IRON TRACKER

Implementation Plan v1.0

Step-by-Step Build Guide with Comprehensive Test Coverage

February 2026 | Confidential

Table of Contents

1. Implementation Overview

This document breaks the Iron Tracker build into 12 sequential milestones, each decomposed into individually testable steps. Every step includes explicit acceptance criteria and a list of tests that must pass before the step is considered complete. Steps are small enough to complete in 1-3 days. Milestones represent shippable increments that can be demonstrated to stakeholders.

1.1 Milestone Summary

#	Milestone	Steps	Duration	Cumulative
M1	Project Scaffolding & CI/CD	6	Week 1	Week 1
M2	Database Schema & Auth	8	Week 2	Week 2
M3	Exercise Library API & UI	7	Week 3	Week 3
M4	Equipment Variant System	8	Weeks 3-4	Week 4
M5	Set Logging Core	10	Weeks 4-5	Week 5
M6	Rest Timer & Logging UX	6	Week 5-6	Week 6
M7	Session Grouping & History	7	Week 6-7	Week 7
M8	Progress Tracking & PRs	9	Weeks 7-8	Week 8
M9	Analytics & Visualization	8	Weeks 9-10	Week 10
M10	Offline-First & PWA	7	Weeks 10-11	Week 11
M11	AI Machine Identification	7	Weeks 11-12	Week 12
M12	AI Coaching & Recommendations	8	Weeks 13-16	Week 16

1.2 Testing Strategy

Unit Tests (Backend): pytest + pytest-asyncio. Every FastAPI endpoint, Pydantic model, and service function. Mock supabase-py calls. Target: 90%+ line coverage on business logic modules.

Unit Tests (Frontend): Vitest + React Testing Library. Every component render, user interaction, and hook behavior. Mock TanStack Query and Supabase client. Target: 85%+ line coverage.

Integration Tests (Backend): pytest with a test Supabase project (dedicated free-tier instance). Real DB operations, RLS verification, auth flow validation.

Integration Tests (Frontend): Vitest with MSW (Mock Service Worker) intercepting Supabase and FastAPI calls. Full user flows within a single screen.

End-to-End Tests: Playwright. Critical paths: sign up, log a set, view history, view chart. Run against the Netlify preview deploy + Render staging + Supabase test project. Target: 15-20 E2E scenarios covering happy paths and key error states.

Contract Tests: Pydantic model exports shared as TypeScript types via a codegen step (pydantic-to-typescript or openapi-typescript from FastAPI's OpenAPI spec). Frontend types are always in sync with backend schemas.

1.3 Conventions

Branch Strategy: main (production), develop (integration), feature/* (per-step branches). Each step merges to develop via PR with CI passing. Milestones merge develop to main and trigger a production deploy.

Commit Format: Conventional Commits: feat(M2-S3): implement RLS policies for sets table.

Test File Naming: Backend: tests/test_{module}.py. Frontend: {Component}.test.tsx or {hook}.test.ts adjacent to the source file.

Step Completion Criteria: Code reviewed, all tests passing in CI, no regression in existing tests, acceptance criteria verified manually or via E2E test.

2. Milestone 1 — Project Scaffolding & CI/CD

Goal: A deployable skeleton with empty screens, CI pipelines running lint and tests on every push, and preview deploys for every PR. No business logic yet.

M1-S1 Initialize Backend Repository

Create the FastAPI project skeleton with a modular folder structure, pyproject.toml for dependency management, and a health endpoint.

Acceptance Criteria:

- ✓ GET /api/health returns {"status": "ok"} with 200
- ✓ Project uses Python 3.12+, FastAPI 0.115+, Pydantic v2, pydantic-settings
- ✓ Folder structure: backend/app/{auth,ai,analytics,cron,core}/ each with __init__.py
- ✓ backend/app/core/config.py uses pydantic-settings BaseSettings with .env loading
- ✓ requirements.txt or pyproject.toml locked with all deps

Test Coverage:

- ▶ Unit: test_health_endpoint returns 200 and correct JSON body
- ▶ Unit: test_config_loads_from_env verifies BaseSettings reads SUPABASE_URL, SUPABASE_KEY, ANTHROPIC_API_KEY with defaults/validation
- ▶ Unit: test_config_missing_required_raises raises ValidationError when required env vars are absent
- ▶ Unit: test_cors_middleware_configured verifies CORS allows configured origins

M1-S2 Initialize Frontend Repository

Create the React + Vite + TypeScript project with TanStack Router (file-based), TanStack Query, MUI with a custom MD3 theme, and stub pages for each bottom nav tab.

Acceptance Criteria:

- ✓ Vite dev server starts without errors
- ✓ TanStack Router file-based routing configured with routes for /log, /history, /stats, /profile
- ✓ MUI ThemeProvider wraps the app with dark-mode-default MD3 color tokens
- ✓ Bottom NavigationBar renders four tabs (Log, History, Stats, Profile) and navigates correctly
- ✓ QueryClientProvider wraps the app with default staleTime and gcTime configuration
- ✓ TypeScript strict mode enabled, no type errors

Test Coverage:

- ▶ Unit: test_app_renders_without_crash — renders <App /> and verifies bottom nav is present
- ▶ Unit: test_bottom_nav_tabs — verifies 4 tabs render with correct labels (Log, History, Stats, Profile)
- ▶ Unit: test_navigation — clicking each tab navigates to the correct route
- ▶ Unit: test_dark_theme_default — verifies MUI theme palette.mode === 'dark'
- ▶ Unit: test_query_client_configured — verifies QueryClient defaults are set
- ▶ Snapshot: test_theme_tokens — snapshot of the theme object to catch accidental changes

M1-S3 Configure CI Pipeline (Backend)

Set up GitHub Actions workflow for the backend: install deps, run pytest, run ruff linter, run mypy type checker on every push and PR.

Acceptance Criteria:

- ✓ Workflow file at .github/workflows/backend-ci.yml
- ✓ Runs on push to develop and main, and on all PRs
- ✓ Steps: checkout, setup Python 3.12, install deps, ruff check, mypy, pytest with coverage report
- ✓ Fails the build on any lint error, type error, or test failure
- ✓ Coverage report uploaded as artifact

Test Coverage:

- ▶ CI Meta: push a deliberately failing test and verify CI reports failure
- ▶ CI Meta: push a ruff violation and verify CI reports failure
- ▶ CI Meta: push clean code and verify CI reports success with coverage artifact

M1-S4 Configure CI Pipeline (Frontend)

Set up GitHub Actions workflow for the frontend: install deps, run Vitest, run ESLint, run TypeScript type check on every push and PR.

Acceptance Criteria:

- ✓ Workflow file at .github/workflows/frontend-ci.yml
- ✓ Runs on push to develop and main, and on all PRs
- ✓ Steps: checkout, setup Node 20, npm ci, eslint, tsc --noEmit, vitest run with coverage
- ✓ Fails the build on any lint error, type error, or test failure
- ✓ Coverage report uploaded as artifact

Test Coverage:

- ▶ CI Meta: push a deliberately failing test and verify CI reports failure
- ▶ CI Meta: push an ESLint violation and verify CI reports failure
- ▶ CI Meta: push clean code and verify CI reports success

M1-S5 Netlify & Render Deploy Configuration

Configure Netlify for frontend preview deploys on PRs and production deploy on main. Configure Render for the backend with environment variables.

Acceptance Criteria:

- ✓ Netlify site connected to the repo with build command: cd frontend && npm ci && npm run build
- ✓ Netlify publish directory: frontend/dist
- ✓ Netlify preview deploys generate a URL for every PR
- ✓ Render web service connected to the repo with root directory: backend
- ✓ Render start command: uvicorn app.main:app --host 0.0.0.0 --port $PORT
- ✓ Environment variables configured on both platforms (no secrets in repo)
- ✓ CORS configured on backend to allow Netlify preview URLs and production URL

Test Coverage:

- ▶ Smoke: deploy to Netlify preview, verify the app loads and bottom nav renders

▶ Smoke: deploy to Render, verify GET /api/health returns 200
▶ Smoke: verify CORS — frontend preview can call backend health endpoint without CORS errors

M1-S6 Supabase Project Setup

Create the Supabase project, configure auth providers, and verify connectivity from both frontend and backend.

Acceptance Criteria:

✓ Supabase project created with PostgreSQL 15
✓ Email auth provider enabled
✓ Google OAuth provider configured (client ID/secret)
✓ Supabase URL and anon key added to frontend env
✓ Supabase URL and service role key added to backend env
✓ @supabase/supabase-js initialized in frontend with createClient()
✓ supabase-py async client initialized in backend lifespan event

Test Coverage:

▶ Integration: test_supabase_frontend_connection — frontend can call supabase.auth.getSession() without error
▶ Integration: test_supabase_backend_connection — backend can call supabase.table('_test_ping').select('*') (create and drop a temp test table)
▶ Integration: test_auth_signup_flow — create a test user via Supabase Auth, verify the user exists, delete the user

3. Milestone 2 — Database Schema & Auth

Goal: All core tables created in Supabase with RLS policies, a working auth flow (signup, login, logout, token refresh) end-to-end, and the FastAPI JWT verification dependency.

M2-S1 Create exercises Table & Seed Data

Create the exercises table for the canonical exercise library. Populate with ~400 seed exercises covering all major muscle groups and equipment types. Exercises are publicly readable but only admins (or the system) can insert seed data.

Acceptance Criteria:

- ✓ Table exercises created with columns: id (uuid PK), name, category, primary_muscles (text[]), secondary_muscles (text[]), movement_type, is_custom, created_by (nullable FK to auth.users), created_at
- ✓ RLS enabled: SELECT allowed for all authenticated users, INSERT/UPDATE/DELETE restricted to rows where created_by = auth.uid() (custom exercises) or via service role (seed data)
- ✓ Seed SQL file inserts ~400 exercises with correct muscle group mappings
- ✓ Indexes on: name (for search), category, primary_muscles (GIN for array containment)

Test Coverage:

- ▶ Integration: test_exercises_readable_by_authenticated — authenticated user can SELECT all seeded exercises
- ▶ Integration: test_exercises_not_readable_by_anon — unauthenticated request returns 0 rows or 401
- ▶ Integration: test_seed_data_count — verify ≥400 exercises exist after seed
- ▶ Integration: test_seed_covers_all_categories — verify exercises exist for push, pull, legs, core, cardio
- ▶ Integration: test_search_by_name — SELECT WHERE name ILIKE '%bench%' returns expected results
- ▶ Integration: test_user_cannot_modify_seed — UPDATE/DELETE on seed exercise (created_by IS NULL) fails for normal user

M2-S2 Create equipment_variants Table

Create the equipment_variants table that stores machine-specific instances of exercises, per user.

Acceptance Criteria:

- ✓ Table equipment_variants with columns: id (uuid PK), user_id (FK), exercise_id (FK), name, equipment_type, manufacturer, weight_increment, weight_unit, seat_settings (jsonb), notes, photo_url, is_default, last_used_at, created_at
- ✓ RLS: all operations restricted to rows where user_id = auth.uid()
- ✓ Foreign key to exercises(id) with ON DELETE CASCADE
- ✓ Unique constraint on (user_id, exercise_id, name) to prevent duplicate variant names per user per exercise
- ✓ Index on (user_id, exercise_id, last_used_at DESC) for MRU sorting

Test Coverage:

▶ Integration: test_create_variant — authenticated user can INSERT a variant linked to an exercise

▶ Integration: test_variant_isolation — user A cannot SELECT user B's variants

▶ Integration: test_variant_update — owner can UPDATE their variant's name and seat_settings

▶ Integration: test_variant_delete — owner can DELETE their variant

▶ Integration: test_duplicate_name_rejected — INSERT same (user_id, exercise_id, name) returns unique constraint error

▶ Integration: test_cascade_on_exercise_delete — deleting a custom exercise cascades to its variants

▶ Integration: test_jsonb_seat_settings — INSERT/SELECT with complex JSONB ({seat_height: 3, back_pad: 2}) works correctly

M2-S3 Create sets Table

Create the sets table — the atomic unit of training data. This is the most critical table for performance and must support rapid inserts and time-range queries.

Acceptance Criteria:

✓ Table sets with columns: id (uuid PK), user_id (FK), exercise_id (FK), variant_id (FK nullable), weight (decimal), weight_unit, reps (int), rpe (decimal nullable), rir (int nullable), set_type, rest_seconds (int nullable), duration_seconds (int nullable), tempo (text nullable), notes (text nullable), logged_at (timestamptz), created_at, session_group (date, generated always as DATE(logged_at))

✓ RLS: all operations restricted to user_id = auth.uid()

✓ Foreign keys to exercises(id) and equipment_variants(id), both with ON DELETE SET NULL for variant_id

✓ Indexes: (user_id, logged_at DESC), (user_id, exercise_id, logged_at DESC), (user_id, session_group)

✓ CHECK constraints: weight >= 0, reps >= 0, rpe BETWEEN 6.0 AND 10.0 OR NULL, rir BETWEEN 0 AND 5 OR NULL

Test Coverage:

▶ Integration: test_insert_set — authenticated user can INSERT a set with required fields

▶ Integration: test_insert_set_with_variant — set linked to a valid variant_id succeeds

▶ Integration: test_insert_set_without_variant — set with variant_id = NULL succeeds (generic logging)

▶ Integration: test_set_isolation — user A cannot SELECT user B's sets

▶ Integration: test_set_query_by_exercise — SELECT WHERE exercise_id = X ORDER BY logged_at DESC returns correct results

▶ Integration: test_set_query_by_session_group — SELECT WHERE session_group = '2026-02-24' returns all sets from that date

▶ Integration: test_check_constraint_weight — INSERT with weight = -5 fails

▶ Integration: test_check_constraint_rpe — INSERT with rpe = 11 fails, rpe = 8.5 succeeds

▶ Integration: test_session_group_generated — INSERT a set at 2026-02-24T23:30:00Z, verify session_group = 2026-02-24

▶ Integration: test_variant_fk_set_null — DELETE variant, verify set's variant_id becomes NULL (not cascade delete)

▶ Performance: test_insert_latency — INSERT 100 sets in sequence, verify avg latency < 50ms

M2-S4 Create personal_records Table & Trigger

Create the personal_records table and a PostgreSQL trigger function that checks for new PRs on every set insert.

Acceptance Criteria:

- ✓ Table personal_records with columns: id (uuid PK), user_id (FK), exercise_id (FK), variant_id (FK nullable), record_type (text: est_1rm, rep_max_1, rep_max_3, rep_max_5, rep_max_8, rep_max_10, max_volume), value (decimal), set_id (FK to sets), achieved_at (timestamptz), created_at
- ✓ RLS: SELECT/DELETE for user_id = auth.uid(), INSERT/UPDATE via trigger only (service role)
- ✓ Unique constraint on (user_id, exercise_id, variant_id, record_type) for upsert semantics
- ✓ Trigger function check_personal_record() fires AFTER INSERT on sets: computes est_1rm via Epley, compares against existing PR, upserts if beaten
- ✓ Trigger also checks rep_max records: e.g., if reps=5 and weight>current 5RM record

Test Coverage:

- ▶ Integration: test_first_set_creates_pr — INSERT a set for a new exercise, verify personal_records rows created for est_1rm and rep_max
- ▶ Integration: test_better_set_updates_pr — INSERT a set with higher est_1rm than existing, verify PR updated with new set_id
- ▶ Integration: test_worse_set_no_pr_change — INSERT a set with lower est_1rm, verify PR unchanged
- ▶ Integration: test_rep_max_tracking — INSERT set at 100kgx5, then 105kgx5, verify rep_max_5 updated to 105
- ▶ Integration: test_volume_record — INSERT set at 100kgx10 (vol=1000), then 80kgx15 (vol=1200), verify max_volume updated
- ▶ Integration: test_variant_specific_pr — PR for variant A does not affect PR for variant B of the same exercise
- ▶ Unit: test_epley_formula — verify e1RM calculation: 100kg x 8 reps = 100 * (1 + 8/30) = 126.67
- ▶ Unit: test_brzycki_formula — verify fallback for >12 reps: weight * 36 / (37 - reps)

M2-S5 Create soreness_reports Table

Create the soreness_reports table for post-workout muscle soreness tracking (carried forward from v0).

Acceptance Criteria:

- ✓ Table soreness_reports with columns: id (uuid PK), user_id (FK), muscle_group (text), soreness_level (int 0-4), reported_at (timestamptz), session_group (date, referencing which workout triggered the prompt), created_at
- ✓ RLS: all operations restricted to user_id = auth.uid()
- ✓ CHECK constraint: soreness_level BETWEEN 0 AND 4

Test Coverage:

- ▶ Integration: test_insert_soreness — authenticated user can report soreness for 'chest' at level 3
- ▶ Integration: test_soreness_isolation — user A cannot see user B's reports
- ▶ Integration: test_invalid_level_rejected — INSERT with soreness_level = 5 fails
- ▶ Integration: test_query_recent_soreness — SELECT WHERE user_id = X AND reported_at > now() - interval '7 days' returns recent reports

M2-S6 Create analytics_cache Table

Create the analytics_cache table for server-side pre-computed analytics (JSONB payloads with type discriminator).

Acceptance Criteria:

- ✓ Table analytics_cache with columns: id (uuid PK), user_id (FK), cache_type (text: weekly_volume, exercise_1rm_trend, muscle_distribution, training_frequency, strength_score), payload (jsonb), period_start (date), period_end (date), computed_at (timestamptz), created_at
- ✓ RLS: SELECT for user_id = auth.uid(), INSERT/UPDATE/DELETE via service role only (backend cron)
- ✓ Unique constraint on (user_id, cache_type, period_start, period_end)
- ✓ Index on (user_id, cache_type, period_end DESC)

Test Coverage:

- ▶ Integration: test_cache_insert_via_service_role — service role can INSERT analytics cache entry
- ▶ Integration: test_cache_readable_by_owner — authenticated user can SELECT their own cache entries
- ▶ Integration: test_cache_not_writable_by_user — normal authenticated user cannot INSERT/UPDATE cache entries
- ▶ Integration: test_cache_upsert — second INSERT with same unique key upserts (ON CONFLICT UPDATE)

M2-S7 Implement FastAPI JWT Verification

Create the get_current_user dependency that verifies Supabase JWTs using JWKS (asymmetric signing). This dependency will protect all authenticated backend endpoints.

Acceptance Criteria:

- ✓ Dependency function get_current_user(authorization: str = Header()) extracts Bearer token
- ✓ Fetches Supabase JWKS from {SUPABASE_URL}/auth/v1/.well-known/jwks.json (cached for 1 hour)
- ✓ Verifies JWT signature, expiry, and issuer using python-jose or PyJWT
- ✓ Extracts user_id from the sub claim
- ✓ Returns a UserContext(user_id=uuid, email=str) dataclass
- ✓ Raises 401 on invalid/expired token, 403 on missing token

Test Coverage:

- ▶ Unit: test_valid_jwt_returns_user — mock JWKS, create a valid JWT, verify UserContext returned with correct user_id
- ▶ Unit: test_expired_jwt_raises_401 — mock JWKS, create an expired JWT, verify HTTPException(401)
- ▶ Unit: test_invalid_signature_raises_401 — sign JWT with wrong key, verify HTTPException(401)
- ▶ Unit: test_missing_auth_header_raises_403 — call without Authorization header, verify HTTPException(403)
- ▶ Unit: test_malformed_token_raises_401 — pass 'Bearer garbage' header, verify HTTPException(401)
- ▶ Unit: test_jwks_caching — two calls within 1 hour make only one HTTP request to JWKS endpoint
- ▶ Integration: test_full_auth_flow — sign up via Supabase, get access token, call a protected endpoint, verify 200

M2-S8 Implement Frontend Auth Flow

Build the login, signup, and session management UI using @supabase/supabase-js, with a React context for auth state and a ProtectedRoute wrapper.

Acceptance Criteria:

- ✓ AuthProvider context wraps the app, exposes: user, session, loading, signUp, signIn, signOut
- ✓ AuthProvider listens to onAuthStateChange and updates state reactively
- ✓ Login screen with email/password fields and a Google OAuth button
- ✓ Signup screen with email, password, confirm password fields
- ✓ ProtectedRoute component redirects to /login if no session
- ✓ All four main tabs wrapped in ProtectedRoute
- ✓ Access token attached to Supabase client automatically (supabase-js handles this)
- ✓ Token refresh handled automatically by supabase-js

Test Coverage:

- ▶ Unit: test_auth_provider_initial_state — renders with loading=true, then resolves to user=null if no session
- ▶ Unit: test_sign_up_calls_supabase — mock supabase.auth.signUp, verify it's called with correct params
- ▶ Unit: test_sign_in_calls_supabase — mock supabase.auth.signInWithPassword, verify correct params
- ▶ Unit: test_sign_out — mock supabase.auth.signOut, verify user state becomes null
- ▶ Unit: test_protected_route_redirects — render ProtectedRoute with no session, verify redirect to /login
- ▶ Unit: test_protected_route_renders_children — render ProtectedRoute with mock session, verify children render
- ▶ Unit: test_auth_state_change — simulate onAuthStateChange callback, verify context updates
- ▶ E2E: test_signup_login_logout — Playwright: sign up with email, verify redirect to /log, sign out, verify redirect to /login

4. Milestone 3 — Exercise Library API & UI

Goal: Users can browse, search, and view exercises. The seeded exercise library is fully navigable. Custom exercise creation works. This milestone builds the foundation for everything that follows.

M3-S1 Exercise List API (Supabase Direct)

Since exercise reads go directly via the Supabase JS client (Pattern A), this step creates the TanStack Query hooks and the API layer for fetching exercises.

Acceptance Criteria:

- ✓ useExercises() hook returns paginated exercises with staleTime: 1 hour
- ✓ useExerciseSearch(query) hook returns filtered exercises with debounced search (200ms)
- ✓ Supabase query: supabase.from('exercises').select('*').ilike('name', `%${query}%`).order('name').range(offset, offset+limit)
- ✓ Query keys follow factory pattern: exerciseKeys.list(filters), exerciseKeys.detail(id)
- ✓ TypeScript types for Exercise match the Supabase table schema exactly

Test Coverage:

- ▶ Unit: test_useExercises_returns_data — mock Supabase, verify hook returns exercises array
- ▶ Unit: test_useExercises_loading_state — verify isLoading=true initially, then false after data
- ▶ Unit: test_useExerciseSearch_debounces — rapid typing triggers only one query after 200ms pause
- ▶ Unit: test_useExerciseSearch_empty_query — empty string returns all exercises (no filter)
- ▶ Unit: test_exercise_type_matches_schema — TypeScript compile-time check that Exercise type includes all table columns
- ▶ Integration: test_exercises_fetch_from_supabase — real Supabase call returns seeded exercises

M3-S2 Exercise List Screen UI

Build the Exercise List screen (Log tab home) with recent exercises, muscle group sections, search bar, and alphabetical list.

Acceptance Criteria:

- ✓ TopAppBar with 'Log' title and camera icon button (placeholder for AI, disabled for now)
- ✓ SearchBar component below app bar with real-time filtering
- ✓ Recent section: horizontal scrolling cards showing last 5-8 logged exercises (empty for new users)
- ✓ By Muscle Group section: collapsible accordion groups (Chest, Back, Shoulders, Arms, Legs, Core)
- ✓ All Exercises section: alphabetical list with sticky alphabet scrubber
- ✓ Empty state for new users with illustration and CTA button
- ✓ Tapping an exercise navigates to /log/:exerciseId

Test Coverage:

- ▶ Unit: test_search_bar_renders — verify search input is present with placeholder
- ▶ Unit: test_search_filters_list — type 'bench' in search, verify only bench-related exercises shown
- ▶ Unit: test_muscle_group_accordion — click 'Chest' header, verify chest exercises expand

▶ Unit: test_exercise_tap_navigates — click an exercise item, verify router navigated to /log/:id

▶ Unit: test_empty_state — render with no recent exercises, verify empty state message and CTA shown

▶ Unit: test_recent_section_horizontal_scroll — verify recent section renders in a horizontal scrollable container

▶ Snapshot: test_exercise_list_item_snapshot — snapshot of a single exercise list item with all fields

M3-S3 Custom Exercise Creation

Allow users to create custom exercises that appear alongside seeded ones. Custom exercises have is_custom=true and created_by=auth.uid().

Acceptance Criteria:

✓ Create Exercise FAB or button accessible from the exercise list (e.g., in 'No results' search state)

✓ Bottom sheet form: name (required), category (dropdown), primary muscles (multi-select chips), secondary muscles (multi-select chips), movement type (compound/isolation toggle)

✓ On save: INSERT to exercises table with is_custom=true, created_by=auth.uid()

✓ New exercise appears immediately in the list (optimistic update)

✓ Validation: name is required, cannot duplicate existing exercise name for this user

Test Coverage:

▶ Unit: test_create_exercise_form_renders — verify all form fields present

▶ Unit: test_create_exercise_validation — submit without name shows error

▶ Unit: test_create_exercise_mutation — mock Supabase insert, verify correct payload sent

▶ Unit: test_optimistic_update — after mutation, verify exercise appears in list before server confirms

▶ Integration: test_custom_exercise_persists — create exercise, refresh page, verify it still appears

▶ Integration: test_custom_exercise_rls — user B cannot see user A's custom exercises

M3-S4 Exercise Detail / Info Screen

Tapping the info icon on an exercise shows its details: muscles targeted, movement type, and (eventually) form tips from AI.

Acceptance Criteria:

✓ Accessible via overflow menu on the Set Logger TopAppBar

✓ Displays: exercise name, category badge, primary muscles as colored chips, secondary muscles as outlined chips, movement type badge, equipment types commonly used

✓ For custom exercises: shows an Edit button and Delete button

Test Coverage:

▶ Unit: test_exercise_detail_renders — verify name, category, muscle chips render correctly

▶ Unit: test_custom_exercise_shows_edit — render with is_custom=true, verify Edit/Delete buttons visible

▶ Unit: test_seed_exercise_no_edit — render with is_custom=false, verify no Edit/Delete buttons

M3-S5 Backend Exercise Endpoints (Optional CRUD)

While most exercise queries go direct to Supabase, provide FastAPI endpoints for operations that benefit from server-side logic: bulk search with ranking, and exercise suggestions.

Acceptance Criteria:

- ✓ GET /api/exercises/search?q={query}&limit=20 returns exercises ranked by relevance (trigram similarity or ts_vector)
- ✓ Response uses Pydantic ExerciseResponse model matching the Supabase schema
- ✓ Endpoint requires valid JWT (get_current_user dependency)
- ✓ GET /api/exercises/{id} returns single exercise detail

Test Coverage:

- ▶ Unit: test_search_endpoint_returns_results — mock Supabase, verify correct response shape
- ▶ Unit: test_search_ranking — verify 'Bench Press' ranks higher than 'Incline Dumbbell Press' for query 'bench'
- ▶ Unit: test_search_empty_query — verify 400 or default results returned
- ▶ Unit: test_search_auth_required — call without JWT, verify 401
- ▶ Unit: test_exercise_response_model — verify Pydantic model serializes correctly

M3-S6 Pydantic-to-TypeScript Type Sync

Set up automatic TypeScript type generation from FastAPI's OpenAPI spec to keep frontend types in sync with backend schemas.

Acceptance Criteria:

- ✓ Script: npm run generate:types runs openapi-typescript against FastAPI's /openapi.json
- ✓ Output types saved to frontend/src/api/types.generated.ts
- ✓ Generated types used in all API call functions and TanStack Query hooks
- ✓ CI step: re-generate types and fail if there are uncommitted changes (drift detection)

Test Coverage:

- ▶ CI: test_types_in_sync — re-generate types, git diff --exit-code fails if types have drifted
- ▶ Unit: test_generated_types_compile — TypeScript compilation succeeds with generated types

M3-S7 Exercise Library E2E Test Suite

End-to-end tests covering the complete exercise browsing experience.

Acceptance Criteria:

- ✓ Playwright test file: tests/e2e/exercises.spec.ts
- ✓ Tests run against preview deploy with seeded Supabase data

Test Coverage:

- ▶ E2E: test_browse_exercises — login, navigate to Log tab, verify exercises are displayed, scroll through list
- ▶ E2E: test_search_exercises — type 'squat' in search bar, verify only squat-related exercises shown, clear search, verify full list restored
- ▶ E2E: test_create_custom_exercise — search for nonexistent name, click 'Create custom', fill form, save, verify it appears in list
- ▶ E2E: test_muscle_group_filter — expand 'Chest' group, verify chest exercises listed, collapse, verify hidden

5. Milestone 4 — Equipment Variant System

Goal: Users can create, edit, and delete equipment variants for any exercise. The variant chip row appears on the Set Logger screen with MRU sorting and progressive disclosure. This is the core differentiating feature.

M4-S1 Variant CRUD Hooks & API Layer

Create TanStack Query hooks for equipment variant CRUD operations via the Supabase JS client.

Acceptance Criteria:

- ✓ useVariants(exerciseId) hook returns all variants for an exercise, ordered by last_used_at DESC
- ✓ useCreateVariant() mutation hook inserts a new variant with optimistic update
- ✓ useUpdateVariant() mutation hook updates variant fields with optimistic update
- ✓ useDeleteVariant() mutation hook deletes with optimistic removal and confirmation dialog
- ✓ All mutations invalidate the variants query on settle
- ✓ TypeScript EquipmentVariant type matches the Supabase schema exactly

Test Coverage:

- ▶ Unit: test_useVariants_returns_data — mock Supabase, verify hook returns variants sorted by last_used_at
- ▶ Unit: test_useVariants_empty — exercise with no variants returns empty array
- ▶ Unit: test_useCreateVariant_optimistic — verify variant appears in cache immediately before server response
- ▶ Unit: test_useCreateVariant_rollback — simulate server error, verify optimistic entry removed from cache
- ▶ Unit: test_useUpdateVariant — mock update, verify cache updated with new data
- ▶ Unit: test_useDeleteVariant — mock delete, verify variant removed from cache

M4-S2 Variant Chip Row Component

Build the horizontal scrollable chip row that appears below the exercise name in the Set Logger, showing equipment variants as MD3 FilterChips.

Acceptance Criteria:

- ✓ VariantChipRow accepts: variants[], selectedId, onSelect, onAddClick
- ✓ Renders MD3 FilterChip for each variant (filled when selected, outlined otherwise)
- ✓ Most-recently-used variant is first in the row
- ✓ Last chip is '+ Add' which calls onAddClick
- ✓ Row scrolls horizontally when variants exceed screen width
- ✓ Hidden entirely when variants.length <= 1 (progressive disclosure)
- ✓ When exactly 1 variant: it's auto-selected silently with no UI

Test Coverage:

- ▶ Unit: test_renders_chips — pass 3 variants, verify 3 chips + 1 Add chip rendered
- ▶ Unit: test_selected_chip_filled — verify selected variant chip has filled styling
- ▶ Unit: test_chip_click_calls_onSelect — click a chip, verify onSelect called with correct variant ID

▶ Unit: test_add_chip_click — click + Add chip, verify onAddClick called
▶ Unit: test_hidden_with_zero_variants — pass 0 variants, verify component renders nothing
▶ Unit: test_hidden_with_one_variant — pass 1 variant, verify component renders nothing (auto-selected)
▶ Unit: test_mru_order — pass variants with different last_used_at, verify most recent is first
▶ Unit: test_horizontal_scroll — pass 10 variants, verify overflow-x: auto on container
▶ Snapshot: test_chip_row_snapshot — snapshot with 3 variants, selected=first

M4-S3 Variant Creation Bottom Sheet

Build the bottom sheet form for creating a new equipment variant, with all fields from the data model.

Acceptance Criteria:

✓ SwipeableDrawer anchored to bottom, draggable, peeks at 60% height
✓ Form fields: name (required), equipment_type (select), manufacturer (autocomplete), weight_increment (number + presets), weight_unit (toggle), seat_settings (dynamic key-value), notes (textarea), photo (file picker)
✓ Save button calls useCreateVariant mutation
✓ Cancel button closes the sheet
✓ Validation: name is required, shows inline error if empty
✓ After save: sheet closes, new variant appears in chip row and is auto-selected

Test Coverage:

▶ Unit: test_sheet_opens — trigger open, verify drawer is visible
▶ Unit: test_sheet_closes_on_cancel — click Cancel, verify drawer closes
▶ Unit: test_name_required_validation — submit without name, verify error message shown
▶ Unit: test_save_calls_mutation — fill all fields, click Save, verify useCreateVariant called with correct payload
▶ Unit: test_equipment_type_options — verify dropdown contains all equipment_type enum values
▶ Unit: test_seat_settings_dynamic — click 'Add Setting', verify a new key-value row appears
▶ Unit: test_seat_settings_remove — add a setting then remove it, verify it disappears
▶ Unit: test_weight_increment_presets — verify preset buttons for 2.5, 5, 10 appear and populate the field

M4-S4 Variant Edit and Delete

Allow editing existing variants via the same bottom sheet form, and deleting with a confirmation dialog.

Acceptance Criteria:

✓ Long-press or tap on a variant chip opens the edit bottom sheet pre-filled with current data
✓ Save updates the variant via useUpdateVariant mutation
✓ Delete button in edit mode shows a confirmation AlertDialog
✓ On delete: variant removed, chip row updates, if deleted variant was selected the next MRU variant is auto-selected

Test Coverage:

▶ Unit: test_edit_prefills_fields — open edit for a variant, verify all fields pre-filled with existing data

▶ Unit: test_edit_saves_changes — modify name, click Save, verify useUpdateVariant called with new name

▶ Unit: test_delete_shows_confirmation — click Delete, verify AlertDialog appears

▶ Unit: test_delete_confirmed — confirm deletion, verify useDeleteVariant called

▶ Unit: test_delete_cancelled — dismiss dialog, verify variant still exists

▶ Unit: test_auto_select_after_delete — delete the selected variant, verify next MRU variant becomes selected

M4-S5 Variant last_used_at Update

Automatically update a variant's last_used_at timestamp when a set is logged using that variant. This drives MRU sorting.

Acceptance Criteria:

✓ After a successful set insert, the variant's last_used_at is updated to now()

✓ This can be a Supabase trigger (AFTER INSERT ON sets) or part of the frontend mutation

✓ Variant query cache is invalidated or optimistically updated to reflect the new timestamp

Test Coverage:

▶ Integration: test_set_insert_updates_variant_timestamp — insert a set, verify variant's last_used_at is updated

▶ Unit: test_mru_resort_after_log — log a set on variant B, verify variant B moves to first position in chip row

M4-S6 Multi-Variant Progress Toggle (Stub)

Add the SegmentedButton toggle to the exercise chart view that switches between 'All Variants' and individual variant views. The chart itself is a stub at this milestone (implemented in M8-M9), but the toggle and data filtering logic are wired up.

Acceptance Criteria:

✓ SegmentedButton component with options: 'All Variants', then one segment per variant name

✓ Selection updates a filter state that will be consumed by chart components

✓ Default selection: 'All Variants'

Test Coverage:

▶ Unit: test_toggle_renders_segments — pass 3 variants, verify 4 segments (All + 3 variant names)

▶ Unit: test_toggle_default_all — verify 'All Variants' is selected by default

▶ Unit: test_toggle_selection_changes_filter — click a variant segment, verify filter state updates

M4-S7 Manufacturer Autocomplete

Build an autocomplete for the manufacturer field that suggests previously entered manufacturers across all of a user's variants.

Acceptance Criteria:

✓ Query: SELECT DISTINCT manufacturer FROM equipment_variants WHERE user_id = auth.uid() AND manufacturer IS NOT NULL

✓ Autocomplete component shows suggestions as user types, allows free-text entry

✓ Cached with staleTime: 10 minutes

Test Coverage:

- ▶ Unit: test_autocomplete_shows_suggestions — mock 3 manufacturers, type first letter, verify suggestions appear
- ▶ Unit: test_autocomplete_free_text — type a new manufacturer not in suggestions, verify it's accepted
- ▶ Unit: test_autocomplete_empty — no prior manufacturers, verify no suggestions but input still works

M4-S8 Equipment Variants E2E Test Suite

End-to-end tests covering the variant lifecycle.

Acceptance Criteria:

Test Coverage:

- ▶ E2E: test_create_first_variant — navigate to exercise, verify no chip row (0 variants), create variant via + Add, verify chip row appears with 1 chip (still hidden since count=1), create second variant, verify chip row now visible with 2 chips
- ▶ E2E: test_switch_variant — with 2 variants, select variant B, verify input fields reload with variant B's last data
- ▶ E2E: test_edit_variant — long-press a variant chip, modify the name, save, verify chip label updated
- ▶ E2E: test_delete_variant — long-press, delete with confirmation, verify chip removed from row

6. Milestone 5 — Set Logging Core

Goal: The complete set logging experience — pre-fill, stepper buttons, numpad, optimistic insert, set list, swipe-to-edit, swipe-to-delete. After this milestone, the app is usable for daily training.

M5-S1 Set Logger Screen Layout

Build the Set Logger screen skeleton with all four zones: variant chips (Zone A), input area (Zone B), action row (Zone C), and session sets list (Zone D).

Acceptance Criteria:

- ✓ Route /log/:exerciseId renders the Set Logger screen
- ✓ TopAppBar with exercise name and overflow menu
- ✓ Zone A: VariantChipRow (from M4)
- ✓ Zone B: weight and reps input fields with large 48px center-aligned font
- ✓ Zone C: Log Set extended FAB (full width, 56dp height)
- ✓ Zone D: empty list area (populated in M5-S5)
- ✓ Layout is responsive: thumb-zone optimized for one-handed use

Test Coverage:

- ▶ Unit: test_set_logger_renders_zones — verify all 4 zones are present
- ▶ Unit: test_exercise_name_in_appbar — verify TopAppBar shows the correct exercise name
- ▶ Unit: test_log_set_button_visible — verify FAB is present with 'Log Set' label
- ▶ Unit: test_weight_field_large_font — verify weight input has fontSize >= 48px
- ▶ Snapshot: test_set_logger_layout_snapshot

M5-S2 Pre-Fill from Last Set

When the Set Logger loads, query the user's most recent set for the current exercise + variant and pre-populate weight and reps.

Acceptance Criteria:

- ✓ useLastSet(exerciseId, variantId) hook queries: SELECT weight, reps, weight_unit, rpe FROM sets WHERE exercise_id=X AND variant_id=Y AND user_id=auth.uid() ORDER BY logged_at DESC LIMIT 1
- ✓ On data load: weight field pre-fills with lastSet.weight, reps field with lastSet.reps
- ✓ If no prior set: weight = '' (empty, cursor focused), reps = 10 (sensible default)
- ✓ Ghost text below inputs shows: 'Last: 3 days ago' or 'First set!' if no history
- ✓ When variant changes (chip switch): pre-fill updates to the new variant's last set

Test Coverage:

- ▶ Unit: test_prefill_from_last_set — mock last set at 100kg x 8, verify inputs show 100 and 8
- ▶ Unit: test_prefill_no_history — mock empty result, verify weight empty, reps 10
- ▶ Unit: test_prefill_ghost_text_recent — last set 2 days ago, verify 'Last: 2 days ago' shown
- ▶ Unit: test_prefill_ghost_text_first — no history, verify 'First set!' shown
- ▶ Unit: test_prefill_updates_on_variant_switch — switch variant, verify new pre-fill values loaded

M5-S3 Stepper Buttons

Add inline stepper buttons adjacent to the weight and reps fields for quick adjustment without opening the numpad.

Acceptance Criteria:

- ✓ Weight field: -5, -2.5 buttons on the left, +2.5, +5 buttons on the right
- ✓ Reps field: -1 button on the left, +1 button on the right
- ✓ Stepper values respect the variant's weight_increment if set (e.g., +5 for selectorized, +2.5 for plates)
- ✓ Tapping stepper updates the displayed value immediately
- ✓ Weight cannot go below 0, reps cannot go below 1
- ✓ Stepper buttons have 48dp minimum touch target with haptic feedback

Test Coverage:

- ▶ Unit: test_weight_plus_five — start at 100, tap +5, verify 105 displayed
- ▶ Unit: test_weight_minus_two_point_five — start at 100, tap -2.5, verify 97.5 displayed
- ▶ Unit: test_weight_floor_zero — start at 2, tap -5, verify 0 (not negative)
- ▶ Unit: test_reps_plus_one — start at 8, tap +1, verify 9
- ▶ Unit: test_reps_floor_one — start at 1, tap -1, verify 1 (not 0)
- ▶ Unit: test_custom_increment — variant has weight_increment=10, verify stepper uses +10/-10 instead of +5/-5
- ▶ Unit: test_touch_target_size — verify stepper button has min-width and min-height >= 48dp

M5-S4 Numpad Bottom Sheet

Build the numeric input bottom sheet that opens when the user taps on the weight or reps field for precise entry.

Acceptance Criteria:

- ✓ Half-screen SwipeableDrawer with drag handle
- ✓ Display area at top showing current value being edited
- ✓ Numpad grid: 0-9, decimal point (weight only), backspace
- ✓ Quick-select row at top: user's last 4 unique values for this field (e.g., last 4 weights)
- ✓ Done button closes the sheet and applies the value
- ✓ Tap outside the sheet also closes and applies
- ✓ For reps: decimal point key is hidden

Test Coverage:

- ▶ Unit: test_numpad_opens_on_weight_tap — tap weight field, verify numpad sheet visible
- ▶ Unit: test_numpad_opens_on_reps_tap — tap reps field, verify numpad sheet visible
- ▶ Unit: test_numpad_digit_input — tap 1, 0, 0, verify display shows '100'
- ▶ Unit: test_numpad_decimal — tap 6, 7, '.', 5, verify display shows '67.5'
- ▶ Unit: test_numpad_backspace — tap 1, 0, 0, backspace, verify display shows '10'
- ▶ Unit: test_numpad_done_applies_value — enter 105, tap Done, verify weight field shows 105
- ▶ Unit: test_numpad_reps_no_decimal — open numpad for reps, verify decimal button is not rendered
- ▶ Unit: test_quick_select — mock last 4 weights [100, 95, 90, 85], verify quick-select row shows these values

▶ Unit: test_quick_select_tap_applies — tap a quick-select value, verify it populates the display

M5-S5 Set Insert Mutation & Optimistic Update

Implement the core set logging mutation: on tapping 'Log Set', insert a new row into the sets table with immediate optimistic cache update.

Acceptance Criteria:

✓ useLogSet() mutation hook: inserts a set to Supabase with exercise_id, variant_id, weight, reps, weight_unit, logged_at=now()

✓ onMutate: immediately adds the new set to the session sets cache

✓ onError: rolls back the optimistic entry, shows an error snackbar

✓ onSettled: invalidates session sets query

✓ After successful log: the set appears in Zone D immediately, the weight/reps fields reset to the just-logged values (ready for another identical set)

✓ Haptic feedback (navigator.vibrate) on successful log

Test Coverage:

▶ Unit: test_log_set_mutation_payload — verify mutation sends correct fields to Supabase insert

▶ Unit: test_optimistic_set_appears — trigger mutation, verify set appears in Zone D before server resolves

▶ Unit: test_optimistic_rollback — simulate server error, verify optimistic set is removed and snackbar shown

▶ Unit: test_fields_reset_after_log — log a set, verify weight/reps fields still show the same values (ready for repeat)

▶ Unit: test_logged_at_timestamp — verify logged_at is set to current time (within 1 second tolerance)

▶ Integration: test_set_persists_in_db — log a set, query Supabase directly, verify row exists with correct data

▶ Integration: test_set_logged_at_server — verify server-side timestamp matches client-sent logged_at

M5-S6 Session Sets List (Zone D)

Display the list of sets logged in the current session for the current exercise, with set number, weight, reps, and optional badges.

Acceptance Criteria:

✓ Query: useSessionSets(exerciseId, sessionGroup=today) returns sets for this exercise today, ordered by logged_at DESC

✓ Each SetRow shows: set number (circled badge), weight x reps, variant name chip (if multiple variants), RPE badge (if set), PR badge (if applicable)

✓ Most recent set at the top of the list

✓ Scrollable if many sets

Test Coverage:

▶ Unit: test_set_row_renders — verify set number, weight, reps displayed correctly

▶ Unit: test_set_row_with_rpe — set has rpe=8.5, verify RPE badge shown

▶ Unit: test_set_row_without_rpe — set has no rpe, verify no RPE badge

▶ Unit: test_sets_ordered_newest_first — log 3 sets, verify first set in list is the most recent

▶ Unit: test_set_number_increments — verify set numbers count from 1 at the bottom (oldest) to N at top (newest)

M5-S7 Swipe-to-Delete Set

Enable swipe-left gesture on set rows to reveal a delete action with confirmation.

Acceptance Criteria:

- ✓ Swipe left on a SetRow reveals a red delete area
- ✓ Completing the swipe triggers a confirmation snackbar with 'Undo' action (5 second window)
- ✓ After 5 seconds without undo: DELETE mutation fires
- ✓ Undo: restores the set to the list (was only hidden, not yet deleted)
- ✓ Optimistic removal from cache on swipe, restore on undo

Test Coverage:

- ▶ Unit: test_swipe_reveals_delete — simulate swipe-left, verify red delete area visible
- ▶ Unit: test_swipe_complete_shows_undo — complete swipe, verify snackbar with 'Undo' appears
- ▶ Unit: test_undo_restores_set — swipe to delete, tap Undo within 5s, verify set reappears
- ▶ Unit: test_delete_after_timeout — swipe to delete, wait 5s, verify DELETE mutation called
- ▶ Unit: test_optimistic_removal — swipe to delete, verify set disappears immediately from list

M5-S8 Swipe-to-Edit Set (Inline)

Enable swipe-right or tap gesture to edit a previously logged set inline.

Acceptance Criteria:

- ✓ Tap on a set row enters edit mode: weight and reps become editable inline
- ✓ Stepper buttons appear adjacent to the inline fields
- ✓ A 'Save' and 'Cancel' button replace the set number badge
- ✓ Save triggers an UPDATE mutation, Cancel restores original values
- ✓ Edit mode only available for sets logged in the current session

Test Coverage:

- ▶ Unit: test_tap_enters_edit_mode — tap a set row, verify fields become editable
- ▶ Unit: test_edit_save — change weight, tap Save, verify UPDATE mutation called with new weight
- ▶ Unit: test_edit_cancel — change weight, tap Cancel, verify original weight restored
- ▶ Unit: test_old_set_not_editable — render a set from yesterday, verify tap does not enter edit mode

M5-S9 Advanced Set Metadata (RPE, RIR, Set Type, Notes)

Add the collapsible metadata row below the Log Set button for advanced fields: RPE, RIR, set type, tempo, and notes.

Acceptance Criteria:

- ✓ Row of filter chips: RPE, Type, Notes (collapsed by default)
- ✓ Tapping RPE chip reveals a horizontal slider or stepper (6.0-10.0 in 0.5 increments)
- ✓ Tapping Type chip reveals a chip group: Working (default), Warmup, Dropset, Failure, AMRAP
- ✓ Tapping Notes chip reveals a text input

✓ Selected metadata is included in the set insert payload

✓ Metadata chips show current values when set (e.g., 'RPE 8.5', 'Warmup')

Test Coverage:

▶ Unit: test_metadata_collapsed_default — verify RPE slider and set type are hidden by default

▶ Unit: test_rpe_chip_expand — tap RPE chip, verify slider appears

▶ Unit: test_rpe_value_in_payload — set RPE to 8.5, log set, verify mutation includes rpe: 8.5

▶ Unit: test_set_type_selection — tap Type chip, select 'Warmup', verify set_type: 'warmup' in payload

▶ Unit: test_notes_input — tap Notes chip, type 'Grip felt wide', verify notes included in payload

▶ Unit: test_metadata_persists_between_sets — set RPE to 8, log set, verify RPE stays at 8 for next set

M5-S10 Set Logging E2E Test Suite

End-to-end tests covering the complete set logging experience.

Acceptance Criteria:

Test Coverage:

▶ E2E: test_log_first_set — navigate to exercise, enter weight 100, reps 8, tap Log Set, verify set appears in list

▶ E2E: test_one_tap_repeat_set — log a set, verify pre-fill is identical, tap Log Set again (1 tap), verify second set logged

▶ E2E: test_stepper_adjustment — pre-fill at 100, tap +5, verify 105, tap Log Set, verify set logged at 105

▶ E2E: test_numpad_entry — tap weight field, enter 67.5 via numpad, tap Done, tap Log Set, verify 67.5 logged

▶ E2E: test_delete_set — log a set, swipe left, confirm, verify set removed

▶ E2E: test_edit_set — log a set, tap to edit, change reps, save, verify updated

▶ E2E: test_variant_switch_prefill — log a set on variant A at 100kg, switch to variant B, verify pre-fill changes to variant B's last weight

7. Milestone 6 — Rest Timer & Logging UX Polish

Goal: A polished logging experience with auto-starting rest timer, haptic feedback, and the quality-of-life features that make the app pleasant to use between sets.

M6-S1 Rest Timer State (Zustand Store)

Create a Zustand store for the rest timer that persists across screen navigation.

Acceptance Criteria:

- ✓ Store: { isRunning, remainingSeconds, totalSeconds, exerciseId, start(seconds), pause(), resume(), reset(), tick() }
- ✓ Timer ticks via setInterval(1000) managed inside the store
- ✓ Store persists in memory (not localStorage) — resets on page refresh
- ✓ When remainingSeconds reaches 0: isRunning becomes false, onComplete callback fires

Test Coverage:

- ▶ Unit: test_start_timer — call start(180), verify isRunning=true, remainingSeconds=180, totalSeconds=180
- ▶ Unit: test_tick — start(180), call tick(), verify remainingSeconds=179
- ▶ Unit: test_pause_resume — start, pause, verify isRunning=false, resume, verify isRunning=true
- ▶ Unit: test_reset — start(180), tick 5 times, reset(), verify remainingSeconds=180
- ▶ Unit: test_completion — start(2), tick twice, verify isRunning=false and onComplete called
- ▶ Unit: test_timer_doesnt_go_negative — start(1), tick twice, verify remainingSeconds=0 (not -1)

M6-S2 Rest Timer Floating Pill Component

Build the floating pill UI that appears below the TopAppBar showing remaining rest time.

Acceptance Criteria:

- ✓ RestTimerPill component reads from Zustand store
- ✓ Renders: linear progress bar (thin, full width), remaining time (MM:SS, centered), +30s and -30s buttons
- ✓ Fixed position below TopAppBar, above all content, z-index above bottom nav
- ✓ Visible only when isRunning=true or remainingSeconds>0
- ✓ Color transitions from blue to amber when remainingSeconds < 30
- ✓ Tapping the pill navigates back to the Set Logger for the associated exercise

Test Coverage:

- ▶ Unit: test_pill_visible_when_running — start timer, verify pill renders
- ▶ Unit: test_pill_hidden_when_idle — no timer running, verify pill not rendered
- ▶ Unit: test_time_display_format — remainingSeconds=90, verify display shows '1:30'
- ▶ Unit: test_color_transition — remainingSeconds=25, verify amber color applied
- ▶ Unit: test_plus_thirty — tap +30s button, verify remainingSeconds increases by 30
- ▶ Unit: test_minus_thirty — timer at 90s, tap -30s, verify remainingSeconds=60
- ▶ Unit: test_minus_thirty_floor — timer at 15s, tap -30s, verify remainingSeconds=0 (not negative)
- ▶ Unit: test_pill_tap_navigates — tap pill body, verify navigation to /log/:exerciseId

M6-S3 Auto-Start on Set Log

Wire the rest timer to auto-start when a set is logged, using the exercise's last-used rest time or a category default.

Acceptance Criteria:

- ✓ After useLogSet mutation succeeds (onSuccess): call timerStore.start(restSeconds)
- ✓ Rest seconds determined by: (1) user's last rest time for this exercise, or (2) category default (compound: 180s, isolation: 90s, hypertrophy: 60s)
- ✓ The rest_seconds from the previous set (if any) is used as the default
- ✓ If the timer is already running when a new set is logged: restart with new duration

Test Coverage:

- ▶ Unit: test_timer_starts_on_log — log a set, verify timer starts
- ▶ Unit: test_timer_uses_last_rest — previous set had rest_seconds=120, verify timer starts at 120
- ▶ Unit: test_timer_uses_category_default — first set (no prior rest), compound exercise, verify timer starts at 180
- ▶ Unit: test_timer_restarts_on_new_set — timer running at 90s, log new set, verify timer restarts from full duration

M6-S4 Rest Time Recording

Automatically record the rest time between sets by measuring the time elapsed from the previous set's logged_at to the current set's logged_at.

Acceptance Criteria:

- ✓ When logging a set: compute rest_seconds = current logged_at - previous set's logged_at (for the same exercise in the same session)
- ✓ Include rest_seconds in the set insert payload
- ✓ If this is the first set of the session for this exercise: rest_seconds = NULL
- ✓ Rest time is displayed on set rows in Zone D as a small label (e.g., '2:30 rest')

Test Coverage:

- ▶ Unit: test_rest_seconds_computed — previous set at T, current set at T+150s, verify rest_seconds=150
- ▶ Unit: test_first_set_no_rest — first set of session, verify rest_seconds=null
- ▶ Unit: test_rest_display_on_set_row — set with rest_seconds=150, verify '2:30 rest' label shown

M6-S5 Notification on Timer Expiry

When the rest timer reaches 0, send a browser notification (if permitted) and haptic vibration for lock-screen awareness.

Acceptance Criteria:

- ✓ Request Notification permission on first timer start
- ✓ On timer complete: new Notification('Rest Complete', { body: 'Time for your next set!' })
- ✓ Also trigger navigator.vibrate([200, 100, 200]) for haptic pulse
- ✓ If notification permission denied: only haptic (no error shown)

Test Coverage:

▶ Unit: test_notification_sent — mock Notification API, verify notification created on timer complete

▶ Unit: test_vibration_triggered — mock navigator.vibrate, verify called with pattern

▶ Unit: test_permission_denied_no_error — mock Notification.permission='denied', verify no notification and no error

M6-S6 Rest Timer E2E

End-to-end test for the full rest timer flow.

Acceptance Criteria:

Test Coverage:

▶ E2E: test_timer_auto_starts — log a set, verify floating pill appears with countdown

▶ E2E: test_timer_persists_across_tabs — log a set on Log tab, switch to History tab, verify pill still visible

▶ E2E: test_timer_plus_thirty — tap +30s on pill, verify time increases

8. Milestone 7 — Session Grouping & History

Goal: Users can browse their training history organized by sessions (derived from set timestamps) and by individual exercises. The History tab becomes fully functional.

M7-S1 Session Grouping Logic (Supabase RPC)

Create a Supabase RPC function that groups sets into sessions using a 90-minute inactivity threshold.

Acceptance Criteria:

- ✓ PostgreSQL function get_user_sessions(p_user_id uuid, p_limit int, p_offset int) returns a table of session summaries
- ✓ Logic: ORDER sets by logged_at, group consecutive sets where the gap between logged_at timestamps is < 90 minutes into the same session
- ✓ Returns: session_start, session_end, duration_minutes, total_sets, exercises (array of {name, sets_count, top_set_weight, top_set_reps}), total_volume
- ✓ Handles midnight-spanning sessions correctly

Test Coverage:

- ▶ Integration: test_single_session — insert 5 sets within 60 minutes, verify 1 session returned
- ▶ Integration: test_two_sessions — insert 3 sets, gap of 2 hours, insert 3 more, verify 2 sessions
- ▶ Integration: test_session_summary — verify returned exercises array and total_volume are correct
- ▶ Integration: test_midnight_span — sets at 23:45 and 00:15, verify same session (gap < 90 min)
- ▶ Integration: test_pagination — create 20 sessions, call with limit=10 offset=0, verify 10 results

M7-S2 Session Timeline Screen

Build the History tab with a chronological list of session summary cards.

Acceptance Criteria:

- ✓ useSessionTimeline() hook calls the RPC with pagination (infinite scroll)
- ✓ Each SessionCard shows: date header, duration, total sets, exercise summary list, total volume
- ✓ Newest sessions at the top
- ✓ Pull-to-refresh reloads the timeline
- ✓ Infinite scroll loads more sessions on reaching the bottom

Test Coverage:

- ▶ Unit: test_session_card_renders — verify date, duration, sets count, exercises displayed
- ▶ Unit: test_timeline_ordered_newest_first — verify first card has the most recent date
- ▶ Unit: test_infinite_scroll_loads_more — scroll to bottom, verify next page of sessions loaded
- ▶ Unit: test_empty_state — no sessions, verify empty state illustration and message

M7-S3 Session Detail Screen

Tapping a session card navigates to a detail view showing all sets in that session, grouped by exercise.

Acceptance Criteria:

- ✓ Route: /history/session?start={timestamp}&end={timestamp}
- ✓ Groups sets by exercise, showing variant chips where applicable
- ✓ Each exercise group shows its sets as a compact table: set#, weight, reps, RPE, rest time
- ✓ Session-level summary bar at the top: duration, total volume, total sets, exercises count

Test Coverage:

- ▶ Unit: test_sets_grouped_by_exercise — session with 3 exercises, verify 3 groups
- ▶ Unit: test_set_table_within_group — exercise with 4 sets, verify 4 rows with correct data
- ▶ Unit: test_summary_bar — verify duration, volume, sets count are correct

M7-S4 Exercise History Screen

A per-exercise history view showing all sets ever logged for one exercise, across all sessions and variants.

Acceptance Criteria:

- ✓ Route: /history/exercise/:exerciseId
- ✓ Flat list of sets, newest first, with date, variant badge, weight x reps, RPE, PR badge
- ✓ Filterable by variant (chip row at top) and date range
- ✓ Shows aggregate stats at top: total sets, total volume, date range, max weight

Test Coverage:

- ▶ Unit: test_exercise_history_renders — verify list of sets with correct data
- ▶ Unit: test_variant_filter — select variant B filter, verify only variant B sets shown
- ▶ Unit: test_aggregate_stats — verify total sets and volume are computed correctly
- ▶ Unit: test_pr_badge_shown — set that is a PR has gold badge

M7-S5 Session Naming (Optional)

Allow users to optionally name sessions (e.g., 'Upper Body Day', 'Leg Day') via long-press on the session card's date header.

Acceptance Criteria:

- ✓ user_session_names table: id, user_id, session_start, session_end, name, created_at (with RLS)
- ✓ Long-press on date header reveals inline text edit field
- ✓ Named sessions show the custom name above the date
- ✓ Names are stored separately from the set data (sessions remain derived)

Test Coverage:

- ▶ Unit: test_long_press_opens_edit — long-press date, verify input field appears
- ▶ Unit: test_name_saves — type 'Push Day', blur, verify name saved
- ▶ Unit: test_name_displays — session with name, verify name shown above date

M7-S6 History Search & Filtering

Add search and filtering capabilities to the session timeline.

Acceptance Criteria:

- ✓ Search bar filters sessions by exercise name (searches within the exercises array)

- ✓ Date range picker (This Week, This Month, 3 Months, Custom Range)
- ✓ Filter persists across navigation within the History tab

Test Coverage:

- ▶ Unit: test_search_filters_sessions — search 'bench', verify only sessions containing bench exercises shown
- ▶ Unit: test_date_range_filter — select 'This Week', verify only this week's sessions shown

M7-S7 History E2E Tests

End-to-end tests for the history experience.

Acceptance Criteria:

Test Coverage:

- ▶ E2E: test_session_appears_after_logging — log sets, navigate to History, verify session card appears
- ▶ E2E: test_session_detail_drill_down — tap session card, verify detail screen with correct sets
- ▶ E2E: test_exercise_history — navigate to exercise history, verify chronological set list

9. Milestone 8 — Progress Tracking & Personal Records

Goal: Real-time PR detection with celebration animation, a comprehensive PR board, and the exercise-level estimated 1RM trend chart.

M8-S1 Real-Time PR Detection (Frontend)

After each set is logged, check the optimistic response against the personal_records cache to determine if a PR was hit, and trigger the celebration UI immediately.

Acceptance Criteria:

- ✓ usePRCheck(exerciseId, variantId) hook compares the just-logged set's computed e1RM against the cached PR
- ✓ If PR: set isPR=true on the set in cache, trigger celebration overlay
- ✓ PR types checked: est_1rm, rep_max at the specific rep count (e.g., new 5RM if reps=5)
- ✓ Falls back to the DB trigger as source of truth (frontend check is for instant UX)

Test Coverage:

- ▶ Unit: test_pr_detected — current PR is 120kg e1RM, new set computes to 125kg, verify isPR=true
- ▶ Unit: test_no_pr — current PR is 120kg, new set is 115kg, verify isPR=false
- ▶ Unit: test_rep_max_pr — current 5RM is 100kg, new set is 105kgx5, verify rep_max PR detected
- ▶ Unit: test_first_set_is_always_pr — no existing PR, any set creates a PR

M8-S2 PR Celebration Overlay Component

Build the animated celebration that plays when a PR is detected.

Acceptance Criteria:

- ✓ Gold banner slides down from top: 'New PR!' title + record detail
- ✓ Confetti particles emit from center (react-confetti or custom)
- ✓ Auto-dismiss after 4 seconds
- ✓ Set row in Zone D receives a permanent gold star badge
- ✓ Respects prefers-reduced-motion: static badge, no animation

Test Coverage:

- ▶ Unit: test_celebration_renders_on_pr — trigger PR, verify banner visible with correct text
- ▶ Unit: test_celebration_auto_dismiss — verify banner gone after 4 seconds (fake timers)
- ▶ Unit: test_reduced_motion — mock prefers-reduced-motion, verify no animation
- ▶ Unit: test_pr_badge_on_set_row — verify gold star badge renders on the PR set

M8-S3 PR Board Screen

A dedicated screen listing all personal records across all exercises.

Acceptance Criteria:

- ✓ Route /stats/prs
- ✓ Grouped by exercise, each showing records across rep ranges

✓ Tappable to navigate to the specific set

Test Coverage:

▶ Unit: test_pr_board_renders — verify exercises listed with their PR records
▶ Unit: test_pr_grouped_by_exercise — verify correct grouping
▶ Unit: test_pr_tap_navigates — tap a PR, verify navigation to exercise history

M8-S4 Estimated 1RM Trend Chart

Build the line chart showing estimated 1RM over time per exercise using Recharts.

Acceptance Criteria:

✓ Recharts LineChart with date on x-axis, e1RM on y-axis
✓ Data points show actual sets as scatter, trend line as smoothed average
✓ Multi-variant: separate lines per variant, color-coded
✓ SegmentedButton toggle for All Variants / individual

Test Coverage:

▶ Unit: test_chart_renders_with_data — pass 20 data points, verify Recharts SVG rendered
▶ Unit: test_chart_empty_state — no data, verify 'Log some sets to see your trend' message
▶ Unit: test_variant_toggle_filters_data — select variant A, verify only variant A data points shown
▶ Unit: test_data_point_tooltip — hover a point, verify tooltip shows date, weight, reps, e1RM

M8-S5 PR Table Per Exercise

Below the 1RM chart, a table showing records at each rep range for the selected exercise.

Acceptance Criteria:

✓ Rows: 1RM, 3RM, 5RM, 8RM, 10RM, Max Volume
✓ Columns: Weight, Date, Variant
✓ Highlight the most recent PR with a subtle animation

Test Coverage:

▶ Unit: test_pr_table_renders — verify all rep range rows present
▶ Unit: test_pr_table_data — verify correct weight and date for each record type

M8-S6 Weekly Volume Bar Chart

Stacked bar chart showing weekly training volume per muscle group.

Acceptance Criteria:

✓ Recharts BarChart with ISO weeks on x-axis
✓ Stacked segments by muscle group, color-coded
✓ Date range selector: 4 weeks, 8 weeks, 12 weeks, All

Test Coverage:

▶ Unit: test_volume_chart_renders — verify stacked bars with correct segment colors
▶ Unit: test_date_range_changes_data — select 4 weeks, verify only 4 bars shown

M8-S7 Backend Analytics Pre-Computation

Create FastAPI endpoints and Supabase RPCs for pre-computing analytics that feed the charts.

Acceptance Criteria:

- ✓ POST /api/analytics/compute triggers computation for authenticated user
- ✓ Supabase RPC compute_exercise_1rm_trend(user_id, exercise_id) returns [{date, e1rm, variant_id}]
- ✓ Supabase RPC compute_weekly_volume(user_id, weeks) returns [{week, muscle_group, volume}]
- ✓ Results cached in analytics_cache table
- ✓ Incremental: only recompute since last computed_at

Test Coverage:

- ▶ Unit: test_1rm_trend_computation — insert known sets, verify e1RM values match Epley formula
- ▶ Unit: test_weekly_volume_computation — insert sets across 3 weeks, verify correct volume sums
- ▶ Unit: test_incremental_computation — compute, add 2 sets, recompute, verify only new data processed
- ▶ Integration: test_cache_stored — compute, verify analytics_cache row exists with correct payload
- ▶ Unit: test_auth_required — call compute endpoint without JWT, verify 401

M8-S8 Muscle Distribution Donut Chart

Nivo pie chart showing volume distribution across muscle groups.

Acceptance Criteria:

- ✓ Nivo ResponsivePie with muscle groups as slices
- ✓ Below the donut: a body silhouette SVG heatmap

Test Coverage:

- ▶ Unit: test_donut_renders — verify SVG pie chart with correct slice count
- ▶ Unit: test_heatmap_colors — verify high-volume muscle is colored red, untrained is gray

M8-S9 Training Frequency Calendar

A GitHub-style heatmap showing training days over the last 12 weeks.

Acceptance Criteria:

- ✓ Nivo Calendar or custom SVG grid
- ✓ Each cell = one day, colored by volume intensity
- ✓ Tap a cell to see that day's session summary in a tooltip

Test Coverage:

- ▶ Unit: test_calendar_renders_84_cells — 12 weeks x 7 days = 84 cells
- ▶ Unit: test_training_day_colored — day with sets has a non-gray color
- ▶ Unit: test_rest_day_gray — day without sets is gray
- ▶ Unit: test_tooltip_on_tap — tap a training day, verify tooltip with volume and exercises

10. Milestone 9 — Stats Dashboard Assembly

Goal: The Stats tab dashboard is assembled from the chart components built in M8, with the weekly snapshot, recent PRs, and top exercises cards.

M9-S1 Weekly Snapshot Card

This-week-vs-last-week comparison card with delta indicators.

Acceptance Criteria:

- ✓ Shows: total sets, total volume, training days
- ✓ Delta: green arrow + percentage for improvement, red for decline

Test Coverage:

- ▶ Unit: test_snapshot_improvement — this week > last week, verify green arrow and positive percentage
- ▶ Unit: test_snapshot_decline — this week < last week, verify red arrow
- ▶ Unit: test_snapshot_no_last_week — first week of training, verify 'First week!' message

M9-S2 Recent PRs Carousel

Horizontal scrolling cards showing the last 3-5 PRs.

Acceptance Criteria:

- ✓ Each card: exercise name, record type, value, date
- ✓ Tappable: navigates to exercise chart view

Test Coverage:

- ▶ Unit: test_carousel_renders_prs — 3 PRs in data, verify 3 cards rendered
- ▶ Unit: test_carousel_empty — no PRs, verify 'Hit your first PR!' message
- ▶ Unit: test_carousel_tap — tap a PR card, verify navigation to exercise chart

M9-S3 Top Exercises Card

Ranked list of most-performed exercises with mini sparkline charts.

Acceptance Criteria:

- ✓ Top 5 exercises by volume
- ✓ Each row: rank number, exercise name, total volume, sparkline of recent trend

Test Coverage:

- ▶ Unit: test_top_five_rendered — verify 5 exercise rows
- ▶ Unit: test_sparkline — verify small SVG chart rendered per exercise

M9-S4 Stats Dashboard Screen Assembly

Combine all dashboard cards into the Stats tab home screen with date range filtering.

Acceptance Criteria:

- ✓ Vertical scroll of cards: Snapshot, PRs, Calendar, Muscle Donut, Top Exercises
- ✓ Date range chip row: This Week, This Month, 3 Months, All Time
- ✓ All cards react to date range change

Test Coverage:

- ▶ Unit: test_all_cards_render — verify all 5 card types present
- ▶ Unit: test_date_range_propagates — select 'This Month', verify all cards' queries include date filter
- ▶ E2E: test_stats_dashboard_loads — login, navigate to Stats, verify dashboard renders with data

M9-S5 Exercise Charts Screen

Dedicated per-exercise screen with Strength/Volume/History tabs.

Acceptance Criteria:

- ✓ Route /stats/exercise/:exerciseId
- ✓ Three tabs: Strength (1RM trend + PR table), Volume (weekly bar chart), History (set list)
- ✓ SegmentedButton for variant filtering

Test Coverage:

- ▶ Unit: test_three_tabs_render — verify Strength, Volume, History tabs present
- ▶ Unit: test_tab_switching — click Volume tab, verify bar chart shown
- ▶ E2E: test_exercise_chart_drill_down — from dashboard, tap exercise, verify chart screen with data

M9-S6 Data Export (CSV/JSON)

Allow users to export their training data.

Acceptance Criteria:

- ✓ Settings screen: 'Export Data' option
- ✓ Formats: CSV (one row per set) and JSON (structured)
- ✓ Includes: all sets, variants, exercises, PRs

Test Coverage:

- ▶ Unit: test_csv_export_format — verify CSV headers and row count match sets count
- ▶ Unit: test_json_export_structure — verify JSON includes sets, variants, and exercises arrays
- ▶ E2E: test_export_downloads — click Export, verify file downloads with correct content

M9-S7 Onboarding Flow

Build the 5-screen onboarding experience for new users.

Acceptance Criteria:

- ✓ Welcome, Profile Setup, First Machine (optional), Tutorial, Ready
- ✓ Coach marks with spotlight overlays on tutorial screen
- ✓ Profile data saved to user_profiles table

Test Coverage:

- ▶ Unit: test_onboarding_step_navigation — verify progression through all 5 screens
- ▶ Unit: test_profile_saves — complete profile, verify data saved to Supabase
- ▶ Unit: test_skip_machine_photo — skip step 3, verify flow continues to tutorial

▶ E2E: test_new_user_onboarding — sign up fresh account, verify onboarding flow completes and redirects to Log tab

M9-S8 Soreness Tracking Prompts

Implement the post-workout soreness prompt system from v0.

Acceptance Criteria:

- ✓ 24-48 hours after a session: prompt appears on app open
- ✓ Body silhouette with tappable muscle groups
- ✓ Each muscle: 0 (no soreness) to 4 (severe) scale
- ✓ Data saved to soreness_reports table

Test Coverage:

- ▶ Unit: test_prompt_timing — mock last session 26 hours ago, verify prompt appears
- ▶ Unit: test_no_prompt_too_soon — session 2 hours ago, verify no prompt
- ▶ Unit: test_muscle_selection_and_rating — tap chest, set 3, verify correct payload
- ▶ Integration: test_soreness_persists — submit report, verify row in soreness_reports

11. Milestone 10 — Offline-First & PWA

Goal: The app works fully offline for set logging and history browsing. Syncs automatically when connectivity returns. Installable as a PWA.

M10-S1 TanStack Query Offline Persistence

Configure PersistQueryClientProvider to persist the query cache to IndexedDB.

Acceptance Criteria:

- ✓ Install @tanstack/query-persist-client-core and idb-keyval
- ✓ Wrap app with PersistQueryClientProvider
- ✓ All query data survives page refresh
- ✓ networkMode: 'offlineFirst' on all mutations

Test Coverage:

- ▶ Unit: test_cache_persists_to_indexeddb — populate cache, unmount, remount, verify cache restored
- ▶ Unit: test_offline_mutation_queued — go offline, log a set, verify mutation in pending state
- ▶ Unit: test_online_mutation_replays — queue mutation offline, go online, verify mutation fires and succeeds

M10-S2 Offline Status Indicator

Show a subtle indicator when the app is offline, without blocking any functionality.

Acceptance Criteria:

- ✓ Chip in TopAppBar: 'Offline — sets will sync when connected'
- ✓ Listens to navigator.onLine and online/offline events
- ✓ Green brief 'Back online — syncing' when connectivity returns

Test Coverage:

- ▶ Unit: test_offline_chip_shown — go offline, verify chip visible
- ▶ Unit: test_online_chip_hidden — online, verify no chip
- ▶ Unit: test_reconnect_message — go offline then online, verify 'Back online' briefly shown

M10-S3 Service Worker (Workbox)

Register a service worker that pre-caches the app shell and uses stale-while-revalidate for API responses.

Acceptance Criteria:

- ✓ Workbox InjectManifest strategy for app shell
- ✓ Runtime cache: Supabase REST calls with stale-while-revalidate
- ✓ Cache-first for exercise library data (rarely changes)

Test Coverage:

- ▶ Unit: test_service_worker_registered — verify navigator.serviceWorker.controller exists after registration

▶ Integration: test_app_loads_offline — load app, go offline, reload, verify app renders from cache

M10-S4 PWA Manifest & Installability

Add a web app manifest and meet all PWA installability criteria.

Acceptance Criteria:

- ✓ manifest.json with name, icons (192, 512), theme_color, display: standalone
- ✓ Meta tags for iOS standalone mode
- ✓ Lighthouse PWA audit passes

Test Coverage:

- ▶ Audit: Lighthouse PWA score ≥ 90
- ▶ Manual: test_add_to_home_screen — verify install prompt appears on supported browsers
- ▶ Manual: test_standalone_mode — installed PWA opens without browser chrome

M10-S5 Sync Conflict Resolution

Handle the edge case where the same set is modified on two devices or the optimistic update conflicts with the server.

Acceptance Criteria:

- ✓ Last-write-wins based on logged_at timestamp
- ✓ If server version is newer than optimistic version: server wins, cache updated
- ✓ Conflict detected via updated_at column added to sets table

Test Coverage:

- ▶ Unit: test_server_wins_newer — server has updated_at T+5, client has T+0, verify server data kept
- ▶ Unit: test_client_wins_newer — client has updated_at T+5, server has T+0, verify client data kept

M10-S6 Offline Set Logging E2E

End-to-end test verifying the complete offline logging experience.

Acceptance Criteria:

Test Coverage:

- ▶ E2E: test_log_sets_offline — go offline, log 3 sets, verify they appear in the session list, go online, verify they sync to Supabase
- ▶ E2E: test_history_available_offline — load history online, go offline, navigate to history, verify data still renders

M10-S7 Performance Audit

Run performance audits to verify the app meets all performance budgets.

Acceptance Criteria:

Test Coverage:

- ▶ Audit: Lighthouse Performance score ≥ 90
- ▶ Audit: FCP < 1.5s on 4G throttle

- ▶ Audit: TTI < 3s on 4G throttle
- ▶ Audit: Bundle size < 200KB gzipped (initial route)

12. Milestone 11 — AI Machine Identification

Goal: Users can photograph a gym machine and receive AI-identified exercise names, equipment type, manufacturer, target muscles, and form tips. The result feeds directly into variant creation.

M11-S1 FastAPI AI Proxy Endpoint

Create the POST /api/ai/identify-machine endpoint that accepts an image, calls Claude, and returns structured results.

Acceptance Criteria:

- ✓ Accepts multipart/form-data with a JPEG/PNG image <5MB
- ✓ Validates JWT via get_current_user
- ✓ Rate-limited: 10 requests/user/day (Redis counter)
- ✓ Sends image to Claude Sonnet with a structured prompt requesting: exercise_names[], equipment_type, manufacturer_guess, primary_muscles[], secondary_muscles[], form_tips[]
- ✓ Parses Claude's response into a Pydantic MachineIdentificationResponse model
- ✓ Returns JSON response within 5 seconds (target)

Test Coverage:

- ▶ Unit: test_identify_returns_structured_response — mock Claude to return expected text, verify Pydantic model parses correctly
- ▶ Unit: test_identify_validates_image_size — send >5MB image, verify 413 error
- ▶ Unit: test_identify_validates_image_type — send a .txt file, verify 415 error
- ▶ Unit: test_identify_auth_required — call without JWT, verify 401
- ▶ Unit: test_identify_rate_limit — mock 10 prior calls today, verify 429 on 11th call
- ▶ Unit: test_prompt_structure — verify the Claude prompt includes the required output schema
- ▶ Unit: test_claude_error_handling — mock Claude API error, verify 502 with user-friendly message

M11-S2 Image Capture UI

Build the camera/gallery picker screen accessible from the Log tab.

Acceptance Criteria:

- ✓ Camera icon in TopAppBar opens /log/identify route
- ✓ Screen shows: camera viewfinder using navigator.mediaDevices.getUserMedia OR file input for gallery
- ✓ Capture/select triggers upload to the identify endpoint
- ✓ Loading state: pulsing machine icon + 'Identifying machine...' text

Test Coverage:

- ▶ Unit: test_camera_opens — mock getUserMedia, verify video element renders
- ▶ Unit: test_gallery_fallback — mock getUserMedia rejection, verify file input shown
- ▶ Unit: test_capture_triggers_upload — capture photo, verify POST to /api/ai/identify-machine
- ▶ Unit: test_loading_state — during API call, verify loading UI shown

M11-S3 AI Result to Variant Creation Flow

Wire the AI response to pre-fill the variant creation bottom sheet.

Acceptance Criteria:

- ✓ On successful identification: open variant creation bottom sheet pre-filled with AI data
- ✓ Exercise name: dropdown allowing selection if multiple exercises identified
- ✓ Equipment type, manufacturer, muscles: pre-filled from AI response
- ✓ Form tips: shown in a read-only expandable section
- ✓ User can edit any pre-filled field before saving

Test Coverage:

- ▶ Unit: test_prefill_from_ai — mock AI response, verify variant form fields pre-filled
- ▶ Unit: test_exercise_dropdown_multiple — AI returns 2 exercise names, verify dropdown with both options
- ▶ Unit: test_form_tips_displayed — verify form tips section renders with AI tips
- ▶ Unit: test_user_can_edit_prefill — change manufacturer, save, verify edited value persisted

M11-S4 Image Hash Caching

Cache AI responses by image hash to avoid duplicate API calls for the same machine photo.

Acceptance Criteria:

- ✓ Compute perceptual hash of the image on the backend (imagehash or dhash)
- ✓ Before calling Claude: check Redis for existing result with this hash
- ✓ If cached: return immediately without Claude API call
- ✓ Cache TTL: 30 days

Test Coverage:

- ▶ Unit: test_cache_hit — send same image twice, verify Claude called only once
- ▶ Unit: test_cache_miss — send two different images, verify Claude called twice
- ▶ Unit: test_cache_expiry — mock expired cache entry, verify Claude called again

M11-S5 AI Error Handling UI

Handle all AI failure modes gracefully.

Acceptance Criteria:

- ✓ Network error: 'Could not reach AI service. Check your connection.'
- ✓ AI couldn't identify: 'Could not identify this machine. Try a clearer photo or create the variant manually.'
- ✓ Rate limit: 'AI features paused for today. Resets at midnight.'
- ✓ All error states provide a 'Manual Create' button fallback

Test Coverage:

- ▶ Unit: test_network_error_ui — mock fetch failure, verify error message and retry button
- ▶ Unit: test_unidentifiable_ui — mock AI response with low confidence, verify message and manual create button
- ▶ Unit: test_rate_limit_ui — mock 429 response, verify rate limit message

M11-S6 AI Token Budget Enforcement

Ensure Claude interactions stay within cost budgets.

Acceptance Criteria:

- ✓ Max input: 1000 tokens (image ~800 + prompt ~200)
- ✓ Max output: 300 tokens
- ✓ Claude max_tokens parameter set to 300 in API call
- ✓ Log token usage per request for cost monitoring

Test Coverage:

- ▶ Unit: test_max_tokens_set — verify Claude API call includes max_tokens=300
- ▶ Unit: test_token_logging — verify usage logged after each call

M11-S7 AI Machine ID E2E

End-to-end test for the AI identification flow.

Acceptance Criteria:

Test Coverage:

- ▶ E2E: test_photo_to_variant — upload a test machine image, wait for AI response, verify variant creation sheet pre-filled, save, verify variant appears in chip row
- ▶ E2E: test_rate_limit_shown — exhaust daily limit, attempt another, verify rate limit message

13. Milestone 12 — AI Coaching & Recommendations

Goal: The deterministic recommendation engine is live (weight suggestions, deload alerts, recovery scores) and the LLM coaching layer provides natural language explanations as a premium feature.

M12-S1 Weight Progression Engine

Build the deterministic algorithm for suggesting next-set weight based on RPE targets and recent performance.

Acceptance Criteria:

- ✓ Input: last N sets for exercise+variant, target RPE, user experience level
- ✓ Algorithm: compute rolling e1RM, apply target RPE percentage, round to nearest weight_increment
- ✓ Output: suggested_weight, rationale_text
- ✓ Stored as a Python service class WeightProgressionEngine

Test Coverage:

- ▶ Unit: test_suggest_increase — last 3 sets all below target RPE, verify suggested weight > last weight
- ▶ Unit: test_suggest_maintain — last set at target RPE, verify same weight suggested
- ▶ Unit: test_suggest_decrease — last 3 sets above target RPE, verify lower weight suggested
- ▶ Unit: test_rounds_to_increment — raw suggestion is 67.3, increment is 2.5, verify 67.5 suggested
- ▶ Unit: test_beginner_faster_progression — beginner level, verify larger weight jumps than advanced

M12-S2 Deload Recommendation Engine

Build the deload detection and recommendation system.

Acceptance Criteria:

- ✓ Monitors: weeks of consecutive volume increase, performance trend (declining e1RM), average RPE
- ✓ Triggers deload alert when: 4+ weeks increasing volume, OR declining e1RM for 2+ weeks, OR avg RPE > 8.5
- ✓ Deload prescription: volume -40-60%, intensity -10-15% for 1 week
- ✓ API: GET /api/recommendations/deload-status returns { should_deload: bool, reason: str, prescription: {...} }

Test Coverage:

- ▶ Unit: test_deload_triggered_by_volume — 4 weeks increasing volume, verify should_deload=true
- ▶ Unit: test_deload_triggered_by_declining_1rm — 3 weeks declining e1RM, verify triggered
- ▶ Unit: test_deload_triggered_by_rpe — avg RPE 9.0 over 2 weeks, verify triggered
- ▶ Unit: test_no_deload_healthy — moderate volume, stable RPE, verify should_deload=false
- ▶ Unit: test_deload_prescription — verify prescribed volume reduction is 40-60%

M12-S3 Muscle Recovery Score Model

Compute per-muscle-group recovery status based on training volume, time since last training, and soreness reports.

Acceptance Criteria:

- ✓ Recovery score: 0-100% per muscle group
- ✓ Factors: hours since last training (exponential decay), volume of last session (higher volume = slower recovery), soreness report (level 3-4 adds recovery penalty)
- ✓ API: GET /api/recommendations/recovery returns [{muscle_group, score, last_trained, recommendation}]

Test Coverage:

- ▶ Unit: test_full_recovery — 72 hours since training, no soreness, verify score ~100%
- ▶ Unit: test_partial_recovery — 24 hours since heavy session, verify score 40-60%
- ▶ Unit: test_soreness_impact — same time elapsed but soreness=4, verify lower score than without soreness
- ▶ Unit: test_all_muscle_groups_returned — verify response includes all major muscle groups

M12-S4 Frontend Weight Suggestion Display

Show the AI-suggested weight on the Set Logger screen below the weight input.

Acceptance Criteria:

- ✓ Small chip below weight field: 'Suggested: 105 kg' with a lightning bolt icon
- ✓ Tapping the chip fills the weight field with the suggestion
- ✓ Chip only shows when a suggestion differs from pre-fill (don't suggest the same weight)
- ✓ Loading state while suggestion computes

Test Coverage:

- ▶ Unit: test_suggestion_displayed — API returns 105, prefill is 100, verify chip shows 'Suggested: 105 kg'
- ▶ Unit: test_suggestion_hidden_same_weight — API returns 100, prefill is 100, verify no chip
- ▶ Unit: test_suggestion_tap_fills — tap chip, verify weight field updates to 105

M12-S5 Deload Alert Banner

Show a deload recommendation banner on the Log tab when triggered.

Acceptance Criteria:

- ✓ Alert banner at top of exercise list: 'Time for a deload week?' with reason text
- ✓ Expandable: shows prescribed volume/intensity reductions
- ✓ Dismissible: 'Got it' button hides for 7 days

Test Coverage:

- ▶ Unit: test_deload_banner_shown — mock should_deload=true, verify banner visible
- ▶ Unit: test_deload_banner_hidden — should_deload=false, verify no banner
- ▶ Unit: test_deload_dismiss — tap 'Got it', verify banner hidden, reappears after 7 days

M12-S6 Recovery Status Indicator

Show per-muscle-group recovery on the Stats dashboard.

Acceptance Criteria:

- ✓ Card on Stats dashboard: muscle recovery heatmap
- ✓ Each muscle colored by recovery score: green (>80%), yellow (50-80%), red (<50%)
- ✓ Tappable: shows detail with last trained date and recommendation

Test Coverage:

- ▶ Unit: test_recovery_heatmap_colors — verify green/yellow/red mapping from score
- ▶ Unit: test_recovery_detail_on_tap — tap a muscle, verify detail tooltip

M12-S7 LLM Coaching Endpoint (Premium)

Build the conversational coaching endpoint that uses Claude to explain recommendations in natural language.

Acceptance Criteria:

- ✓ POST /api/ai/coach with { query: string, context_weeks: 4 }
- ✓ Backend injects: last 4 weeks of training data, recovery scores, active PRs, user profile
- ✓ System prompt instructs Claude to explain deterministic engine outputs, not make independent programming decisions
- ✓ Token budget: 2000 input, 500 output
- ✓ Rate limit: 5 queries/user/day

Test Coverage:

- ▶ Unit: test_coach_returns_response — mock Claude, verify structured coaching response
- ▶ Unit: test_context_injection — verify training data included in Claude prompt
- ▶ Unit: test_rate_limit — 6th query returns 429
- ▶ Unit: test_auth_required — no JWT returns 401

M12-S8 Weekly Progress Summary (Cron)

A cron job that generates weekly progress reports using the coaching LLM.

Acceptance Criteria:

- ✓ Runs every Monday at 6 AM UTC via Render cron
- ✓ For each active user: compute week's stats, pass to Claude, store generated summary
- ✓ Summary covers: PRs hit, volume changes, consistency, recommendations for next week
- ✓ Delivered as a card on the Stats dashboard and optionally via email

Test Coverage:

- ▶ Unit: test_weekly_summary_generation — mock user data and Claude, verify summary text generated
- ▶ Unit: test_summary_stored — verify summary saved to analytics_cache with correct type
- ▶ Integration: test_cron_processes_active_users — 3 active users, verify 3 summaries generated
- ▶ Unit: test_inactive_user_skipped — user with no sets in 14 days, verify no summary generated

14. Launch Checklist

Every item must be verified before the v1.0 production deploy.

Category	Item	Status
Testing	All unit tests pass (backend + frontend)	□
Testing	All integration tests pass	□
Testing	All E2E tests pass against staging	□
Testing	Backend coverage ≥90% on business logic	□
Testing	Frontend coverage ≥85%	□
Performance	Lighthouse Performance ≥90	□
Performance	Lighthouse PWA ≥90	□
Performance	Lighthouse Accessibility ≥90	□
Performance	FCP < 1.5s on 4G throttle	□
Performance	Bundle size < 200KB gzipped (initial route)	□
Security	All RLS policies verified with cross-user tests	□
Security	No API keys in frontend code	□
Security	JWT verification covers all backend endpoints	□
Security	CORS restricted to production domain only	□
Infrastructure	Render always-on ($7/mo) activated	□
Infrastructure	Supabase backup schedule configured	□
Infrastructure	Error monitoring (Sentry) configured	□
Infrastructure	Uptime monitoring configured	□
Data	Exercise seed data reviewed for accuracy	□
Data	Database indexes verified with EXPLAIN ANALYZE on key queries	□
UX	Empty states tested for all screens	□
UX	Error states tested for all network failure modes	□
UX	Dark mode verified on all screens	□
UX	Light mode verified on all screens	□
Legal	Privacy policy published	□
Legal	Terms of service published	□

End of Implementation Plan.