Petfolio

System Design & Architecture

For engineers – current state of the app

`pet_link/` (Flutter, Firebase, Riverpod)

Table of Contents

1. Product & Scope (Current)

- **Goal**: Shared hub for pet care (owners, family, sitters).
- **Platforms**: iOS, Android, Web, Desktop (Flutter).
- **Core capabilities (implemented)**:
  - User authentication via Firebase Auth.
  - Pet profiles with photos.
  - Care plans (feeding, meds, notes).
  - Timezone-aware local notifications.
  - Shareable read-only profiles (QR / link).
  - Sitter dashboard & task completion.
  - Lost & Found poster generation.

1.1 Users & Roles

- **Owner**
  - Creates/manages pets & care plans.
  - Manages handoffs and access tokens.
- **Co-caretakers / Family**
  - Share owner-like access on same account.
- **Sitter / Walker**
  - Time-boxed access via QR/link.
  - Completes tasks, views care plan.
- **Public Viewer**
  - Read-only access to shared/lost pet pages.

2. High-Level Architecture

- **Frontend**
  - Flutter app under `pet_link/lib/`.
  - `main.dart` initializes Firebase + timezone, wraps app in Riverpod `ProviderScope`.
  - `PetfolioApp` (MaterialApp) configures routes, theme, and startup flow.
- **Backend**
  - Firebase Auth for identity.
  - Firestore for all primary domain data.
  - Firebase Storage for pet images and generated posters.
  - Local device notifications for reminders (no push yet).

2.1 Logical Components

- **App Shell**
  - Startup logic (`AppStartup`), auth guard (`AuthWrapper`), `MainScaffold` with bottom navigation.
- **Features**
  - `auth/` – sign in/up, auth wrapper.
  - `pets/` – pet list, detail, edit/create.
  - `care_plans/` – care plan editor & task engine.
  - `sharing/` – access tokens, QR, sitter dashboard, public profiles.
  - `lost_found/` – lost reports and posters.
  - `onboarding/` – welcome & success flows.

- **Core & Services**
  - `core/widgets/` – shared UI components.
  - `services/` – network, local storage, error handling.

3. Flutter App Structure

- **Entry point**
  - `lib/main.dart`
    - Initializes Flutter bindings.
    - Initializes timezone database.
    - Calls `Firebase.initializeApp` with `firebase_options.dart`.
    - Runs `PetfolioApp` inside a `ProviderScope`.
- **App root**
  - `app/app.dart`
    - Defines `PetfolioApp` and `MaterialApp`.
    - Configures routes and `onGenerateRoute`.
    - Wires theme (`theme.dart`) and configuration (`config.dart`).

3.1 App Shell & Startup

- `AppStartup`
  - Performs one-time startup tasks (e.g., loading user, initializing services).
  - Wraps the main scaffold, ensuring app is ready before showing UI.
- `AuthWrapper`
  - Listens to auth state.
  - Routes unauthenticated users to login/signup.
  - Routes authenticated users to the main experience (dashboard, sitter view).
- `MainScaffold`
  - Houses top-level navigation (bottom nav) and app bar.
  - Switches between main tabs: pets, care/tasks, profile, etc.

4. Navigation & UX Flow

- **Routing**
  - `MaterialApp` with static `routes` and `onGenerateRoute`.
  - Typed `RouteNames` in `config.dart`.
  - `onGenerateRoute` handles:
    - Edit pet (`EditPetPage`) with optional `pet` argument.
    - Pet detail (`PetDetailPage`) with required `Pet`.
    - Share pet (`SharePetPage`) with required `Pet`.
    - Public profile (`PublicPetProfilePage`) with `tokenId`.
- **Future**
  - `go_router` is planned but **not yet integrated**.

4.1 Key Flows (Owner)

- **Onboarding**
  i. User lands on `WelcomeView`.
  ii. Signs up / logs in.
  iii. Creates first pet via guided flow.
  iv. Sees `SuccessView` and transitions to main app.
- **Daily usage**
  - View pet list & status.
  - Open pet detail to see care plan, notes, and lost status.
  - Edit care plan, manage reminders.
  - Generate QR / link for sitter or lost poster.

4.2 Key Flows (Sitter & Public)

- **Sitter handoff**
  i. Owner opens `SharePetPage` to create an `AccessToken`.
  ii. App generates QR code for the access link.
  iii. Sitter scans QR -> deep link / web route.
  iv. Sitter sees `SitterDashboardPage` with tasks and care plan.
- **Public profile / lost poster**
  - Public user lands on `PublicPetProfilePage` using token id.
  - App enforces read-only access using token role and expiry.

5. Domain Model Overview

> Firestore schemas are defined in more detail in the Firestore spec; this section reflects the **current implemented** model.

- **User**
  - `{ id, email, displayName, photoUrl, roles[], createdAt, updatedAt }`
- **Pet**
  - `{ id, ownerId, name, species, breed, dateOfBirth, weightKg, heightCm, photoUrl, isLost, createdAt, updatedAt }`
- **CarePlan**
  - `{ id, petId, dietText, feedingSchedule[], medications[], emergencyContacts, medicalHistory, behavioralNotes }`

5.1 Domain Model (Sharing & Activity)

- **AccessToken**
  - `{ id, petId, grantedBy, grantedTo, role, expiresAt, notes, contactInfo }`
- **TaskCompletion**
  - `{ id, petId, careTaskId, completedBy, completedAt }`
- **LostReport**
  - `{ id, petId, ownerId, createdAt, lastSeenLocation, notes, posterUrl }`
- **Not yet implemented (planned)**
  - `WeightEntry` for health tracking.
  - Messaging collections (owner–sitter chat).
  - Push notification events.

5.2 Firestore Collections (Current)

- Expected top-level collections:
  - `/users`
  - `/pets`
  - `/carePlans`
  - `/accessTokens`
  - `/taskOccurrences` or `/taskCompletions`
  - `/lostReports`
- **Notes**
  - Exact collection names / nesting should be confirmed against `firestore.rules` and repositories in `lib/features/**/data/`.

6. State Management & Providers

- **State management**
  - `flutter_riverpod` used across the app.
  - Providers typically live in each feature folder under `features/**`.
- **Common patterns**
  - Repository providers for Firestore access.
  - Stream/`AsyncValue` providers for live pet lists, care plans, and sitter tasks.
  - Scoped providers for currently selected pet or current access token.
- **Guidelines**
  - Prefer using existing providers; avoid duplicating data sources.
  - Drive UI from `AsyncValue` for loading/error states.

7. Core Feature Areas – Auth

- **Authentication**
  - Firebase Auth (email/password) integration.
  - `AuthWrapper` used as a route guard:
    - Shows login/signup when unauthenticated.
    - Shows `MainScaffold` or sitter dashboards when authenticated.
- **User state**
  - Exposed via Riverpod provider(s).
  - Used to scope Firestore queries to the current user (e.g., pets where `ownerId == user.id`).

7.1 Core Feature Areas – Pets & Care Plans

- **Pets**
  - CRUD via `EditPetPage` and `PetDetailPage`.
  - Photo upload using Firebase Storage; metadata stored in `Pet` document.
  - Additional fields: emergency contacts, medical history, behavior notes.
- **Care Plans**
  - One primary care plan per pet (current assumption).
  - Contains diet, feeding schedule, meds, special instructions.
  - Feeding/med times feed into local notification scheduling logic.

7.2 Core Feature Areas – Sharing & Handoffs

- **AccessToken system**
  - Each token binds a pet to a sitter with a role and expiry.
  - Used to authorize sitter views and actions (task completion, notes).
- **QR / link sharing**
  - Access tokens encoded into short URLs.
  - `qr_flutter` used to render QR codes.
- **Sitter Dashboard**
  - Task list for sitter, based on care plan.
  - Allows checking off tasks -> writes `TaskCompletion`.

7.3 Core Feature Areas – Lost & Found

- **Lost mode**
  - Owner can mark a pet as `isLost = true`.
  - Creates a `LostReport` with last location and notes.
- **Poster generation**
  - Service composes a poster (image) and uploads to Firebase Storage.
  - `posterUrl` stored on the `LostReport`.
- **Public sharing**
  - Generated link / QR for public posters.
  - Firestore rules restrict modification to the owner only.

8. Security & Access Control

- **Authentication**
  - All owner and sitter actions are tied to a Firebase user.
- **Authorization**
  - Firestore rules enforce:
    - Owners can only see/manage their own pets and related documents.
    - Sitter access is mediated via `AccessToken` documents.
    - Public views have read-only access and are scoped by token role.
- **Data privacy**
  - Shared tokens are time-boxed by `expiresAt`.
  - Sensitive fields are omitted from public views where appropriate.

8.1 Firestore Rules (Highlights)

- Rules (see `firestore.rules`) enforce:
  - `read/write` on `/pets` where `request.auth.uid == resource.data.ownerId`.
  - Read-only token-based access for sitter/public views.
  - Write access to `TaskCompletion` only for authenticated users with valid token.
  - Restricted access to `/lostReports` to owner or token holders where required.
- **Engineers should**:
  - Update rules in sync with schema changes.
  - Keep public surface area minimal.

9. Testing & Quality

- **Automated tests**
  - Tests live under `test/`:
    - `unit/` – pure Dart / business logic.
    - `widget/` – widget-level tests.
    - `integration/` – flows involving Firebase / navigation.
  - 457 tests, 100% pass rate
- **Build & test harness**
  - `test/run_tests.dart` and `test/test_config.dart` for project-specific setup.
  - CI is expected to run `flutter test` across these suites.

9.1 Error Handling & Resilience

- **Centralized error handling**
  - `services/error_handler.dart` provides a shared mechanism to:
    - Log or report errors (e.g., to Firebase Crashlytics in future).
    - Map domain and network errors to user-friendly messages.
- **UX patterns**
  - Use Riverpod `AsyncValue` error states to show:
    - Snackbars/toasts for transient errors.
    - Inline retry buttons or error placeholders.
  - Avoid silent failures; prefer explicit user feedback.

10. Known Gaps & Future Work

- **Navigation**
  - Migrate to `go_router` for declarative routing and deep-linking.
- **Messaging**
  - Owner–sitter chat system using Firestore collections.
  - UI components under a new `messaging/` feature module.
- **Push notifications**
  - Integrate FCM for:
    - Task completion events.
    - Lost pet updates.

10.1 Future Work (Cont.)

- **Weight tracking**
  - `WeightEntry` model and charting UI under pets.
- **Offline & caching**
  - Use `hive` or similar for offline profiles and reminders.
- **UX & accessibility**
  - Dark mode theming.
  - Accessibility improvements (larger text, screen reader labels).
- **Refactoring**
  - Consolidate routing once `go_router` is adopted.
  - Keep feature modules decoupled, with shared abstractions in `core/` and `services/`.

11. How to Extend Safely

- **When adding features**
  - Reuse existing repositories and providers where possible.
  - Respect Firestore rules and update them with any new collections/fields.
  - Store shared UI in `core/widgets` or `app/widgets`.
- **When changing schemas**
  - Update:
    - Firestore rules.
    - Repositories and DTOs.
    - Test fixtures under `test/`.
- **When in doubt**
  - Prefer additive changes over breaking changes.

12. Summary

- Petfolio is a Flutter + Firebase app with:
  - Clear feature modules.
  - Riverpod-based state management.
  - Strong sharing and handoff model (owners ↔️ sitters).
  - A foundation for real-time, multi-user pet care.
- This document should be kept in sync with:
  - `README.md`
  - Firestore rules and domain models in code.