Product Requirements Document (PRD)

Simple Contacts CRUD — Flutter + Firebase (Firestore) — Android & Web

---

1. Purpose

Build a small, secure, cross-platform CRUD application using **Flutter** and **Firebase Firestore** that runs as a native Android app and as a Flutter Web app (which can also be loaded inside an Android WebView). The app will allow creating, searching, editing and deleting simple contact records (Name and Contact Number).

Goals:

- Single codebase for Android & Web
- Realtime sync using Firestore (so updates show on all clients)
- Proper validation and UX (popups, inline table, search)
- Secure by default (Firestore Security Rules, input validation, no direct secret storage)

---

2. High-level User Flows

2.1 Launch / Home

- On first launch show Home screen with a prominent **Create** button.
- If there is at least one record stored, show a **Search** input (partial match) and a table/list of records below.

2.2 Create Record

- Clicking **Create** opens a modal popup with two fields:
    - **Name** — text field, max 40 characters, letters and spaces only (no digits/symbols). Required.
    - **Contact Number** — numeric field, exactly 10 digits allowed. Required.
- Buttons: **Save** (disabled unless both fields valid), **Cancel** (closes modal).
- On Save: create a Firestore document under contacts collection with the fields: name, contactNumber, createdAt, updatedAt, createdBy (uid placeholder until Auth added).

2.3 Search & List

- When at least one record exists, show the Search box above the list.
- Search should allow partial matches on name and contactNumber (client-side filtering or Firestore queries with simple constraints). Minimum 1 character to trigger.
- Display results in a responsive table layout on wide screens and a card/list layout on narrow screens (mobile). Table columns: Name, Contact Number, Created At, Actions.
- Each row has **Edit** (pencil) and **Delete** (trash) icons on the right.

2.4 Edit

- Clicking Edit opens the same popup prefilled.
- Validate same rules as Create.
- On Save: update Firestore updatedAt timestamp and fields.

2.5 Delete

- Clicking Delete opens a confirmation dialog: “Are you sure you want to delete this record? This action cannot be undone.” Buttons: **Delete**, **Cancel**.
- On Delete: remove document from Firestore and show a toast/snackbar confirming deletion; the UI list updates automatically via real-time listener.

---

3. Functional Requirements (detailed)

3.1 Fields & Validation

- **Name**:
    - Type: string
    - Display limit: 40 characters
    - Allowed characters: A–Z, a–z, space (no digits, no punctuation)
    - Validation messages: "Name is required" / "Use only letters and spaces" / "Max 40 characters"
- **Contact Number**:
    - Type: string of digits
    - Exactly 10 digits required
    - Validation messages: "Contact number is required" / "Enter exactly 10 digits"
- **Save button state** should enable only when both fields are valid.

3.2 Search

- Partial search for both Name and Contact Number (case-insensitive). Implementation options:
    - Client-side filter from a local stream of loaded documents (works for small datasets).
    - For larger datasets use Firestore queries with >=/< range queries on indexed fields and/or a simple startAt()/endAt() for prefix search.

3.3 List Display

- Show results in a table on web/desktop. On mobile use a single-column card list.
- Each record shows Name, Contact Number, CreatedAt or relative time.

3.4 Edit/Delete

- Edit updates the document in-place.
- Delete requires confirmation dialog.
- Concurrency: last write wins (document update is atomic). Consider version or timestamp-based conflict detection for advanced use.

3.5 Realtime & Offline

- Use Firestore listeners (collection.snapshots()) to receive real-time updates across devices.
- Enable Firestore offline persistence so the app continues to work when offline and syncs when online.

---

4. Data Model

Firestore collection: contacts
Document fields:

- name : string
- contactNumber : string
- createdAt : timestamp (Firestore server timestamp when created)
- updatedAt : timestamp (Firestore server timestamp when updated)
- createdBy : string (user id placeholder; if no auth used, store 'anonymous')

Indexes: single-field indexes on name and contactNumber (default). For advanced search, consider a composite index.

---

5. Security & Best Practices

5.1 Firestore Security Rules (required)

- Only allow reads/writes from authenticated users (preferred). If using open access for demonstration, restrict writes to validated fields and reject malformed values.
- Example rule checks:
    - request.resource.data.name exists, is string, length <= 40, and matches [A-Za-z]+.
    - request.resource.data.contactNumber exists, is string, length == 10, and matches ^[0-9]{10}$.
    - createdAt / updatedAt should be set via server timestamp or validated.

If user authentication not added initially, restrict by other rules (e.g., require a specific API key via Cloud Function) — but **do not** leave database open for public writes in production.

5.2 Client-side Input Validation

- Validate inputs on the client before sending to Firestore to reduce malformed calls and improve UX.

5.3 Avoid embedding secrets

- Never hardcode API keys or service account credentials in the client app.
- Firebase client SDK uses public config (allowed) but no private credentials.

5.4 Secure Hosting

- Use HTTPS (Firebase Hosting provides SSL automatically).
- If embedding Web inside Android WebView, enable setJavaScriptEnabled(true) carefully and avoid exposing native interfaces that accept untrusted input.

---

6. Non-functional Requirements

- Responsive UI for web and mobile.
- Minimal latency for CRUD operations (Firestore typically fast; show loading indicators for network ops).
- Accessibility: tappable targets, readable fonts.
- Internationalization-ready (strings in one place).

---

7. UX / UI Details

- Home screen: AppBar with title, Create FAB or button.
- Create/Edit modal: use showDialog() (material) or a Dialog widget with TextFormFields, validators, and focused input.
- Search: top-of-list search box with clear icon and placeholder "Search by name or contact".
- Table: DataTable on web (with pagination if many records) and a ListView on mobile.
- Actions: trailing icons for edit & delete; show tooltips on hover (web).

---

8. Error Handling & Notifications

- Show user-friendly error messages on network failures or invalid operations.
- On network errors, gracefully fall back to offline mode and show indication that data will sync when online.
- Show success snackbars after create/update/delete.

---

9. Testing & QA

- Unit tests for validation logic (name/contact rules).
- Widget tests for modal behaviour and list rendering.

- Integration test for end-to-end create → list → edit → delete flow.
- Security tests for Firestore rules via Firebase Emulator Suite.

---

10. Deployment & Hosting

- Use flutter build web to produce web assets.
- Host on Firebase Hosting using firebase deploy.
- Android: build using flutter build apk or App Bundle and publish to Play Store.
- Web inside Android WebView: create a thin Flutter app with a WebView pointing to your hosted https://<project>.web.app URL.

---

11. Acceptance Criteria

- Create popup opens with correct validators and Save enables only on valid input.
- Records appear in list/table and update in real-time across devices.
- Search filters records partially by name or contact.
- Edit and Delete operations work with confirmation and proper feedback.
- Firestore Security Rules prevent malformed writes.
- Offline operations queue and sync when online.

---

12. Deliverables

- Flutter project (single codebase) with Firestore integration.
- Firestore security rules file.
- README with setup & deployment steps.
- Unit & widget tests for validation and core flows.
