Individual Project Sprint 2 Retrospective & Plan

Owen Newberry — Forge Workout

Sprint 2 Overview

- **Sprint length**: 5 weeks
- **Total features**: 2
- **Total requirements**: 2
- Focus: athlete workout logging and recurring workout scheduling (moved from Sprint 1).

Sprint 2 Features

- Workout logging — athletes record sets (weight, reps, notes) and persist to the backend.
- Recurring workout scheduling — define repeatable workout patterns (daily/weekly/custom), apply templates to athletes, and auto-generate plans.

Sprint 2 Requirements (Acceptance Criteria)

- As an athlete, I should be able to log my workout (weight, reps, etc.) so that I can save my data to the database.
    - Acceptance: Log form validates input, returns success, and saved record is visible in history within the app.
- As a trainer, I should be able to create and assign recurring workout schedules so that plans auto-populate for athletes on scheduled days.
    - Acceptance: Trainer can define recurrence rules (daily/weekly/custom), assign to athletes or templates, and the system generates plan records for the target dates.

Sprint 2 Weekly Milestones

- **Week 1 — Finalize workout & schedule data models**
  - Deliver: final `logs` and `schedules` schema, API endpoints for create/list, unit tests for serialization.
  - Tasks: review plan/exercise/log shapes; design `schedule` relation and recurrence storage (RFC-like rules or simplified fields).
- **Week 2 — Build workout logging UI**
  - Deliver: logging screen with per-set inputs, validation, and save flow.
  - Tasks: wire UI → backend, add local prefill (SharedPreferences), handle edge parsing.
- **Week 3 — Implement recurring scheduling backend**
  - Deliver: schedule creation endpoint, recurrence expansion logic (generate plans for upcoming window).

Architecture & Implementation Notes

- Frontend: Flutter (shared codebase for web/mobile/desktop where feasible).
- Backend: PocketBase (SQLite) for MVP speed; collections: `users`, `templates`, `plans`, `logs`, `schedules`, `videos`.
- Important shapes:
  - `log` record: `{ athlete, plan, exerciseId, sets (JSON string), createdAt }`.
  - `schedule` record: `{ trainer, athletes, templateId, rule (simple RFC-like recurrence string), startDate, endDate }`.
- Robustness techniques:
  - Defensive parsing for `sets` field to handle double-encoded JSON and locale number formats.
  - Client-side fallback for list endpoints: fetch recent items and filter locally when

Testing Strategy

- Unit tests: `sets` parsing, createLog serialization, schedule recurrence expansion logic, client-side filters.
- Widget tests: `LogEntry` prefill, logging form validation, schedule creation UI.
- Network/mocked tests: simulate server behavior for schedule creation, recurrence expansion and list fallbacks (using `MockClient`).
- Integration tests: run end-to-end flows against a disposable PocketBase instance (Docker) to confirm persistence and generated plans.

Demo & Acceptance Flow

1. Trainer creates a recurring schedule (e.g., "Upper body M/W/F") and assigns a template.
2. System generates plan records for the upcoming period (preview shows generated dates).
3. Athlete opens today's plan, performs logged sets, and saves — log appears in the database and is visible in the athlete's history.
4. Trainer reviews generated schedule and resolves conflicts (if any).

Risks & Mitigations

- Complex recurrence rules and calendar edge-cases (DST, leap days, exceptions).
  - Mitigation: start with a limited recurrence feature set (daily, weekly, custom by weekday) and expand only after stability.
- PocketBase list/filter edge-cases (400 responses) that can block plan/log retrieval.
  - Mitigation: client-side fallback + verbose debug logs; add targeted unit tests to reproduce and prevent regressions.
- UX complexity for trainers defining recurrence and resolving conflicts.
  - Mitigation: provide presets + clear previews and conflict warnings; ship a simple mode first.

Success Metrics (Sprint)

- Feature completion: implement 2/2 features.
- Acceptance tests passing: target 100% for defined requirements.
- Test coverage: raise unit/widget coverage for critical logic (parsing, schedule expansion).
- Reliability: schedule generation success rate > 98% in integration runs.

Retrospective — What Went Well

- Rapid iteration thanks to PocketBase and Flutter cross-platform tooling.
- Clear domain focus from DNA Sports Center informing feature prioritization.

Retrospective — Areas To Improve

- Harden server-side filters and relations handling to avoid unexpected 400s.
- Improve schedule rule testing and monitoring for edge-cases.