Indian Mutual Fund Robo-Advisor — Phase 2 (Version 0.2)

Author: Ravi Shankar

Document: Software Requirements Specification (SRS) — Phase 2 (Registration & Analytics MVP)

Derived from: Phase 1 SRS (existing MVP single-user prototype).

---

1. Executive summary

Phase 2 extends the Phase 1 single-user prototype by adding a simple, privacy-minded **user registration** step and lightweight **analytics** so the team can measure prototype interest and basic funnel metrics. The Phase-2 changes are intentionally minimal (KISS) and designed for rapid implementation with low operational overhead, while remaining compatible with future migration to full authentication and cloud services.

Key additions - Registration step (shown after risk-score results). - Minimal email validation (format-only) and a required consent checkbox. - Server-side persistence using **SQLite** (recommended) with an easy CSV-export path. - Minimal analytics: counts and simple segmentations (city, country, completed questionnaire, reached recommendations). - Backwards-compatible design so Phase-1 UI/code can be extended rather than re-written.

---

2. Scope and constraints

2.1 In scope (Phase 2)

- Add a registration UI presented **after** the user completes the risk questionnaire and views the risk category.
- Capture: `name`, `email`, `city`, `country`, `timestamp`, and `event flags` (questionnaire_completed, recommendations_viewed).
- Store records in a local **SQLite** database file shipped with the app (configurable path).
- Simple email format validation (no OTP/email sending).
- Consent checkbox required for storing data: short KISS text.
- Admin utilities to export user records to CSV.
- Lightweight analytics queries and a small admin page showing counts and basic breakdowns.

2.2 Out of scope (Phase 2)

- Full authentication (login/password, OAuth) — **design will allow future addition**.
- Sending emails (welcome, verification) or OTPs.
- Production-grade dashboards or analytics platforms (Power BI / Looker). Only simple built-in reporting.
- Payment or investment execution.
- GDPR/Privacy legal review — follow basic consent; user to obtain legal review if needed.

3. User stories / Flows

3.1 Primary flow — Guest -> Questionnaire -> Register -> Recommendations

1. User loads app (Phase-1 behavior unchanged).
2. User completes 13-question risk questionnaire.
3. App calculates risk score and displays Risk Category and description.
4. App shows a concise registration prompt (see wording below) with input fields: Name, Email, City, Country, and a consent checkbox.
5. Buttons: `[Register & Show Recommendations]` and `[Skip & View Recommendations]`.
6. Registration is optional: user can skip to view recommendations. (KISS: capture interest without blocking experience.)
7. If user registers and consents, record is stored in SQLite and event flags set.
8. App displays recommendations filtered as Phase 1 defined.

3.2 Alternate flows

- If user submits invalid email format -> show inline validation error.
- If database write fails -> show a friendly error and allow the user to continue to recommendations (do not block the user).

---

4. Functional requirements (FR)

Each FR ID continues the Phase-1 naming pattern where relevant.

FR-R1 — Registration UI (post-risk) - FR-R1.1: After risk calculation, present a registration card with fields: `Full Name` (string), `Email` (string, format-checked), `City` (string), `Country` (drop-down with top countries + Other). Require consent checkbox with text (see 4.6). - FR-R1.2: Provide two actions: `Register & Show Recommendations` and `Skip & View Recommendations`. - FR-R1.3: On successful registration, persist record and set `questionnaire_completed = true` and `recommendations_viewed = true` when recommendations page loads.

FR-R2 — Validation & Errors - FR-R2.1: Validate email format on client and server. If invalid, display E101: "Please enter a valid email address.". - FR-R2.2: Make `Name` optional (KISS) but `Email` and consent checkbox mandatory to register. - FR-R2.3: If persistence fails, show non-blockingE102: "We couldn't save your details right now — you can still view recommendations." Log error server-side.

FR-R3 — Persistence & Export - FR-R3.1: Persist registrations in a SQLite database (single table `registrations`). - FR-R3.2: Include a small admin page reachable via a local-only route or hidden UI token to export registrations to CSV. - FR-R3.3: Provide a simple `vacuum` / backup note for the DB file in README.

FR-R4 — Analytics & Metrics - FR-R4.1: Record events/flags: `questionnaire_completed`, `registered`, `recommendations_viewed`, and `timestamp`. - FR-R4.2: Admin overview page shows: - Total registered users (distinct emails). - Number of users who completed questionnaire vs. those who registered. - Breakdown by country and top 10 cities. - Funnel: % completed questionnaire →

% registered → % viewed recommendations. - FR-R4.3: Allow exporting simple aggregated counts to CSV.

FR-R5 — Privacy & Consent - FR-R5.1: Show consent checkbox with minimal wording; record consent boolean and timestamp in DB. - FR-R5.2: Provide a short Privacy Note on the registration card: "We will use this info only to improve this prototype. No emails will be sent."

FR-R6 — Future auth compatibility - FR-R6.1: Schema and data access patterns must include a `user_id` field (nullable) so that when auth is added, registrations can be linked to authenticated accounts.

---

5. Data model

Database: SQLite (file: `data/robo_advisor_phase2.db` by default). Configurable via environment variable or a small config file.

Table — registrations - `id` INTEGER PRIMARY KEY AUTOINCREMENT - `name` TEXT NULL - `email` TEXT NOT NULL - `city` TEXT NULL - `country` TEXT NULL - `consent` BOOLEAN NOT NULL - `consent_ts` DATETIME NOT NULL - `questionnaire_completed` BOOLEAN NOT NULL DEFAULT 1 - `recommendations_viewed` BOOLEAN NOT NULL DEFAULT 0 - `risk_score` INTEGER NULL - `risk_category` TEXT NULL - `created_ts` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP - `user_id` TEXT NULL -- placeholder for future auth linkage

Indexes - Index on `email` (unique constraint optional — recommended to avoid duplicates). - Index on `country` and `city` for fast analytics queries.

Notes - SQLite chosen because: zero infra, easy local backup, reliable SQL queries for analytics, easy to ship with prototype. For larger scale, migrating to PostgreSQL / Supabase / Firebase is straightforward.

---

6. Minimal admin UI (KISS)

A small admin route (`/admin` or a hidden button) exposes: - Overview cards: total registrations, completed questionnaires, recommendations viewed. - Tables: latest 50 registrations (mask or avoid showing PII in shared environments). - Export button: `Export all registrations (CSV)`.

Security: admin route should be accessible locally (localhost) only in MVP. For remote deployments, put behind simple auth in hosting environment.

---

7. UI copy & wording (chosen)

After considering the three variants you liked (Direct, Soft, Value-focused), the Phase-2 SRS uses a short combined, clear and locally-appropriate line that balances value + invitation:

> **Register to access personalized mutual fund recommendations and help us improve this prototype.**

Consent checkbox text (KISS): - `I agree to share my details for prototype research and to receive no emails.`

Inline privacy note (below the form): - `We store your name, email, and location only to understand interest in this prototype. No marketing emails will be sent.`

Button labels: - Primary: `Register & Show Recommendations` - Secondary: `Skip & View Recommendations`

(You can tweak the text later; these are designed to be concise and familiar to an Indian audience.)

---

8. API & implementation notes

Implementation should require minimal changes to existing Streamlit code.

Suggested implementation steps 1. Add a lightweight DB helper module (`db.py`) with functions: `init_db()`, `save_registration(payload)`, `export_csv()`. 2. After calculating risk score in existing flow, render a registration form (Streamlit `st.form`) with the fields specified. 3. On submit: validate email regex server-side and call `save_registration`. 4. If save succeeds, write a small analytics event and continue to recommendation rendering logic. 5. Add admin page to Streamlit app accessible by a secret query param or hidden button. Use `st.experimental_singleton` / caching carefully for DB connection reuse. 6. Add README notes for DB backup and migration steps.

Email validation - Use a simple regex check for format only (no DNS/SMTP check): KISS.

Error handling - If DB insert fails, log exception and allow user to continue. - Surface user-friendly message: "Unable to save your details — you can still view recommendations."

---

9. Non-functional requirements

- **Performance**: Registration write must not add >200ms latency to recommendation display in normal local runs.
- **Availability**: The app must continue to function even if registration persistence fails.
- **Security**: Data stored in SQLite file. In shared environments, document that PII is stored in the DB file and recommend not sharing the file.
- **Maintainability**: Keep DB layer separate, use migrations if schema changes in future.
- **Privacy**: Store only the specified fields. No emails sent.

---

10. Acceptance criteria

The Phase-2 feature is accepted when these are met: 1. A user completes the questionnaire, sees risk category, is offered the registration form as described. 2. A user who registers (fills email & consents) has a row saved in SQLite with correct flags and timestamps. 3. Admin export produces a CSV with the saved fields and timestamps. 4. Analytics admin page shows coherent counts (total registrations, questionnaire completions, recommendations viewed) consistent with DB. 5. If DB write fails, the user

can still view recommendations (app is non-blocking). 6. Email format validation prevents malformed emails from being stored.

---

11. Migration & future roadmap (notes)

- When ready to add authentication, add a `users` table and populate `user_id` in `registrations` linking to authenticated users.
- Consider moving DB to PostgreSQL or Supabase for multi-instance concurrency if the app will be hosted for many users.
- To enable email campaigns or verification, implement SMTP integration and opt-in flows with double opt-in.

---

12. Implementation estimate (developer-facing)

A small rough estimate for an experienced Streamlit developer building on the existing prototype: - DB helper + schema + tests: 4–6 hours - Registration form UI + validations: 2–4 hours - Admin export + overview: 3–5 hours - Integration, QA, README: 2–4 hours

(Estimates are narrow because this is an incremental change to an existing working prototype.)

Appendix A — Example SQL (SQLite)

```
CREATE TABLE registrations (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT,
  email TEXT NOT NULL,
  city TEXT,
  country TEXT,
  consent BOOLEAN NOT NULL,
  consent_ts DATETIME NOT NULL,
  questionnaire_completed BOOLEAN NOT NULL DEFAULT 1,
  recommendations_viewed BOOLEAN NOT NULL DEFAULT 0,
  risk_score INTEGER,
  risk_category TEXT,
  created_ts DATETIME DEFAULT CURRENT_TIMESTAMP,
  user_id TEXT
);
CREATE INDEX idx_reg_email ON registrations(email);
CREATE INDEX idx_reg_country ON registrations(country);
```

End of SRS Phase 2

You can edit this document in-place. If you'd like, I can also produce a downloadable Word or PDF version.