# Plan to Align Lovable Website & Supabase Scoring Infrastructure

## Overview

To **reformat the Lovable website and database**, we will **align the front-end, data models, and scoring logic** and implement a cohesive CI/CD pipeline. The Lovable application is a React + TypeScript project (Vite build) with Tailwind and ShadCN UI components. It currently uses Supabase (PostgreSQL + Edge Functions) for backend logic. **Scoring is entirely server-side** – the client invokes Supabase functions (like score\_prism and score\_fc\_session) and renders the results, with no client-side scoring calculations. Our goals are to:

* **Preserve and organize the existing UI** (pages like *Results*, *Assessment*, *UserDashboard*, and components like ResultsV2, OverlayChips, TraitPanel) while improving consistency and maintainability[[1]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L6-L13).
* **Solidify the data model** – ensure core tables (assessment\_sessions, assessment\_responses, profiles, etc.) and reference data (types, traits, overlays) are consistent, with proper constraints and indices for integrity and performance.
* **Harden the scoring API** – keep the scoring logic in versioned Supabase Edge Functions, so updates can roll out without breaking historical data. We’ll continue the version tagging (e.g. *PRISM v1.2.0* in code) for transparency[[2]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L2-L5) and possibly introduce parameters or config flags for new scoring versions.
* **Establish unified CI/CD workflows** – tie together database migrations, function deployments, and front-end builds into one pipeline (e.g. via GitHub Actions). Automated tests and smoke checks will be included so that every pull request and release is validated.

By addressing front-end, database, backend API, and deployment, we ensure **everything is aligned** and scoring is “set up the right way” across the stack. Below is a detailed plan for each aspect:

## 1. Front-End Restructuring (UI/UX Preservation & Improvements)

**Retain the current React/TypeScript UI structure** while reorganizing for clarity and future growth. The existing pages (Results.tsx, Assessment.tsx, UserDashboard.tsx, etc.) and key components (e.g. ResultsV2, OverlayChips, TraitPanel) will **remain functionally intact**[[1]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L6-L13), but we’ll improve how they are structured within the project. Key steps:

* **Group related components and pages:** Organize files by feature/domain. For example, move result-related components (ResultsV2, OverlayChips, TraitPanel) under a results or assessment feature directory. This makes it clear these pieces form the results page UI. Similarly, assessment-taking components can be under an assessment/ folder. The goal is to reorganize without rewriting, so the UI behaves exactly as before.
* **Preserve Tailwind and shadcn/ui usage:** Keep the styling approach consistent. We won’t change the design system – Tailwind utility classes and shadcn/ui primitives will continue to be used in the revamped structure. This ensures the look-and-feel and responsiveness remain the same. (For example, TraitPanel will still display the Neuroticism bar using Tailwind-styled divs[[3]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/components/Results/TraitPanel.tsx#L6-L14)).
* **No client-side scoring logic:** Ensure that the front-end continues *not* to perform any scoring calculations. It should only call the backend APIs and render results. In the refactor, double-check that pages like Results.tsx call the appropriate Supabase functions or RPCs to get data. Currently, the Results page first tries a secure RPC (get\_profile\_by\_session) if a share token is present[[4]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L119-L127), then falls back to reading the session or invoking an edge function for results[[5]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L149-L157). We will maintain this pattern so the client remains a thin layer.
* **Refine UX where needed:** While preserving functionality, we can make minor UX improvements. For example: ensure loading and error states are handled gracefully on the Results page (the code already manages a loading spinner and error messages via state[[6]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L36-L44)[[7]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L139-L147)). We might unify the results-fetch logic so that if a user has a valid session and is authenticated, we rely on the direct Supabase query (which RLS will permit for the owner), and use the edge function only when necessary. This simplifies the number of calls and speeds up the UX. Any tweaks will be tested to not regress the user experience.
* **Migrate to SSR or static export (optional):** In anticipation of deployment needs, configure Vite for SSR/static generation of certain pages. This is mainly for future scalability (e.g., prerendering marketing pages or improving SEO on result sharing). We can set up the build to produce a static version of the site for hosting, without changing the app’s code. All interactive pages (assessment flow, dashboard) will still hydrate into a single-page app. SSR considerations will not affect how scoring works, but we’ll ensure the build step (possibly npm run build with an SSR adapter) is integrated into CI.

By keeping the existing React app structure and just **reorganizing and polishing**, we ensure the user-facing side is stable. The assessment flow and results visualization remain consistent, meeting the UX priority of preserving current pages while setting a foundation for future improvements.

## 2. Database & Data Model Alignment (Supabase Schema)

We will **align and reinforce the Supabase database schema** so that the data model is consistent with the application’s needs and is ready for future scoring updates. The core tables – **assessment\_sessions**, **assessment\_responses**, **profiles** – and supporting reference tables (e.g. type\_prototypes, trait/overlay tables) will be kept and refined, rather than radically changed. Key actions in this phase:

* **Audit and preserve core tables:** We confirm that assessment\_sessions, assessment\_responses, and profiles remain the central tables linking a user’s assessment session to their answers and computed profile. No renaming or removal of these tables – instead, we **add any needed columns or constraints** to align with the evolving logic. For example, a recent migration added a share\_token column to assessment\_sessions (backfilled for existing rows) and made it NOT NULL with a unique index[[8]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L10-L18). We will ensure such additions remain in place, as they enable secure sharing of results via token.
* **Enforce relational integrity:** Add or verify **foreign key constraints** and cascade rules between tables. For instance, ensure each assessment\_response.session\_id references a valid session. In fact, a migration has already set up assessment\_responses.session\_id with a FOREIGN KEY to assessment\_sessions(id) ON DELETE CASCADE[[9]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250902_owner_can_read_responses.sql#L59-L66) (so that deleting a session will clean up its responses). Similarly, if not already, profiles.session\_id should reference assessment\_sessions(id) (likely with cascade as well, since a profile is essentially the result of a session). We will review the schema for any missing foreign keys or cascade rules and add them to prevent orphaned records and keep data aligned.
* **Apply field constraints and indexes:** Where appropriate, tighten the schema with CHECK constraints, NOT NULLs, and indexes for performance. For example, in the profiles table, ensure fields like type\_code or results\_version are not null once a profile is completed. We see ongoing refinements in migrations – e.g., adding new columns and then updating existing rows and setting defaults[[10]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql#L1-L9)[[11]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql#L56-L64). We’ll continue this approach: introduce new fields (with defaults or backfill), update all rows for consistency, then enforce NOT NULL or add an index. This process ensures a **gradual hardening** of the schema without breaking current functionality.
* **Retain and sync reference data:** The application relies on various reference tables and views (such as type\_prototypes, trait definitions, and overlay configurations). These must stay in sync with the scoring logic. For instance, the type\_prototypes table contains the Model A function blocks for each type and is loaded by the scoring function at runtime[[12]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L8-L16). We will preserve this table and ensure it always has complete data for all types (the scoring function expects 8 entries per type and falls back to hardcoded values if any are missing[[13]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L2-L10)[[14]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L14-L18)). Similarly, if there’s a traits table defining personality trait descriptors or thresholds, we keep it updated so the UI (like any trait display or future expansion of TraitPanel) reflects accurate info. In short, all **supporting lookup tables and views remain intact** and aligned with their usage in code.
* **Implement ongoing migrations for optimization:** Treat this as a **refinement, not a rewrite** of the database. The project’s migration history shows a pattern of incremental improvements – e.g., adding triggers to update updated\_at timestamps on key tables[[15]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250824164349_18a12c1b-628f-4ac9-84ed-2875dfab11f7.sql#L12-L20), creating views for consistent result calculations[[16]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250824062456_87fc50e2-bc40-4971-9efc-9e816aec4965.sql#L4-L12)[[17]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250824062456_87fc50e2-bc40-4971-9efc-9e816aec4965.sql#L32-L40), and tightening RLS policies. We will continue with this method. For example, we’ll verify that every table that needs an updated\_at trigger has one (and add if not), ensure important query patterns have indexes (e.g., if the app frequently queries profiles by user\_id or sessions by status, add indexes accordingly), and refine RLS (Row Level Security) policies so they remain secure but usable. As an illustration, an **RLS hardening** was done to allow users to read their own sessions and responses while blocking others[[18]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250902_owner_can_read_responses.sql#L38-L46)[[19]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250902_owner_can_read_responses.sql#L54-L62). We’ll double-check that these policies exist and are correct, and add any missing policies to align with the app’s access patterns.

By solidifying the database schema in this manner, **data model alignment** is achieved. All components of the app (frontend, edge functions) will be working with a consistent and secure data structure. This reduces bugs (no more mismatched fields) and boosts performance (with proper indexing and constraints). Moreover, it lays the groundwork for evolving the scoring logic (e.g., adding new result fields or tables) without breaking the existing data.

## 3. Server-Side Scoring Logic & API Robustness

Scoring is the heart of the application, and we will ensure it’s **structured “the right way”** on the server with proper versioning, testing, and integration. The current setup uses **Supabase Edge Functions** for scoring (score\_prism for the main PRISM algorithm, and score\_fc\_session for the forced-choice submodule) and related operations (finalizeAssessment, get-results-by-session, etc.). Our plan for the scoring/API layer:

* **Keep scoring on the server (Supabase Edge Functions):** We will not introduce any client-side scoring computations, preserving the security and consistency benefits. The React app will continue to call the protected endpoints. For example, when a user completes an assessment, the client calls the finalizeAssessment function (which in turn invokes score\_prism server-side)[[20]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/finalizeAssessment/index.ts#L117-L125). Similarly, to fetch results for display, the client either uses a Supabase RPC or calls the get-results-by-session function to retrieve the computed profile[[4]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L119-L127)[[5]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L149-L157). We will maintain and document these flows clearly so there is a single source of truth for scoring.
* **Version the scoring logic:** The main scoring function currently logs its version as **PRISM v1.2.0** in its header comment and output[[2]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L2-L5)[[21]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L1079-L1086). We will continue this versioning scheme. Every time we adjust the algorithm or calibration, we will bump the version string (e.g., to v1.3 or v2.0 as appropriate) and update any references. The profiles table already stores the results\_version and version of each result[[22]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L1032-L1041)[[21]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L1079-L1086), allowing the system to differentiate profiles computed with older logic. For instance, the database views use p.results\_version to filter or interpret results appropriately[[23]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250824062456_87fc50e2-bc40-4971-9efc-9e816aec4965.sql#L26-L34)[[24]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250824062456_87fc50e2-bc40-4971-9efc-9e816aec4965.sql#L70-L73). Going forward, **whenever scoring changes**, we will: *(a)* update the version constant in the function code, *(b)* include logic to handle backward compatibility if needed (for example, if v2.0 uses new fields, the code or DB views might need to branch on version), and *(c)* add a note in code comments or a dedicated config table tracking the current scoring version. This ensures API consumers (like the front-end) can always know which version produced a given result, and we can support multiple versions during transitions.
* **Robust edge function implementation:** Strengthen the edge functions by handling errors and ensuring idempotency. The score\_prism function already uses a service-role Supabase client (bypassing RLS) to safely read/write data and logs structured events (e.g., evt:scoring\_start and evt:scoring\_complete)[[25]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L1-L9)[[26]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L1134-L1141). We will build on this by making sure each function:
* Validates input thoroughly (e.g., check that session\_id is provided and well-formed, as is done in score\_fc\_session[[27]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_fc_session/index.ts#L14-L22)).
* Returns clear error messages or statuses if something goes wrong (the finalize function already captures scoring errors and returns a 422 with the message[[28]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/finalizeAssessment/index.ts#L123-L131)[[29]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/finalizeAssessment/index.ts#L156-L164)). We’ll audit each function to cover edge cases (like no responses submitted, duplicate submissions, etc.) so that the API is predictable and robust.
* Is idempotent or safely repeatable when possible. For example, calling finalizeAssessment twice on the same session currently detects an existing profile and simply returns it without re-scoring[[30]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/finalizeAssessment/index.ts#L50-L59)[[31]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/finalizeAssessment/index.ts#L83-L91). This prevents double computation and race conditions. We will ensure such checks exist wherever needed (perhaps also in score\_prism to avoid scoring twice in parallel).
* **Coordinate score\_prism with score\_fc\_session:** The forced-choice (FC) scoring function is separate (score\_fc\_session), but the main scoring uses its output. In PRISM v1.2.0, after the Likert and other scores are processed, the code pulls in FC scores (it queries the fc\_scores table for the given session/version)[[32]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L12-L20). We will make sure that whenever an assessment is finalized, both parts are executed in the correct order. One approach is already used: the finalizeAssessment function can invoke score\_prism, which internally will fetch or compute FC results as needed. If we find any timing issues (say, score\_prism needs FC data ready), we might explicitly call score\_fc\_session first. However, since score\_prism looks up fc\_scores by session and version, we need to ensure score\_fc\_session ran earlier in the workflow. If not currently guaranteed, we will adjust the finalize logic to invoke score\_fc\_session (or incorporate its logic) before score\_prism. This guarantees the final profile includes both Likert and forced-choice analysis.
* **API endpoint consistency:** All client-facing API calls will be reviewed for consistency and security. The front-end uses a mix of Supabase RPC (for a shareable results retrieval) and direct edge function calls[[4]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L119-L127)[[5]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L149-L157). We will ensure these endpoints remain valid after refactoring. For instance, we created the secure function get\_profile\_by\_session(session\_id, share\_token) to replace a broad RLS policy on profiles[[33]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L16-L24)[[34]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L41), and the client now uses this RPC when a share token is present[[4]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L119-L127). This is a good pattern for secure access. We will continue to expose only the necessary data via either RPC or edge functions, and not directly open up tables with lax policies. Every new API addition will follow this practice (define a PostgreSQL function or use an edge function with proper validation, rather than weakening RLS).
* **Thorough testing of scoring:** As we set up “scoring the right way,” we will also create tests or use existing QA suites to validate the scoring outputs. The repository already includes a QA suite for validation logic and even E2E tests for the assessment flow[[35]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/QA.md#L59-L68)[[36]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/QA.md#L79-L88). We will extend this to cover the scoring functions. For example, we can simulate a full assessment submission (perhaps by calling score\_prism on known fixture data) and verify the profile outputs (type\_code, confidence, etc.) match expected values. This gives us confidence that changes to the algorithm (or database calibration data) produce correct results. Such tests can be integrated into the CI pipeline (see next section) so that any divergence in scoring triggers a test failure before deployment.

By focusing on these measures, the **scoring infrastructure will be robust and maintainable**. The API layer will clearly delineate responsibilities (each function doing one job, e.g., finalizing session vs. computing scores vs. fetching results) and enforce versioning. This setup makes it easy to evolve the algorithm – we can introduce a new version flag or function without breaking older results, and deploy it behind the scenes. Overall, the server-side logic remains the authoritative “brain” of the assessment, which is exactly how it should be for security and consistency.

## 4. Unified CI/CD Deployment Pipeline

Finally, we will implement a **unified continuous integration and deployment workflow** that ties together all the above aspects. This ensures that whenever we push changes (whether to the front-end, database schema, or edge functions), everything stays in sync and undergoes proper checks. Key elements of the CI/CD setup:

* **GitHub Actions for CI:** We will create a GitHub Actions workflow (e.g., .github/workflows/ci.yml) that runs on each pull request and on merges to main. This workflow will have steps to cover building and testing all parts of the system. For example:
* **Install dependencies and lint** – run npm ci (or npm install) to get all packages, then npm run lint and npm run type-check. This catches any TypeScript errors or lint violations early.
* **Run unit and QA tests** – execute the test suites (as defined in package scripts). The project’s QA includes library integrity tests, validation logic tests, and E2E tests[[35]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/QA.md#L59-L68)[[36]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/QA.md#L79-L88). We will ensure npm run qa:all (or a similar command) is run in CI. The pipeline will fail if any test fails, blocking a bad change from merging. (As noted in the QA docs, we might also use a pre-commit hook for developers, but CI is the safety net on the server side)[[37]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/QA.md#L165-L173). Additionally, we can include a quick **smoke test** step: for instance, spin up the app in a test environment and hit a couple of endpoints (maybe using a script similar to verify:smoke). This could call the health-check endpoints or ensure that a basic page load returns 200.
* **Check database migrations** – incorporate the Supabase migration workflow into CI. We will use the Supabase CLI to validate and even apply migrations in a test scenario. For example, run supabase db lint or a **dry-run migration** against a temporary database to ensure the SQL scripts have no errors. If possible, we can start a Postgres service in CI, apply all migrations, and run some simple queries (perhaps using the existing scripts/preflight.ts which executes the ground-zero SQL and runs verification queries[[38]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/scripts/preflight.ts#L161-L169)[[39]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/scripts/preflight.ts#L170-L178)). This step ensures our DB changes won’t break in production.
* **Build the front-end** – run the Vite build (e.g., npm run build). Possibly do this twice if we have an SSR build and a static build. The build step should produce the production-ready assets. As part of this, we can also generate a preview artifact (for example, upload the static build to GitHub Actions as a preview, or deploy to a temporary environment for QA to look at). The CI should fail if the build fails or if there are missing environment variables, etc.
* **Compile and package Edge Functions** – ensure the Supabase Edge Functions compile successfully. We might run a command like npm run functions:compile (if defined) to transpile the TypeScript functions to the Deno bundle that Supabase expects. Any compile error in our functions (score functions, etc.) will then be caught early. This is important because these functions run in Deno; type mismatches or missing imports would cause runtime errors if not caught. By compiling in CI, we verify that the latest code for score\_prism, finalizeAssessment, etc., is syntactically correct and ready to deploy.
* **Continuous Deployment (CD):** Once the above CI steps pass on the main branch, we can have automated deployment steps:
* **Apply database migrations to production** – Use the Supabase CLI (supabase db deploy) or a GitHub Action to run the new migrations on the Supabase project. This will update the schema (adding columns, constraints, etc. that were in the migration files). By running this first in deployment, we ensure the database is ready *before* new code that depends on it goes live. For instance, if a new scoring version expects a new column in profiles, we must add that column (via migration) before deploying the new score\_prism function.
* **Deploy edge functions** – Following DB migration, use supabase functions deploy --all (or deploy individual functions) through the CLI. This pushes the latest code for score\_prism, score\_fc\_session, and others to the Supabase environment. Because our migrations ran first, any new database references in the functions should now exist (preventing runtime errors). We will version control these functions in the repo, so the deployment uses the exact code that was tested in CI.
* **Deploy the front-end** – Finally, release the updated website. If we are hosting on a platform (e.g., Vercel, Netlify, or Supabase Hosting), we’ll integrate with that. For a static site, this might involve uploading the built assets (from the build step) to an object storage or CDN. If using Supabase Storage or a Cloudflare Pages, etc., we script that in the Action. The key is that the front-end is deployed **after** the backend changes, so that any new API endpoints or fields it relies on are already available. (We will also ensure environment variables like the Supabase anon key, URL are correctly configured in the deployment environment – the front-end uses these for API calls).
* **Post-deployment smoke test:** Optionally, after deployment, the pipeline can ping a few endpoints to ensure the app is up. For example, call the /functions/v1/get-results-by-session with a test input or simply request the homepage. This confirms that the system is live with the new changes. If any of these quick checks fail, the deployment can be marked as failed for immediate investigation.
* **GitHub PR integration:** The workflow described will run on PRs in a slightly different mode – e.g., it might skip the deploy steps but run all tests and build steps, giving developers feedback. We can also incorporate **Supabase DB review apps** if available: spin up a temporary database or schema for a PR to test migrations in isolation. If not, the local container approach suffices. Once a PR is approved and merged, the CD steps trigger on the main branch to do the real deployment.
* **Monitoring and notifications:** We’ll configure the pipeline to output logs and metrics for each step. If lint/test fails, the Action will comment on the PR with the failure. For deployments, we might integrate Slack or email notifications on success/failure. Also, since Supabase functions logging is crucial (we rely on those console logs for events), we may set up a log drain or at least ensure we can view function logs easily after deployment to catch any unexpected errors early. This is more of an ops consideration, but part of “infrastructure done right” is having visibility into issues once code is live.

By **tying Supabase migrations, edge function deploys, and front-end builds into one pipeline**, we achieve a true unified deployment. This means no more manual steps out of sync – the code in Git (for both website and database) becomes the single source of truth, and each change is automatically tested and released. It reduces the chance of deploying a front-end that calls a missing API or a function that expects a DB column that wasn’t added. In effect, the CI/CD pipeline enforces that **everything remains aligned** across the stack, which is exactly our overarching goal.

## Conclusion

This comprehensive plan addresses the reformatting and alignment of the Lovable application across all layers. We preserve the things that already work well (the UI/UX, the core data schema, the server-side scoring principle) while restructuring the project for clarity and future growth. By solidifying the data model, enforcing versioned and robust scoring logic, and implementing an integrated deployment pipeline, we set up a foundation where **feature development and scoring iterations can proceed rapidly and safely**. Every code or database change will go through the same rigorous process – from development (with clear prompts for Codex/Copilot to follow this blueprint) to automated testing and deployment via PRs.

In summary, this ensures that **the website and database are fully aligned with each other**, and that scoring is implemented “the right way” – server-driven, versioned, and infrastructure-backed – so the application remains reliable and scalable as it evolves. [[2]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L2-L5)[[8]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L10-L18)

[[1]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L6-L13) [[4]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L119-L127) [[5]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L149-L157) [[6]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L36-L44) [[7]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx#L139-L147) Results.tsx

<https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/pages/Results.tsx>

[[2]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L2-L5) [[12]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L8-L16) [[13]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L2-L10) [[14]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L14-L18) [[21]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L1079-L1086) [[22]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L1032-L1041) [[25]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L1-L9) [[26]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L1134-L1141) [[32]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts#L12-L20) index.ts

<https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_prism/index.ts>

[[3]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/components/Results/TraitPanel.tsx#L6-L14) TraitPanel.tsx

<https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/src/components/Results/TraitPanel.tsx>

[[8]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L10-L18) [[33]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L16-L24) [[34]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql#L34-L41) 20250820002713\_932dced6-5d6b-4315-ab32-e0958463ec2d.sql

<https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250820002713_932dced6-5d6b-4315-ab32-e0958463ec2d.sql>

[[9]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250902_owner_can_read_responses.sql#L59-L66) [[18]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250902_owner_can_read_responses.sql#L38-L46) [[19]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250902_owner_can_read_responses.sql#L54-L62) 20250902\_owner\_can\_read\_responses.sql

<https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250902_owner_can_read_responses.sql>

[[10]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql#L1-L9) [[11]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/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/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250819204241_c41c4435-4bb3-4138-83d7-16154aba90ed.sql>

[[15]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250824164349_18a12c1b-628f-4ac9-84ed-2875dfab11f7.sql#L12-L20) 20250824164349\_18a12c1b-628f-4ac9-84ed-2875dfab11f7.sql

<https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250824164349_18a12c1b-628f-4ac9-84ed-2875dfab11f7.sql>

[[16]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250824062456_87fc50e2-bc40-4971-9efc-9e816aec4965.sql#L4-L12) [[17]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250824062456_87fc50e2-bc40-4971-9efc-9e816aec4965.sql#L32-L40) [[23]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250824062456_87fc50e2-bc40-4971-9efc-9e816aec4965.sql#L26-L34) [[24]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250824062456_87fc50e2-bc40-4971-9efc-9e816aec4965.sql#L70-L73) 20250824062456\_87fc50e2-bc40-4971-9efc-9e816aec4965.sql

<https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/migrations/20250824062456_87fc50e2-bc40-4971-9efc-9e816aec4965.sql>

[[20]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/finalizeAssessment/index.ts#L117-L125) [[28]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/finalizeAssessment/index.ts#L123-L131) [[29]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/finalizeAssessment/index.ts#L156-L164) [[30]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/finalizeAssessment/index.ts#L50-L59) [[31]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/finalizeAssessment/index.ts#L83-L91) index.ts

<https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/finalizeAssessment/index.ts>

[[27]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_fc_session/index.ts#L14-L22) index.ts

<https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/supabase/functions/score_fc_session/index.ts>

[[35]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/QA.md#L59-L68) [[36]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/QA.md#L79-L88) [[37]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/QA.md#L165-L173) QA.md

<https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/QA.md>

[[38]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/scripts/preflight.ts#L161-L169) [[39]](https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/scripts/preflight.ts#L170-L178) preflight.ts

<https://github.com/ultomotone/prism-discover-yourself/blob/0ba5ba0e0b519b121868099cd880a7363f4e4ce3/scripts/preflight.ts>