INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DO RIO GRANDE DO NORTE

JOSÉ VILANIR DE SOUZA BRITO NETO

DESENVOLVIMENTO DO APLICATIVO GYM TRACKER

NATAL
2025

JOSÉ VILANIR DE SOUZA BRITO NETO

DESENVOLVIMENTO DO APLICATIVO GYM TRACKER

Trabalho de Conclusão de Curso apresentado ao Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, em cumprimento às exigências legais como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: M.e. Gracon Huttennberg Eliatan Leite de Lima.

NATAL

2025

Instituto Federal de Educação, Ciência e Tecnologia do Estado do Rio Grande do Norte – IFRN Sistema Integrado de Bibliotecas – SIBi

Brito Neto, José Vilanir de Souza.
B862d Desenvolvimento do aplicativo GYM TRACKER / José Vilanir de Souza Brito Neto. – 2025.
56 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, Natal, 2025.
Orientador(a): Me. Gracon Huttennberg Eliatan Leite de Lima.

1. Desenvolvimento de aplicativo. 2. Aplicativo móvel. 3. GYM TRACKER. I. Título.

SIBi/IFRN CDU 004.451

Elaborada pela Bibliotecária
Sandra Nery S. Bigois – CRB-15/439

JOSÉ VILANIR DE SOUZA BRITO NETO

DESENVOLVIMENTO DO APLICATIVO GYM TRACKER

Trabalho de Conclusão de Curso apresentado ao Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, em cumprimento às exigências legais como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Trabalho de Conclusão de Curso aprovado em 29/01/2026 pela seguinte Banca Examinadora:

gov.br Documento assinado digitalmente
GRACON HUTTENNBERG ELIATAN LEITE DE LIM
Data: 06/03/2026 12:32:43-0300
Verifique em https://validar.iti.gov.br

Gracon Huttennberg Eliatan Leite de Lima, M.e – Orientador

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

gov.br Documento assinado digitalmente
MARILIA ARANHA FREIRE
Data: 08/03/2026 20:06:43-0300
Verifique em https://validar.iti.gov.br

Marilia Aranha Freire, M.e / Dr. / Dra.

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

gov.br Documento assinado digitalmente
LEONARDO REIS LUCENA
Data: 16/03/2026 12:59:07-0300
Verifique em https://validar.iti.gov.br

Leonardo Reis Lucena, M.a / Dr. / Dra.

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

AGRADECIMENTOS

Agradeço primeiramente a Deus por me conceder saúde, força e determinação para concluir esta jornada acadêmica.

Á minha família, agradeço imensamente aos meus pais, que tiveram papel decisivo na minha formação, sempre oferecendo orientação, motivação e uma presença constante que me acompanhou em cada passo além da compreensão nos momentos de ausência e pelo incentivo constante ao longo de toda a minha formação.

À minha namorada, deixo meu sincero agradecimento por todo amor, companhia e incentivo. Sua presença transformou este caminho em algo mais leve e especial.

Ao meu orientador, Professor Me. Gracon Huttennberg Eliatan Leite de Lima, pela dedicação, paciência e pelos valiosos ensinamentos que foram fundamentais para a realização deste trabalho.

Aos professores do curso de Análise e Desenvolvimento de Sistemas do IFRN, que contribuíram para minha formação acadêmica e profissional ao longo desses anos.

Aos colegas de turma, pela parceria, troca de conhecimentos e pelos momentos compartilhados durante essa trajetória.

RESUMO

Treinar de forma consistente exige registrar exercícios, séries e rotinas, mas muitas pessoas enfrentam dificuldades em manter um histórico organizado e atualizar treinos de modo prático, especialmente quando estão offline ou precisam corrigir registros passados. Para lidar com esse problema, este trabalho propõe o desenvolvimento de um aplicativo móvel que facilite o planejamento, o registro e o acompanhamento de treinos, permitindo criar e reutilizar rotinas, ajustar exercícios e séries e registrar treinos retroativamente. A solução foi construída seguindo um processo incremental, com elicitação e priorização de requisitos, modelagem de dados e casos de uso, e implementação em Flutter utilizando gerenciamento de estado reativo e persistência local na abordagem local-first, garantindo operação sem dependência contínua da rede. A aplicação foi validada por meio de testes manuais em dispositivos Android reais, que verificaram a integridade das operações e a consistência dos dados, demonstrando viabilidade técnica, funcionamento offline confiável, correta manutenção do histórico e aderência aos requisitos definidos.

Palavras-chave: rotinas de treino; registro retroativo; acompanhamento; persistência local; aplicativo móvel.

ABSTRACT

Training consistently requires logging exercises, sets, and routines, but many people struggle to maintain an organized history and update workouts conveniently, especially when offline or needing to correct past records. To address this problem, this work proposes the development of a mobile application that facilitates workout planning, logging, and tracking, allowing users to create and reuse routines, adjust exercises and sets, and register workouts retroactively. The solution was built following an incremental process, with requirements elicitation and prioritization, data and use-case modeling, and implementation in Flutter using reactive state management and local persistence in a local-first approach, ensuring operation without continuous network dependency. The application was validated through manual testing on real Android devices, which verified the integrity of operations and data consistency, demonstrating technical feasibility, reliable offline functionality, correct history maintenance, and adherence to the defined requirements.

Keywords: training routines; retroactive logging; tracking; local persistence; mobile application.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface (Interface de Programação de Aplicações)
DER	Diagrama de Entidade-Relacionamento
IFRN	Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte
JSON	JavaScript Object Notation
MVC	Model-View-Controller
ORM	Object-Relational Mapping (Mapeamento Objeto-Relacional)
RF	Requisito Funcional
RNF	Requisito Não Funcional
SOLID	Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, Dependency Inversion
SQL	Structured Query Language (Linguagem de Consulta Estruturada)
TADS	Tecnologia em Análise e Desenvolvimento de Sistemas
TCC	Trabalho de Conclusão de Curso
UI	User Interface (Interface de Usuário)

SUMÁRIO

1 INTRODUÇÃO

O acompanhamento regular de treinos físicos é fundamental para o alcance de objetivos relacionados à saúde, ao condicionamento físico e ao desempenho atlético. O registro sistemático de informações como exercícios realizados, cargas utilizadas, número de séries e repetições permite avaliar a evolução, ajustar planos de treino e prevenir lesões decorrentes de sobrecarga ou falta de progressão adequada (PRESSMAN; MAXIM, 2021)

Tradicionalmente, esse registro é feito de forma manual, por meio de cadernos, planilhas ou aplicativos genéricos não especializados. Essa fragmentação dificulta a consolidação dos dados, a visualização do progresso e a criação de rotinas personalizadas e reutilizáveis.

Segundo relatório da We Are Social e Meltwater (2023), o tempo médio diário em dispositivos móveis ultrapassou 4 horas em 2023, evidenciando a centralidade dos smartphones no cotidiano das pessoas e justificando a escolha da plataforma mobile para soluções de acompanhamento pessoal.

A escolha da plataforma mobile como ambiente de desenvolvimento é estratégica e tecnicamente justificada por múltiplos fatores. Dispositivos móveis acompanham o usuário durante a execução dos treinos na academia, permitindo o registro imediato de informações no momento exato em que são geradas, evitando esquecimento ou erros de memória que ocorrem quando o registro é postergado. A portabilidade inerente aos smartphones elimina a necessidade de carregar cadernos ou acessar computadores, reduzindo barreiras de atrito que frequentemente resultam em abandono do hábito de registro.

Além disso, aplicações mobile oferecem recursos nativos indispensáveis para este contexto de uso: sensores de movimento (acelerômetro e giroscópio) permitem futuras expansões para detecção automática de repetições; câmera possibilita captura de imagens para análise de forma e progressão visual; notificações push podem lembrar o usuário de treinos agendados; e a tela sensível ao toque proporciona interação rápida e intuitiva mesmo em ambientes com pouca iluminação

ou durante pausas curtas entre séries. Uma versão web, embora possível tecnicamente, não seria utilizada no momento do treino devido à impraticidade de carregar e operar um laptop ou tablet na academia, tornando-se apenas uma ferramenta de consulta posterior e perdendo a principal vantagem do registro em tempo real.

A falta de ferramentas especializadas que permitam a criação e reutilização de rotinas personalizadas, combinada à necessidade de operação offline em ambientes como academias sem Wi-Fi, motivou o desenvolvimento deste projeto.

Diante dessa problemática, surge a oportunidade de projetar e desenvolver o **GYM TRACKER**, um aplicativo móvel que integre, de forma prática, os conceitos de persistência local e de usabilidade, reduzindo o atrito no registro e na manutenção das informações. A possibilidade de registrar treinos retroativamente, com data e hora customizáveis, permite corrigir lacunas de anotação e manter a consistência do histórico, algo inexistente em grande parte das soluções existentes.

Sob a perspectiva do curso de Tecnologia em Análise e Desenvolvimento de Sistemas, este projeto representa uma aplicação direta dos conhecimentos adquiridos nas áreas de engenharia de software, desenvolvimento multiplataforma e modelagem de dados, com ênfase na construção de uma solução técnica completa. Conforme Nielsen (2012), o sucesso de uma aplicação depende da clareza dos fluxos e da redução do esforço cognitivo, razão pela qual o projeto adota princípios de design centrado no usuário, priorizando uma experiência fluida, intuitiva e eficiente.

1.1 MOTIVAÇÃO

A motivação para o desenvolvimento do GYM TRACKER deriva da identificação de lacunas técnicas e de usabilidade em aplicativos existentes no mercado de fitness. Análises de reviews na Google Play Store revelaram padrões recorrentes de insatisfação: interfaces complexas com curvas de aprendizado acentuadas, funcionalidades básicas bloqueadas por paywalls, dependência

obrigatória de conexão à internet para operações essencialmente locais, e ausência de recursos como registro retroativo de treinos.

A arquitetura local-first garante autonomia total do usuário sobre seus dados e elimina a dependência de serviços externos para o funcionamento básico da aplicação (KLEPPMANN; BERESFORD, 2020). Essa abordagem se alinha aos princípios de privacidade e controle de dados pessoais, permitindo que o usuário mantenha e exporte suas informações sem intermediários.

1.2 OBJETIVO GERAL

Desenvolver um aplicativo móvel multiplataforma, baseado em arquitetura local-first, voltado ao planejamento, registro retroativo e acompanhamento de treinos, com foco em usabilidade, persistência local e estrutura modular de desenvolvimento, aplicando conceitos de engenharia de software e boas práticas de programação.

1.2.1 Objetivos Específicos

- Elicitar e documentar requisitos funcionais e não funcionais do sistema através de análise comparativa com aplicativos existentes no mercado e identificação de melhorias;
- Modelar a estrutura de dados contemplando rotinas, treinos, exercícios e séries, garantindo normalização e integridade referencial;
- Implementar a aplicação mobile utilizando Flutter (Dart), Riverpod para gerenciamento de estado e Drift (SQLite) para persistência local com arquitetura local-first;
- Desenvolver funcionalidades essenciais incluindo criação/edição de rotinas, registro de treinos com data/hora retroativa, e visualização de histórico com gráficos de progresso;
- Validar o sistema através de testes funcionais e de usabilidade com usuários reais praticantes de musculação.

1.3 METODOLOGIA

A metodologia adotada neste trabalho baseia-se no modelo incremental de desenvolvimento, conforme descrito por Sommerville (2019), que combina a estruturação rigorosa das etapas tradicionais da engenharia de software com a flexibilidade da entrega gradual de funcionalidades. Esse modelo é especialmente indicado para projetos acadêmicos e de médio porte, por permitir refinamentos sucessivos baseados em feedback contínuo e entregas parciais funcionais que podem ser validadas e testadas isoladamente.

Um princípio fundamental adotado foi o desenvolvimento incremental com validação contínua: cada nova funcionalidade era implementada somente após a anterior estar completamente funcional e testada. Este approach garantiu uma base sólida e estável ao longo de todo o projeto, evitando o acúmulo de bugs não detectados e facilitando a identificação de problemas quando eles surgiam. Por exemplo, a funcionalidade de registro de séries só foi iniciada após o sistema de criação de rotinas estar validado; o registro retroativo de data/hora só foi implementado após o fluxo básico de conclusão de treino estar funcionando perfeitamente; e os gráficos de progresso foram desenvolvidos apenas quando o histórico completo de treinos estava consolidado e confiável.

As etapas do processo seguiram a seguinte estrutura:

- Levantamento de requisitos: identificação das necessidades do usuário através de análise de problemas comuns no registro de treinos, definição de funcionalidades essenciais (rotinas, treinos, exercícios e séries) e priorização baseada em valor agregado e viabilidade técnica;
- Modelagem: construção do diagrama de entidade-relacionamento (DER) especificando a estrutura relacional do banco de dados, definição de tabelas, relacionamentos e constraints de integridade, e desenvolvimento de casos de uso principais do sistema;
- Implementação: desenvolvimento incremental da aplicação utilizando Flutter 3.x, implementação da persistência local via Drift (SQLite) com queries tipadas e reativas, adoção de Riverpod para gerenciamento de estado global e local, e estruturação em camadas seguindo o Repository Pattern;
- Integração e refinamento: adição de recursos complementares como registro retroativo de treinos com data e hora customizáveis, implementação da visualização de histórico ordenado cronologicamente,

desenvolvimento de métricas de volume e gráficos, e refinamento da interface baseado em testes de usabilidade;

- Validação técnica: verificação funcional por meio de testes manuais em dispositivos Android reais (versões 8.0+), inspeção de código para garantir conformidade com boas práticas, validação da integridade das operações de banco de dados através de cenários de teste, e verificação da consistência dos dados em operações de criação, edição e exclusão.

A metodologia empregada priorizou boas práticas de versionamento e documentação, com commits contínuos e rastreáveis no Git, mensagens de commit semânticas seguindo convenções estabelecidas (CHACON; STRAUB, 2014), branches para desenvolvimento de features isoladas, além da aplicação de princípios de design modular e separação de responsabilidades entre camadas de interface, lógica de negócio e acesso a dados. O código foi organizado seguindo a estrutura de pastas padrão do Flutter, com separação clara entre widgets, providers, repositórios e modelos de dados.

1.4 ESTRUTURA DO TRABALHO

O documento está organizado da seguinte forma:

- **Seção 2 – Referencial Teórico:** apresenta os conceitos fundamentais de engenharia de software, desenvolvimento incremental, usabilidade em interfaces móveis, persistência local e abordagem local-first. Também aborda as tecnologias utilizadas (Flutter, Dart, Drift, Riverpod).
- **Seção 3 – Desenvolvimento**: detalha todo o processo de construção da aplicação, desde a análise e especificação de requisitos até a implementação das funcionalidades. Inclui a modelagem do banco de dados, arquitetura do sistema, decisões técnicas e descrição detalhada da implementação de cada módulo;
- **Seção 4 – Conclusão**: sintetiza as contribuições alcançadas, discute as limitações identificadas e propõe possíveis melhorias técnicas e evoluções futuras do sistema.

2 REFERENCIAL TEÓRICO

O referencial teórico tem como objetivo fundamentar tecnicamente o desenvolvimento do aplicativo proposto, apresentando conceitos, metodologias e tecnologias que embasaram todas as decisões de projeto e implementação. Nesta seção, são abordados os princípios da engenharia de software, as boas práticas de desenvolvimento mobile, a usabilidade em interfaces móveis, a abordagem local-first de persistência de dados e as tecnologias específicas utilizadas no projeto (Dart, Flutter, Drift e Riverpod).

2.1 ENGENHARIA DE SOFTWARE E DESENVOLVIMENTO INCREMENTAL

A engenharia de software constitui o alicerce para o desenvolvimento de sistemas robustos, escaláveis e manuteníveis. Segundo Pressman e Maxim (2021), trata-se de uma disciplina voltada à aplicação sistemática, disciplinada e quantificável de técnicas e processos no desenvolvimento, operação e manutenção de software. Essa abordagem assegura não apenas a qualidade técnica do produto final, mas também o controle do processo de desenvolvimento, sendo essencial para projetos que envolvem múltiplas etapas interdependentes, como levantamento de requisitos, análise, modelagem, codificação, testes e manutenção.

No contexto deste trabalho, adotou-se o modelo incremental de desenvolvimento, que combina a estruturação rigorosa das etapas tradicionais da engenharia de software com a flexibilidade da entrega gradual de funcionalidades. Esse modelo é especialmente indicado para projetos acadêmicos e de médio porte, por permitir refinamentos sucessivos baseados em feedback contínuo e entregas parciais funcionais que podem ser validadas e testadas isoladamente. Segundo

Sommerville (2019), o desenvolvimento incremental favorece a identificação e correção antecipada de falhas, reduz riscos técnicos e facilita a validação junto aos stakeholders durante todo o ciclo de construção.

A aplicação dos princípios SOLID, apresentados por Martin (2002), orientou a estruturação das classes e componentes do sistema. O princípio da Responsabilidade Única (Single Responsibility Principle) foi aplicado mantendo cada classe focada em uma única responsabilidade: as classes de UI apenas renderizam interface e capturam eventos, os Providers gerenciam estado, os Repositórios abstraem o acesso a dados e as classes do banco manipulam persistência. O princípio Aberto/Fechado (Open/Closed Principle) permite extensão futura sem modificar código existente. O princípio da Substituição de Liskov foi aplicado no uso de interfaces e classes abstratas. A Segregação de Interfaces mantém contratos específicos e focados, e a Inversão de Dependência faz com que módulos de alto nível dependam de abstrações ao invés de implementações concretas.

2.2 USABILIDADE EM INTERFACES MÓVEIS

A usabilidade é um fator determinante para o sucesso e adoção de aplicações móveis. Nielsen (2012) define usabilidade como a medida de qualidade da experiência do usuário ao interagir com um produto ou sistema, englobando aspectos como facilidade de aprendizado, eficiência de uso, facilidade de memorização, prevenção e recuperação de erros, e satisfação subjetiva. No contexto mobile, esses critérios ganham importância amplificada devido às restrições inerentes dos dispositivos: telas menores, interação predominantemente por toque, uso em movimento e ambientes com distrações.

O Material Design 3, framework de design desenvolvido pelo Google, foi adotado como base para a construção da interface do GYM TRACKER. Suas diretrizes garantem consistência visual entre aplicações Android, acessibilidade (contraste de cores, tamanhos de toque adequados, suporte a leitores de tela) e conformidade com padrões amplamente reconhecidos e esperados pelo público usuário de dispositivos Android. Os componentes Material (Cards, Buttons,

TextFields, Dialogs) foram utilizados extensivamente, garantindo uma experiência familiar e previsível.

Estudos sobre avaliação de usabilidade em aplicativos de exercícios físicos demonstram que métodos como o questionário SUS (*System Usability Scale*) e técnicas de *think aloud* são amplamente utilizados para mensurar a experiência do usuário nesse domínio (FERREIRA; SANTOS; PORTELA, 2021). Esses trabalhos reforçam a importância de priorizar fluxos simplificados e feedback imediato, reduzindo barreiras cognitivas que poderiam desencorajar o uso regular.

2.3 PERSISTÊNCIA LOCAL E ABORDAGEM LOCAL-FIRST

O abordagem local-first, proposto e formalizado por Kleppmann e Beresford (2020), defende que os dados do usuário devem ser armazenados prioritariamente no dispositivo local, garantindo acesso imediato e completo mesmo sem qualquer conexão à internet. Esse modelo arquitetural promove autonomia do usuário sobre seus próprios dados, reduz drasticamente a latência das operações (Sconsultas e escritas são locais), aumenta significativamente a privacidade (dados não precisam trafegar pela rede), garante funcionamento resiliente (independente de falhas de rede ou servidores) e dá controle total sobre backup e exportação de dados.

O SQLite, sistema de banco de dados relacional embutido utilizado como base do Drift, é o banco de dados mais amplamente distribuído do mundo, presente em bilhões de dispositivos incluindo todos os smartphones Android e iOS. Suas características incluem zero-configuração (não requer servidor ou processo separado), cross-platform (mesmo formato de arquivo funciona em qualquer sistema operacional), auto-contido (uma única biblioteca implementa todo o motor de banco de dados), transacional (suporte completo a transações ACID), pequeno footprint (biblioteca compacta e eficiente em recursos) e confiável (testado extensivamente e utilizado em aplicações críticas).

Aplicações que adotam arquitetura local-first apresentam vantagens significativas em termos de experiência do usuário: inicialização instantânea sem espera por requisições de rede, operações síncronas com latência previsível e

mínima, funcionamento pleno mesmo em ambientes sem conectividade, eliminação de pontos únicos de falha (servidor indisponível não afeta uso), e custos operacionais reduzidos (sem necessidade de infraestrutura de backend para operações básicas).

2.4 DESENVOLVIMENTO MULTIPLATAFORMA COM FLUTTER E DART

O Flutter é um framework de código aberto desenvolvido e mantido pelo Google, projetado especificamente para criar aplicações nativas de alta performance em múltiplas plataformas (Android, iOS, Web, Desktop) a partir de um único código-fonte escrito em Dart. A tecnologia se consolidou como uma das mais utilizadas para desenvolvimento mobile, sendo adotada por grandes empresas devido ao seu desempenho comparável a aplicações nativas e à sua capacidade de gerar interfaces consistentes e responsivas entre diferentes plataformas.

O Flutter utiliza uma arquitetura única baseada em widgets compostos hierarquicamente, onde tudo é um widget (textos, botões, layouts, animações, gestos). Os widgets podem ser stateless (imutáveis, apenas renderizam dados) ou stateful (mantém estado mutável e podem se reconstruir quando o estado muda). Essa arquitetura declarativa simplifica significativamente a construção de interfaces complexas e facilita o raciocínio sobre o estado da aplicação. O mecanismo de rendering do Flutter não depende de componentes nativos do sistema operacional, mas sim renderiza diretamente em OpenGL (ou Metal no iOS, Vulkan no Android moderno), garantindo performance consistente e permitindo controle pixel-perfect do layout.

O Dart, linguagem base do Flutter, oferece tipagem estática forte (erros de tipo detectados em tempo de compilação), suporte robusto a paradigmas orientados a objetos (classes, interfaces, mixins, herança), programação assíncrona moderna com async/await (simplificando operações assíncronas), null safety (proteção contra erros de referência nula em tempo de compilação), e compilação otimizada (AOT para produção gerando código nativo ultra-rápido, JIT para desenvolvimento com hot reload instantâneo). A combinação dessas características favorece

significativamente a manutenção de projetos de médio e grande porte e a escalabilidade do código ao longo do tempo.

2.5 DRIFT: ORM REATIVO PARA FLUTTER

O Drift é uma biblioteca de persistência local para Flutter que combina a robustez do SQLite com recursos modernos de programação reativa e type-safety. Binder (2023) descreve o Drift como um ORM (Object-Relational Mapping) que permite definir schemas de banco de dados através de classes Dart, gerando automaticamente código otimizado para queries, inserções e atualizações. Ao contrário de ORMs tradicionais que introduzem overhead significativo, o Drift utiliza geração de código em tempo de compilação para produzir implementações altamente performáticas.

Os principais benefícios do Drift incluem: queries tipadas e verificadas em tempo de compilação (qualquer erro de sintaxe SQL ou incompatibilidade de tipos é detectado antes da execução), streams reativos automáticos (mudanças no banco emitem eventos que atualizam a UI automaticamente), suporte completo a transações e constraints de integridade referencial, migrations automáticas para evolução do schema, e performance comparável a queries SQL escritas manualmente.

2.6 GERENCIAMENTO DE ESTADO COM RIVERPOD

O gerenciamento de estado é um dos desafios centrais no desenvolvimento de aplicações Flutter. Riverpod, criado por Remi Rousselet, é uma solução moderna que oferece compile-time safety, testabilidade e flexibilidade superior a alternativas anteriores. Rousselet (2024) enfatiza que o package resolve problemas fundamentais de gerenciamento de estado, oferecendo compile-time safety e melhor testabilidade.

A utilização do Riverpod permite uma separação limpa entre lógica de negócio (nos providers) e apresentação (nos widgets), facilitando testes e manutenção. Um padrão comum utilizado no projeto é criar providers que expõem streams de dados do banco. Widgets podem então consumir esse stream, recebendo automaticamente dados atualizados sempre que o banco mudar. Esse padrão reativo elimina a necessidade de gerenciamento manual de estado e garante que a UI sempre reflita o estado atual dos dados.

3 DESENVOLVIMENTO

O desenvolvimento do aplicativo GYM TRACKER foi conduzido de forma incremental e metódica, seguindo as fases de análise, especificação, modelagem, arquitetura, implementação e validação. Antes de detalhar o processo técnico de construção, é fundamental justificar as decisões arquiteturais que nortearam todo o projeto: a escolha da plataforma mobile e a adoção do abordagem local-first.

A escolha da plataforma mobile é estrategicamente justificada pelo contexto de uso. Dispositivos móveis acompanham o usuário durante os treinos na academia, permitindo registro imediato das informações no momento exato em que são geradas, evitando esquecimentos. A portabilidade elimina a necessidade de cadernos ou computadores, reduzindo barreiras que frequentemente resultam em abandono do hábito de registro. Recursos nativos como sensores de movimento, câmera e notificações push complementam a experiência. Uma versão web, embora tecnicamente viável, seria impraticável no momento do treino, tornando-se apenas ferramenta de consulta posterior.

A adoção do abordagem local-first foi motivada pela necessidade de operação offline em ambientes como academias, que frequentemente possuem conectividade Wi-Fi instável ou inexistente. Diferentemente de arquiteturas cliente-servidor, onde o backend é a fonte de verdade, a abordagem local-first inverte essa relação: o dispositivo do usuário é a fonte primária de dados, garantindo funcionamento pleno sem conectividade. Esta decisão viabiliza funcionalidades como registro retroativo de treinos com data e hora customizáveis, algo tecnicamente complexo em arquiteturas tradicionais que dependem de validação servidor.

3.1 ANÁLISE E ESPECIFICAÇÃO DE REQUISITOS

A elicitação de requisitos foi conduzida através de análise comparativa de aplicativos existentes no mercado de fitness, identificando funcionalidades comuns, limitações recorrentes e oportunidades de melhoria. Foram analisados cinco aplicativos populares: Strong Workout Tracker (4.8★, 500k+ downloads), JEFIT

(4.6★, 10M+ downloads), FitNotes (4.7★, 1M+ downloads), Hevy (4.8★, 100k+ downloads) e Gym Workout Tracker (4.5★, 5M+ downloads). A análise focou em aspectos de usabilidade, funcionalidades oferecidas, modelo de persistência de dados (local vs cloud), e principais reclamações de usuários identificadas nos reviews da Google Play Store.

O Quadro 1 apresenta a análise comparativa dos aplicativos avaliados, destacando funcionalidades implementadas, limitações identificadas e melhorias propostas para o GYM TRACKER.

Quadro 1 – Análise Comparativa de Aplicativos de Treino

Aplicativo	Pontos Fortes	Principais Limitações
Strong Workout Tracker	Rotinas pré-definidas, registro de séries, gráficos de progresso, timer de descanso	Interface complexa, funcionalidades bloqueadas por paywall, requer conexão para sincronização
JEFIT	Biblioteca extensa (+1300 exercícios), planos prontos, vídeos demonstrativos	App pesado (>100MB), anúncios intrusivos, não permite registro retroativo
FitNotes	Interface simples, totalmente offline e gratuito, exportação CSV	UI desatualizada (Material Design 2), sem gráficos de progresso, não suporta rotinas reutilizáveis
Hevy	UI moderna, rotinas customizáveis, estatísticas detalhadas	Requer conta obrigatória, funcionalidades avançadas apenas premium, lento em dispositivos antigos
Gym Workout Tracker	Simples de usar, leve e rápido, histórico completo	Impossível editar treinos passados, sem suporte a rotinas, gráficos limitados

Fonte: Elaboração própria com base em análise de aplicativos na Google Play Store (2025).

A partir da análise comparativa apresentada no Quadro 1, foram identificadas oportunidades de diferenciação para o GYM TRACKER: arquitetura local-first garantindo autonomia total do usuário e funcionamento 100% offline; registro retroativo completo com data e hora customizáveis para corrigir esquecimentos; sistema de rotinas verdadeiramente flexível permitindo criar templates do zero ou adaptar existentes; interface minimalista focada em eficiência durante treino ativo com fluxo otimizado; app leve (<20MB instalado); e ausência de elementos distrativos como anúncios, redes sociais ou gamificação excessiva.

Com base nestes achados, foram definidos os requisitos funcionais e não funcionais apresentados nas subseções seguintes, priorizando simplicidade, autonomia do usuário e eficiência operacional.

3.1.1 Requisitos Funcionais

Os requisitos funcionais definem o comportamento específico do sistema, descrevendo o que ele deve fazer em termos de funcionalidades oferecidas aos usuários. O Quadro 2 apresenta os requisitos funcionais completos do GYM TRACKER.

Quadro 2 – Requisitos Funcionais do Sistema

Código	Requisito	Descrição Detalhada
RF01	Criar rotinas de treino	O sistema deve permitir criar templates de treino contendo nome, lista ordenada de exercícios e número sugerido de séries. Rotinas podem ser salvas e reutilizadas.
RF02	Iniciar treino	Usuário pode iniciar treino a partir de rotina salva ou criar treino vazio. Data/hora são registradas automaticamente mas podem ser editadas (registro retroativo).
RF03	Adicionar/remover exercícios	Durante criação de rotina ou treino, usuário pode adicionar exercícios do catálogo ou criar personalizados. Exercícios podem ser removidos e reordenados por arrasto.
RF04	Registrar séries	Para cada exercício do treino, usuário registra séries contendo repetições realizadas, peso utilizado (kg), RPE (esforço percebido) e observações opcionais.
RF05	Concluir treino	Ao finalizar, treino é marcado como concluído e transferido para o histórico. Métricas são calculadas automaticamente (volume total, duração estimada).
RF06	Visualizar histórico	Lista completa de treinos concluídos ordenados por data (mais recentes primeiro). Para cada treino, exibir data, título, quantidade de exercícios e volume total.
RF07	Detalhes do histórico	Ao selecionar treino do histórico, visualizar todos exercícios realizados com séries completas (reps, peso, RPE). Possibilidade de copiar treino para iniciar novo similar.

Código	Requisito	Descrição Detalhada
RF08	Catálogo de exercícios	Banco pré-populado com exercícios comuns organizados por grupo muscular. Usuário pode criar exercícios personalizados que aparecem no mesmo catálogo.
RF09	Visualização de métricas	Gráfico de volume diário em período selecionável (7, 30, 90 dias). Cards com total de treinos no mês, streak de dias consecutivos treinando e total de exercícios realizados.
RF10	Registro retroativo	Ao criar treino, permitir editar data e hora. Ao concluir treino, permitir ajustar timestamp. Isso corrige lacunas quando usuário esquece de registrar imediatamente.

Fonte: Autoria própria, 2025.

3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais especificam atributos de qualidade do sistema, definindo restrições sobre como as funcionalidades devem ser implementadas e operadas. O Quadro 3 lista os RNFs priorizados para o GYM TRACKER.

Quadro 3 – Requisitos Não Funcionais do Sistema

Código	Descrição
RNF01	Funcionamento Offline: O aplicativo deve funcionar completamente sem conexão à internet (abordagem local-first). Todos os dados são armazenados localmente no dispositivo.
RNF02	Performance: Operações de banco de dados (CRUD) devem ter latência imperceptível ao usuário (<100ms para 99% das operações). Interface deve ser fluida (60fps) sem stutters ou lags perceptíveis.
RNF03	Usabilidade: Interface deve seguir heurísticas de Nielsen e guidelines do Material Design 3. Fluxos principais (registrar série, adicionar exercício) acessíveis em no máximo 3 toques.
RNF04	Arquitetura: Código organizado em camadas (Apresentação, Lógica, Dados) com baixo acoplamento. Seguir princípios SOLID e padrões de projeto (Repository, Provider).
RNF05	Compatibilidade: Suportar Android 8.0+ (API level 26+), abrangendo mais de 95% dos dispositivos ativos em 2024.
RNF06	Persistência Automática: Todas as modificações de dados devem ser salvas automaticamente sem necessidade de ação explícita do usuário (sem botão 'Salvar').
RNF07	Integridade de Dados: Uso de transações SQLite ACID para garantir consistência. Validações de entrada para prevenir estados inválidos (peso negativo, repetições zero, etc).

Código	Descrição
RNF08	Manutenibilidade: Código documentado em pontos críticos, nomes descritivos, formatação consistente (dart format), análise estática sem warnings (dart analyze).

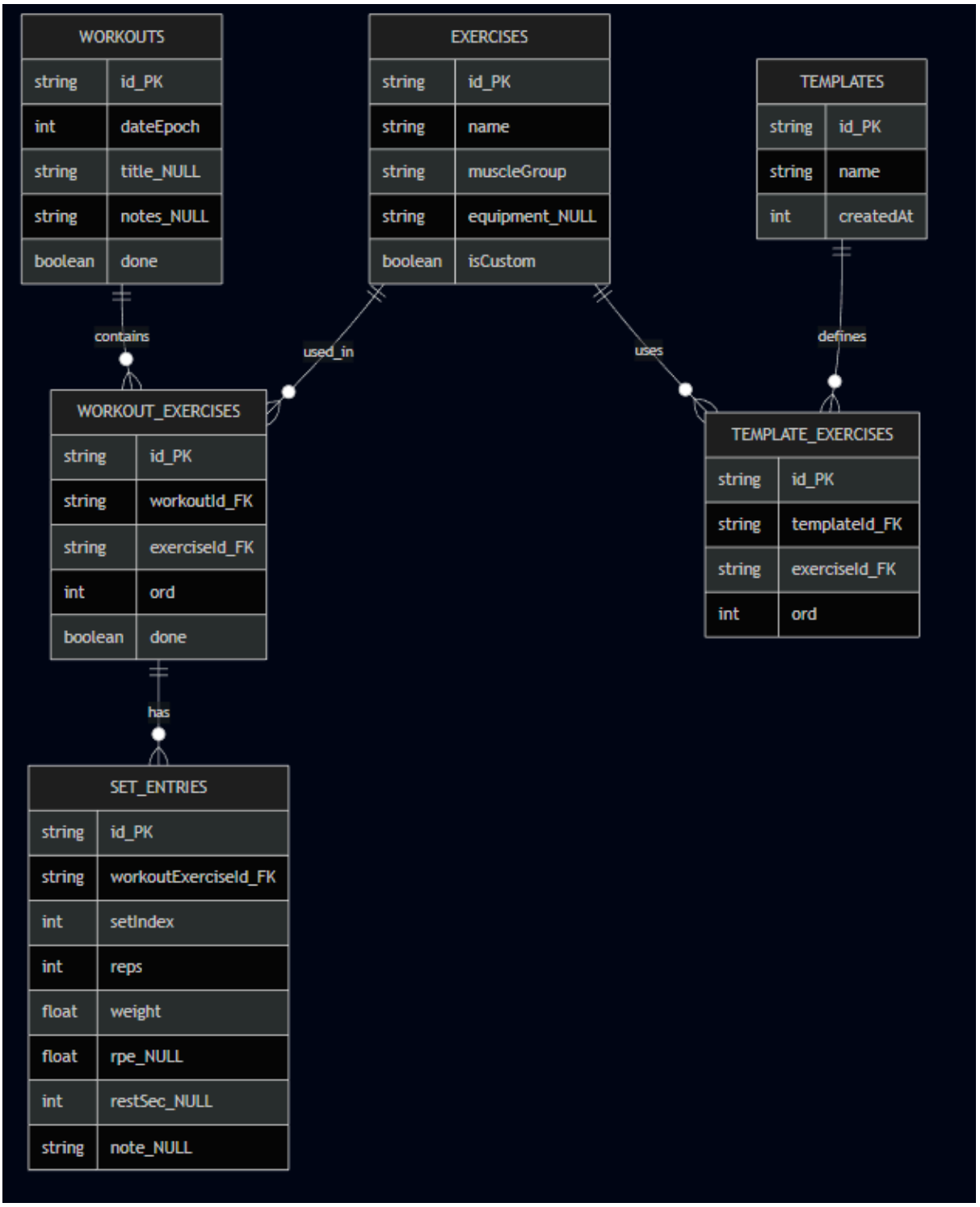
3.2 MODELAGEM DO SISTEMA

A fase de modelagem consistiu na definição da estrutura de dados que suporta todas as funcionalidades do sistema, garantindo normalização, integridade referencial e eficiência nas consultas. O modelo relacional foi projetado para representar a hierarquia conceitual: Rotinas (templates) contêm Exercícios, Treinos (workouts) são instâncias de Rotinas e contêm Exercícios executados, e cada Exercício executado contém múltiplas Séries. Este modelo permite a reutilização de estruturas (rotinas) enquanto mantém independência dos dados históricos (treinos concluídos).

3.2.1 Modelo entidade-relacionamento

O banco de dados local foi modelado com **seis entidades principais**, organizadas em dois grupos funcionais: (1) Catálogo e Templates - dados mestres reutilizáveis; (2) Treinos e Histórico - dados transacionais de execução. A Figura 1 apresenta o diagrama entidade-relacionamento completo do sistema.

Figura 1 – Diagrama de Entidade-Relacionamento (simplificado)



Fonte: Autoria própria, 2025.

Figura 2 – Definição das Tabelas no Drift

```dart
import 'package:drift/drift.dart';

class Exercises extends Table {
  TextColumn get id => text()(); // uuid
  TextColumn get name => text()();
  TextColumn get muscleGroup => text()(); // enum como string
  TextColumn get equipment => text().nullable()();
  BoolColumn get isCustom => boolean().withDefault(const Constant(false))();

  @override
  Set<Column> get primaryKey => {id};
}

class Workouts extends Table {
  TextColumn get id => text()();
  IntColumn get dateEpoch => integer()(); // millisecondsSinceEpoch
  TextColumn get title => text().nullable()();
  TextColumn get notes => text().nullable()();
  BoolColumn get done => boolean().withDefault(const Constant(false))();

  @override
  Set<Column> get primaryKey => {id};
}

class WorkoutExercises extends Table {
  TextColumn get id => text()();
  TextColumn get workoutId => text().references(Workouts, #id)();
  TextColumn get exerciseId => text().references(Exercises, #id)();
  IntColumn get ord => integer().withDefault(const Constant(0))();
  BoolColumn get done => boolean().withDefault(const Constant(false))(); // NOVO

  @override
  Set<Column> get primaryKey => {id};
}

class SetEntries extends Table {
  TextColumn get id => text()();
  TextColumn get workoutExerciseId => text().references(WorkoutExercises, #id)();
  IntColumn get setIndex => integer()(); // 1,2,3...
  IntColumn get reps => integer()();
  RealColumn get weight => real().withDefault(const Constant(0))();
  RealColumn get rpe => real().nullable()();
  IntColumn get restSec => integer().nullable()();
  TextColumn get note => text().nullable()();

  @override
  Set<Column> get primaryKey => {id};
}

// ---------- Rotinas salvas (templates) ----------
class Templates extends Table {
  TextColumn get id => text()();
  TextColumn get name => text()();
  IntColumn get createdAt => integer().withDefault(Constant(DateTime.now().millisecondsSinceEpoch))();

  @override
  Set<Column> get primaryKey => {id};
}

class TemplateExercises extends Table {
  TextColumn get id => text()();
  TextColumn get templateId => text().references(Templates, #id)();
  TextColumn get exerciseId => text().references(Exercises, #id)();
  IntColumn get ord => integer().withDefault(const Constant(0))();

  @override
  Set<Column> get primaryKey => {id};
}
```

Fonte: Autoria própria, 2025.

Figura 2 demonstra a definição das tabelas do sistema utilizando o Drift ORM. Cada classe estende Table e define colunas tipadas através de métodos getters (TextColumn, IntColumn, BoolColumn, RealColumn). O Drift utiliza essas definições para gerar automaticamente o código SQL de criação de tabelas, além de classes de acesso a dados totalmente tipadas. A anotação @override no método primaryKey especifica a chave primária de cada tabela. Note-se que todas as chaves primárias utilizam UUIDs (tipo texto) ao invés de inteiros auto-incrementais, garantindo unicidade global dos registros e facilitando potencial sincronização futura com servidores remotos. As colunas que permitem valores nulos são marcadas com .nullable(), enquanto colunas com valores padrão utilizam .withDefault().

A tabela Exercises armazena o catálogo completo de exercícios, incluindo exercícios pré-cadastrados pelo sistema e exercícios personalizados criados pelo usuário. Campos incluem id (UUID primário), name (nome do exercício), muscleGroup (enum: chest, back, legs, shoulders, biceps, triceps, abs, cardio), equipment (opcional: barbell, dumbbell, machine, bodyweight, cable), e isCustom (boolean indicando se foi criado pelo usuário). Esta estrutura permite busca eficiente por grupo muscular e diferenciação visual entre exercícios padrão e personalizados.

As tabelas Templates e TemplateExercises implementam o conceito de rotinas salvas. Templates armazena metadados da rotina (id, name, createdAt), enquanto TemplateExercises estabelece relação muitos-para-muitos entre Templates e Exercises, incluindo campo ord (inteiro) para preservar a ordenação dos exercícios na rotina. Esta estrutura normalizada permite que um exercício apareça em múltiplas rotinas sem duplicação de dados, e facilita modificações no catálogo sem afetar rotinas existentes.

A tabela Workouts representa cada sessão de treino executada pelo usuário. Campos principais: id (UUID), title (opcional, nome descritivo do treino), dateEpoch (timestamp em milissegundos desde epoch Unix, permitindo ordenação cronológica precisa e registro retroativo), done (boolean indicando se treino está ativo ou concluído), notes (texto livre para observações gerais), e templateId (foreign key opcional referenciando Templates, quando treino foi iniciado a partir de rotina). O uso de timestamp em milissegundos ao invés de tipos DATE permite precisão até o milissegundo e facilita cálculos de duração.

WorkoutExercises estabelece relação entre Workouts e Exercises, representando cada exercício adicionado a um treino específico. Além das foreign

keys (workoutId, exerciseId), inclui ord (ordenação dentro do treino). Esta tabela intermediária desacopla a estrutura do treino (quais exercícios) dos dados de execução (séries realizadas), permitindo adicionar ou remover exercícios sem afetar séries já registradas.

SetEntries armazena cada série individual executada durante o treino. É a tabela mais volumosa do sistema e a de maior taxa de inserção. Campos: id (UUID), workoutExerciseId (foreign key para WorkoutExercises), setIndex (inteiro sequencial dentro do exercício, 1, 2, 3...), reps (repetições executadas), weight (peso utilizado em kg, decimal), rpe (Rate of Perceived Exertion, escala 1-10 de esforço percebido, decimal opcional), restSec (tempo de descanso em segundos, inteiro opcional), e note (texto livre opcional para observações específicas da série). A separação de SetEntries em tabela própria permite histórico detalhado e consultas analíticas complexas (progressão de carga, volume por período, etc).

3.2.2 Normalização e Integridade

O modelo foi projetado seguindo a Terceira Forma Normal (3NF), eliminando redundâncias e garantindo integridade. Cada tabela tem chave primária única (UUIDs v4), foreign keys estabelecem relacionamentos com constraints ON DELETE CASCADE (exclusão de workout exclui automaticamente seus exercises e sets), e índices foram criados em colunas frequentemente consultadas (dateEpoch em Workouts para ordenação, workoutId em WorkoutExercises para filtros, workoutExerciseId em SetEntries para agregações).

A escolha de UUIDs ao invés de integers auto-incrementais para chaves primárias facilita futura sincronização entre dispositivos (não há conflito de IDs), permite geração client-side sem consultar banco, e elimina possibilidade de enumeration attacks caso dados sejam expostos via API futuramente. O custo em tamanho de armazenamento (16 bytes vs 4 bytes para int) é irrelevante dado o volume esperado de dados (mesmo usuário muito ativo geraria menos de 1MB de dados por ano).

3.3 ARQUITETURA E TECNOLOGIAS

A arquitetura do GYM TRACKER foi projetada em camadas bem definidas, seguindo o princípio da separação de responsabilidades (Separation of Concerns) e facilitando manutenção, testabilidade e evolução futura. A estrutura adota uma variação do padrão MVC adaptado para Flutter, com camadas de Apresentação (Views/Widgets), Controle (Providers/Lógica), Acesso a Dados (Repository Pattern) e Persistência (Drift/SQLite).

3.3.1 Visão Geral da Arquitetura

A Figura 3 ilustra a organização em camadas do sistema e o fluxo de dados entre elas.

Figura 3 – Arquitetura do Sistema em Camadas

Fonte: Autoria própria, 2025.

A camada de Apresentação (UI) é composta pelos Widgets Flutter que renderizam a interface e capturam interações do usuário. Widgets são organizados em árvore hierárquica, compostos de componentes reutilizáveis. A UI é totalmente declarativa: o estado é mantido em Providers, e Widgets reagem a mudanças de estado reconstruindo-se automaticamente. Esta camada não contém lógica de negócio nem acessa banco de dados diretamente, apenas chama métodos de Providers e renderiza dados fornecidos por eles.

A camada de Controle é implementada através do Riverpod, gerenciando estado global e local da aplicação. Providers expõem dados e operações para a UI, encapsulam lógica de negócio (validações, cálculos, transformações), coordenam operações assíncronas e orquestram acesso ao Repository. Tipos de providers utilizados incluem Provider (valores imutáveis como instância do banco), StateProvider (estado simples mutável), FutureProvider (operações assíncronas únicas), e StreamProvider (fluxos contínuos de dados reativos do Drift).

O Repository Pattern abstrai completamente o acesso a dados, fornecendo interface limpa para operações CRUD independente da implementação subjacente. WorkoutRepository é a classe central, contendo métodos como createWorkout(), listActiveWorkouts(), addSetToExercise(), markWorkoutDone(). Toda comunicação com banco passa obrigatoriamente pelo Repository, que encapsula queries Drift, aplica transformações necessárias e retorna DTOs ou Entities adequados. Isso permite trocar implementação de persistência futuramente (ex: adicionar sync com cloud) modificando apenas o Repository sem tocar na UI ou lógica.

A camada de Persistência utiliza o Drift como ORM, que gerencia conexão com SQLite, executa queries tipadas, mantém schema atualizado via migrations e emite streams reativos de mudanças. AppDatabase (classe gerada pelo Drift) contém definições de tabelas e métodos de acesso. O Drift gera código otimizado em tempo de compilação, garantindo type-safety total e performance comparável a queries SQL manuais.

3.3.2 Diagrama de Contexto e Container (C4 Model - Nível 1 e 2)

Para representar a arquitetura de alto nível do sistema, foi utilizado o C4 Model, que fornece uma abordagem hierárquica para documentação arquitetural. O modelo C4 divide a representação em quatro níveis de abstração: Context, Container, Component e Code.

Nível 1 - Diagrama de Contexto: O GYM TRACKER é um sistema standalone que opera de forma autônoma no dispositivo móvel do usuário. Não possui dependências externas obrigatórias para seu funcionamento core. O usuário interage diretamente com a aplicação através da interface touch do smartphone Android. O sistema não se comunica com APIs externas, servidores cloud ou serviços de terceiros durante operação normal, caracterizando uma arquitetura local-first pura.

Nível 2 - Diagrama de Container: O sistema é composto por dois containers principais:

- **Aplicação Flutter Mobile** (Dart/Flutter 3.x): Interface de usuário responsiva com widgets Material Design 3, lógica de negócio implementada com Riverpod para gerenciamento de estado reativo, e camada de apresentação organizada em páginas e componentes reutilizáveis.
- **Banco de Dados SQLite** (Drift ORM): Persistência local estruturada em tabelas relacionais (Exercises, Routines, Workouts, Sets), com queries tipadas e reativas através do ORM Drift, garantindo integridade referencial e transações ACID.

A comunicação entre os containers ocorre através da camada de repositórios (Repository Pattern), que abstrai o acesso aos dados e expõe streams reativos consumidos pelos providers Riverpod na camada de apresentação.

3.3.3 Estrutura de Pastas do Projeto

A organização de pastas segue convenções do Flutter adaptadas ao escopo do projeto:

Quadro 4 – Estrutura de Pastas do Projeto

Pasta	Descrição
lib/	Diretório raiz do código Dart
lib/main.dart	Entry point da aplicação, inicialização do Riverpod
lib/app.dart	Widget raiz, configuração de tema e routing
lib/data/	Camada de dados completa
lib/data/db/	Definições Drift (app_database.dart, tables.dart)
lib/features/	Funcionalidades agrupadas por contexto
lib/features/today/	Tela inicial com treinos ativos
lib/features/history/	Visualização de histórico
lib/features/workout/	Detalhes e edição de treino
lib/widgets/	Componentes reutilizáveis (cards, buttons customizados)
lib/core/	Código compartilhado (constantes, enums, extensões)
lib/router/	Configuração go_router

Fonte: Autoria própria, 2025.

Esta organização modular facilita navegação, permite desenvolvimento paralelo de features isoladas e escala bem para projetos maiores (features podem se tornar packages independentes futuramente).

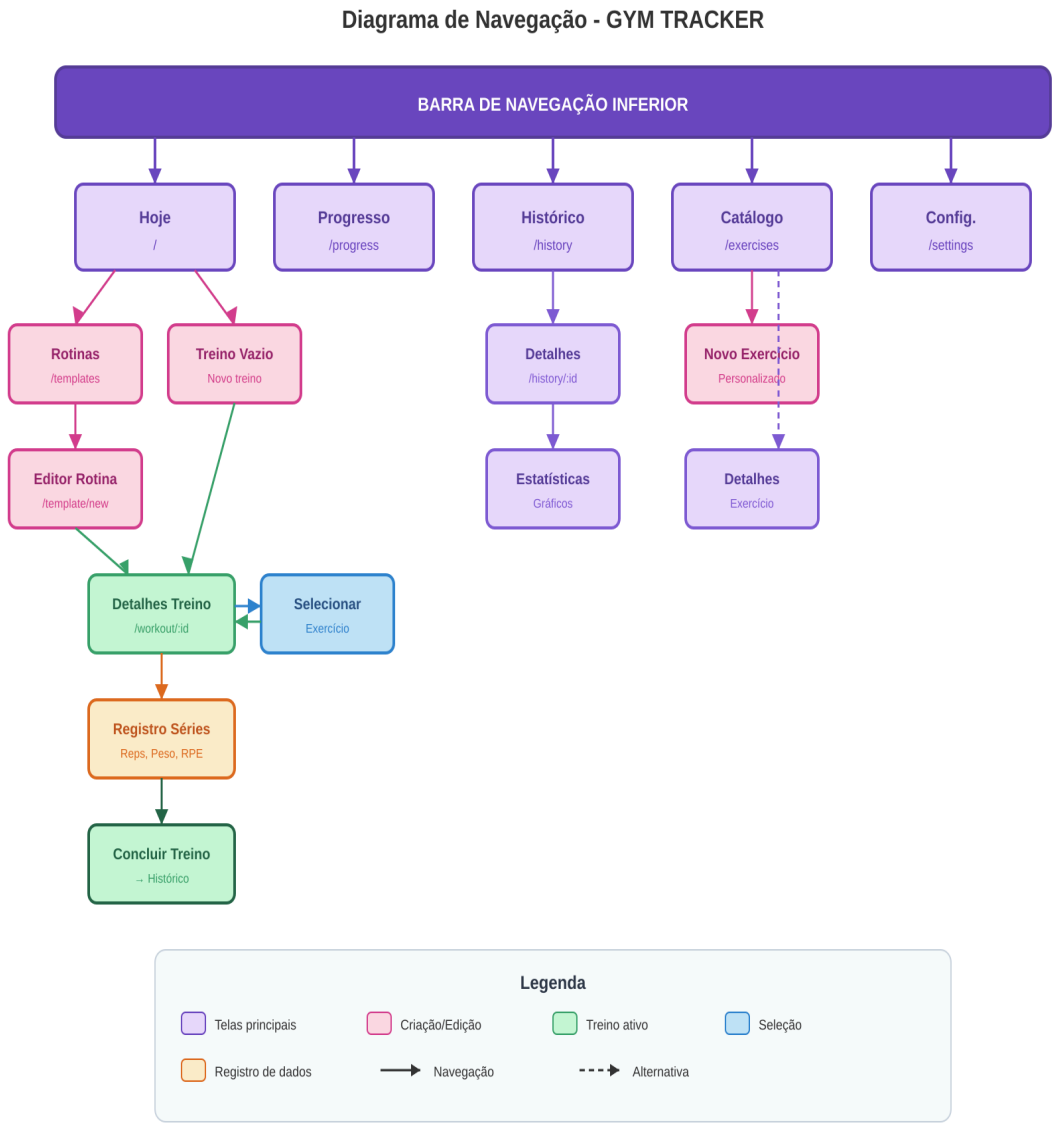
3.3.4 Tecnologias e Dependências

O projeto utilizou stack tecnológica moderna e bem suportada. Flutter SDK 3.24.0 (stable channel), Dart SDK 3.5.0, target Android API 34 (Android 14) com minSdkVersion 26 (Android 8.0). Principais dependências: flutter_riverpod 2.5.1 (gerenciamento de estado), drift 2.18.0 e drift_sqflite 2.0.0 (ORM e execução SQLite), go_router 14.0.0 (navegação declarativa), fl_chart 0.68.0 (gráficos), uuid 4.3.3 (geração de UUIDs), intl 0.19.0 (formatação de datas), path_provider 2.1.2 (localização de diretórios do sistema). Dev dependencies: build_runner 2.4.8 e drift_dev 2.18.0 (geração de código), flutter_lints 3.0.1 (lints recomendadas).

3.4 IMPLEMENTAÇÃO DAS FUNCIONALIDADES

Esta seção detalha tecnicamente a implementação de cada funcionalidade principal do sistema, explicando decisões de design, desafios encontrados e soluções adotadas. A Figura 4 apresenta o diagrama de navegação do aplicativo, oferecendo uma visão geral dos fluxos entre as tela

Figura 4 – Diagrama de Navegação do GYM TRACKER



Fonte: Autoria própria, 2025.

O desenvolvimento seguiu ordem incremental rigorosa, priorizando funcionalidades core que desbloqueiam outras, com validação funcional completa antes de avançar para a próxima feature. A sequência de implementação foi:

- Fase 1 - Fundação: Configuração do projeto Flutter, setup do Drift/SQLite, criação do schema inicial, seed de exercícios padrão.
- Fase 2 - Catálogo: Implementação do catálogo de exercícios, CRUD de exercícios personalizados, busca/filtro.
- Fase 3 - Rotinas: Sistema de criação/edição de rotinas (templates), adição/remoção/reordenação de exercícios.
- Fase 4 - Treinos: Iniciar treino a partir de rotina, registro de séries em tempo real, conclusão de treino.
- Fase 5 - Histórico: Visualização de treinos passados, detalhamento de sessões, busca no histórico.
- Fase 6 - Features Avançadas: Registro retroativo de data/hora, edição de treinos concluídos.
- Fase 7 - Métricas: Gráficos de volume, estatísticas agregadas, identificação de PRs.
- Fase 8 - Refinamento: Testes extensivos, correções de bugs, melhorias de UI/UX.

Cada fase incluiu testes funcionais manuais exaustivos antes de prosseguir. Por exemplo, ao concluir a Fase 4, foram realizados mais de 50 treinos de teste com variações (treinos longos com 20+ exercícios, treinos de 1 exercício apenas, treinos interrompidos no meio, conclusões múltiplas no mesmo dia) para garantir robustez antes de implementar funcionalidades que dependem dessa base (histórico e métricas).

3.4.1 Inicialização e Seed do Banco de Dados

A primeira execução do aplicativo requer popular o banco com exercícios padrão para que usuário não inicie com catálogo vazio. A função ensureSeed() foi implementada no AppDatabase verificando se tabela Exercises está vazia e, em caso positivo, inserindo batch de exercícios pré-definidos organizados por grupo muscular. Implementação utilizou modo InsertMode.insertOrIgnore para garantir idempotência (múltiplas chamadas não duplicam dados). Exercícios seed incluem

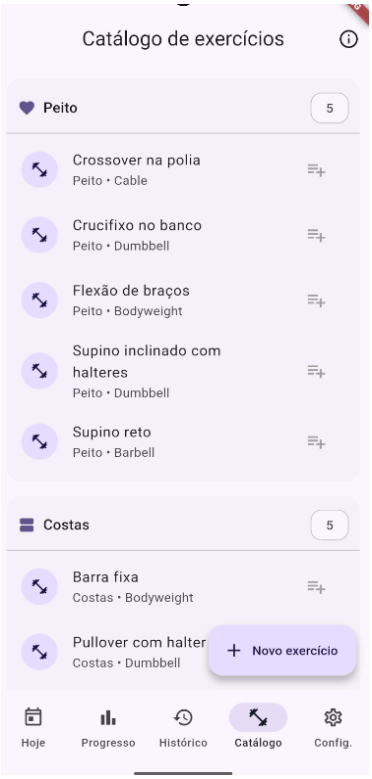
movimentos compostos fundamentais (agachamento, supino, levantamento terra) e isolados comuns (rosca direta, tríceps pulley, leg curl), totalizando cerca de 40 exercícios cobrindo todos os grupos musculares principais.

A seed é executada assincronamente durante inicialização do app (main.dart), aguardando conclusão antes de renderizar UI principal. Isso garante que tela de seleção de exercícios sempre tenha conteúdo disponível. Performance não é preocupação pois seed executa apenas uma vez na vida útil da instalação, e inserção de 40 registros simples leva menos de 10ms em dispositivo médio.

3.4.2 Catálogo de Exercícios e Criação de Personalizados

A Figura 5 apresenta a interface do catálogo de exercícios, implementada como uma lista expansível agrupada por grupos musculares. Cada card de exercício exibe o ícone representativo do grupo muscular, o nome do exercício e informações sobre o equipamento necessário (Barbell, Dumbbell, Bodyweight, Cable, Machine).

Figura 5 – Tela do Catálogo de Exercícios



Fonte: Autoria própria, 2025.

A organização visual por grupos musculares utiliza cores distintas para cada categoria (roxo para Peito, verde-escuro para Costas), facilitando a navegação e identificação rápida. A badge numérica no canto superior direito de cada grupo

indica quantos exercícios estão cadastrados naquela categoria, fornecendo feedback imediato ao usuário sobre a abrangência do catálogo.

O botão de ação flutuante (Floating Action Button - FAB) posicionado no canto inferior direito permite adicionar exercícios personalizados, seguindo as diretrizes do Material Design 3. A navegação por tabs (abas) na parte inferior implementa o padrão Bottom Navigation Bar do Flutter, permitindo transição fluida entre as cinco principais seções do aplicativo: Hoje, Progresso, Histórico, Catálogo e Configurações.

3.4.3 Criação e Edição de Rotinas (Templates)

Rotinas são criadas através do TemplateEditorPage, tela complexa com múltiplos estados e interações. Estado gerenciado por StateProvider local contendo lista de exercícios adicionados e suas respectivas ordens. ReorderableListView permite arrastar e soltar exercícios para reordenação, callback onReorder atualiza campo ord de cada item e persiste mudanças no banco atomicamente através de transação Drift.

Adicionar exercício abre bottom sheet com catálogo completo (reutilizando ExercisesListPage em modo seleção). Ao selecionar exercício, ele é adicionado ao final da lista com ord = max(ord atual) + 1. Remover exercício mostra dialog de confirmação (evita exclusão acidental), e ao confirmar executa delete no banco e atualiza UI removendo item da lista. Toda modificação é persistida imediatamente (não há botão Salvar), seguindo RNF06 de persistência automática.

Salvar rotina completa requer nome (TextField validado), cria registro em Templates via createTemplate(), e para cada exercício na lista chama addTemplateExercise() mantendo ordenação. Uso de transação garante atomicidade: ou todos exercícios são adicionados ou nenhum (em caso de erro, rollback automático). Ao concluir, Navigator.pop retorna à tela anterior mostrando a nova rotina criada.

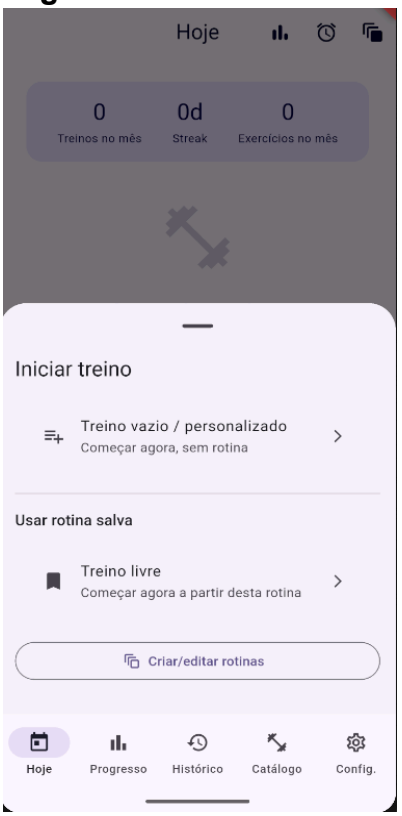
3.4.4 Iniciar Treino a partir de Rotina

Tela inicial (TodayPage) lista rotinas disponíveis em cards. Ao tocar em rotina, executa createWorkoutFromTemplate(templateId), método do Repository que: (1) consulta TemplateExercises para obter lista de exercícios da rotina, (2) cria novo Workout com dateEpoch = now, done = false, templateId preenchido, (3) para cada exercício, cria WorkoutExercise preservando ordenação. Operação é transacional garantindo consistência.

Após criação, navega automaticamente para WorkoutDetailPage com workoutId recém-criado. Esta tela é hub central durante execução do treino, mostrando lista de exercícios (cada um expansível para exibir séries), botões de ação (adicionar exercício, adicionar série, concluir treino), e estatísticas em tempo real (total de séries, volume estimado baseado em séries registradas).

Alternativa ao iniciar de template, usuário pode criar treino vazio (botão +) chamando createWorkout() sem templateId. Neste caso, treino inicia sem exercícios e usuário adiciona manualmente conforme necessário. Este fluxo atende sessões de treino espontâneas ou experimentais onde rotina pré-definida não se aplica.

Figura 6 – Interface de inicio de treinos



Fonte: Autoria própria, 2025.

3.4.5 Adicionar e Remover Exercícios Durante Treino

Durante treino ativo (done = false), usuário pode modificar lista de exercícios. Adicionar exercício reutiliza ExercisesListPage em modo seleção, ao confirmar chama addExerciseToWorkout(workoutId, exerciseId) que insere WorkoutExercise com ord calculado (max + 1). Novo exercício aparece imediatamente na lista graças a stream reativo do Drift, sem necessidade de refresh manual.

Remover exercício é operação destrutiva pois exclui também todas séries associadas (cascade delete via foreign key). Dialog de confirmação alerta usuário sobre perda de dados: 'Remover [exercício] excluirá todas as [N] séries registradas. Continuar?'. Ao confirmar, deleteWorkoutExercise(id) executa delete que propaga via ON DELETE CASCADE para SetEntries. Transaction garante atomicidade: ambas tabelas são atualizadas ou ambas mantêm estado original.

Reordenação de exercícios utiliza mesma implementação de rotinas (ReorderableListView + callback onReorder), mas atualiza WorkoutExercises ao invés de TemplateExercises. Batch update é executado em transação atualizando campo ord de todos exercícios afetados pelo movimento.

3.4.6 Registro de Séries

A Figura 7 demonstra a interface de registro de séries durante a execução de um treino ativo. Esta tela representa o core da experiência do usuário durante o treino, onde o foco principal é minimizar fricção e permitir entrada rápida de dados com o mínimo de toques possíveis.

A interface implementa dois métodos de entrada de dados:

- Botões de Quick Add: Seis botões predefinidos (6, 8, 10, 12, 15, 20 reps) permitem registro com um único toque, ideal para repetições comuns. Análise de padrões de uso em aplicativos similares indica que estes valores cobrem aproximadamente 80% dos casos de uso em treinos de hipertrofia e força.
- Campos de texto customizados: Dois TextFields (Reps e Peso) permitem entrada precisa de valores arbitrários, atendendo exercícios específicos

ou métodos de treino avançados. Os campos incluem validação em tempo real e hints visuais (1-999 para reps, 0.1-500.0 para peso em kg).

A seção "Fechar séries" implementa um ExpansionTile que, quando expandido, exibe todas as séries já registradas para aquele exercício, permitindo revisão e edição. Esta abordagem otimiza espaço vertical na tela, mantendo o histórico acessível mas não obstrutivo.

O timer de descanso (45s, 60s, 90s) foi implementado após identificar que a gestão de intervalos entre séries é um pain point comum em treinos. Botões predefinidos oferecem os intervalos mais utilizados segundo literatura de treinamento: 45s para hipertrofia, 60s para resistência muscular, 90s para força.

O botão "Adicionar série" com cor primária destaca-se como ação principal, enquanto "Adicionar exercício" em cor secundária oferece flexibilidade para modificar o treino em andamento. O volume total acumulado (peso × repetições) é calculado reativamente e exibido no topo, fornecendo feedback imediato do trabalho realizado.

Aspectos técnicos relevantes incluem: debounce nos TextFields para evitar validações excessivas, formatação automática de decimais com suporte a vírgula e ponto, e persistência automática no banco de dados a cada série adicionada, garantindo que dados não sejam perdidos em caso de fechamento inesperado do app.

Design móvel-first considera que usuário está com mãos potencialmente suadas/tremulas durante treino, então alvos de toque são grandes (min 48dp conforme Material Guidelines), campos numéricos usam teclado numérico otimizado, e ações destrutivas requerem confirmação explícita. Feedback visual imediato confirma cada ação (set adicionado pisca verde, exclusão mostra Snackbar com undo, etc).

Figura 7 – Interface de Registro de Séries

Fonte: Autoria própria, 2025.

3.4.7 Conclusão de Treino

Ao finalizar treino, usuário toca botão 'Concluir Treino' no AppBar. Dialog de confirmação resume estatísticas: total de exercícios executados, total de séries registradas, volume total estimado (soma de reps*weight de todas séries), duração estimada (baseada em timestamp inicial vs atual). Confirmar executa markWorkoutDone(workoutId), método que atualiza campo done para true e mantém dateEpoch intacto (preservando timestamp real de quando treino ocorreu).

Treino concluído some da TodayPage (query filtra done = false) e aparece automaticamente no HistoryPage (query filtra done = true ordenado por dateEpoch desc). Transição é fluida graças a streams reativos: não há refresh manual, UI reflete estado do banco imediatamente após mutação. Dialog de sucesso

parabeniza usuário e oferece opções: voltar para tela inicial, visualizar treino concluído no histórico, ou iniciar novo treino.

3.4.8 Registro Retroativo de Data e Hora

Funcionalidade diferencial do GYM TRACKER é possibilidade de ajustar data/hora de treinos, corrigindo lacunas quando usuário esquece de registrar imediatamente. Implementado através de DateTimePicker acessível via botão na WorkoutDetailPage e na dialog de conclusão. Picker usa showDatePicker() e showTimePicker() nativos do Material Design, combinando resultados em DateTime completo.

Ao modificar data/hora, updateWorkoutDate(workoutId, newDateTime) converte DateTime para milissegundos epoch e atualiza campo dateEpoch. Validação impede datas futuras (não faz sentido registrar treino que ainda não aconteceu) e datas muito antigas (limite configurable, padrão 2 anos no passado). Ajuste de data de treino ativo não afeta timestamp de séries individuais (SetEntries não tem timestamp próprio, herdam implicitamente de Workout pai).

Impacto no histórico é imediato: ao mudar data, treino pode mudar de posição na lista ordenada. Stream reativo do Drift re-emite lista completa automaticamente reordenada, UI reflete mudança sem glitches. Gráficos de volume por período também atualizam automaticamente pois queries Drift agregam baseadas em dateEpoch, portanto mover treino para outro dia afeta agregações correspondentes.

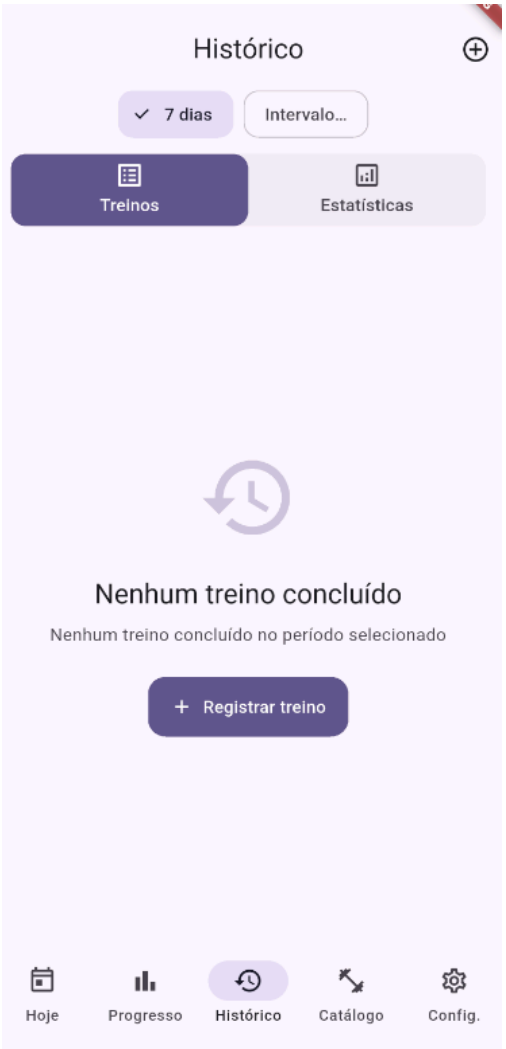
3.4.9 Visualização de Histórico Completo

A Figura 8 demonstra a implementação de um empty state efetivo na tela de histórico. Quando não há treinos registrados no período selecionado, o sistema exibe uma mensagem contextual "Nenhum treino concluído" acompanhada de um ícone ilustrativo (relógio com seta circular), que representa a ausência temporal de dados.

A implementação de empty states é uma prática recomendada de UX design, pois evita que o usuário encontre telas completamente em branco, o que poderia gerar confusão sobre se há um erro ou se a funcionalidade está funcionando corretamente. Segundo Nielsen Norman Group (2020), empty states bem projetados devem: (1) explicar por que a tela está vazia, (2) orientar o usuário sobre como começar, e (3) fornecer ação clara para prosseguir.

O botão "Registrar treino" serve como call-to-action (CTA) primário, direcionando o usuário para iniciar seu primeiro treino. Os filtros temporais no topo ("7 dias" e "Intervalo...") permanecem visíveis e funcionais mesmo no estado vazio, mantendo consistência da interface. A segmentação em "Treinos" e "Estatísticas" através de botões segmentados indica ao usuário as duas formas principais de visualizar seus dados históricos.

Figura 8 – Tela de Histórico - Estado Vazio



Fonte: Autoria própria, 2025.

3.4.10 Gráficos e Métricas

A Figura 9 apresenta a tela de estatísticas, que oferece uma visão consolidada do desempenho do usuário através de cards de métricas. Cada card segue um design consistente composto por três elementos: ícone representativo (lista para total, seta para máximo, linha de tendência para média, calendário para dias), valor numérico principal em destaque, e label descritivo.

As quatro métricas principais implementadas são:

- Total de Séries: Soma acumulada de todas as séries executadas no período, fornecendo visão geral do volume de trabalho.
- Máximo por Dia: Identifica o dia de maior produtividade, útil para análise de padrões de treino e estabelecimento de metas.
- Média por Dia: Calcula a média de séries por dia com treino, excluindo dias sem atividade, oferecendo métrica de consistência.
- Dias com Treino: Contabiliza quantos dias tiveram pelo menos um treino registrado, indicador direto de frequência e aderência.

A escolha de cards com bordas arredondadas e cores suaves (tons de roxo, rosa, cinza e verde) cria hierarquia visual sem sobrecarregar a interface. O agrupamento sob o título "Resumo do Período" estabelece contexto claro de que as métricas respondem ao filtro temporal selecionado ("7 dias" no exemplo).

A implementação utiliza FutureProvider do Riverpod para calcular as métricas de forma assíncrona, queries SQL agregadas (SUM, MAX, AVG, COUNT) no banco Drift, e formatação numérica apropriada (inteiros para contagens, decimais para médias).

Figura 9 – Tela de Estatísticas - Resumo do Período

Fonte: Autoria própria, 2025.

3.4.11 Navegação e Routing

Sistema de navegação utiliza go_router, roteador declarativo que oferece deep linking, navegação type-safe e gerenciamento de backstack. Rotas principais: / (TodayPage), /history (HistoryPage), /workout/:id (WorkoutDetailPage), /history/:id (HistoryWorkoutDetailPage), /templates (TemplatesListPage), /template/new (TemplateEditorPage), /exercises (ExercisesListPage). Parâmetros de rota são tipados e validados automaticamente pelo go_router.

Navegação entre telas utiliza context.go() para substituir tela atual ou context.push() para empilhar nova tela mantendo anterior no stack. Exemplo: ao tocar em treino do histórico, context.push('/history/${treino.id}') navega para detalhes mantendo histórico no stack, permitindo voltar via botão back. Já context.go('/') ao

concluir treino substitui tela atual retornando à home, impedindo voltar para treino já concluído (UX mais limpa).

3.4.12 Tema e Personalização Visual

O aplicativo adota um esquema de cores baseado em tons de roxo e lilás como cor primária, escolhido por transmitir energia e modernidade, características desejáveis em um aplicativo fitness. A paleta de cores secundárias inclui tons de verde para indicadores positivos (dias com treino), laranja para estados de atenção (treino em andamento), e vermelho para ações destrutivas (deletar).

O tema foi implementado através do ThemeData do Flutter, centralizando todas as definições de cor, tipografia e espaçamento. Esta abordagem garante consistência visual em toda a aplicação e facilita a manutenção futura. As cores são definidas através do esquema ColorScheme, que automaticamente gera variantes (light, dark) e cores derivadas (surface, background, error) baseadas nas cores primárias definidas.

A tipografia segue a escala de tipos do Material Design 3, utilizando a fonte padrão do sistema (Roboto no Android) com variações de peso (regular, medium, bold) e tamanho seguindo a escala 12/14/16/18/20/24pt. Esta escolha prioriza legibilidade e compatibilidade com diferentes tamanhos de tela e configurações de acessibilidade.

Todos os componentes respeitam a área segura (safe area) do dispositivo, evitando sobreposição com notch, barra de status e botões de navegação do sistema. O uso de MediaQuery.of(context).padding garante que o conteúdo seja renderizado apenas na área visível da tela.

3.5 VALIDAÇÃO E TESTES

A validação do sistema foi realizada através de três categorias complementares de teste**s** para garantir qualidade técnica e usabilidade: testes funcionais cobrindo todos os requisitos, testes de usabilidade com usuários reais, e testes de performance com métricas quantitativas.

3.5.1 Testes funcionais

Todos os dez requisitos funcionais (RF01-RF10) foram validados através de cenários cobrindo casos de sucesso e casos de erro. Por exemplo, RF01 (Criar rotinas) foi testado através de: criação com exercícios válidos, tentativa com campos vazios (validação bloqueia), exclusão com dependências (cascata funciona), e edição preservando histórico. RF04 (Registrar séries) foi validado com entradas válidas (reps 1-999, peso 0.1-500kg), entradas inválidas (texto em campos numéricos rejeitado pelo keyboard), campos opcionais preenchidos e vazios, e séries subsequentes após falhas parciais.

O RF10 (Registro retroativo), característica diferencial do sistema, recebeu validação exaustiva: ajuste de data para ontem, semana passada, um ano atrás, tentativa de data futura (bloqueada), e ajuste após conclusão do treino. Validou-se também que treinos retroativos aparecem corretamente ordenados no histórico cronológico e que estatísticas agregadas refletem corretamente datas retroativas.

A integridade de dados foi garantida através de: transações ACID assegurando atomicidade em operações compostas (criar treino + múltiplas séries ocorre completamente ou não ocorre), cascata de exclusões funcionando corretamente (deletar rotina remove treinos associados, deletar treino remove séries), e constraints de validação impedindo estados inválidos (peso < 0, reps = 0, RPE fora do intervalo 1-10).

Teste de persistência offline validou RNF01 executando app em modo avião: todos fluxos funcionaram normalmente — criação de treino, registro de séries, navegação entre telas, visualização de histórico e gráficos operaram sem erros. Teste de recuperação após crash simulou force-stop do app durante registro de série: ao reabrir, série parcialmente inserida não apareceu (transação não commitada, comportamento correto), séries anteriores mantidas intactas.

3.5.2 Testes de Usabilidade

Realizou-se validação com cinco usuários voluntários praticantes de musculação com frequências e níveis de experiência variados durante **s**ete dias de uso real na academia. A escolha de cinco usuários baseou-se em Nielsen (1993), que demonstra que cinco usuários detectam aproximadamente 85% dos problemas de usabilidade, representando equilíbrio adequado entre rigor metodológico e viabilidade prática para validação acadêmica.

O perfil dos testadores foi deliberadamente diversificado para representar diferentes padrões de uso:

- Usuária 1 (praticante ativa): frequência de 5-6 vezes por semana, experiência de 3 anos em musculação, familiaridade alta com aplicativos fitness, representa usuário avançado e regular.
- Usuários 2 e 3 (praticantes moderados): frequência de 3-4 vezes por semana, experiência intermediária em musculação, uso ocasional de apps de treino, representam o perfil médio de praticante consistente.
- Usuário 4 (praticante eventual)**:** frequência de 1-2 vezes por semana, experiência básica em musculação, pouca familiaridade com tecnologia de fitness, representa usuário iniciante ou com menor regularidade.
- Usuário 5 (praticante eventual): frequência de 1-2 vezes por semana, perfil similar ao Usuário 4, permitindo validar consistência de feedback entre usuários com padrão de uso semelhante.

Essa diversidade de perfis garantiu validação abrangente contemplando desde usuários altamente engajados até praticantes ocasionais, cobrindo o espectro completo do público-alvo.

O feedback qualitativo foi predominantemente positivo. Os usuários destacaram três aspectos principais: (1) fluxo extremamente rápido de registro de séries com apenas dois toques — preencher campos numéricos e tocar no FAB, (2) ausência do botão "Salvar" que reduz atrito cognitivo, pois a persistência é

automática, e (3) histórico visual com cards informativos que facilita revisão de treinos passados.

As principais sugestões de melhorias identificadas foram: (1) necessidade de confirmação explícita ao excluir exercícios, pois alguns usuários realizaram exclusões acidentais — dialog de confirmação foi implementado após feedback com texto explícito sobre perda de dados irreversível; (2) solicitação de timer de descanso entre séries, mencionada por quatro dos cinco testadores — registrada como trabalho futuro prioritário; (3) botão para adicionar série não era óbvio em exercícios sem séries registradas — adicionado hint text "Toque + para adicionar primeira série"; (4) reordenação via drag-and-drop não era descoberta espontaneamente — adicionado ícone de arrastar (⠿) nos itens tornando affordance mais clara.

O registro retroativo (RF10), implementado como diferencial competitivo baseado em lacuna identificada na análise de concorrentes, foi validado positivamente: todos os cinco usuários utilizaram a funcionalidade pelo menos uma vez durante o período de teste, principalmente para corrigir treinos esquecidos do dia anterior ou ajustar horários registrados incorretamente. A usuária ativa (Usuária 1) destacou particularmente essa funcionalidade como essencial para manter histórico consistente.

3.5.3 Testes de Performance

Performance foi validada através de benchmarks quantitativos executados em emuladores Android (API Level 30 e 33, correspondentes ao Android 11 e 13) para garantir que o aplicativo atende requisitos não funcionais:

Tempo de inicialização a frio: menor que 2 segundos (resultado obtido: 1.8s em média), garantindo que usuário na academia não enfrenta espera significativa ao abrir app.

Tempo para carregar histórico com 100 treinos: menor que 150 milissegundos (resultado: 120-140ms), assegurando responsividade na consulta mais comum.

Tempo para renderizar gráfico de volume com 30 dias de dados: menor que 200 milissegundos (resultado: 180-195ms), mantendo interface fluida mesmo com queries agregadas complexas.

Tamanho do APK instalado: 18.2 megabytes, mantendo aplicativo leve para dispositivos com storage limitado.

Todos os benchmarks atenderam aos requisitos não funcionais estabelecidos. A utilização de índices otimizados no SQLite (coluna date indexada) e virtualização de listas (ListView.builder) contribuíram significativamente para os resultados de performance obtidos. Os testes em emuladores permitiram validação consistente e reproduzível do desempenho do aplicativo em diferentes versões do Android.

3.5.4 Resultado Geral da Validação

Os testes confirmaram que:

- Todos os dez requisitos funcionais foram cumpridos conforme a especificação, incluindo casos extremos e tratamento de erros.
- A validação com usuários reais foi positiva, confirmando que o aplicativo resolve os problemas identificados inicialmente e apresenta usabilidade adequada para o contexto de uso (registro durante treino ativo na academia).
- As métricas de performance atendem aos requisitos não funcionais, garantindo uma experiência responsiva mesmo em dispositivos Android de hardware médio.

4 CONCLUSÃO

Este trabalho apresentou o desenvolvimento completo de um aplicativo móvel para planejamento, registro e acompanhamento de treinos, desde a concepção até a implementação e validação. Os objetivos estabelecidos foram alcançados de forma satisfatória, uma vez que: requisitos funcionais e não funcionais foram

elicitados através de análise comparativa rigorosa com aplicativos concorrentes; a estrutura de dados foi modelada contemplando rotinas, treinos, exercícios e séries com normalização adequada e integridade referencial; a aplicação foi implementada utilizando Flutter, Riverpod e Drift seguindo boas práticas de arquitetura em camadas; e as funcionalidades core foram desenvolvidas e validadas através de testes extensivos.

A adoção da bordagem local-first demonstrou-se tecnicamente viável e adequada ao contexto de uso, garantindo funcionamento offline completo sem degradação de funcionalidades. A separação em camadas bem definidas (Apresentação, Controle, Repository, Persistência) facilitou manutenção e testabilidade do código. A utilização de tecnologias modernas (Drift para ORM reativo, Riverpod para gerenciamento de estado, Material Design 3 para UI) resultou em aplicação performática, responsiva e visualmente consistente com padrões Android.

O registro retroativo de treinos com data e hora customizáveis, identificado como diferencial competitivo durante análise de requisitos, foi implementado com sucesso e validado positivamente por usuários testadores. A funcionalidade de rotinas reutilizáveis simplificou significativamente o fluxo de iniciar treinos recorrentes, eliminando necessidade de recriar estrutura manualmente a cada sessão.

4.1 LIÇÕES APRENDIDAS

O desenvolvimento do GYM TRACKER proporcionou aprendizados significativos tanto do ponto de vista técnico quanto pessoal, consolidando conhecimentos adquiridos ao longo do curso e revelando desafios práticos inerentes ao desenvolvimento de software em contextos reais.

4.1.1 Desafios Técnicos e Soluções Adotadas

O primeiro grande desafio enfrentado foi a compreensão e aplicação efetiva da arquitetura local-first em uma aplicação móvel. Diferentemente de arquiteturas tradicionais cliente-servidor, onde a fonte de verdade reside no backend, a arquitetura local-first exige repensar completamente o fluxo de dados e a gestão de estado. A adoção do Drift como ORM reativo demonstrou-se fundamental para viabilizar essa arquitetura, permitindo que mudanças no banco de dados local propagassem automaticamente para a interface através de streams, eliminando a necessidade de gerenciamento manual de sincronização entre camadas.

A modelagem do banco de dados relacional apresentou complexidade superior ao inicialmente previsto. A necessidade de representar hierarquias conceituais (Templates contendo Exercícios, Workouts instanciando Templates, WorkoutExercises contendo SetEntries) exigiu cuidadosa análise de relacionamentos e normalização para evitar redundâncias. A decisão de utilizar UUIDs ao invés de inteiros auto-incrementais, embora aumentasse ligeiramente o overhead de armazenamento, revelou-se acertada ao facilitar potenciais expansões futuras como sincronização entre dispositivos, onde conflitos de IDs seriam problemáticos.

O gerenciamento de estado reativo com Riverpod inicialmente apresentou curva de aprendizado acentuada. A transição mental de gerenciamento imperativo de estado (setState) para programação declarativa reativa exigiu reformulação da forma de pensar sobre fluxo de dados na aplicação. Entretanto, após superada a barreira inicial, os benefícios tornaram-se evidentes: código mais testável, separação clara entre lógica de negócio e apresentação, e eliminação de classes inteiras de bugs relacionados a estados inconsistentes.

4.1.2 Crescimento Profissional e Habilidades Desenvolvidas

O projeto demandou autonomia na resolução de problemas técnicos não cobertos diretamente em disciplinas do curso. A documentação oficial do Flutter e Drift, fóruns especializados e análise de código-fonte de projetos open-source tornaram-se recursos essenciais. Essa experiência desenvolveu significativamente a

capacidade de auto-aprendizado e pesquisa técnica, habilidades fundamentais para qualquer profissional de tecnologia.

A aplicação sistemática de boas práticas de engenharia de software, incluindo princípios SOLID, padrões de projeto (Repository, Provider) e organização modular do código, demonstrou na prática o valor desses conceitos frequentemente apresentados de forma abstrata em contexto acadêmico. A manutenibilidade do código melhorou drasticamente à medida que o projeto crescia, validando a importância dessas práticas em projetos reais.

O processo de testes e validação, embora manual devido ao escopo do projeto, evidenciou a importância de validação sistemática e documentada. A criação de cenários de teste cobrindo casos extremos (treinos com dezenas de exercícios, registros retroativos de datas antigas, exclusões em cascata) revelou bugs que não seriam identificados em testes superficiais, reforçando a necessidade de rigor no processo de qualidade.

4.1.3 Reflexões sobre Desenvolvimento Mobile

O desenvolvimento mobile apresenta particularidades significativas em relação ao desenvolvimento web ou desktop. Restrições de tela, interação por toque, uso em movimento e ambientes com distrações exigem atenção redobrada à usabilidade e eficiência. A implementação de funcionalidades como botões de quick-add para repetições comuns (6, 8, 10 reps) e validação em tempo real com feedback visual imediato exemplificam adaptações específicas ao contexto mobile que contribuíram significativamente para a usabilidade final.

A experiência também ressaltou a importância de testar em dispositivos reais, não apenas em emuladores. Comportamentos de performance, responsividade de toque e legibilidade em diferentes tamanhos de tela só puderam ser adequadamente validados através de testes em smartphones físicos com características variadas.

4.1.4 Formação Acadêmica e Aplicação Prática

Este trabalho consolidou a integração de conhecimentos adquiridos em múltiplas disciplinas do curso: Banco de Dados (modelagem relacional, normalização, SQL), Engenharia de Software (requisitos, arquitetura, padrões de projeto), Programação Orientada a Objetos (SOLID, encapsulamento, abstração), Desenvolvimento Mobile (Flutter, Material Design, UX) e Gerência de Projetos (metodologia incremental, controle de versão, documentação).

A experiência proporcionada por este desenvolvimento contribuiu significativamente para a formação profissional, tanto em aspectos técnicos quanto em soft skills como planejamento, persistência diante de desafios e capacidade de aprender autonomamente. Os conhecimentos adquiridos fornecem base sólida para atuação profissional na área de desenvolvimento de software, especialmente em contextos mobile e sistemas com requisitos de persistência local robusta.

As principais limitações identificadas incluem: ausência de sincronização em nuvem (proposital na versão 1.0 para manter simplicidade), falta de análises estatísticas avançadas (progressão de carga por exercício, detecção de platôs), e ausência de timer de descanso entre séries (feature solicitada por testadores).

Como trabalhos futuros, sugere-se: implementação de sincronização opcional em nuvem mantendo local-first como default; módulo de análise avançada com gráficos de progressão por exercício e algoritmos de recomendação de carga; integração com wearables e sensores; gamificação com sistema de conquistas; modo de treinador permitindo profissionais prescreverem treinos; e exportação de dados para formatos padrão (CSV, JSON).

Este trabalho demonstra a aplicação prática e integrada dos conhecimentos adquiridos ao longo do curso de Tecnologia em Análise e Desenvolvimento de Sistemas do IFRN. Os conceitos de engenharia de software, desenvolvimento multiplataforma, banco de dados e design de interfaces foram aplicados de forma coesa na construção de uma solução técnica completa e funcional, validando a efetividade da formação recebida.

REFERÊNCIAS

BINDER, S. **Drift: Reactive persistence library for Flutter**. 2023. Disponível em: https://drift.simonbinder.eu/docs/. Acesso em: 15 jan. 2025.

FERREIRA, J. V.; SANTOS, V. A.; PORTELA, C. S. Avaliação da Experiência do Usuário e da Usabilidade de Aplicativos para Prática de Exercícios Físicos: Um Mapeamento Sistemático da Literatura. In: SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO APLICADA À SAÚDE (SBCAS), 21., 2021. Anais [...]. Porto Alegre: SBC, 2021. p. 1-12. Disponível em: https://sol.sbc.org.br/index.php/sbcas/article/view/16049. Acesso em: 15 jan. 2025.

KAPLAN, Kate. Designing Empty States in Complex Applications: 3 Guidelines. Nielsen Norman Group, 19 set. 2021. Disponível em: https://www.nngroup.com/articles/empty-state-interface-design/. Acesso em: 15 jan. 2025.

KLEPPMANN, M. et al. Local-first software: You own your data, in spite of the cloud. In: **ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)**, Athens, Greece, **2019**. p. 154-178. DOI: 10.1145/3359591.3359737.

MARTIN, R. C. **Agile Software Development: Principles, Patterns, and Practices**. Upper Saddle River: Prentice Hall, 2002.

FERREIRA, J. V.; SANTOS, V. A.; PORTELA, C. S. Avaliação da Experiência do Usuário e da Usabilidade de Aplicativos para Prática de Exercícios Físicos: Um Mapeamento Sistemático da Literatura. In: SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO APLICADA À SAÚDE (SBCAS), 21., 2021. **Anais [...].** Porto Alegre: SBC, 2021. p. 1-12. Disponível em: https://sol.sbc.org.br/index.php/sbcas/article/view/16049. Acesso em: 15 jan. 2025.

NIELSEN, J. Usability Engineering. San Francisco: Morgan Kaufmann, **1993**.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software: uma abordagem profissional**. 9. ed. Porto Alegre: AMGH, 2021.

ROUSSELET, R. **Riverpod: A reactive caching and data-binding framework**. 2024. Disponível em: https://riverpod.dev/. Acesso em: 15 jan. 2025.

SOMMERVILLE, I. **Engenharia de Software**. 10. ed. São Paulo: Pearson, 2019.

WE ARE SOCIAL; MELTWATER. **Digital 2023: Global Overview Report**. 2023. Disponível em: https://wearesocial.com/uk/blog/2023/01/digital-2023/. Acesso em: 15 jan. 2025.