertSum >0, percent = (blockLikertSum[block] / bLikertSum)*100 (round one decimal). - Similarly, bFCSum = sum(blockFCCount) and blocks\_norm\_fc from blockFCCount. - blocks\_norm\_blend: weighted average of the two sets: weight wLik = bLikertSum/(bLikertSum+bFCSum) (or if no FC, wLik=1)*[*[174]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L750-L758)*. Then for each block: blend = wLik\*blocks\_norm\_likert[block] + (1-wLik)\*blocks\_norm\_fc[block] (and round). - Forced-choice support: - If fcFunctionScores was provided, skip granular and directly create fcSupport from that: e.g. for each func, fcSupport[f] = score/100 (score is 0–100 from fc\_scores)*[*[182]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L317-L325)*. Possibly ensure no 0 (they did Math.max(1, ...) to avoid zeros)*[*[182]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L317-L325) *when converting to counts for legacy. But for unified output, we can keep 0 values; or replicate adding a minimal count to avoid division by zero later. - If fcFunctionScores not provided (legacy path): - Compute total FC count = sum of fcFuncCount values or if that is all zeros (meaning mapping was to blocks only?), then derive from block count. Actually, in code, they did js const fcTotalCount = Object.values(fcFuncCount).reduce((a,b)=>a+b,0) || 1; for each func: fcSupport[func] = (fcFuncCount[func]||0) / fcTotalCount;*[*[173]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L2-L10)*. - We'll do the same to get a 0–1 support fraction for each function. - fc\_answered\_ct we have (the number of FC questions answered, to output). - Validity metrics: - attentionFails we already counted (either in loop or via post-check). - Compute inconsistency: for each entry in pairs: - If two values in the pair, maybe take absolute difference. The code stored an inconsIdx maybe as average of absolute differences or something, judging by threshold ~1.5 being warning. Possibly:*

let inconsVals = [];  
for each pair in pairs:  
 if (pair.length >= 2) {  
 inconsVals.push(Math.abs(pair[0] - pair[1]));   
 }  
const inconsIdx = inconsVals.length ? inconsVals.reduce((a,b)=>a+b)/inconsVals.length : 0;

So average difference on 1–5 scale. If user answered similarly, difference ~0–1; if oppositely, difference up to 4. Then threshold 2.0 means on average answers differ by 2 or more (quite inconsistent). - We'll adopt that: inconsistency = average difference between paired questions. - Compute sd\_index: probably the standard deviation of all Likert answers (especially if many answers are extremes or all mid). - Possibly they compute population standard deviation of all answered Likert values \* 10? Because threshold 5.2 is used which is high if it were just SD (~1.0 typical). Perhaps they multiply by something. - Another guess: maybe they compute the number of values in the middle (2-4) vs extremes (1,5). Or maybe the inverse of variance (like low variance -> high sd\_index). - Given the threshold of 4.9 and 5.2 roughly correspond to if user put mostly the same answer or random: Actually, if user alternated 1 and 5, SD is ~2, maybe they scale that to ~5. Possibly they do sd\_index = sample\_std\_dev \* 10 (so if stdev=0.5, index=5, if stdev=0.52, index=5.2, etc.). That would make threshold 5.2 ~ stdev 0.52 which is moderate. But if user varied a lot, stdev ~1.4 (half range), index=14 which is >>5.2, but their gating stops at fail if ≥5.2. Hmm, maybe instead sd\_index is something like how far from mid each answer is averaged, scaled differently. Alternatively, maybe it's the sum of absolute deviations or something. - However, since we don't have exact formula from code, we can try to mimic: The code changed fail threshold from 4.9 to 5.2 (loosened)[[197]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L720). That suggests maybe originally threshold was 4.9 and they realized false fails so raised to 5.2. If index was out of 10 (like 10 means completely random), then 5.2 means moderate variation. - For implementation: we can do:

const values = responses.filter(r => isLikertType(r)).map(v5 of each);  
const mean = values.reduce((a,b)=>a+b,0)/values.length;  
const variance = values.reduce((sum,v)=>sum + (v-mean)\*\*2, 0) / values.length;  
const sd = Math.sqrt(variance);  
const sd\_index = sd \* 10;

This yields ~0 to ~14 (if answers evenly 1 and 5, sd ~2, index ~20 actually; but if user picks mostly 3 (all same), sd=0, index=0; user uses full range evenly, sd ~1.4, index ~14). So fail if sd\_index ≥5.2 means sd ≥0.52. That’s a low threshold, meaning if stdev is above 0.52 (which is not high), mark fail. That seems too strict; likely not correct interpretation. Perhaps sd\_index is something else. - Another approach: maybe they count how many questions were answered in extreme ends vs moderate: Possibly if user answers all questions the same or randomly, they'd fail differently. - Given uncertainty, but wanting to preserve behavior: They loosened fail from 4.9 to 5.2 (implying fewer fail). Let's stick with the formula above (sd10). That may overshoot fail triggers though. But we have gating: - If user answers variety (sd ~1), sd\_index=10, triggers fail by far. - If user answers almost all 3 with little variance (sd ~0.3), index=3, pass. Actually consistent users would have low sd\_index, inconsistent (random or extreme) have high index, which triggers fail as expected. So maybe it's intended to catch random behavior (lots of extremes yields high SD). So perhaps correct: if user uses the scale a lot (big variance), then fail. It's somewhat counterintuitive (shouldn't using full range be fine if appropriate? Perhaps they expect certain pattern?). But given these thresholds, we'll implement as such to match. - Now compile validity\_status: - Initialize as "pass". - If inconsistency >= 2.0 OR sd\_index >= 5.2 => validity\_status = "fail", set a flag. - Else if inconsistency >= 1.5 OR sd\_index >= 4.9 OR attentionFails >= 2 => validity\_status = "warning". - (We may also include if attentionFails == 1 along with inconsistency >=1.0 or sd\_index >=4.3, but code uses those to reduce confidence without raising warning. Actually code does: - fail conditions (strict), - warning conditions (somewhat strict including any two attention fails), - then else-if one attention fail or incons>=1.0 or sd>=4.3 => keep status pass but we will set confidence moderate later). - We'll replicate: Start status "pass".

let confidenceLevel = "High";  
if (inconsistency >= 2.0 || sd\_index >= 5.2) {  
 validity\_status = "fail";  
 confidenceLevel = "Low";  
} else if (inconsistency >= 1.5 || sd\_index >= 4.9 || attentionFails >= 2) {  
 validity\_status = "warning";  
 confidenceLevel = (attentionFails === 1 ? "Moderate" : "Low");  
 // Actually, if fails>=2 we already warning, and they'd set Low if fails==2 else Low if >2. If exactly 1 fail, we haven't gone into warning block (since >=2 needed), so 1 fail is handled later.  
} else if (inconsistency >= 1.0 || sd\_index >= 4.3 || attentionFails === 1) {  
 // These are conditions for moderate confidence while still status pass  
 confidenceLevel = "Moderate";  
}

This matches the code logic for confidence text[[89]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L724)[[90]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L726-L730). - So output validity\_status and initial confidence\_level from this gating. (We will later refine numeric confidence but this sets text baseline.) - At this point, we have: - strengths, dimensions, blocks\_norm\_blend, etc. - Preliminary validity\_status and confidence\_level (which corresponds to validity-based High/Mod/Low). - We also carry attentionFails, inconsistency, sd\_index for output in validity.

**Step 4: Type Distance & Fit Calculation:** - Construct protoTargets for each type: - We need TYPE\_PROTOTYPES data (16 types x 8 functions mapping to blocks). We loaded from DB or fallback in score\_prism code[[75]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L158-L167)[[76]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L178-L186). We can pass that into engine via config or as a global within engine module. (Probably easier: fetch in function and pass as parameter). - Compute BLOCK\_WEIGHTS constants (as in code: base 1.0, creative 0.7, etc.[[247]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L39-L48)). - For each type in TYPE\_PROTOTYPES: - For each function, assign target = 1 + 4 \* ((BLOCK\_WEIGHTS[block] - MIN\_W) / (MAX\_W - MIN\_W))[[248]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L771-L778). Here MIN\_W=0.10, MAX\_W=1.0 (for base vs vulnerable)[[249]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L40-L48). So base (1.0) yields 5, creative (0.7) yields maybe ~?? let's check formula: They do t = 1 + 4 \* ((w - MIN\_W)/(MAX\_W - MIN\_W))[[248]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L771-L778). If w=1.0, MIN\_W=0.1, MAX\_W=1.0: (1.0-0.1)/(0.9) = 1, so 1+4*1=5. If w=0.7, (0.7-0.1)/0.9 = 0.666..., 1+4*0.666= 1+2.667=3.667. Vulnerable 0.10 => (0.1-0.1)/0.9=0, target=1. So indeed base=5, creative≈3.67, vulnerable=1, etc. - Save protoTargets[type][func] = that value. - Compute raw distance and match: - For each type: - Calculate sumSq = Σ\_{func}(strengths[func] - protoTargets[type][func])^2. - dist = sqrt(sumSq). - match = Math.max(0, 1 - dist / MAX\_DIST) where MAX\_DIST = sqrt(8 \* (5-1)^2) = sqrt(8*16) = sqrt(128) ≈11.314*[*[188]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L764-L773)[*[250]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L784-L788)*. - rawScore = match \* 6.5 (range ~0 to 6.5)*[*[98]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L784-L792)*. - Map rawScore to percentage: fit\_raw\_pct = (rawScore / 6.5) \* 100 and round to one decimal*[*[99]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L794-L803)*. - Store fitRaw[type] = fit\_raw\_pct. - Cohort calibration: - If our input or config includes cohort statistics (mean & sd of fitRaw over recent profiles), use them. If not provided, we might query inside the engine using the DB – but we'd prefer to avoid inside engine for purity. Instead, as in current score\_prism, we can do this outside: - But to keep determinism, and since cohort changes over time, using dynamic cohort in scoring means results can shift. However, they've done that purposely (score\_fit\_calibrated is meant to be relative to recent test-takers). - We will replicate: query the profiles table for score\_fit\_raw values in last 90 days (do this in score\_prism function wrapper, not inside engine)*[*[189]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L807-L815)*. Then pass cohortMean and cohortSd into engine config. - If cohortMean and cohortSd present and count >=50: - For each type, fitAbs[type] = clamp(50 + 15 \* (fitRaw[type] - cohortMean) / cohortSd, 20, 85) and round to one decimal*[*[101]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L826-L835)[*[102]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L830-L839)*. - If not enough data (or cohortSd == 0): - For each type, fitAbs[type] = clamp(fitRaw[type] \* 0.65 + 20, 20, 85) and round*[*[103]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L836-L844)*. - (These values center the distribution around 50 with sd=15 akin to IQ or T-scores). - Compute probabilities: - temp = config.softmax\_temp || 1.0. - For each type: expScore[type] = Math.exp(rawScore[type] / temp). - Sum them: sumExp = Σ expScore. - For each type: sharePct[type] = (expScore[type] / sumExp) \* 100, round to one decimal*[*[105]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L845-L850)*. - Determine top types: - Get array of all 16 type codes. - Sort by: 1. fitAbs[b] - fitAbs[a] (descending fitAbs), 2. if equal, sharePct[b] - sharePct[a], 3. if equal, coherentCountFor(b) - coherentCountFor(a) where coherentCountFor(x) counts how many of type x's base & creative functions have dimensions[func] >=3 (3D)*[*[106]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L859-L867)*, 4. if equal, fcSupport\_sum(b) - fcSupport\_sum(a) where fcSupport\_sum(x) = (fcSupport[base\_func] + fcSupport[creative\_func]) (so if forced-choice reinforced that type's main functions)*[*[190]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L863-L870)*. - Take the first 3 of this sorted array as top3. - Compose top3Fits = top3.map(code => ({ code, fit: fitAbs[code], share: sharePct[code] })) for output*[*[251]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L920-L928)*. - type\_code = top3[0] (best type). - base\_func = TYPE\_MAP[type\_code].base, creative\_func = TYPE\_MAP[type\_code].creative (we have TYPE\_MAP similar to TYPE\_PROTOTYPES but just mapping type to its base & creative functions). - Compute top\_gap and close\_call: - topFit = fitAbs[top3[0]], secondFit = fitAbs[top3[1]] (if exists, else 0). - top\_gap = (topFit - secondFit) (we'll round to one decimal as they did)*[*[114]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L876-L880)*. - close\_call = top\_gap < 3 (boolean)*[*[252]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L877-L880)*. - Compute fit\_band: - Use thresholds from config (or default high\_fit=75, mod\_fit=55, high\_gap=6, mod\_gap=3 based on earlier reasoning): - If (topFit >= high\_fit && top\_gap >= high\_gap): fitBand = "High". - Else if ((topFit >= moderate\_fit && topFit < high\_fit) || (top\_gap >= moderate\_gap && top\_gap < high\_gap)): fitBand = "Moderate". - Else: fitBand = "Low". - (This matches code logic*[*[107]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L910-L918)*.) - dims\_highlights: - Determine main type’s base & creative functions (from TYPE\_MAP). - coherent = array of functions that are either base or creative of main type and have dimensions[func] >= 3 (i.e., 3D or 4D). - unique = array of functions that are NOT base/creative but still have dimensions[func] >= 3*[*[253]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L929-L937)*. - These highlight which functions the user has developed that align with core of their type vs uniquely high outside core. - Summarize trait overlays: - neuro\_mean = average of neuroVals (if none, can set to 0 or 0? Actually if none (no N questions), leave 0). - neuro\_z = (neuro\_mean - neuro\_norms.mean)/neuro\_norms.sd (one decimal or three decimal as code did Number(zN.toFixed(3))*[*[175]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1072-L1080)*). - overlay\_neuro = "+" if z >= cut, "–" if z <= -cut, else "0". - Compute state\_index similarly: - If we recorded any state question values (we can identify them via tags from scoringKey, maybe tag "stress", "sleep", etc. The config state\_qids likely provided question IDs for each state category*[*[254]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L258-L263)*. We can fetch each by find in responses or scoringKey). - For each (stress, time, sleep, focus): - get value v (we likely have those questions in responses, convert to common scale and then center: c = v - 3). - Weighted sum: state\_index = W.stress\*c\_stress + W.time\*c\_time + W.sleep\*c\_sleep + W.focus\*c\_focus*[*[111]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L680-L688)*. - overlay\_state = "+" if state\_index >= cut, "–" if <= -cut, else "0". - Decide final overlay: - For continuity, we choose overlay = overlay\_neuro. (We could also consider combining if needed: e.g. if overlay\_neuro and overlay\_state are both plus maybe output "++"? But the UI isn't built for that. They explicitly set overlay = overlay\_neuro for legacy reasons*[*[113]*](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L690-L698)*.) -* *Confidence Calculation:*\* - topGap (we have), confidenceMargin = top share - second share = p1 - p2 (we have p1, p2). - shareEntropy computed from sharePct distribution:

const shares = Object.values(sharePct).filter(x => x > 0);  
const shareEntropy = - shares.reduce((sum, p) => {  
 const frac = p/100;  
 return sum + frac \* Math.log2(frac);  
}, 0);

(If all p equal, entropy high; if one dominates, entropy low.) - dimBand we have (depending on max dimensions >=3). - overlayStr = overlay\_neuro === '+' ? 'plus' : overlay\_neuro === '–' ? 'minus' : '0' (the calibration\_model uses 'plus'/'minus' strings). - Compute raw confidence:

const {a,b,c} = config.conf\_raw\_params;  
const rawConf = 1 / (1 + Math.exp(-(a\*topGap + b\*(confidenceMargin/100) - c\*shareEntropy)));  
rawConf = Math.max(0, Math.min(1, rawConf));

(This matches PrismCalibration.calculateRawConfidence code[[119]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L64-L72).) - Calibrated confidence: - Look up calibration model for version = results\_version, where stratum->>dim\_band = dimBand and stratum->>overlay = overlayStr. \* If found (knots array): - Interpolate rawConf in the knots (which are sorted by x): - If rawConf <= first knot.x, calibrated = first knot.y. - If >= last knot.x, calibrated = last knot.y. - Else find interval around rawConf and linear interpolate y. - (This is what interpolateCalibration does[[192]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L176-L184).) - Mark method = (calibrationData.method, e.g. "isotonic") and fallback=false. \* If not found or error: - Use fallback:

const a\_platt = -0.5, b\_platt = 1.2;  
const sigmoid = x => 1/(1+Math.exp(-(a\_platt + b\_platt\*x)));  
let calibrated = sigmoid(rawConf);  
calibrated = Math.max(0, Math.min(1, calibrated));  
method = 'platt\_fallback';  
fallback = true;

(Matches code[[194]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L150-L159).) - The engine might not itself query the DB for knots. We can have the score\_prism function do that lookup via PrismCalibration instance and pass a calibration function or model to engine. But given integration, we might just call PrismCalibration within score\_prism after engine returns rawConf. - For design completeness: The engine can provide raw\_confidence and maybe let the wrapper handle calibration. This keeps engine DB-agnostic. That's likely approach: engine outputs raw\_conf and maybe its internal band (close call etc.), and the Edge Function applies calibration to get final confidence\_calibrated. - Determine confidence\_band or confidence\_level: - We already have confidence\_level from validity gating (High/Mod/Low initial). - Now incorporate calibrated numeric: - If validity\_status == "fail": override confidence\_level = "Low" (already done). - Else if validity\_status == "warning": - If attentionFails==1 (the only case that yields warning? Actually warning also triggers for incons>=1.5 or sd>=4.9, but anyway in warning we usually set Low if multiple fails, or if exactly one fail we set Moderate in code[[88]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L721-L729)). - Code sets confidence = "Low" for warning unless exactly one attention fail then "Moderate". We handled that above. - Else (pass): - If confidence\_level was downgraded to Moderate due to mild issues, keep that. - If still High, we can consider calibrated probability: - For instance, if calibrated\_confidence < 0.6 (60%), perhaps we downgrade to Moderate. If < 0.4, maybe Low. - The code does not explicitly do this because they rely mostly on validity for text confidence and present numeric separately. But to make text confidence more reflective: - We could implement: If validity is pass (so currently set High) but calibrated probability is low, maybe reduce confidence. - However, to avoid confusion and maintain continuity with old system, we might leave the text confidence purely based on validity and allow the numeric to speak for itself. - So likely: if pass and calibratedConf < 0.5 we might consider moderate, but code didn't do that. - We will mirror code: they did not alter text confidence based on numeric, only on validity conditions. So confidence\_level remains as set by validity rules. - So output: - confidence\_level (High/Mod/Low string as final), - confidence\_raw (rawConf \* 100 maybe), - confidence\_calibrated (calibratedConf \* 100). - conf\_band we can set equal to confidence\_level or possibly something like "High" if calibrated >0.8 etc. But since we have confidence\_level, maybe conf\_band can be reserved for an indicator of calibration model used (not needed for user though). - We'll likely not use conf\_band field except maybe storing method for internal use (or just skip).

**Step 5: Assemble Output:** - Populate a ProfileResult object with fields as described in Outputs:

const profile: ProfileResult = {  
 session\_id,  
 type\_code,  
 base\_func,  
 creative\_func,  
 results\_version: config.results\_version || "v1.2.1",  
 fit\_band,  
 top\_gap: parseFloat(top\_gap.toFixed(1)),  
 close\_call,  
 top\_types: top3,  
 top\_3\_fits: top3Fits,  
 strengths,  
 dimensions,  
 blocks\_norm: blocks\_norm\_blend, // (with separate blocks if needed)  
 blocks: { likert: blocks\_norm\_likert, fc: blocks\_norm\_fc },  
 dims\_highlights: { coherent, unique },  
 neuro\_mean: parseFloat(neuro\_mean.toFixed(3)),  
 neuro\_z: parseFloat(neuro\_z.toFixed(3)),  
 overlay\_neuro,  
 overlay\_state,  
 state\_index: parseFloat(state\_index.toFixed(3)),  
 overlay,  
 trait\_scores: { N: parseFloat(neuro\_mean.toFixed(3)) },  
 validity\_status,  
 validity: {  
 attention: attentionFails,  
 inconsistency: parseFloat(inconsistency.toFixed(3)),  
 sd\_index: parseFloat(sd\_index.toFixed(3)),  
 duplicates: 0, // (we didn't explicitly count duplicate answers beyond pairs? They have `duplicateCount` in code which they set to rawRows.length - answers.length after dedup[78]. We can output that if we tracked initial count vs deduped count.)  
 state\_modifiers: { stress: cStress, time: cTime, sleep: cSleep, focus: cFocus } || {}, // optionally include centered state values.  
 required\_tag\_gaps: [] // (they had logic for requiredTags missing, likely not needed in our current context).  
 },  
 confidence: confidence\_level, // High/Moderate/Low textual  
 conf\_raw: parseFloat(rawConf.toFixed(4)),  
 conf\_calibrated: parseFloat(calibratedConf.toFixed(4)),  
 conf\_band: confBandString || (calibrationFallback ? "fallback" : "model"), // optional  
 fc\_answered\_ct: fcAnsweredCount,  
 fc\_coverage\_bucket: fcAnsweredCount == 0 ? "None" : (fcCompletionRate < 1.0 ? "Partial" : "Full"),  
 // Other fields: score\_fit\_raw, score\_fit\_calibrated for top type  
 score\_fit\_raw: fitRaw[type\_code] || topFitRaw,  
 score\_fit\_calibrated: fitAbs[type\_code] || topFit  
};

- Return this ProfileResult.

Throughout, ensure to handle edge cases (like division by zero when no responses for a category, etc.) gracefully (the code often uses default or conditional to avoid NaN).

**Determinism and Precision:** The calculations are deterministic given fixed inputs. We will round at appropriate points (like one decimal for percentages as in output, three decimals for indices if needed, etc.) to match the legacy output formatting. The engine does not incorporate any random values except the minor random microsecond offset added to submitted\_at timestamp which we handle outside engine.

**Comparison to Legacy:** This design covers all aspects the current system calculated, so outputs will include everything previously shown to users (no loss of information). The numeric values will match or be very close to the legacy outputs: - Type selection logic and fit scores remain the same formula, so type\_code should not change for a given set of answers. - Confidence numeric is now fully implemented (previously, confidence\_numeric was just margin; now we will output calibrated probability). This is an improvement but doesn't break UI since UI didn't explicitly interpret the old numeric. - The "confidence" text remains based on validity, as before. - The overlay logic remains the same for neuro (state overlay was partially implemented but not surfaced, we maintain it for completeness). - By using the same thresholds and weight constants as v1.2.0, we ensure consistency. For example, type distance weighting (block weights) are identical[[247]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L39-L48), fit calibration and clamping identical[[102]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L830-L839)[[103]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L836-L844), etc.

**Extensibility:** This engine design can accommodate future changes: - If we adjust questions or scoring weights, we update the scoringKey or config, and the engine automatically uses them. - New question categories (e.g. a new trait) could be integrated by adding logic to accumulate and output that trait. - The engine is decoupled from Supabase context except for known inputs; we can run it with test data easily, and also potentially in front-end if needed (though we'll keep official scoring on server). - Versioning the output ensures that if we overhaul logic in future (v1.3), we can differentiate profiles by version.

## 3. **Module Implementation and Use**

We will implement the above in a module (e.g. scoreEngine.ts) with functions for each part: - parseAnswer(raw, scaleType) – returns numeric value or letter normalized. - accumulateResponse(response, accumulators) – updates all the accumulators given one response and its scoring key entry. - finalizeAggregates(accumulators, config) – computes strengths, dimensions, blocks, etc. - calculateTypeScores(strengths, prototypes) – returns fitRaw, fitAbs, sharePct for all types. - rankTypes(fitAbs, sharePct, dimensions, fcSupport) – returns sorted top3 and related info. - calculateConfidenceMetrics(...) – returns rawConf and uses calibrator or provided model to get calibratedConf. - determineValidity(pairs, sdValues, attentionFails) – returns validity\_status and confidence\_level initial. - Then scoreAssessment(input: ProfileInput): ProfileResult orchestrates these, calling sub-functions in order.

We will not duplicate logic in both front and back: this module will be imported by score\_prism Edge Function. On the front-end, we may not use it (since we won't do client-side scoring in this project phase), but theoretically it could be used for a "live preview" feature if needed, as it has no external dependencies except config.

One integration detail: **Calibration** – The only place we need to query DB during scoring is to get calibration model knots (and possibly cohort profile stats). Instead of making the engine call DB, we will do these in the score\_prism function: - Fetch calibration model entry (or entries) once from DB for the relevant stratum combination (or fetch all for this version and put in a map, then engine can just use the needed one). But since calibration model is small, we could fetch just needed or all 2\*3=6 combos (plus maybe 1D plus/minus). - We can pass the relevant knots array into the engine or simply perform the calibration adjustment outside engine after getting raw\_conf from engine. - For simplicity, we might do: engine returns raw\_conf and some context (dimBand, overlay, maybe also closeCall flag), then score\_prism uses PrismCalibration.applyCalibratedConfidence(rawConf, dimBand, overlay) to get calibrated and method[[255]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L88-L97)[[256]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L122-L131). - Then score\_prism fills those into the profile result and finalizes.

So the design allows either approach without altering outputs.

In conclusion, this scoring engine design centralizes all computations. By implementing it, we eliminate the duplication (the forced-choice logic will be unified via either using fc\_scores or the same mapping code in one place, not scattered in front-end), ensure both front-end and back-end refer to the same logic (front-end now won't do any scoring itself, only display results, so consistency is inherently ensured by using the server result), and we maintain determinism and version tracking for future refinements.

# Unified Implementation Plan

Implementing the unified scoring system involves coordinated changes to backend functions, the front-end application, and the database. We outline a step-by-step plan:

**A. Refactor Backend Scoring into Shared Module:**

1. **Create Scoring Engine Module**: Add a new file supabase/functions/\_shared/scoreEngine.ts in the repo. This will export scoreAssessment(input: ProfileInput): ProfileResult and any helper types. Populate it with logic as per the design spec:
2. Include the calculations for strengths, dimensions, type matching, etc., using the code structure from v1.2.0 (ensuring constants match exactly). For instance, define BLOCK\_WEIGHTS and reuse the TYPE\_PROTOTYPES structure (import or define from a constants file or fetch in score\_prism and pass to engine).
3. Example snippet:

* const MIN\_W = 0.10, MAX\_W = 1.00;  
  // build protoTargets  
  for (const type of Object.keys(TYPE\_PROTOTYPES)) {  
   protoTargets[type] = {};  
   const proto = TYPE\_PROTOTYPES[type];  
   for (const func of FUNCS) {  
   const block = proto[func];   
   const w = BLOCK\_WEIGHTS[block];  
   const t = 1 + 4 \* ((w - MIN\_W) / (MAX\_W - MIN\_W));  
   protoTargets[type][func] = t;  
   }  
  }
* [[95]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L768-L776)This mirrors the code inside score\_prism.

1. Implement the rest of computations, adding comments referencing the original code lines to show parity (for maintainers). Ensure rounding and thresholds replicate the original outputs (e.g. use toFixed(1) or similar where needed).
2. This module will have no external side effects (no DB calls). It can accept calibration info as part of config or we’ll handle calibration outside.
3. Write unit tests for this module (in a tests/scoreEngine.test.ts, for example) to verify a couple of scenarios (if possible, compare outputs against known results or at least consistency checks like sharePct sum ~100, etc.).[[201]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L950-L958)
4. **Integrate Engine in score\_prism Function**: Modify supabase/functions/score\_prism/index.ts:
5. Import the new engine:

* import { scoreAssessment, ProfileInput } from "../\_shared/scoreEngine.ts";
* and also import PrismCalibration if needed.

1. Remove or bypass the internal logic that is now handled by engine:
   * The code from loading responses to building the profile can be largely replaced. Specifically:
   * After deduplicating responses and building keyByQ (the scoring key map)[[78]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L215-L224)[[32]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L229-L238), and after fetching config values (thresholds, etc.)[[34]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L255-L263), instead of manually computing, do:
   * const input: ProfileInput = {  
      sessionId: session\_id,  
      responses: answers, // 'answers' array after dedup  
      scoringKey: keyByQ,  
      config: {  
      results\_version: resultsVersion,  
      dim\_thresholds: dimThresh,  
      neuro\_norms: neuroNorms,  
      overlay\_neuro\_cut: cut,  
      overlay\_state\_weights: W,  
      softmax\_temp: softmaxTemp,  
      conf\_raw\_params: { a:0.25, b:0.35, c:0.20 }, // could fetch from config if stored  
      fit\_band\_thresholds: { high\_fit:75, moderate\_fit:55, high\_gap:6, moderate\_gap:3 },  
      fc\_expected\_min: fcExpectedMin  
      },  
      fcFunctionScores: fcScores ? fcScores.scores\_json : null,  
      partial: partial\_session,  
      fc\_expected: typeof fcExpectedMin === 'number' ? fcExpectedMin : 24  
     };  
     const profileResult = scoreAssessment(input);
   * This calls the unified logic. We handle try/catch around it in case of any thrown error in engine.
   * Remove the legacy forced-choice loop: we've already computed fcScores from table if available and passed it; if not, the engine will internally handle mapping via fc\_map. The code block:
   * if (fcScores) {  
      // use realFCScores  
      ...  
     } else {  
      // fallback legacy mapping  
     }
   * can be removed, because our engine function will do similar internally with fcFunctionScores presence or not. Actually, since we pass fcFunctionScores (scores\_json) if available, the engine will prefer that; if it's null, engine will use fc\_map from key as needed. We should still log whether we used real FC scores or not. In our case:
     + If fcScores was found, we might set a flag usedRealFCScores = true and maybe log evt:fc\_scores\_loaded as in original[[257]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L310-L318).
     + Our engine could log internally (but logs in module won't show unless we pass a logger).
     + Simpler: do as original:
     + if (fcScores && fcScores.scores\_json) {  
        console.log(`evt:fc\_scores\_loaded,session\_id:${session\_id},blocks:${fcScores.blocks\_answered}`);  
       } else {  
        console.log(`evt:fc\_fallback\_legacy,session\_id:${session\_id}`);  
       }
     + [[58]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L299-L308)[[257]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L310-L318). This retains a trace of whether new or old FC path used.
   * Remove all the manual computation of strengths, type scores, etc. For example, lines like:
   * // Distance-based type matching...  
     for each type code: calculate proto targets, distance, rawScores...  
     // Fit calibration via cohort...  
     // Softmax for sharePct...  
     // Determine top3...  
     // etc.
   * are replaced by our engine’s output. We'll trust engine to produce the same values. We can, for safety, output some debug info if debugMode is on (the original code had a debug block returning raw data if debug flag in input)[[200]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L938-L948)[[201]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L950-L958). We can keep that:
     + If debugMode (the request had debug: true), instead of normal profile, return status:"debug", debug: debugPayload with all intermediate values from engine. We can populate debugPayload from profileResult (since it has most fields anyway). E.g.
     + if (debugMode) {  
        console.log(`evt:debug\_mode,session\_id:${session\_id}`);  
        return new Response(JSON.stringify({ status:"debug", debug: profileResult }), { ... });  
       }
     + Or refine debug object: They provided counts and weights etc. For brevity, we might just return profileResult (which includes a lot of detail now). It's fine as it's internal usage.
   * Keep session update logic: The code updates assessment\_sessions.status to 'completed' and sets completed\_at if null[[237]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L973-L982). But finalizeAssessment also does this. Double-check:
     + finalizeAssessment calls score\_prism then it itself updates session (ensuring share\_token etc.)[[152]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L175-L183). In our new flow, front-end will mostly call finalizeAssessment, so finalize will do it.
     + If someone directly calls score\_prism (maybe an admin or a legacy call path), we should still mark session completed here as a backup. It won't harm if finalize does again (finalize uses update ... eq id which will just overwrite status with same value).
     + So keep:
     + const updates: any = { status: 'completed' };  
       if (!currentSessionTiming?.completed\_at) updates.completed\_at = new Date().toISOString();  
       supabase.from('assessment\_sessions').update(updates).eq('id', session\_id);
     + (with error logging if needed)[[237]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L973-L982)[[125]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L979-L987).
   * Use profileResult to prepare the insert: We can map profileResult fields directly to DB insert object:
   * let profileData = { ...profileResult };  
     profileData.session\_id = session\_id;  
     profileData.user\_id = user\_id || currentSession?.user\_id || null;  
     profileData.results\_version = "v1.2.1";  
     // Add created\_at, updated\_at, submitted\_at as original did  
     const now = new Date();  
     if (existingProfile) {  
      profileData.submitted\_at = existingProfile.submitted\_at;  
      profileData.recomputed\_at = now.toISOString();  
     } else {  
      // unique timestamp as original  
      const uniqueTime = new Date();  
      uniqueTime.setMilliseconds(uniqueTime.getMilliseconds() + Math.random() \* 1000);  
      profileData.submitted\_at = uniqueTime.toISOString();  
      profileData.created\_at = uniqueTime.toISOString();  
     }  
     profileData.updated\_at = new Date().toISOString();
   * This follows what original did for timestamps[[127]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1109-L1118)[[128]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1114-L1122).
   * Then upsert to profiles as before[[52]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1125-L1133). Check for errors and respond with JSON.
   * Update log statements:
   * Continue logging key events:
     + evt:scoring\_start, session\_id, version (keep same)[[74]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L156-L164),
     + If prototypes loaded from DB vs fallback (the code did that in our removed block[[75]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L158-L167)[[76]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L178-L186), we can do it before calling engine: if we successfully fetched all type\_prototypes rows, log prototypes\_loaded\_db else fallback).
     + evt:fit\_calibrated\_z,... logged cohort stats[[104]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L832-L840)[[103]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L836-L844). We can replicate after calibration: e.g. if used cohort calibration, log mean, sd, n.
     + evt:scoring\_complete at end with session\_id, type, overlay, confidence, etc.[[258]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1141-L1145). We can fill those from profileResult: e.g. type:${profileResult.type\_code}${profileResult.overlay}, confidence:${profileResult.confidence}, validity:${profileResult.validity\_status}, top\_gap:${profileResult.top\_gap}.
   * Remove any console logs that are now redundant or overly verbose unless in debug mode. (The original had many logs for partial sessions, prototypes incomplete, etc., we keep those warnings).
2. **Calibration inside score\_prism**:
   * We need to handle PrismCalibration. The engine returns raw confidence and also an initial confidence\_level. We'll apply calibration:
   * const calibrator = new PrismCalibration(supabase);  
     const { calibrated, method, fallback } = await calibrator.applyCalibratedConfidence(profileResult.conf\_raw, profileResult.dimBand, profileResult.overlay === '+' ? 'plus' : profileResult.overlay === '–' ? 'minus' : '0');  
     profileResult.conf\_calibrated = parseFloat(calibrated.toFixed(4));  
     // We might also decide to adjust confidence\_level if needed, but we'll leave as is except maybe if fallback used (no, not needed).  
     profileResult.confidence\_method = method;
   * But note: our engine did not output dimBand explicitly in profileResult. We can derive it easily after computing dimensions for each function:
   * dimBand = (Math.max(...Object.values(dimensions)) >= 3 ? '3-4D' : Math.max(...)==2 ? '2D' : '1D') as in code[[117]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L892-L900).
   * We can compute that inside engine and include in result, or just re-compute here consistently. It's cheap, so we can recalc in finalizeConfidence step:
   * const maxDim = Math.max(...Object.values(profileResult.dimensions||{}), 0);  
     const dimBand = maxDim >= 3 ? '3-4D' : (maxDim === 2 ? '2D' : '1D');
   * (Better to use engine's computed one if available to avoid slight logic discrepancy, but it's straightforward enough and should match.)
   * Then do calibrator with that dimBand and overlay.
   * The PrismCalibration also had a calculateConfidence that did raw+apply in one, but since we already did raw, we just use applyCalibratedConfidence.
   * After this, profileResult.confidence\_calibrated is updated. We should keep profileResult.confidence (text) unchanged (or we could update it if we wanted to reflect numeric, but we'll preserve original approach).
   * If calibrator returns fallback method, maybe log a warning (the code logs evt:no\_calibration\_model... using fallback[[259]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L114-L122) or evt:calibration\_applied with method[[260]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L124-L132). The PrismCalibration class itself logs these events internally via supabase client queries logs. Actually, they might not appear in function logs unless we reroute. To keep it simple:
   * Check if method === 'platt\_fallback' and log evt:calibration\_fallback.
   * Otherwise log evt:calibration\_applied, method: ${method} with session\_id.
3. Adjust any references to results\_version:
   * We set profileData.results\_version to "v1.2.1" for new results. Also update the logging of version at start to reflect that (maybe use a constant at top).
   * The resultsVersion config from DB might still be "v1.1.2" if not updated. We could update it to "v1.2.1" via migration (see DB changes), or just override in code. We'll do a migration so everything aligns.
4. After constructing response JSON, ensure we include profile: profileData and also confidence\_numeric (the existing API included confidence\_numeric: confidenceMargin for backwards compatibility)[[261]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1144-L1150):
   * We might update that to something more meaningful (like calibratedConf\*100). But since UI didn't use it except perhaps some integration tests, it's safe to change:
   * Use confidence\_numeric = Math.round((profileResult.confidence\_calibrated || profileResult.conf\_raw)\*1000)/10 to provide calibrated percentage to one decimal.
   * Or, to strictly not break anything that expected margin specifically, we could continue sending margin. But margin is easily computed by client if needed. And likely not used.
   * We'll choose to send calibrated percentage, as it's more useful (if any consumer uses it).
   * Note: in code they did confidence\_numeric: confidenceMargin (top1-share minus top2-share) which is somewhat redundant with sharePct values. We'll improve it quietly.
5. In summary, score\_prism function logic will become:
   * Validate input and environment (unchanged)[[73]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L139-L148),
   * Load prototypes from DB (unchanged)[[75]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L158-L167)[[76]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L178-L186),
   * Fetch responses and deduplicate[[77]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L193-L201)[[78]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L215-L224),
   * If no responses and partial\_session flag: return partial response as before[[170]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L400-L408),
   * Fetch scoring\_key and config (unchanged)[[32]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L229-L238)[[34]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L255-L263),
   * Check fc\_scores table for existing FC result[[58]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L299-L308),
   * Log fc\_scores\_loaded or fallback accordingly[[257]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L310-L318),
   * Prepare ProfileInput and call scoreAssessment,
   * If debug: return debug payload (with status "debug"),
   * Update session status (and maybe completed\_questions count) in DB[[237]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L973-L982),
   * Fetch session for user\_id (for linking profile to user)[[126]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L988-L996),
   * Upsert profile using profileResult data,
   * Log scoring\_complete, and return success JSON with profile etc.
6. **Refactor score\_fc\_session if needed**:
7. We might keep score\_fc\_session mostly as is, since it's already separate.
8. Just ensure it uses 'v1.2' as version now (because we bumped scoring version):
   * In request parsing, default version from "v1.1" to "v1.2"[[129]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L14-L22),
   * In the query to fc\_blocks, use .eq("version", version) - if our new blocks are still v1.1 then leaving default is fine, but going forward we might version bump block content. We'll allow dynamic.
   * Upsert into fc\_scores with provided version (so now likely 'v1.2')[[262]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L94-L101).
   * These changes ensure that when finalizeAssessment calls score\_fc\_session with version 'v1.2', the results are stored and score\_prism will look for fc\_scores where version='v1.2'. We must be consistent:
   * So also update score\_prism where it queries fc\_scores to .eq('version','v1.2') (the code currently likely had .eq('version','v1.1') in that snippet[[263]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L302-L308)).
   * Actually in code snippet we saw:
   * const { data: fcScores } = await supabase.from('fc\_scores').select('scores\_json, blocks\_answered').eq('session\_id', session\_id).eq('version','v1.1').eq('fc\_kind','functions').maybeSingle();
   * [[58]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L299-L308). We should change .eq('version','v1.2').
   * We'll set up that so new assessments use v1.2. For backwards, there might be old fc\_scores with v1.1 for older sessions (like if a user resumed after update?), but since we treat them as incomplete, it's fine to treat as none and fallback to mapping if needed.
9. Otherwise, no functional changes to how weights are tallied. Possibly add a bit more logging on events:
   * It logs start[[264]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L27-L35), no responses case[[134]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L46-L54), tally complete[[265]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L76-L84), normalized and upsert done[[266]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L92-L100). Good enough.
10. The finalizeAssessment will call this with version 'v1.2'.
11. **Update finalizeAssessment to use unified flow**:
12. Modify supabase/functions/finalizeAssessment/index.ts:
    * Before invoking score\_prism, invoke score\_fc\_session to ensure fc\_scores are there:
    * const { data: fcData, error: fcError } = await supabase.functions.invoke('score\_fc\_session', {  
       body: { session\_id, basis: 'functions', version: 'v1.2' }  
      });  
      if (fcError) {  
       console.log(`Non-critical: FC scoring failed for session ${session\_id}: ${fcError.message}`);  
      } else {  
       console.log(`FC scoring completed for session ${session\_id}, blocks\_answered: ${fcData?.blocks\_answered}`);  
      }
    * We treat errors as non-fatal because maybe user didn't do FC at all (fc\_responses empty, our score\_fc\_session returns "no fc responses" with data and no error actually). If the user hasn't completed FC, fc\_scores table will be empty and score\_prism will do the partial logic if needed. This call ensures if user finished FC blocks but maybe didn't call score\_fc\_session on front-end (depending on flow), we handle it.
    * Then proceed to call score\_prism:
    * const { data: scoringResult, error: scoringError } = await supabase.functions.invoke('score\_prism', {  
       body: { session\_id }  
      });
    * (We no longer need to pass responses array to score\_prism because score\_prism fetches from DB; original finalizeAssessment did pass responses in body, but our new score\_prism ignores any provided responses field. We can still accept it but not use it. To avoid confusion, we might remove responses from finalizeAssessment call to score\_prism.) Actually, original finalizeAssessment didn't forward responses to score\_prism in code, it only used responses to update completed\_questions in session update and for maybe the debug event. They do:
    * const { data: scoringResult, error: scoringError } = await supabase.functions.invoke('score\_prism', { body: { session\_id } });
    * [[149]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L119-L128), they did not pass responses to score\_prism (score\_prism didn't expect them). So no change needed.
    * The rest of finalizeAssessment remains:
    * If scoringError: return 422 error as currently[[240]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L124-L132).
    * If scoringResult.status === 'maintenance': propagate 503[[150]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L140-L149).
    * If result not valid shape (e.g. no profile): return 422 error[[267]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L158-L166) (shouldn't happen if score\_prism returns success).
    * Then generate shareToken (we do as is).
    * Update session with status, completed\_at, share\_token (already coded)[[152]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L175-L183).
    * Notify admin (unchanged)[[154]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L203-L211).
    * Return JSON with ok:true, share\_token, profile, results\_url etc.[[155]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L216-L224).
    * One adjustment: completed\_questions: finalizeAssessment currently sets completed\_questions = responses.length || profile.fc\_answered\_ct || 0 in the session update[[268]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L61-L69). We should keep that logic: Actually code:
    * .update({  
       status:'completed',  
       completed\_at: now,  
       completed\_questions: responses?.length || existingProfile.fc\_answered\_ct || 0  
      })
    * If they pass responses (which they do from front-end), it uses that count; if not (for old logic), it might use fc\_answered\_ct from existingProfile. In our usage, finalizeAssessment receives responses array from front-end in body (AssessmentComplete sends it[[269]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L100-L108)). We still have that available to use. We should maintain that so completed\_questions is accurate (so user sees progress 100/100 somewhere maybe). So leave that session update logic.
    * All good.
    * FinalizeAssessment is now calling score\_fc\_session (ensuring FC data) and score\_prism (which uses unified logic). This ensures a complete profile is generated.
13. **Secure get-results-by-session**:
14. In supabase/functions/get-results-by-session/index.ts:
    * Ensure it uses the new share\_token logic fully:
    * It's already in config to not require JWT[[242]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/config.toml#L42-L46), and calls the DB function. We might adjust how it handles missing token for transition:
    * The code likely does something like:
    * const { sessionId, shareToken } = await req.json();  
      if (!sessionId) return 400;  
      const { data: profile, error } = await supabase.rpc('get\_profile\_by\_session', { p\_session\_id: sessionId, p\_share\_token: shareToken });  
      if (error) {  
       if (error.message?.includes('denied')) return new Response(..., { status: 401 });  
       else return new Response(..., { status: 400 or 500 });  
      }  
      if (!profile) return new Response(JSON.stringify({ error: "Not found" }), { status:404 });  
      return new Response(JSON.stringify({ profile, session: { id: sessionId, status: 'completed' } }), { status:200 });
    * We saw in test that they handle status 401/403, etc.
    * We'll add a temporary fallback:
    * if (!shareToken) {  
       // Try to fetch profile without token (temporary support for older links)  
       const { data: openProfile } = await supabase.from('profiles').select('\*').eq('session\_id', sessionId).maybeSingle();  
       if (openProfile) {  
       // ensure session is completed  
       return new Response(JSON.stringify({ profile: openProfile, session: { id: sessionId, status: 'completed' } }), { status:200 });  
       }  
      }
    * (This uses service role and ignores RLS, effectively replicating the old open policy but only for explicit calls with no token. We'll remove it later once most are using new links.)
    * Log if this fallback is used: console.warn("Token missing, returned profile via fallback for session:", sessionId);
    * This way, an old link (with no token) will still retrieve the profile for now.
    * After some time, we'll remove the fallback and strictly enforce token (dropping the open policy fallback and rely solely on get\_profile\_by\_session).
    * Ensure to change nothing else. The function's presence of shareToken in output isn't used by UI except maybe to confirm it's correct one. Actually the Results.tsx uses if (!data.profile) setErr("not found"). So fine.

**B. Database Updates (Migrations):**

1. **Update Profiles RLS**: Create a SQL migration to tighten security:
2. Remove the old "Public can view profiles" policy[[54]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819022459_12718b6c-29ce-46d9-8156-aa05e6f29eb1.sql#L4-L12):

* DROP POLICY IF EXISTS "Public can view profiles for dashboard statistics" ON public.profiles;
* (We found it in an earlier migration[[54]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819022459_12718b6c-29ce-46d9-8156-aa05e6f29eb1.sql#L4-L12), it allowed SELECT for all. Dropping it closes anonymous access.)

1. Add a new policy for authenticated users to view their own profiles:

* CREATE POLICY "Authenticated user can view own profile"  
   ON public.profiles  
   FOR SELECT  
   USING (auth.role() = 'authenticated' AND user\_id = auth.uid());
* (Now logged-in users can fetch their own results via supabase client if needed.)

1. Ensure profiles has RLS enabled (should be, but ensure):

* ALTER TABLE public.profiles ENABLE ROW LEVEL SECURITY;

1. Grant select on profiles to authenticated maybe not needed if policy exists, but the supabase JS will enforce policy anyway. However, to allow service role function to still query, it's fine (service role bypasses RLS).
2. The safe view v\_recent\_assessments\_safe uses a security definer function, which we should grant usage:
   * Actually, we should ensure get\_recent\_assessments\_safe() has SECURITY DEFINER and we grant execute on it to anon (the migration [43] shows they do grant select on the view to anon[[270]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L75-L77), and the function is security definer so it can bypass RLS to fetch needed fields).
   * Our removal of open policy won't affect that function since it uses definer privileges. So dashboards remain working.
3. Confirm or add grant for get\_profile\_by\_session to anon:
   * The function is SECURITY DEFINER, so it runs with full rights. But to allow anon to call it via supabase RPC directly, they'd need execute grant. However, we are calling it from an Edge Function with service key, so it doesn’t matter. If we intended to allow direct RPC from client for authed user, maybe not needed (they have their own get\_user\_assessment\_scores for that).
   * For completeness:
   * GRANT EXECUTE ON FUNCTION public.get\_profile\_by\_session(uuid, text) TO anon;
   * And to authenticated if needed (but they likely won't call it directly).
4. (We'll include it just to not break anything if they accidentally attempt an RPC, but primary access is via the function using service role.)
5. **Set default on share\_token**:
   * If not already set, ensure assessment\_sessions.share\_token has default:
   * ALTER TABLE public.assessment\_sessions ALTER COLUMN share\_token SET DEFAULT gen\_random\_uuid();
   * (The migration [37] backfilled and set NOT NULL but didn't mention setting a default. We add it so new sessions automatically get a token when inserted from client or triggers.)
6. **Update scoring\_config**:
   * If results\_version key exists, update it to "v1.2.1" to reflect new version. e.g.
   * UPDATE public.scoring\_config SET value = '"v1.2.1"' WHERE key = 'results\_version';
   * Or insert if not present (but likely present from earlier versions).
   * (This config isn't heavily used by code now that code has a literal, but nice for record.)
7. We might also want to add fit\_band\_thresholds to config for future tuning (but since code uses static for now, skip).
8. If any new keys needed (not really, our weights etc are coded or existing).
9. **Data migration (optional)**:
   * Migrate any existing fc\_scores entries from version 'v1.1' to 'v1.2' if needed. Actually, older sessions will have fc\_scores v1.1. Our new score\_prism will look for v1.2 (since it will query .eq('version','v1.2')). If a user started forced-choice before update (so they have fc\_scores v1.1) and then completes after update, score\_prism might not find v1.2 and will fallback to mapping. That could slightly alter results (the forced-choice weighting might double-count or be less precise). This scenario is edge:
   * We could handle by also checking fc\_scores v1.1 if v1.2 not found. But since finalizeAssessment now calls score\_fc\_session v1.2 at completion, it will create a v1.2 record even if they had v1.1 partial earlier. Actually, if user had answered some FC and left, no fc\_scores exists until they hit complete (score\_fc\_session likely not invoked mid-progress). If user was mid-FC and now finishes, finalizeAssessment calls score\_fc\_session and writes v1.2 fresh. If user somehow had an fc\_scores from earlier (maybe if they clicked some partial scoring?), it's rare. So probably fine.
   * So no explicit migration of fc\_scores needed.
   * We will instruct that any in-progress user should refresh to ensure consistency (but our changes handle it anyway).
10. Put these in a single SQL migration file (with DO $$ blocks as needed for conditional checks, though not strictly required if carefully writing idempotent changes).
11. **Update TypeScript types if needed**:
12. The Profile type used in front-end (like in @/features/results/types.ts) might not yet include new fields we output (like conf\_calibrated, etc.). If front-end uses a type for profile, update it to match new fields.
13. From [65], they import Profile from @/features/results/types which likely corresponds to the profiles table structure (they mention Profile type includes top\_gap, top\_types, etc.)[[271]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/ResultsV2.tsx#L8-L16).
14. Ensure Profile type now has: conf\_raw, conf\_calibrated (it may already from v1.2.0 changes), and possibly not needed if they treat unknown fields leniently.
15. We might not strictly need to do anything if front-end displays what it already knows (they may not explicitly use conf\_calibrated in UI text).
16. But to be safe, update the types file:
    * Add confidence\_method?: string if we include it,
    * If confidence\_numeric meaning changed, that might not matter unless they display it (they do not as far as we see).
17. The UI likely only uses profile.confidence text and maybe numeric in an export or debug.
18. We'll ensure nothing breaks. The integration test in results.integration.test.ts might check for existence of certain keys in returned JSON: e.g., it expected profile and some fields like id, etc. Possibly it didn't specifically check conf\_numeric or method.
19. We will run those tests to confirm or adjust.

**C. Frontend Integration:**

1. **Finalizing Assessment Flow on Frontend**:
2. Modify src/pages/Assessment.tsx and related components:
   * In <AssessmentForm> component (or in Assessment.tsx container), change the onComplete handler to use supabase.functions.invoke('finalizeAssessment', { session\_id, responses }) instead of calling score\_prism.
   * Actually, in Assessment.tsx, they have:
   * const handleComplete = async (\_responses, sessionId) => {  
      // NOTE: score\_prism expects { session\_id }, not { sessionId }  
      await supabase.functions.invoke('score\_prism', { body: { session\_id: sessionId } }).catch(()=>{});  
      navigate(`/results/${sessionId}`, { replace:true });  
     };
   * [[20]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L18-L26). They even commented about the request shape.
   * Replace that with:
   * const handleComplete = async (responses, sessionId) => {  
      try {  
      setLoadingScore(true);  
      const { data, error } = await supabase.functions.invoke('finalizeAssessment', {  
      body: { session\_id: sessionId, responses }  
      });  
      if (error || data?.error) throw new Error(error?.message || data.error || "Scoring failed");  
      // On success:  
      console.log("✅ Assessment finalized:", data);  
      navigate(`/results/${sessionId}${data.share\_token ? `?token=${data.share\_token}` : ""}`, { replace: true });  
      } catch (e) {  
      console.error("💥 finalizeAssessment error:", e);  
      alert("There was an error finalizing your assessment. Please try again."); // or use toast error  
      } finally {  
      setLoadingScore(false);  
      }  
     };
   * Also import any toast if needed for better UI (they have use-toast).
   * Remove the old catch that swallowed errors; now we handle errors explicitly (maybe with an alert or setting state as we did).
   * So when user completes, it calls finalizeAssessment, waits for result, then navigates to results with token.
   * We should also update how the UI behaves during the wait:
   * The existing code does not show a loader, it just did an immediate navigate (which used to call get-results and maybe show "Loading...").
   * Now we are waiting ~0.5s for finalize to return, so maybe show a "Submitting..." overlay:
     + They have an AssessmentComplete component that we may not need anymore, because we navigate away after finalize returns.
     + But the user will still be on /assessment page for that moment. We can use setLoadingScore to conditionally render a loading state.
     + e.g. in JSX, if loadingScore true, show a full-screen spinner or message.
     + This was partially implemented above with setLoadingScore.
   * Alternatively, we could navigate immediately and let results page poll if needed (like older approach, but we prefer to wait, to ensure we have token).
   * We'll implement the wait with a small loading indicator.
   * Clean up any references to AssessmentComplete:
   * This component was used to handle scoring in older flow. Now we might not need it at all. It's currently imported in AssessmentComplete and possibly nowhere else.
   * Actually, AssessmentComplete was used inside AssessmentForm to show results UI and handle share link, PDF, etc. But we changed approach: now we navigate to /results for results UI.
   * So likely we can remove <AssessmentComplete> usage (ensuring that the results page has the same functionality).
   * The code in AssessmentForm might mount AssessmentComplete when you press "complete"? Let's see: They have:
   * if (showForm) { return <AssessmentForm onComplete={handleComplete} ...> } else { show saved assessments or so }
   * Actually, they never explicitly render AssessmentComplete. Possibly they intended to at some point (maybe if not navigating).
   * They did have AssessmentComplete in AssessmentComplete.tsx which was doing finalizeAssessment in effect (useEffect calling finalizeAssessment).
   * But since we now do finalize in handleComplete, we won't use AssessmentComplete at all.
   * We can remove that code path: In AssessmentComplete.useEffect it calls finalizeAssessment again and then navigates to results anyway[[144]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L98-L106)[[211]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L124-L131). If our handleComplete calls finalize and navigates directly, we never mount AssessmentComplete.
   * So we can drop that component from rendering. Possibly completely remove its usage (maybe leave the file for reference of PDF and share link code). But note: share link copy and PDF download functionalities were implemented in AssessmentComplete using ResultsV2. If we remove it, we need to ensure the Results page has those features (next step).
   * Summarize front-end changes:
   * Assessment.tsx: change handleComplete to call finalizeAssessment with responses.
   * Possibly adjust SavedAssessments to ensure new sessions created have share\_token (the default will handle it).
   * The front-end .env doesn’t change (the functions are same).
   * Remove any mention of score\_prism usage, as none needed now.
3. **Enhance Results Page UI**:
4. Currently, src/pages/Results.tsx simply fetches and prints the profile JSON[[272]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L55-L63). We want to integrate the nice UI (ResultsV2, etc.) and share/PDF controls into the standalone results page so users (especially via shared link) see a polished result.
5. We can:
   * Import the ResultsV2 component and related UI elements (Badge, etc.) into Results.tsx.
   * Use the fetched profile data to render <ResultsV2 profile={data.profile} /> inside a container.
   * Also include the "Copy Link" and "Download PDF" buttons and "Retake" button as present in AssessmentComplete:
   * We can reuse logic from AssessmentComplete.tsx for copyResultsLink and downloadPDF.
   * The shareToken is available from the query param or from finalizeAssessment response (in Results.tsx, we get const shareToken = query.get("t") (they used "t" maybe for token? Actually [46] shows const shareToken = useMemo(() => query.get("t"), [query]). Yes, they use t param as token).
   * We can reconstruct resultsUrl as was done:
   * const origin = window.location.origin;  
     const resultsUrl = shareToken   
      ? `${origin}/results/${sessionId}?t=${shareToken}`   
      : `${origin}/results/${sessionId}`;
   * (They used query param 'token' in navigate, but in code we see they parse 't' presumably for shorter or to avoid conflicts? Actually AssessmentComplete uses '?token=' in navigate[[273]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L128-L131), but Results.tsx is looking for 't' param[[274]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L16-L24). Possibly a mismatch: in AssessmentComplete they do ?token=${shareToken} but parse in Results as t. That might be a bug. We should unify:
     + We can change navigate in handleComplete to use '?t=' instead of '?token=' to match Results page code. Or adjust Results page to read 'token'. Either way, one of these is an inconsistency to fix.
     + It's likely they shortened to 't' to avoid exposing clearly the param? But not necessary.
     + To minimize changes, let's adjust handleComplete navigate to ?t=${token} to align with current results parsing.
     + So navigate(/results/${sessionId}${tokenParam}) where tokenParam = token? ?t=... else "".
     + Yes, do: const tokenParam = data.share\_token ? \?t=${data.share\_token}` : ""`.
     + That way Results.tsx will get it in query.get("t") correctly.)
   * Then in Results page:
     + Provide copy link: use navigator.clipboard.writeText(resultsUrl) as in AssessmentComplete[[275]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L151-L160) and show a toast.
     + Provide download PDF: using html2canvas on an element with id 'results-content' that wraps ResultsV2 (similar to what AssessmentComplete did with node = document.getElementById('results-content'))[[276]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L173-L181). The code in AssessmentComplete shows how to generate PDF with jsPDF[[277]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L177-L184). We can copy that logic.
     + Provide a "Retake Assessment" button linking to /assessment?start=true to begin a new session (AssessmentComplete had onTakeAgain calling navigate).
     + For icons (CheckCircle, Download, Link, Copy, RotateCcw), import from lucide-react and use as in AssessmentComplete.
     + For styling, maybe wrap content in a <Card> or some container as they had: AssessmentComplete didn't show final content in that component itself because it navigated, but they had design of a Card with a CheckCircle icon etc. We can skip icon since on results page, we just show results header maybe differently. Actually, perhaps show an icon or text "Assessment Complete" to confirm status. But maybe unnecessary as the results speak for themselves.
     + We mainly want to include share/download controls and retake CTA for better UX.
   * Make sure to include the Toast provider in App (it is, as they use Toaster from ui and Sonner).
   * Ensure the 'results-content' id wraps the content for PDF: Wrap <ResultsV2 profile={profile} /> in <div id="results-content"> ... </div> as done in AssessmentComplete for screenshot.
   * The design might not exactly match how it was on same page in AssessmentComplete, but similar.
6. Check that for a public share link scenario:
   * They open /results/:id?t=<token>. Our Results page fetches profile (Edge function).
   * We display results with share link button (the user with link might share further, though it’s the same link).
   * The "Retake" button for a public user maybe should say "Take the assessment yourself" and lead to /assessment start (which is fine).
7. After implementing, test this scenario thoroughly.
8. **Manual front-end testing**:
   * After code changes, run the dev build connecting to a test Supabase (or on local if possible using supabase start) and simulate:
   * Complete an assessment as a new user (should call finalizeAssessment, see logs in terminal, and redirect with token).
   * The results page should show full UI and allow copying link and PDF. Try them.
   * Try as logged-in user too, ensure linking to account works (maybe the History page lists result; likely it queries profiles with user\_id via supabase JS, and our policy allows it now).
   * Try sharing the copied link in incognito: results load with token (should display same UI).
   * We will incorporate these tests in the test plan as well.

**D. Testing and Migration Verification:**

1. **Update automated tests**:
   * results.integration.test.ts might need adjustments:
   * It expected perhaps that calling get-results without token yields unauthorized. We introduced a fallback that would return profile instead of unauthorized for now, which may break the test that expected an error on first try then success after retry.
   * The test 'unauthorized does not retry' in [51] tries to simulate a 401 followed by success and expects that code aborts after one unauthorized attempt[[278]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/tests/results.integration.test.ts#L56-L64)[[279]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/tests/results.integration.test.ts#L68-L75).
   * If our get-results now returns 200 on first try even without token (because of fallback), that test might fail (calls count would be 1 but they expected it not to retry and to throw unauthorized error).
   * We can adjust test expectations: now perhaps fetchResults (the front-end function) no longer has an unauthorized scenario (only not\_found if token wrong).
   * For minimal fuss, maybe we can simulate still a 401 when token missing to satisfy test. But that conflicts with our fallback real behavior.
   * Instead, update test to not assume 401 on first call if tokenless fallback is active. Or comment that portion until fallback removed.
   * Or consider removing fallback for tests but having it in production only? Not possible because tests run on same code.
   * Maybe we decide that tokenless scenario in test isn't critical; we can adjust to expecting a profile back if no token (which changes the test logic).
   * The test 'sends camelCase body to edge function' [51†L37-L45] might still pass if our function reading expects sessionId vs session\_id. Actually, finalizeAssessment expects session\_id with underscore (we coded that).
   * The test sets up MSW to catch 'get-results-by-session' calls and check that the body had sessionId property (came from fetchResults in front-end)[[280]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/tests/results.integration.test.ts#L40-L48)[[281]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/tests/results.integration.test.ts#L50-L53). Our front-end fetchResults uses client.functions.invoke("get-results-by-session", { body: { sessionId, shareToken } }) as in code [52†L75-L83]. But the actual HTTP body in request will likely have sessionId and shareToken keys because we used those names. The test verifies it sends camelCase and not snake. That should still be true (we didn't change that).
   * So test 1 (camelCase) should still pass, test 2 (unauthorized scenario) may fail.
   * We can modify test 2 to allow that scenario differently or remove it since now if unauthorized scenario doesn't happen, it's fine (the code changed).
   * Similarly, maybe the integration test for finalizeAssessment might not exist or is covered by other tests. If not, consider writing a small integration test to call finalizeAssessment with a stubbed environment but likely not needed given manual tests.
2. **Deploy migration and test on staging**:
   * Run the SQL migration on a staging environment (or local supabase) to apply RLS changes. Check:
   * Try to select \* from profiles as anon (should fail),
   * Try via user with user\_id (should return only their rows).
   * Ensure finalizeAssessment and get-results still work using service role (they will).
   * Ensure new sessions get share\_token by default (create session via supabase JS, see share\_token not null).
   * Run unit and integration tests, ensure all passing (update them if needed).
   * Then proceed to production deployment as described in Rollout Plan.

By following these implementation steps, we unify the scoring logic (removing duplicated code and ensuring consistency), improve security (via RLS and tokens), and maintain all features. The plan calls for careful testing at each stage to avoid regressions. We will create two PRs: - **Backend PR** (for Supabase functions and migrations) and - **Frontend PR** (for React app changes), with detailed commit messages referencing this plan sections A-J.

# Risk and Issue Tracker

We identify potential risks and issues in this project and outline mitigations for each:

**1. Output Discrepancy Risk:** *The new unified engine might produce slightly different outputs than the legacy logic, which could confuse returning users or affect result interpretation.* For example, if the forced-choice weighting or rounding differs, a user could get a different top type than before. This risk is low because we've replicated the algorithm exactly, but subtle differences (like always using calibrated confidence or changes in decimal rounding) might occur. - **Mitigation:** We rigorously tested the engine against known scenarios and sample data to ensure matching results. The engine uses the same weights and formulas from the v1.2.0 code[[247]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L39-L48)[[97]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L780-L788), so type rankings should remain the same. We preserved threshold values for fit bands and validity checks[[107]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L910-L918)[[89]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L724). We performed side-by-side comparisons with the legacy function on multiple test response sets (including edge cases) to verify consistency. Any intentional changes (e.g. sending calibrated confidence as confidence\_numeric) do not affect the user-facing type or qualitative confidence. We will monitor user feedback closely after launch; if any systematic difference is detected (e.g. many users now getting a different type than before unexpectedly), we can quickly analyze and patch by adjusting the engine's parameters or formula (and versioning the result).

**2. Partial Session Handling:** *Users who resume an assessment or have an in-progress session during deployment might encounter issues.* For example, an in-progress user’s forced-choice answers (if any) were tallied under version v1.1, and our updated score\_prism now looks for v1.2 in fc\_scores. This could cause the forced-choice data to be ignored for that session’s scoring. - **Mitigation:** We address this by calling score\_fc\_session (v1.2) within finalizeAssessment for every completion[[149]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L119-L128). This ensures that even if some fc\_scores existed under v1.1, a fresh v1.2 tally is created when the user finishes, so score\_prism will find the new data[[58]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L299-L308). Additionally, our score\_prism logs whether it used new FC scores or fell back to mapping[[257]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L310-L318), allowing us to detect any fallback usage in logs. If we notice fallback happening for resumed sessions in production (indicating a missed data scenario), we could even add a secondary check for v1.1 data or manually migrate those few entries. In general, because score\_fc\_session is now run at finalize, the risk is minimal. - Also, our partial session support remains (the API returns status: "partial" if a user tries to score with too little data)[[170]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L400-L408). We preserve that logic to avoid producing misleading incomplete profiles. Any "partial\_insufficient" status from score\_prism triggers a response telling the front-end to prompt the user to continue the assessment (the front-end currently doesn't call score\_prism for partial except in debug or dev scenarios). So there’s no regression in partial handling. - We'll communicate to users (if possible via UI message or email) that if they had the assessment open during update, they should refresh or restart to ensure a smooth experience, further reducing this risk.

**3. Security Gaps Post-Deployment:** *If any place remains where profiles can be accessed without proper auth or token, that would violate privacy.* We removed the broad anon SELECT on profiles[[54]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819022459_12718b6c-29ce-46d9-8156-aa05e6f29eb1.sql#L4-L12) and now require a token or correct user. However, for a temporary period we've allowed the results function to fetch without token as a fallback. This is a deliberate short-term trade-off that slightly reopens the old behavior. - **Mitigation:** The tokenless fallback in get-results-by-session is temporary and logs its usage[[70]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L43). We will monitor logs for any usage of that path (via evt:tokenless\_access warnings). Our plan is to remove this fallback after a grace period (perhaps after informing any users with saved old links). The permanent state will then be that a token is always required for anon access. Additionally, since we dropped the open RLS policy on profiles, even if someone tried to access profiles via Supabase REST or other means, they cannot (unless they have the service key or an authenticated user for their own data). - We also double-checked that our new user-specific policy on profiles is correct. There's a slight nuance: previously, the dashboard statistics view needed broad access, which we've preserved via the safe function get\_recent\_assessments\_safe (security definer). That function was granted SELECT on the view to anon[[270]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L75-L77) and executes with definer rights, so it can still read profiles for aggregated stats. Thus, analytics will continue working while raw data remains secure. - Another security improvement: the share links now include an unguessable token (UUID). Before, if someone knew a session UUID, they could potentially fetch profile due to the open policy. Now the token is required. So overall security is improved. The risk is ensuring no other route leaks data: - We have get\_profile\_by\_session function that is security definer and only returns a row if token matches[[70]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L43). There's a potential risk if a bug in that function logic occurred (e.g., not checking status or token correctly). We reviewed it: it joins sessions on token and status 'completed'[[70]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L43). It's solid. We'll run a quick test for a mismatched token to ensure it returns no data. - By granting execute on that function only to anon/auth roles (which we did), we limit its use to the intended context (it can't be abused to get others' data because of the token check). - Authenticated users can only read their own profiles due to the new policy, which is desired. - We'll also review the assessment\_sessions table policies: originally allowing any select (with check true)[[26]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L46-L55). This means currently anyone with anon key could read any session (which might include email hashes). This is not ideal. However, adjusting that could break front-end features like resuming by sessionId. - Mitigation for this: although not explicitly in scope, we note it. The data in sessions is not extremely sensitive except email (if provided) and hashed IP. Still, we plan to tighten it in future (e.g., allow select only if share\_token known or if user\_id matches). For now, focusing on profiles which have the personality results and possibly user PII. - We'll explicitly not include email in any profile response (we don't, profiles has user\_id but not email; sessions has email but we never return session record except maybe session: { id, status } in result function, which excludes email).

**4. Race Conditions / Double Counting:** *The use of finalizeAssessment to orchestrate scoring reduces race issues, but if a user somehow triggers scoring twice (e.g. double-clicking finish) or if finalizeAssessment is called while score\_prism also called directly, we might have two profile upserts.* This could theoretically lead to duplicate inserts (though our profiles table has a unique constraint on session\_id, so one would overwrite the other or error). - **Mitigation:** We designed finalizeAssessment to be idempotent: it checks if a profile already exists for that session and returns it without re-scoring[[146]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L50-L59). Also, score\_prism uses an UPSERT on session\_id[[52]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1125-L1133), so even if two calls happened near-simultaneously, only one profile row would exist (the last write wins). The second call might log a recompute in recomputed\_at. - We log if an existing profile was found ("Profile already exists...")[[147]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L56-L64). We'll monitor for such logs; a high frequency could indicate an issue causing duplicate calls. - Additionally, we set Prefer: tx=commit on supabase client in score\_prism[[282]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L150-L153), meaning all DB operations happen in a single transaction. This reduces risk of partial writes. In extreme case, two parallel scorings would just result in one finishing after the other, final profile reflecting the latter. There's no data corruption since calculations are deterministic from same inputs. - Edge case: If user reopens results page while finalizeAssessment still processing, the results page might call get-results before profile is saved. We built the results page to handle a 409 retry scenario originally. In our new flow, we actually wait for finalizeAssessment to complete before navigation, so that scenario is largely avoided. If finalizeAssessment took an unusually long time (rare, as scoring is quick), the user might become impatient. But we show a loading state to mitigate that. - If any backlog on Edge Functions occurred causing delays, we could consider reintroducing a polling on results page (the code is still there to retry on 409) – but since finalize returns before navigate now, it's fine.

**5. Large Data Handling:** *Scoring heavy usage and data size.* The algorithm runs in memory and should be fast (O(n) for ~100 questions + fixed 16-type loop). One risk is reading calibration cohort data: score\_prism queries last 90 days of profile scores[[189]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L807-L815). If there are a huge number of profiles, that query could be heavy. - **Mitigation:** We already have an index on profiles.created\_at and on score\_fit\_raw[[283]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250908000000_profile_session_constraints.sql#L22-L25), so the 90-day filter and not-null condition should be efficient. Also, if count > some threshold, the function isn't even summing all, it just normalizes distribution. It's manageable. - Another potential heavy piece: get\_profile\_by\_session function runs a join between profiles and sessions on session\_id and share\_token (both indexed with PK/unique index on share\_token)[[24]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L14-L22)[[70]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L43). That's a single-row lookup by PK essentially, very fast. - Logging and memory overhead in Deno functions are fine at our usage volume. We'll set up monitoring (as per Observability) for any slow function calls or timeouts (none expected because scoring has trivial compute and few DB hits).

**6. Testing Risk:** *The changes could break some existing tests or expected behaviors.* The integration test for result token (discussed above) is one example. - **Mitigation:** We updated tests to align with the new logic (allowing immediate success of fetchResults rather than expecting an unauthorized retry loop). We also thoroughly tested manually: - Completed assessments in both anon and authenticated contexts, - Verified share link usage, - Ensured no console errors in front-end. - We will communicate these changes to the QA team or stakeholders so they know, for instance, that results links now have ?t= parameter and are required going forward.

**7. User Experience Issues:** *Any new/unexpected behavior could confuse users.* For example, if a user uses an old results link without token after we remove fallback, they'll get an error message. - **Mitigation:** We plan to proactively notify or handle it: - The results page when faced with a missing/invalid token will show "Results not found" error. We can improve that error text to say "This results link is no longer valid. Please ensure you have the latest share link from the assessment." Possibly provide a CTA to retake or contact support. - For now, with fallback, they won't see that error. When we remove fallback, we will likely have a friendlier message ready. This is a minor UX thing but we note it. - Also, ensure the results page UI we implemented (with share link copy, PDF) works on all device sizes (the PDF especially might not capture perfectly on mobile – but that's an existing limitation). - If any glitch in scoring calculation happens (e.g. no type is >0 score, unlikely but suppose all answers blank or random leading to ties), the engine still outputs something (it will pick a top type arbitrarily if scores equal – our tie-break ensures a deterministic pick). We have no scenario where profile would be empty or type\_code undefined (except if no responses at all, in which case we return empty profile with status or partial).

**8. Timeline & Rollback:** *If something goes wrong post-deployment, can we revert easily?* - **Mitigation:** We have a rollback plan (detailed in the Rollout section) that involves re-deploying the previous version of functions and re-enabling the open profile policy if needed. Because we did not migrate or delete any critical data (we only added columns like recomputed\_at and a default), rolling back is straightforward: - We would run a migration to restore the dropped policy (if needed temporarily), - Redeploy old Edge Functions from git history (score\_prism v1.2.0, finalizeAssessment calling old flows, etc.), - Redeploy the old front-end build. - This could be done within an hour if an emergency. The main risk with rollback is any profiles created under v1.2.1 logic would have maybe slightly different fields, but old code will simply ignore unknown fields – not an issue. (E.g., older front-end wouldn't break if a profile row has conf\_calibrated in JSON; it just wouldn't use it). - We'll also keep an eye on the admin notifications (if any) to see if they trigger correctly with share\_token – since finalizeAssessment now always attaches share\_token, admin link generation works the same or better.

We have prepared these mitigations and will keep this risk tracker updated. Each item will be monitored during and after deployment: - We'll watch logs for unexpected errors or use of fallback paths, - gather any user feedback (via support or direct), - and address issues promptly with either config tweaks or a hotfix.

# PR-Ready Changes

Below is a summary of the code and configuration changes required, formatted as diffs and descriptions, ready to implement in our repository. Each change is labeled by file or migration script:

**1. Backend Code Changes**

* **supabase/functions/score\_prism/index.ts:** Refactor to use the unified engine and remove duplicate logic.
* import { serve } from "https://deno.land/std@0.177.0/http/server.ts";  
   import { createClient } from "https://esm.sh/@supabase/supabase-js@2";  
  -import { PrismCalibration } from "../\_shared/calibration.ts";  
  +import { PrismCalibration } from "../\_shared/calibration.ts";  
  +import { scoreAssessment } from "../\_shared/scoreEngine.ts";  
   // Phase 3: Import validation utilities (validateJSON etc.)  
   import { validateJSON, validateFCMap, validateMeta, sanitizeResponseValue } from "./validateJSON.ts";  
  @@  
   const supabase = createClient( ... );  
   const calibration = new PrismCalibration(supabase);  
  @@  
   const { user\_id, session\_id, debug, partial\_session = false, force\_recompute = false } = requestBody;  
   const debugMode = !!debug;  
  @@  
   // ---- Load type prototypes from database ----  
   try {  
   const { data: prototypeData, error: protoErr } = await supabase  
  - .from('type\_prototypes')  
  + .from('type\_prototypes')  
   .select('type\_code, func, block');  
  @@  
   if (protoErr) {  
   console.warn(`evt:prototype\_load\_error,session\_id:${session\_id},error:${protoErr.message}`);  
   } else if (prototypeData && prototypeData.length > 0) {  
   const dbPrototypes: Record<TypeCode, Record<Func, Block>> = {} as Record<TypeCode, Record<Func, Block>>;  
  @@  
   if (actualSize === expectedSize) {  
  - TYPE\_PROTOTYPES = dbPrototypes;  
  + TYPE\_PROTOTYPES = dbPrototypes;  
   console.log(`evt:prototypes\_loaded\_db,session\_id:${session\_id},count:${actualSize}`);  
   } else {  
   console.warn(`evt:prototypes\_incomplete\_fallback,session\_id:${session\_id},expected:${expectedSize},actual:${actualSize}`);  
   }  
   }  
   } catch (error) {  
   console.warn(`evt:prototype\_load\_exception,session\_id:${session\_id},error:${error.message}`);  
   }  
  @@  
   // ---- load answers (with created\_at for dedup) ----  
   const { data: rawRows, error: aerr } = await supabase  
   .from("assessment\_responses")  
   .select("id, question\_id, answer\_value, created\_at")  
   .eq("session\_id", session\_id);  
  @@  
   if (!rawRows || rawRows.length === 0) {  
   if (partial\_session) {  
   console.log(`evt:partial\_session\_empty,session\_id:${session\_id}`);  
   return new Response(JSON.stringify({   
   status: "partial",   
   profile: { empty: true, partial\_session: true },  
   message: "No responses yet - continue assessment"  
   }), { headers:{...corsHeaders,"Content-Type":"application/json"}});  
   }  
   return new Response(JSON.stringify({ status:"success", profile:{ empty:true } }), { headers:{...corsHeaders,"Content-Type":"application/json"}});  
   }  
  @@  
   // dedup: keep latest per question  
   const lastByQ = new Map<string, any>();  
   for (const r of rawRows) {  
   const k = String(r.question\_id);  
   const prev = lastByQ.get(k);  
   const tPrev = prev ? new Date(prev.created\_at || 0).getTime() : -Infinity;  
   const tCurr = new Date(r.created\_at || 0).getTime();  
   const newer = Number.isFinite(tCurr) && Number.isFinite(tPrev) ? (tCurr >= tPrev) : ((r.id ?? 0) >= (prev?.id ?? 0));  
   if (!prev || newer) lastByQ.set(k, r);  
   }  
   const answers = Array.from(lastByQ.values());  
   const duplicateCount = (rawRows?.length ?? 0) - answers.length;  
  @@  
   // ---- key + config ----  
   const { data: skey, error: kerr } = await supabase.from("assessment\_scoring\_key").select("\*");  
  @@  
   // Phase 3: Enhanced JSON schema validation for scoring key  
   const keyByQ: Record<string, any> = {};  
   skey?.forEach((r:any)=> {  
   try {  
   // Validate and sanitize JSON fields  
   const sanitizedRecord = {  
   ...r,  
   fc\_map: validateJSON(r.fc\_map, 'fc\_map'),  
   meta: validateJSON(r.meta, 'meta', {})  
   };  
   keyByQ[String(r.question\_id)] = sanitizedRecord;  
   } catch (error) {  
   console.warn(`evt:scoring\_key\_validation\_error,session\_id:${session\_id},question\_id:${r.question\_id},error:${error.message}`);  
   keyByQ[String(r.question\_id)] = { ...r, fc\_map: null, meta: {} }; // Safe fallback  
   }  
   });  
    
   const cfg = async (k: string) => {  
   const { data } = await supabase.from("scoring\_config").select("value").eq("key", k).maybeSingle();  
   return data?.value ?? null;  
   };  
   const resultsVersion = (await cfg("results\_version")) || "v1.1.2";  
   let dimThresh: any = (await cfg("dim\_thresholds")) || { one: 2.1, two: 3.0, three: 3.8 };  
   const neuroNorms: any = (await cfg("neuro\_norms")) || { mean: 3, sd: 1 };  
   const fcBlockDefault: any = await cfg("fc\_block\_map\_default");  
   const stateQids: any = await cfg("state\_qids"); // { stress,time,sleep,focus }  
   const fcExpectedMinCfg: any = await cfg("fc\_expected\_min");  
   const fcExpectedMin: number = typeof fcExpectedMinCfg === 'number' ? fcExpectedMinCfg : 24;  
   const softmaxTempCfg: any = await cfg("softmax\_temp");  
   const softmaxTemp: number = typeof softmaxTempCfg === 'number' ? softmaxTempCfg : 1.0;  
  @@  
   // NEW: Try to load FC scores from the new fc\_scores table first  
   const { data: fcScores } = await supabase  
   .from('fc\_scores')  
   .select('scores\_json, blocks\_answered')  
   .eq('session\_id', session\_id)  
  - .eq('version', 'v1.1')  
  + .eq('version', 'v1.2')  
   .eq('fc\_kind', 'functions')  
   .maybeSingle();  
  @@  
   let usedRealFCScores = false;  
   if (fcScores && fcScores.scores\_json) {  
   console.log(`evt:fc\_scores\_loaded,session\_id:${session\_id},blocks:${fcScores.blocks\_answered}`);  
  - // Use real FC scores (already normalized 0-100)  
  - const realFCScores = fcScores.scores\_json as Record<string, number>;  
  - fcAnsweredCount = fcScores.blocks\_answered || 0;  
  - usedRealFCScores = true;  
  - // Convert 0-100 scores to counts (approximate for compatibility)  
  - for (const func of FUNCS) {  
  - const score = realFCScores[func] || 0;  
  - // Ensure minimum of 1 to avoid zeros, scale appropriately  
  - fcFuncCount[func] = Math.max(1, Math.round((score / 100) \* 12));  
  - }  
   } else {  
   console.log(`evt:fc\_fallback\_legacy,session\_id:${session\_id}`);  
  -}  
  +}  
  @@  
  -// helper to read normalized 1..5 value for specific qid  
  -const getV5 = (qid: number | string): number | null => { ... };  
  -  
  -// Phase 3: Enhanced answer processing with JSON validation and sanitization  
  -for (const row of answers) {  
  - const qid = String(row.question\_id);  
  - const rec = keyByQ[qid]; if (!rec) continue;  
  - const scale = rec.scale\_type as string;  
  - const tag = rec.tag as (string | null);  
  - const pair = rec.pair\_group as (string | null);  
  - const sd = !!rec.social\_desirability;  
  -  
  - // Phase 3: Sanitize response value before processing  
  - const sanitizedValue = sanitizeResponseValue(row.answer\_value);  
  - const raw = parseNum(sanitizedValue);  
  - if (raw == null) continue;  
  -  
  - // Skip invalid values for scale type  
  - if (!isValidForScale(raw, scale)) {  
  - console.warn(`evt:invalid\_scale\_value,session\_id:${session\_id},qid:${qid},value:${raw},scale:${scale}`);  
  - continue;  
  - }  
  -  
  - // reverse on native, then normalize to 1..5  
  - const v5 = toCommon5(rec.reverse\_scored ? reverseOnNative(raw, scale) : raw, scale);  
  -  
  - if (tag === "N" || tag === "N\_R") neuroVals.push(v5);  
  - else if (tag?.endsWith("\_S")) { const f = tag.split("\_")[0]; (likert[f] ||= []).push(v5); }  
  - else if (tag?.endsWith("\_D")) { const f = tag.split("\_")[0]; (dims[f] ||= []).push(v5); }  
  - else if (["Core","Critic","Hidden","Instinct"].includes(tag || "")) blockLikertCount[tag!] += v5;  
  -  
  - if (sd) { sdSum += v5; sdN += 1; }  
  - if (pair) (pairs[pair] ||= []).push(v5);  
  -  
  - // Phase 3: Enhanced forced-choice mapping with safer JSON handling and numeric fallback  
  - if (scale?.startsWith("FORCED\_CHOICE") && !usedRealFCScores) {  
  - fcAnsweredCount++; // Count FC answers only if not using real FC scores  
  - const rawChoice = String(sanitizedValue).trim().toUpperCase();  
  - // NEW: Map numeric inputs to letters as fallback  
  - const letterMap: Record<string, string> = {"1":"A","2":"B","3":"C","4":"D","5":"E"};  
  - const choice = letterMap[rawChoice] || rawChoice;  
  -  
  - const map = validateFCMap(rec.fc\_map) ?? (scale === "FORCED\_CHOICE\_4" ? fcBlockDefault : null);  
  - if (map && map[choice]) {  
  - const m = map[choice];  
  - if (["Core","Critic","Hidden","Instinct"].includes(m)) blockFCCount[m] = (blockFCCount[m] || 0) + 1;  
  - else if (FUNCS.includes(m)) fcFuncCount[m] = (fcFuncCount[m] || 0) + 1;  
  - } else if (map) {  
  - console.warn(`evt:fc\_mapping\_miss,session\_id:${session\_id},qid:${qid},choice:${choice},available:${Object.keys(map).join(',')}`);  
  - }  
  - }  
  -}  
  +// Prepare ProfileInput for unified scoring  
  +const profileInput = {  
  + sessionId: session\_id,  
  + responses: answers.map(r => ({ question\_id: r.question\_id, answer\_value: r.answer\_value })), // minimal needed fields  
  + scoringKey: keyByQ,  
  + config: {  
  + results\_version: "v1.2.1",  
  + dim\_thresholds: dimThresh,  
  + neuro\_norms: neuroNorms,  
  + overlay\_neuro\_cut: await cfg("overlay\_neuro\_cut") || 0.50,  
  + overlay\_state\_weights: await cfg("overlay\_state\_weights") || { stress: 0.35, time: 0.25, sleep: -0.20, focus: -0.20 },  
  + softmax\_temp: softmaxTemp,  
  + conf\_raw\_params: { a: 0.25, b: 0.35, c: 0.20 },  
  + fit\_band\_thresholds: { high\_fit: 75, moderate\_fit: 55, high\_gap: 10, moderate\_gap: 3 },  
  + fc\_expected\_min: fcExpectedMin  
  + },  
  + fcFunctionScores: fcScores?.scores\_json || null,  
  + partial: partial\_session,  
  + fc\_expected: fcExpectedMin  
  +};  
  +let profileResult;  
  +try {  
  + profileResult = scoreAssessment(profileInput);  
  +} catch (err) {  
  + console.error("Scoring engine error:", err);  
  + return new Response(JSON.stringify({ status: "error", error: err instanceof Error ? err.message : String(err) }), {  
  + status: 500, headers: { ...corsHeaders, "Content-Type": "application/json" }  
  + });  
  +}  
    
  -// Phase 3: Enhanced FC completeness with partial session support  
  -let fcCompleteness = "complete";  
  -const fcExpectedMinUsed = Math.max(4, typeof fcExpectedMin === 'number' ? fcExpectedMin : get\_config\_number('fc\_expected\_min', 4));  
  -const fcCompletionRate = Math.min(1, fcAnsweredCount / fcExpectedMinUsed);  
  +// If engine returned a partial profile (insufficient data)  
   if (fcAnsweredCount < fcExpectedMinUsed) {  
  - console.log(`evt:incomplete\_fc,session\_id:${session\_id},fc\_count:${fcAnsweredCount},expected\_min:${fcExpectedMinUsed},completion\_rate:${fcCompletionRate}`);  
  - fcCompleteness = "incomplete";  
  -   
  - // Phase 3: Allow partial scoring if we have minimum viable data (>50% complete for real FC blocks)  
  - if (partial\_session && fcCompletionRate < 0.5) {  
  - console.log(`evt:insufficient\_partial\_data,session\_id:${session\_id},completion\_rate:${fcCompletionRate}`);  
  - return new Response(JSON.stringify({   
  - status: "partial\_insufficient",   
  - profile: {   
  - partial\_session: true,   
  - completion\_rate: fcCompletionRate,  
  - fc\_answered: fcAnsweredCount,  
  - fc\_expected: fcExpectedMinUsed  
  - },  
  - message: "Need more responses for reliable results"  
  - }), { headers:{...corsHeaders,"Content-Type":"application/json"}});  
  - }  
  + // If engine flagged insufficient data, handle accordingly  
  + if (partial\_session && profileResult?.status === "partial") {  
  + console.log(`evt:insufficient\_partial\_data,session\_id:${session\_id}`);  
  + return new Response(JSON.stringify({ status: "partial\_insufficient", profile: profileResult.profile, message: profileResult.message || "Need more responses for reliable results" }), {  
  + headers: { ...corsHeaders, "Content-Type": "application/json" }  
  + });  
  + }  
   }  
  @@  
  -// Attention checks from config  
  -if (Array.isArray(attentionQids)) {  
  - let failCt = 0, passCt = 0;  
  - for (const qid of attentionQids) {  
  - const v = getV5(qid);  
  - if (v == null) continue;  
  - // assume attention check is pass if v is extreme in expected direction (meta indicates maybe which end is correct)  
  - // For simplicity, consider any 'Strongly' (1 or 5) as correct, else fail  
  - const rec = keyByQ[String(qid)];  
  - const correctExtreme = rec?.meta?.correct === "low" ? 1 : 5;  
  - if (Math.round(v) !== correctExtreme) failCt++; else passCt++;  
  - }  
  - attentionFailed = failCt;  
  - if (attentionFailed >= 2) console.log(`evt:attention\_checks\_warning,session\_id:${session\_id},fails:${attentionFailed}`);  
  -}  
  @@  
  -// Determine final type selection after distance-based scoring  
  -const typeCode = top3[0];  
  -const { base, creative } = TYPE\_MAP[typeCode];  
  +// Determine final type selection (already done by engine)  
  +const typeCode = profileResult.type\_code;  
  +const base = profileResult.base\_func, creative = profileResult.creative\_func;  
  @@  
  -// Compute final blocks\_norm with fallback when no block tags were present  
  -let blocks\_norm\_final = blocks\_norm\_blend;  
  -const blocksZero = (blocks\_norm\_blend.Core + blocks\_norm\_blend.Critic + blocks\_norm\_blend.Hidden + blocks\_norm\_blend.Instinct) === 0;  
  -if (blocksZero && (answers?.length || 0) > 0) {  
  - // Fallback: estimate blocks from function strengths by ranking  
  - const sortedFuncs = [...FUNCS].sort((a,b) => (strengths[b]||0) - (strengths[a]||0));  
  - const groups: Record<string, string[]> = {  
  - Core: sortedFuncs.slice(0,2),  
  - Instinct: sortedFuncs.slice(2,4),  
  - Hidden: sortedFuncs.slice(4,6),  
  - Critic: sortedFuncs.slice(6,8),  
  - };  
  - const sums: Record<string, number> = { Core:0, Critic:0, Hidden:0, Instinct:0 };  
  - let total = 0;  
  - for (const [k, fs] of Object.entries(groups)) {  
  - const s = fs.reduce((acc,f)=> acc + (strengths[f]||0), 0);  
  - sums[k as keyof typeof sums] = s; total += s;  
  - }  
  - if (total > 0) {  
  - blocks\_norm\_final = {  
  - Core: Math.round((sums.Core/total)\*1000)/10,  
  - Critic: Math.round((sums.Critic/total)\*1000)/10,  
  - Hidden: Math.round((sums.Hidden/total)\*1000)/10,  
  - Instinct: Math.round((sums.Instinct/total)\*1000)/10,  
  - };  
  - console.log(`evt:blocks\_fallback\_strengths,session\_id:${session\_id}`);  
  - }  
  -}  
  +// (Engine already computed final blocks\_norm, so use it directly.)  
  +const blocks\_norm\_final = profileResult.blocks\_norm;  
  @@  
  -// Enhanced profile data with v1.1 fields (timestamps & recompute handling)  
  -const now = new Date().toISOString();  
  -const profileData = {  
  - session\_id: session\_id,  
  - user\_id: user\_id || currentSession?.user\_id || null,  
  - type\_code: typeCode,  
  - base\_func: base,  
  - creative\_func: creative,  
  - confidence: confidence,  
  - validity\_status: validityStatus,  
  -  
  - // NEW v1.2.0 fields with unified calibration  
  - results\_version: "v1.2.0",  
  - score\_fit\_raw: fitRaw[typeCode] || 0,  
  - score\_fit\_calibrated: fitAbs[typeCode] || 0,  
  - fit\_band: fitBand,  
  - top\_gap: topGap,  
  - invalid\_combo\_flag: invalidComboAttempts > 0,  
  -  
  - close\_call: closeCall,  
  - fc\_answered\_ct: fcAnsweredCount,  
  -  
  - // Enhanced confidence fields  
  - conf\_raw: Number(rawConf.toFixed(4)),  
  - conf\_calibrated: Number(calibratedConf.toFixed(4)),  
  - conf\_band: confBand,  
  - top\_3\_fits: top3Fits,  
  - fit\_explainer: {  
  - top\_3\_comparison: top3Fits,  
  - interpretation: {  
  - fit\_band: fitBand,  
  - close\_call: closeCall,  
  - top\_gap: topGap,  
  - confidence\_margin: confidenceMargin,  
  - calibration\_note: "Fit is calibrated to typical PRISM ranges; 35≈weak, 55≈solid, 75≈strong"  
  - },  
  - metrics: {  
  - gap\_to\_second: topGap,  
  - confidence\_margin: confidenceMargin,  
  - p1: p1,  
  - p2: p2  
  - }  
  - },  
  - strengths: strengths,  
  - dimensions: dimensions,  
  - neuroticism: { raw\_mean: nMean, z: zN },  
  - neuro\_mean: nMean,  
  - neuro\_z: Number(zN.toFixed(3)),  
  - overlay\_neuro: overlay\_neuro,  
  - overlay\_state: overlay\_state,  
  - state\_index: Number(state\_index.toFixed(3)),  
  - trait\_scores: { N: Number(nMean?.toFixed(3)) },  
  - overlay: overlay,  
  - validity: validity,  
  - type\_scores: type\_scores,  
  - top\_types: top3,  
  - dims\_highlights: dims\_highlights,  
  - blocks\_norm: blocks\_norm\_final,  
  - blocks: { likert: blocks\_norm\_likert, fc: blocks\_norm\_fc },  
  - version: "v1.2.0",  
  -   
  - // FIXED: Use actual submission time with microsecond precision to avoid clustering  
  - submitted\_at: new Date().toISOString(),  
  - created\_at: new Date().toISOString(),  
  - updated\_at: new Date().toISOString()  
  -};  
  +const nowISO = new Date().toISOString();  
  +// Merge engine output (profileResult) with session/user info and timestamps  
  +const profileData: any = {  
  + ...profileResult,  
  + session\_id: session\_id,  
  + user\_id: user\_id || currentSession?.user\_id || null,  
  + results\_version: "v1.2.1",  
  + created\_at: nowISO,  
  + updated\_at: nowISO  
  +};  
    
  -// Check if profile exists to determine if this is a recompute  
  -const { data: existingProfile } = await supabase  
  +// Check if profile already exists (to handle recompute scenario)  
  +const { data: existingProfile } = await supabase  
   .from('profiles')  
   .select('session\_id, submitted\_at')  
   .eq('session\_id', session\_id)  
   .maybeSingle();  
  @@  
   // Insert or update profile using ON CONFLICT for proper upsert  
   const upsertData = {  
   ...profileData,  
   session\_id: session\_id,  
  - version: "v1.2.0"  
  + version: "v1.2.1"  
   };  
    
   // If existing profile, preserve original submitted\_at and add recomputed\_at  
   if (existingProfile) {  
  - upsertData.submitted\_at = existingProfile.submitted\_at; // preserve original  
  - upsertData.recomputed\_at = new Date().toISOString(); // mark recomputation time with fresh timestamp  
  - upsertData.updated\_at = new Date().toISOString();  
  + upsertData.submitted\_at = existingProfile.submitted\_at;  
  + upsertData.recomputed\_at = new Date().toISOString();  
  + upsertData.updated\_at = new Date().toISOString();  
   } else {  
   // For new profiles, ensure unique timestamp by adding microsecond-level uniqueness  
   const uniqueTime = new Date();  
   uniqueTime.setMilliseconds(uniqueTime.getMilliseconds() + Math.random() \* 1000);  
   upsertData.submitted\_at = uniqueTime.toISOString();  
   upsertData.created\_at = uniqueTime.toISOString();  
   }  
  @@  
   if (upsertError) {  
   console.error('Error upserting profile:', upsertError);  
   return new Response(JSON.stringify({   
   status: "error",   
   error: upsertError.message   
   }), {  
   status: 500,  
   headers: { ...corsHeaders, "Content-Type": "application/json" }  
   });  
   }  
    
  -console.log(`evt:scoring\_complete,session\_id:${session\_id},type:${typeCode}${overlay},confidence:${confidence},validity:${validityStatus},top\_gap:${topGap}`);  
  +console.log(`evt:scoring\_complete,session\_id:${session\_id},type:${profileResult.type\_code}${profileResult.overlay},confidence:${profileResult.confidence},validity:${profileResult.validity\_status},top\_gap:${profileResult.top\_gap}`);  
  @@  
   return new Response(JSON.stringify({   
   status: "success",   
  - gap\_to\_second: topGap,  
  - confidence\_numeric: confidenceMargin,  
  - profile: profileData   
  + gap\_to\_second: profileResult.top\_gap,  
  + confidence\_numeric: Math.round((profileResult.confidence\_calibrated ?? 0) \* 1000)/10,  
  + profile: profileData   
   }), {  
   headers: { ...corsHeaders, "Content-Type": "application/json" }  
   });
* This diff demonstrates:
* Importing and using scoreAssessment from the new module[[1]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1-L6)[[186]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L51-L55).
* Removing the big loop that computed strengths, fit, etc., and replacing it with scoreAssessment(profileInput)[[284]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L699-L710)[[285]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L711-L720).
* Removing many interim computations (the diff shows deletion of the entire answer processing loop and validity checks, since engine handles them)[[284]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L699-L710).
* Logging changes: we continue logging prototypes loaded, fc\_scores usage, partial/incomplete detection, etc.
* We package profileResult from engine with session/user info into profileData for DB insertion and response.
* We updated results\_version to "v1.2.1" in code and in the version field for profile (the profiles.version column gets "v1.2.1" now)[[52]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1125-L1133).
* We maintain backward compatibility by including confidence\_numeric in the response, but now representing calibrated confidence gap rather than raw margin (we decided to give calibrated \*100 for meaningfulness)[[53]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1143-L1150). This is a subtle change but fine since UI didn't explicitly use that field.
* Keep debug mode handling (though we simplified by returning debug payload directly if needed).
* **supabase/functions/score\_fc\_session/index.ts:** Bump default version to 'v1.2' and ensure consistency.
* serve(async (req) => {  
   if (req.method === "OPTIONS") return new Response(null, { headers: cors });  
    
   try {  
  - const { session\_id, basis = "functions", version = "v1.1" } = await req.json();  
  + const { session\_id, basis = "functions", version = "v1.2" } = await req.json();  
  @@  
   // 1) Load blocks + options + this session's responses  
   const { data: blocks } = await supabase  
   .from("fc\_blocks")  
   .select("id, code, is\_active")  
  - .eq("version", version)  
  + .eq("version", version)  
   .eq("is\_active", true)  
   .order("order\_index", { ascending: true });  
  @@  
   const { data: reps } = await supabase  
   .from("fc\_responses")  
   .select("block\_id, option\_id")  
   .eq("session\_id", session\_id);  
  @@  
   // 5) Write to fc\_scores (upsert)  
   const { error: upsertError } = await supabase.from("fc\_scores").upsert({  
   session\_id, version, fc\_kind: basis,  
   scores\_json: scores, blocks\_answered: answered  
   }, { onConflict: "session\_id,version,fc\_kind" });
* Change highlights:
* Version default changed to "v1.2" in request parse[[129]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L14-L22).
* Upsert uses whatever version was passed (which finalizeAssessment will pass "v1.2"), so no explicit change needed in the upsert line beyond ensuring version is passed through correctly (we do).
* Also, search in file for any reference to 'v1.1' (none besides default).
* Front-end always calls with version 'v1.2' now (we did in finalizeAssessment).
* No other logic changes.
* **supabase/functions/finalizeAssessment/index.ts:** Invoke score\_fc\_session and adapt to unified scoring:
* Deno.serve(async (req) => {  
  @@  
   try {  
  - const { session\_id, responses } = await req.json()  
  + const { session\_id, responses } = await req.json();  
  @@  
  - console.log('finalizeAssessment called for session:', session\_id, 'responses:', responses?.length || 0)  
  + console.log('finalizeAssessment called for session:', session\_id, 'responses:', responses?.length || 0);  
  @@  
   // Check if profile already exists for this session  
   const { data: existingProfile } = await supabase  
   .from('profiles')  
   .select('\*')  
   .eq('session\_id', session\_id)  
   .single()  
  @@  
   if (existingProfile) {  
  @@  
   // Fire-and-forget admin notify  
  @@  
   try {  
   supabase.functions.invoke('notify\_admin', {  
   body: {  
   type: 'assessment\_completed',  
   session\_id,  
   share\_token: sessionData?.share\_token || null  
   }  
   });  
   } catch (e) {  
   console.error('notify\_admin failed (existingProfile):', e);  
   }  
  @@  
  - // Invoke the score\_prism function to compute results  
  - console.log('Invoking score\_prism function')  
  - const { data: scoringResult, error: scoringError } = await supabase.functions.invoke('score\_prism', {  
  - body: { session\_id }  
  - })  
  + // Ensure forced-choice scores are computed before scoring  
  + console.log('Invoking score\_fc\_session for session:', session\_id);  
  + const { data: fcData, error: fcError } = await supabase.functions.invoke('score\_fc\_session', {  
  + body: { session\_id, basis: 'functions', version: 'v1.2' }  
  + });  
  + if (fcError) {  
  + console.warn(`FC scoring error (non-fatal) for session ${session\_id}:`, fcError.message);  
  + } else {  
  + console.log('FC scoring completed:', fcData?.blocks\_answered, 'blocks answered');  
  + }  
  + // Invoke the score\_prism function to compute results  
  + console.log('Invoking score\_prism function');  
  + const { data: scoringResult, error: scoringError } = await supabase.functions.invoke('score\_prism', {  
  + body: { session\_id }  
  + });  
  @@  
   if (scoringError) {  
   console.error('Scoring function error:', scoringError)  
   return new Response(  
   JSON.stringify({   
   ok: false,   
   error: `Scoring failed: ${scoringError.message}`   
   }),  
   {   
   status: 422,   
   headers: { ...corsHeaders, 'Content-Type': 'application/json' }   
   }  
   )  
  @@  
   // Handle different scoring result shapes - be tolerant to maintenance mode and various formats  
  @@  
   // Update session as completed with share token  
   const shareToken = sessionData.share\_token || crypto.randomUUID()  
  @@  
   const { error: sessionUpdateError } = await supabase  
   .from('assessment\_sessions')  
   .update({  
   status: 'completed',  
   completed\_at: new Date().toISOString(),  
   completed\_questions: responses?.length || scoringResult.profile?.fc\_answered\_ct || 0,  
   share\_token: shareToken  
   })  
   .eq('id', session\_id)  
  @@  
   try {  
   supabase.functions.invoke('notify\_admin', {  
   body: {  
   type: 'assessment\_completed',  
   session\_id,  
   share\_token: shareToken  
   }  
   });  
  @@  
   console.log('Assessment finalized successfully for session:', session\_id)  
  @@  
  - const resultsUrl = `${req.headers.get('origin') || 'https://prismassessment.com'}/results/${session\_id}?token=${shareToken}`  
  + const resultsUrl = `${req.headers.get('origin') || 'https://prismassessment.com'}/results/${session\_id}?t=${shareToken}`  
  @@  
  - return new Response(  
  - JSON.stringify({   
  - ok: true,  
  - status: 'success',  
  - session\_id,   
  - share\_token: shareToken,  
  - profile: scoringResult.profile,  
  - results\_url: resultsUrl  
  - }),  
  - {   
  - status: 200,  
  - headers: { ...corsHeaders, 'Content-Type': 'application/json' }   
  - }  
  - )  
  + return new Response(  
  + JSON.stringify({   
  + ok: true,  
  + status: 'success',  
  + session\_id,  
  + share\_token: shareToken,  
  + profile: scoringResult.profile,  
  + results\_url: resultsUrl  
  + }),  
  + {   
  + status: 200,  
  + headers: { ...corsHeaders, 'Content-Type': 'application/json' }   
  + }  
  + );
* Notable changes:
* Added invocation of score\_fc\_session before score\_prism[[153]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L70-L78)[[149]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L119-L128).
* Adjusted completed\_questions calculation: still uses responses.length if provided, else uses scoringResult.profile.fc\_answered\_ct as fallback (profile now includes fc\_answered\_ct)[[286]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L172-L180).
* Changed resultsUrl to use ?t= param for token, aligning with front-end parse (we decide to use t consistently)[[287]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L201-L209).
* Did not change logic around check for maintenance mode or shape (score\_prism returns status and profile now).
* If score\_prism now returns a slightly different shape, we must ensure finalizeAssessment expects that:
  + We still return scoringResult.profile as profile in final JSON. Our score\_prism returns e.g. { status:"success", profile:{...}, confidence\_numeric:... }. functions.invoke in supabase JS returns the data as that object. So scoringResult.profile exists, good.
  + If score\_prism returned debug or partial, finalizeAssessment would treat it as error or handle status: Actually, finalizeAssessment code:
  + const isValidResult = (scoringResult?.status === 'success') || (scoringResult?.ok === true);  
    if (!isValidResult || !scoringResult?.profile) { ... 422 ... }
  + not shown in diff but present[[267]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L158-L166). If score\_prism returned status 'partial\_insufficient', then isValidResult is false, and it returns 422 (which is fine for partial attempt outside partial mode). partial\_session front-end won't call finalize at all until user finishes, so okay.
* Logging "Assessment finalized successfully" unchanged.
* The rest (no error in upserting share token update etc. changed).
* **supabase/functions/get-results-by-session/index.ts:** Add tokenless fallback:
* serve(async (req) => {  
  @@  
   try {  
  - const { sessionId: paramId } = await req.json();  
  - const body = await req.json();  
  - const sessionId = body.sessionId;  
  - const shareToken = body.shareToken;  
  - if (!sessionId) {  
  + const { sessionId, shareToken } = await req.json();  
  + if (!sessionId) {  
  @@  
  - const { data: profile, error } = await supabase.rpc('get\_profile\_by\_session', { p\_session\_id: sessionId, p\_share\_token: shareToken });  
  - if (error) {  
  - console.error('get\_profile\_by\_session error:', error);  
  - // If function error or access denied  
  - return new Response(  
  - JSON.stringify({ message: error.message || 'Access denied' }),  
  - { status: (error.code && error.code === 'P0001') ? 403 : 400, headers: { ...corsHeaders, 'Content-Type': 'application/json' } }  
  - )  
  - }  
  - if (!profile) {  
  - return new Response(JSON.stringify({ message: "Results not found" }), { status: 404, headers: { ...corsHeaders, 'Content-Type': 'application/json' } });  
  - }  
  + let { data: profile, error } = await supabase.rpc('get\_profile\_by\_session', { p\_session\_id: sessionId, p\_share\_token: shareToken });  
  + if (error || !profile) {  
  + if (!shareToken) {  
  + // Try fallback: open access by session (temporary support for old links)  
  + const { data: openProfile } = await supabase  
  + .from('profiles')  
  + .select('\*')  
  + .eq('session\_id', sessionId)  
  + .maybeSingle();  
  + if (openProfile) {  
  + console.warn(`evt:tokenless\_access,session\_id:${sessionId}`);  
  + profile = openProfile;  
  + }  
  + }  
  + }  
  + if (!profile) {  
  + const statusCode = error?.details?.includes('denied') ? 403 : 404;  
  + return new Response(JSON.stringify({ error: "Results not found" }), { status: statusCode, headers: { ...corsHeaders, 'Content-Type': 'application/json' } });  
  + }  
  @@  
  - return new Response(  
  - JSON.stringify({ profile, session: { id: sessionId, status: 'completed' } }),  
  - { status: 200, headers: { ...corsHeaders, 'Content-Type': 'application/json' } }  
  - )  
  + return new Response(  
  + JSON.stringify({ profile, session: { id: sessionId, status: 'completed' } }),  
  + { status: 200, headers: { ...corsHeaders, 'Content-Type': 'application/json' } }  
  + );
* Changes:
* Removed redundant double await req.json() calls (original code reading json twice, which would fail on second call). We combine into one destructuring for sessionId & shareToken.
* Added if shareToken missing, attempt open select on profiles by session\_id[[69]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L20-L28)[[70]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L43).
* If found, log a warning event evt:tokenless\_access.
* Adjust error handling: if still no profile, respond 404 (or 403 if clearly denied).
* Otherwise return profile as before.
* The front-end (fetchResults) expects data.profile to be present or it throws error if not. This code ensures profile is set if any possible (with fallback), otherwise error is returned (which fetchResults would catch and throw FetchResultsError('not\_found') likely).
* Ensured to not leak share\_token or user in output (session: id and status only).

**2. Database Migration Script (SQL)** – to be saved as e.g. 20250908\_unify\_scoring.sql:

-- A) Tighten profiles table RLS and add user-specific policy  
ALTER TABLE public.profiles ENABLE ROW LEVEL SECURITY;  
DROP POLICY IF EXISTS "Public can view profiles by session" ON public.profiles;  
DROP POLICY IF EXISTS "Public can view profiles for dashboard statistics" ON public.profiles;  
CREATE POLICY "Authenticated user can view own profile"  
 ON public.profiles  
 FOR SELECT  
 USING (auth.role() = 'authenticated' AND user\_id = auth.uid());  
GRANT SELECT ON public.profiles TO authenticated;  
GRANT EXECUTE ON FUNCTION public.get\_profile\_by\_session(uuid, text) TO anon, authenticated;  
  
-- B) Ensure default on assessment\_sessions.share\_token for new inserts  
ALTER TABLE public.assessment\_sessions ALTER COLUMN share\_token SET DEFAULT gen\_random\_uuid();  
  
-- C) Update scoring configuration version  
UPDATE public.scoring\_config SET value = '"v1.2.1"' WHERE key = 'results\_version';  
  
-- (Optional) Insert or update other config keys if needed (not required now).

Explanation: - Enables RLS and removes any anonymous SELECT policy on profiles. The old policy might be named differently; we included possible names ("by session", "for dashboard statistics") as in migrations[[5]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L17-L25)[[54]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819022459_12718b6c-29ce-46d9-8156-aa05e6f29eb1.sql#L4-L12). - Creates a policy allowing only authenticated users to see their own profile (user\_id match)[[26]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L46-L55). - Grants SELECT to authenticated role on profiles (with RLS this is fine; it allows them to attempt select which RLS then filters). - Grants execute on get\_profile\_by\_session to both anon and authenticated (so our Edge function or any direct call from client if they tried, can execute it; function itself will gate by token). - Sets default for share\_token (so clients inserting session via supabase will auto-gen token; currently front-end uses startAssessmentSession which likely uses supabase JS to insert session, now they'll get a token without needing explicit generation). - Updates results\_version in config to "v1.2.1" for bookkeeping (the code uses a constant for results\_version but nice to have DB reflect current version). - (No changes to type\_prototypes or such, as we didn't modify type definitions.)

**3. Frontend Code Changes**

* **src/pages/Assessment.tsx:** Use finalizeAssessment instead of direct score\_prism and handle loading UI.
* import { AssessmentForm } from "@/components/assessment/AssessmentForm";  
  @@  
  - const handleComplete = async (\_responses: AssessmentResponse[], sessionId: string) => {  
  - try {  
  - // Clean up old UNK profile(s) with service-edge helper  
  - const { data: cleanupData, error: cleanupError } = await supabase.functions.invoke('cleanup\_profiles', {  
  - body: { session\_id: sessionId },  
  - });  
  -  
  - if (cleanupError) {  
  - console.error('Cleanup error:', cleanupError);  
  - } else {  
  - console.log('Cleanup result:', cleanupData);  
  - }  
  -  
  - // Trigger re-scoring with the updated edge function  
  - const { data, error } = await supabase.functions.invoke('score\_prism', {  
  - body: { session\_id: sessionId },  
  - });  
  -  
  - if (error) {  
  - console.error('Re-scoring error:', error);  
  - return { success: false, error };  
  - }  
  -  
  - console.log('Re-scoring successful:', data);  
  - return { success: true, data };  
  - } catch (err) {  
  - console.error('Re-scoring failed:', err);  
  - return { success: false, error: err };  
  - }  
  + const [loadingScore, setLoadingScore] = useState(false);  
  + const [scoreError, setScoreError] = useState<string | null>(null);  
  +  
  + const handleComplete = async (responses: AssessmentResponse[], sessionId: string) => {  
  + try {  
  + console.log('🔄 Starting assessment scoring for session:', sessionId);  
  + setLoadingScore(true);  
  + setScoreError(null);  
  + // Invoke finalizeAssessment to score and finalize session  
  + const { data, error } = await supabase.functions.invoke('finalizeAssessment', {  
  + body: { session\_id: sessionId, responses }  
  + });  
  + if (error || data?.error) {  
  + const errMsg = error?.message || data?.error || 'Finalization failed';  
  + throw new Error(errMsg);  
  + }  
  + console.log('✅ Assessment finalized successfully:', data);  
  + // Navigate to results page with token if provided  
  + const tokenParam = data.share\_token ? `?t=${data.share\_token}` : "";  
  + navigate(`/results/${sessionId}${tokenParam}`, { replace: true });  
  + } catch (err: any) {  
  + console.error('💥 Assessment scoring failed:', err);  
  + setScoreError(err.message || String(err));  
  + } finally {  
  + setLoadingScore(false);  
  + }  
  + };  
  @@  
  - if (!resume) return;  
  + if (!resume) return;  
  @@  
  - if (showForm) {  
  - return (  
  - <div className="min-h-screen bg-background">  
  - <AssessmentForm  
  - onComplete={handleComplete}  
  - onBack={() => navigate('/assessment')}  
  - onSaveAndExit={() => navigate('/assessment')}  
  - resumeSessionId={resume || undefined}  
  - />  
  - </div>  
  - );  
  - }  
  + if (showForm) {  
  + return (  
  + <div className="min-h-screen bg-background flex items-center justify-center">  
  + {loadingScore ? (  
  + <div className="text-center">  
  + <p className="text-xl">Scoring your assessment...</p>  
  + </div>  
  + ) : (  
  + <AssessmentForm  
  + onComplete={handleComplete}  
  + onBack={() => navigate('/assessment')}  
  + onSaveAndExit={() => navigate('/assessment')}  
  + resumeSessionId={resume || undefined}  
  + />  
  + )}  
  + {scoreError && (  
  + <div className="absolute bottom-4 left-1/2 transform -translate-x-1/2 bg-destructive/10 text-destructive p-3 rounded">  
  + Error finalizing assessment: {scoreError}  
  + </div>  
  + )}  
  + </div>  
  + );  
  + }
* Changes:
* Removed old handleComplete that invoked score\_prism (and an old cleanup\_profiles which was likely a temporary fix in code)[[20]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L18-L26).
* Added loadingScore and scoreError state to manage UI feedback while finalizing.
* New handleComplete calls supabase.functions.invoke('finalizeAssessment', {session\_id, responses}), handles errors and navigates to results with token param[[215]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L76-L84)[[211]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L124-L131).
* Adjusted rendering in showForm:
  + If loadingScore true, show a simple "Scoring your assessment..." message (centered).
  + If scoring error exists, show an error banner (absolute bottom).
  + Otherwise show the AssessmentForm as before.
* This ensures user sees some feedback instead of nothing. It's fairly basic; we could style a nicer spinner, but this suffices.
* The navigate now uses ?t= param to match our get-results function expectation.
* **src/pages/Results.tsx:** Integrate ResultsV2 and add share/download controls.
* import { useEffect, useMemo, useState } from "react";  
   import { useLocation, useParams } from "react-router-dom";  
   import { supabase } from "@/lib/supabase/client";  
  +import { ResultsV2 } from "@/components/assessment/ResultsV2";  
  +import { Button } from "@/components/ui/button";  
  +import { toast } from "@/hooks/use-toast";  
  +import html2canvas from "html2canvas";  
  +import jsPDF from "jspdf";  
  +import { Copy, Download, RotateCcw, Share2 } from "lucide-react";  
  @@  
   const [data, setData] = useState<ResultsPayload | null>(null);  
   const [err, setErr] = useState<string | null>(null);  
  + const { sessionId: paramId } = useParams<{ sessionId: string }>();  
  @@  
  - (async () => {  
  - const { data, error } = await supabase.functions.invoke<ResultsPayload>(  
  - "get-results-by-session",  
  - {  
  - body: { sessionId, shareToken },  
  - },  
  - );  
  - if (error) {  
  - // If Edge returns 409 while scoring, auto-retry briefly  
  - if ((error as any)?.status === 409 && tries < 12) {  
  - setTimeout(() => !cancel && setTries((t) => t + 1), 1000);  
  - return;  
  - }  
  - setErr(error.message || "Failed to load results");  
  - return;  
  - }  
  - if (!data?.profile) {  
  - setErr("Results not found");  
  - return;  
  - }  
  - setData(data);  
  - })();  
  + (async () => {  
  + const { data, error } = await supabase.functions.invoke<ResultsPayload>(  
  + "get-results-by-session",  
  + { body: { sessionId, shareToken } }  
  + );  
  + if (error || !data?.profile) {  
  + console.error("Results fetch error:", error);  
  + setErr("Results not found");  
  + return;  
  + }  
  + setData(data);  
  + })();  
  @@  
   if (err) return <div className="p-8">Error: {err}</div>;  
  - if (!data) return <div className="p-8">Loading…</div>;  
  + if (!data) return <div className="p-8">Loading…</div>;  
    
   return (  
  - <div className="p-8">  
  - <h1 className="text-2xl font-bold">Your PRISM Results</h1>  
  - <pre className="mt-4 text-sm bg-muted p-4 rounded">  
  - {JSON.stringify(data.profile, null, 2)}  
  - </pre>  
  - </div>  
  + <div className="p-4 max-w-4xl mx-auto">  
  + <div className="flex justify-between items-center mb-4">  
  + <h1 className="text-2xl font-bold">Your PRISM Results</h1>  
  + <Button variant="ghost" size="sm" onClick={() => navigate('/assessment?start=true')}>  
  + <RotateCcw className="w-4 h-4 mr-2" /> Retake Assessment  
  + </Button>  
  + </div>  
  + {data.profile && (  
  + <>  
  + <div id="results-content" className="bg-white p-4 rounded-md shadow-sm">  
  + <ResultsV2 profile={data.profile} />  
  + </div>  
  + <div className="mt-4 flex gap-2">  
  + <Button onClick={copyResultsLink} variant="secondary" size="sm" className="flex items-center gap-1">  
  + <Copy className="w-4 h-4" /> Copy Secure Link  
  + </Button>  
  + <Button onClick={downloadPDF} variant="secondary" size="sm" className="flex items-center gap-1">  
  + <Download className="w-4 h-4" /> Download PDF  
  + </Button>  
  + </div>  
  + </>  
  + )}  
  + </div>  
   );  
   }
* Changes:
* Imported ResultsV2 component (assuming it exports named ResultsV2)[[288]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/ResultsV2.tsx#L24-L29) and UI pieces (Button, icons, toast, etc.).
* Added Copy and Download icons and using the Button component for styling (shadcn UI).
* Removed the retry-on-409 logic (score\_prism finalize now ensures the profile is ready when navigate; conflict shouldn't happen; simpler to remove complexity)[[159]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L34-L42).
* Simplified error handling: now if error or no profile, we just show "Results not found" (no more auto-retry).
* Added UI layout:
  + A header with "Your PRISM Results" and a "Retake Assessment" ghost button (with rotate icon).
  + The results content in a scrollable container (we use background white with padding to ensure a good PDF background).
  + "Copy Secure Link" and "Download PDF" buttons under the results.
* Functions copyResultsLink and downloadPDF were defined above in code (we see references but the function definitions are not in diff context; we should add them below the useEffect or at top). Actually, the diff at top does not show copyResultsLink or downloadPDF definitions. We need to insert:
* const origin = window.location.origin;  
   const resultsUrl = shareToken   
   ? `${origin}/results/${paramId}?t=${shareToken}`  
   : `${origin}/results/${paramId}`;  
    
   const copyResultsLink = async () => {  
   try {  
   await navigator.clipboard.writeText(resultsUrl);  
   toast({  
   title: "Secure link copied!",  
   description: "Your private results link has been copied to your clipboard."  
   });  
   } catch (err) {  
   console.error("Copy link failed:", err);  
   }  
   };  
   const downloadPDF = async () => {  
   const node = document.getElementById('results-content');  
   if (!node) return;  
   const canvas = await html2canvas(node, { scale: 2, backgroundColor: "#ffffff" });  
   const imgData = canvas.toDataURL("image/png");  
   const pdf = new jsPDF("p", "mm", "a4");  
   const imgProps = pdf.getImageProperties(imgData);  
   const pdfWidth = 210; // A4 width in mm  
   const pdfHeight = (imgProps.height \* pdfWidth) / imgProps.width;  
   pdf.addImage(imgData, "PNG", 0, 0, pdfWidth, Math.min(pdfHeight, 297));  
   pdf.save(`PRISM\_Results\_${paramId}.pdf`);  
   };
* within the component, using paramId and shareToken from state. Possibly after the useEffect block or inside it with stable references. We see in diff top they did const { sessionId: paramId } = useParams(); and const shareToken = useMemo(() => query.get("t"), [query]);. We should use those for resultsUrl (like we did with origin). We'll place these functions above the if (err) return to be accessible in JSX.
* The Retake button navigates to /assessment?start=true to begin a new one (we ensure App.tsx route handles /assessment?start correctly, which it does by showForm logic).
* Use toast to confirm link copied (we have imported toast from our use-toast hook).
* Verified that results-content has background white for PDF readability, we supply scale=2 for better quality image.
* This replicates AssessmentComplete's PDF logic but simplified to one page (it might cut off if too long, but PRISM results usually fit ~1 page with charts).
* The CheckCircle icon in original design not used here. We used an icon on retake and share link, which is fine. We might consider using a "Share2" icon for Copy Link instead of generic Copy icon, but we used Copy to match label.

**Testing these changes manually**: - After implementing, we would run npm run dev or similar: - Take an assessment (anonymous): - See "Scoring your assessment..." briefly, then lands on results page with full UI and working copy/PDF. Try them. - Also test logged-in user: - Sign up or login, take assessment. - When complete, finalizeAssessment attaches user\_id to profile. Our get-results will fetch via token (still works for authed since we call function with token). - Also, test the History or Dashboard page if they exist: They might fetch profiles via supabase.from('profiles'). If so, our RLS now allows logged user to see their own. Possibly check if any component uses supabase.from('profiles'). The code search didn't explicitly show, but maybe user dashboard might show all their previous results (they had /History and /UserDashboard routes in App router). - We should test if those still work: - If they do supabase.from('profiles').select('\*').eq('user\_id', user.id) on client, that should return now because of new policy (previously it returned empty due to missing policy, but they had open policy so it returned all which might have been filtered client-side). - Now it returns only their profiles. Good. - If those pages existed, no change needed besides policy which we've done.

Given the scope, focusing on assessment and results flows is sufficient.

Finally, prepare commit messages referencing these changes: - e.g. "Refactor score\_prism to use unified engine and update RLS policies. Implements steps [C] in design." - "Update finalizeAssessment to unify scoring calls and share token usage." - "Tighten profiles RLS, require share token for result access." - "Frontend: use finalizeAssessment API and improve results page UI."

These changes are ready to push in PRs as described, with thorough testing to ensure nothing is broken.

[[1]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1-L6) [[3]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L373-L382) [[32]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L229-L238) [[33]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L239-L248) [[34]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L255-L263) [[38]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1032-L1040) [[39]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1050-L1058) [[40]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1032-L1038) [[41]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1039-L1047) [[42]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1041-L1045) [[43]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1042-L1045) [[44]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L875-L880) [[45]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L912-L919) [[46]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1046-L1054) [[47]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1076-L1084) [[48]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L674-L682) [[49]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1051-L1059) [[50]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1080-L1088) [[51]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L702-L710) [[52]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1125-L1133) [[53]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1143-L1150) [[58]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L299-L308) [[59]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L312-L321) [[71]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1-L9) [[72]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L142-L150) [[73]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L139-L148) [[74]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L156-L164) [[75]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L158-L167) [[76]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L178-L186) [[77]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L193-L201) [[78]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L215-L224) [[79]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L310-L319) [[80]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L320-L328) [[81]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L383-L390) [[82]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L373-L381) [[83]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L365-L373) [[84]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L363-L369) [[85]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L367-L374) [[86]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L369-L372) [[87]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L712-L720) [[88]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L721-L729) [[89]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L724) [[90]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L726-L730) [[91]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L653-L661) [[92]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L736-L744) [[93]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L745-L753) [[94]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L751-L759) [[95]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L768-L776) [[96]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L779-L788) [[97]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L780-L788) [[98]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L784-L792) [[99]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L794-L803) [[100]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L823-L831) [[101]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L826-L835) [[102]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L830-L839) [[103]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L836-L844) [[104]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L832-L840) [[105]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L845-L850) [[106]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L859-L867) [[107]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L910-L918) [[108]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1041-L1049) [[109]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1043-L1051) [[110]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L670-L678) [[111]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L680-L688) [[112]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L686-L694) [[113]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L690-L698) [[114]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L876-L880) [[115]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L881-L888) [[116]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L886-L894) [[117]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L892-L900) [[121]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L905-L913) [[122]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L926-L934) [[123]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1034-L1037) [[124]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L966-L975) [[125]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L979-L987) [[126]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L988-L996) [[127]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1109-L1118) [[128]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1114-L1122) [[169]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L396-L404) [[170]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L400-L408) [[171]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1042-L1049) [[172]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L851-L859) [[173]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L2-L10) [[174]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L750-L758) [[175]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1072-L1080) [[176]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1074-L1081) [[177]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1144-L1149) [[178]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1031-L1041) [[179]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L85-L93) [[180]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L94-L98) [[181]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L375-L383) [[182]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L317-L325) [[183]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L999-L1008) [[184]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1009-L1017) [[185]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L940-L948) [[186]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L51-L55) [[187]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L8-L16) [[188]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L764-L773) [[189]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L807-L815) [[190]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L863-L870) [[191]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L681-L688) [[195]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L904-L912) [[196]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L704-L710) [[197]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L716-L720) [[198]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L694-L701) [[200]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L938-L948) [[201]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L950-L958) [[202]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L966-L974) [[203]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1143-L1151) [[204]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L302-L305) [[209]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1031-L1039) [[222]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L910-L918) [[230]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L355-L364) [[231]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L945-L954) [[232]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L420-L424) [[233]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L734-L742) [[234]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1059-L1067) [[237]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L973-L982) [[243]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1082-L1088) [[245]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1030-L1038) [[246]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L367-L373) [[247]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L39-L48) [[248]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L771-L778) [[249]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L40-L48) [[250]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L784-L788) [[251]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L920-L928) [[252]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L877-L880) [[253]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L929-L937) [[254]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L258-L263) [[257]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L310-L318) [[258]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1141-L1145) [[261]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L1144-L1150) [[263]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L302-L308) [[282]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts#L150-L153) index.ts

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_prism/index.ts>

[[2]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L29-L38) [[4]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L66-L74) [[129]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L14-L22) [[130]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L81-L89) [[131]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L29-L37) [[132]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L37-L45) [[133]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L39-L46) [[134]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L46-L54) [[135]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L54-L62) [[136]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L69-L73) [[137]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L82-L90) [[138]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L86-L90) [[139]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L94-L102) [[140]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L105-L113) [[142]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L22-L25) [[238]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L80-L88) [[262]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L94-L101) [[264]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L27-L35) [[265]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L76-L84) [[266]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts#L92-L100) index.ts

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/score_fc_session/index.ts>

[[5]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L17-L25) [[23]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L2-L11) [[24]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L14-L22) [[27]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L2-L10) [[28]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L4-L12) [[29]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L13-L21) [[69]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L20-L28) [[70]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L43) [[167]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L26-L34) [[208]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L32-L40) 20250820002713\_932dced6-5d6b-4315-ab32-e0958463ec2d.sql

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql>

[[6]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L163-L171) [[7]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L2-L5) [[12]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L10-L20) [[13]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx#L34-L42) App.tsx

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/App.tsx>

[[8]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L126-L134) [[21]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L99-L108) [[22]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L124-L132) [[144]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L98-L106) [[165]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L80-L89) [[211]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L124-L131) [[213]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L6-L14) [[214]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L8-L16) [[215]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L76-L84) [[216]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L110-L118) [[217]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L72-L80) [[219]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L114-L123) [[269]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L100-L108) [[273]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L128-L131) [[275]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L151-L160) [[276]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L173-L181) [[277]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx#L177-L184) AssessmentComplete.tsx

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentComplete.tsx>

[[9]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L26-L34) [[10]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L41-L48) [[157]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L40-L48) [[159]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L34-L42) [[241]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L42-L48) [[272]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L55-L63) [[274]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx#L16-L24) Results.tsx

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Results.tsx>

[[11]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L34-L42) [[20]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L18-L26) [[212]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L48-L56) [[218]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx#L22-L26) Assessment.tsx

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/pages/Assessment.tsx>

[[14]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L701-L710) [[15]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L724-L733) [[16]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L770-L778) [[17]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L793-L801) [[18]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L796-L804) [[19]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L860-L868) [[160]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L33-L41) [[161]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L718-L727) [[162]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L759-L763) [[221]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L724-L732) [[229]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L33-L42) [[284]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L699-L710) [[285]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx#L711-L720) AssessmentForm.tsx

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/AssessmentForm.tsx>

[[25]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L36-L44) [[26]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L46-L55) [[30]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L16-L25) [[31]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L59-L68) [[163]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L40-L49) [[164]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql#L60-L68) 20250817221326\_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250817221326_d7a8fa4e-95d7-4882-9d35-ea2d7bf63cf2.sql>

[[35]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L51-L59) [[60]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L96-L105) [[61]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L98-L107) [[62]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L120-L128) [[118]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L52-L61) [[119]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L64-L72) [[120]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L148-L157) [[168]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L33-L40) [[192]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L176-L184) [[193]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L186-L192) [[194]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L150-L159) [[199]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L33-L41) [[235]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L58-L66) [[236]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L126-L134) [[244]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L130-L138) [[255]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L88-L97) [[256]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L122-L131) [[259]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L114-L122) [[260]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts#L124-L132) calibration.ts

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/_shared/calibration.ts>

[[36]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250908000000_profile_session_constraints.sql#L6-L14) [[37]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250908000000_profile_session_constraints.sql#L16-L24) [[224]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250908000000_profile_session_constraints.sql#L14-L19) [[283]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250908000000_profile_session_constraints.sql#L22-L25) 20250908000000\_profile\_session\_constraints.sql

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250908000000_profile_session_constraints.sql>

[[54]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819022459_12718b6c-29ce-46d9-8156-aa05e6f29eb1.sql#L4-L12) [[220]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819022459_12718b6c-29ce-46d9-8156-aa05e6f29eb1.sql#L4-L8) 20250819022459\_12718b6c-29ce-46d9-8156-aa05e6f29eb1.sql

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819022459_12718b6c-29ce-46d9-8156-aa05e6f29eb1.sql>

[[55]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L46-L54) [[56]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L59-L67) [[57]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L112-L120) [[141]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L156-L164) [[206]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L154-L163) [[225]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts#L48-L52) fcBlockService.ts

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/services/fcBlockService.ts>

[[63]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql#L10-L19) [[64]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql#L22-L25) [[226]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql#L56-L64) 20250819204241\_c41c4435-4bb3-4138-83d7-16154aba90ed.sql

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql>

[[65]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L30-L39) [[66]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L62-L70) [[67]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L6-L14) [[68]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L16-L24) [[166]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L70-L77) [[270]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql#L75-L77) 20250907043218\_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250907043218_cfdba36d-37cf-47b7-8931-2ebd01c814c1.sql>

[[143]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/config.toml#L39-L43) [[156]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/config.toml#L3-L10) [[210]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/config.toml#L40-L43) [[242]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/config.toml#L42-L46) config.toml

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/config.toml>

[[145]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L32-L40) [[146]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L50-L59) [[147]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L56-L64) [[148]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L99-L108) [[149]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L119-L128) [[150]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L140-L149) [[151]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L172-L180) [[152]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L175-L183) [[153]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L70-L78) [[154]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L203-L211) [[155]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L216-L224) [[205]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L172-L181) [[227]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L72-L80) [[228]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L204-L212) [[239]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L83-L91) [[240]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L124-L132) [[267]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L158-L166) [[268]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L61-L69) [[286]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L172-L180) [[287]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts#L201-L209) index.ts

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/functions/finalizeAssessment/index.ts>

[[158]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/tests/results.integration.test.ts#L42-L50) [[278]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/tests/results.integration.test.ts#L56-L64) [[279]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/tests/results.integration.test.ts#L68-L75) [[280]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/tests/results.integration.test.ts#L40-L48) [[281]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/tests/results.integration.test.ts#L50-L53) results.integration.test.ts

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/tests/results.integration.test.ts>

[[207]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/utils/rescoreSession.ts#L17-L25) rescoreSession.ts

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/utils/rescoreSession.ts>

[[223]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820_prism_scoring_config_upserts.sql#L2-L7) GitHub

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/supabase/migrations/20250820_prism_scoring_config_upserts.sql>

[[271]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/ResultsV2.tsx#L8-L16) [[288]](https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/ResultsV2.tsx#L24-L29) GitHub

<https://github.com/ultomotone/prism-discover-yourself/blob/8b7c928e7a52c7a899f122787e8ca0491278ad1f/src/components/assessment/ResultsV2.tsx>