Report

Date: January 31, 2025

To: Dr : Mfonobong Uko

From: Shahzad Sadruddin (Student ID # **2513806**)

RE: Mobile App Development and Design

Dear Professor Uko,

A comprehensive mobile bodybuilding application was developed to support younger generations, bodybuilders, and health-conscious individuals in managing their fitness routines, monitoring daily progress, and maintaining long-term wellness habits. The platform features an interactive catalogue of exercises with animated demonstrations that guide users in performing movements safely and effectively—an approach shown to improve exercise adherence and reduce injury risk (Schoeppe et al., 2016). The design emphasizes visual guidance, structured routines, and personalized tracking tools to enhance user motivation and long-term engagement.

The interface is built using a wide range of Flutter UI components, including Stateful and Stateless widgets, gradient cards, custom canvas rendering, and Flutter View integrations. These elements support sophisticated layouts such as the home dashboard, detailed workout screens, and nested sub-screens that display exercise patterns, sets, repetitions, and weight configurations (**See Exhibit I)**. The UI prioritizes clarity, accessibility, and intuitive navigation, aligning with research demonstrating that well-designed mobile interfaces significantly improve user engagement in health applications (Harrison et al., 2013).

Accessibility is further strengthened through WCAG-aligned features such as dynamic text scaling, semantic labels, high-contrast elements, and multimodal feedback **(See Exhibit II).**

Beyond exercise tracking, the application includes a customizable meal-planning module that enables users to log meals, monitor caloric intake, and maintain balanced nutrition. This holistic approach reflects evidence that digital diet-tracking tools positively influence nutritional behavior and long-term health outcomes (Dennison et al., 2013). The dashboard also provides a visual overview of workout distribution using intuitive muscle-group icons, helping users maintain balanced training routines across chest, shoulders, back, legs, and core **(See Exhibit III).**

Architecturally, the application follows Flutter's declarative paradigm and adopts a Model-View-View Model (MVVM) structure. The View layer consists of reusable widget components, while the View Model layer—implemented using various classes which manages business logic, input validation, and progress calculations. The Model layer defines core entities such as Exercise, Workout Plan, and Meal. For data persistence, the system employs a hybrid strategy: **SQLite** stores structured user information including age, height, weight, calorie logs, and workout history, while **shared_preferences** manages lightweight settings and user preferences. This combination ensures efficient querying, long-term data retention, and a seamless offline experience. State management is handled through the provider package, enabling predictable UI updates and optimized performance **(See Exhibit IV).**

The selection of tools and libraries was driven by goals of maintainability, performance, and a premium user experience. MVVM with Provider reduces boilerplate and enhances scalability, while Stateful Widget and CustomPaint enable fine-grained control over custom exercise animations that appeal to visually oriented users (Yang, Maher, & Conroy, 2015). Libraries such as **fl_chart** and **percent_indicator** transform raw fitness metrics into motivational visualizations that reinforce habit formation (Mercer et al., 2016) – **(See Exhibit V**). Additional packages—including **dart:math** for animation logic, **google_fonts**

for typography, and authentication libraries for Google, Apple, and Facebook—support secure onboarding and a polished, modern user experience **(See Exhibit VI).**

The application supports both beginners and advanced users through carefully designed user journeys. Beginners benefit from large exercise cards, animated demonstrations, and simple set-logging controls that make learning proper form intuitive. All users gain value from the Progress dashboard, which presents growth trends through interactive charts, and from the Nutrition module, which provides structured meal logging to support healthy eating habits. Personalized experiences are enabled through SQLite and shared_preferences, which store user metrics, preferences, and historical data.

Together, these design decisions create a technically robust, visually engaging, and user-centric fitness application that empowers individuals to learn exercises, monitor progress, and maintain sustainable health habits through evidence-based design and modern mobile architecture.

References

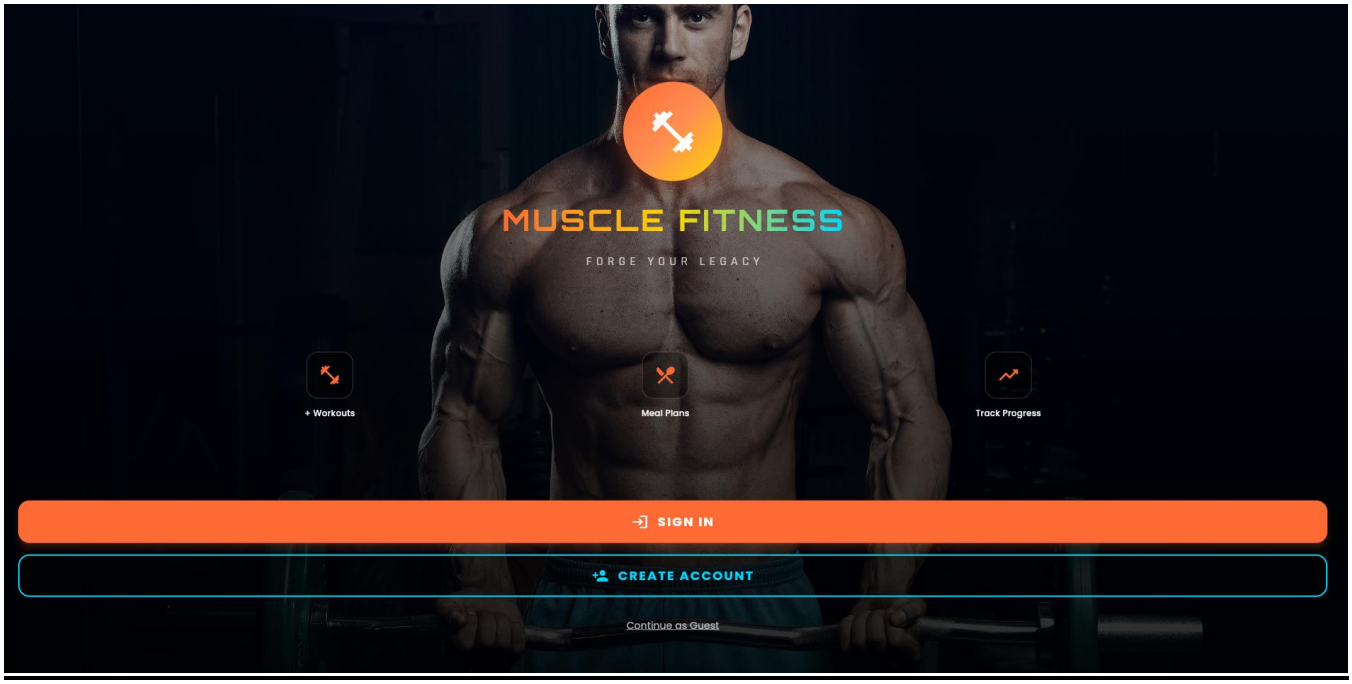
- Schoeppe, S., Alley, S., Van Lippevelde, W., Bray, N. A., Williams, S. L., Duncan, M. J., & Vandelanotte, C. (2016). *Efficacy of interventions that use apps to improve diet, physical activity and sedentary behaviour: A systematic review.* International Journal of Behavioral Nutrition and Physical Activity, 13(1), 127.
- Harrison, R., Flood, D., & Duce, D. (2013). *Usability of mobile applications: Literature review and rationale for a new usability model.* Journal of Interaction Science, 1(1), 1–16.
- Dennison, L., Morrison, L., Conway, G., & Yardley, L. (2013). *Opportunities and challenges for smartphone applications in supporting health behavior change: Qualitative study.* Journal of Medical Internet Research, 15(4), e86.

- Yang, C. H., Maher, J. P., & Conroy, D. E. (2015). *Implementation of behavior change techniques in mobile applications for physical activity.* American Journal of Preventive Medicine, 48(4), 452–455.
- Mercer, K., Li, M., Giangregorio, L., Burns, C., & Grindrod, K. (2016). *Behavior change techniques present in wearable activity trackers: A critical analysis.* JMIR mHealth and uHealth, 4(2), e40.

Exhibit I

App Landing Screen

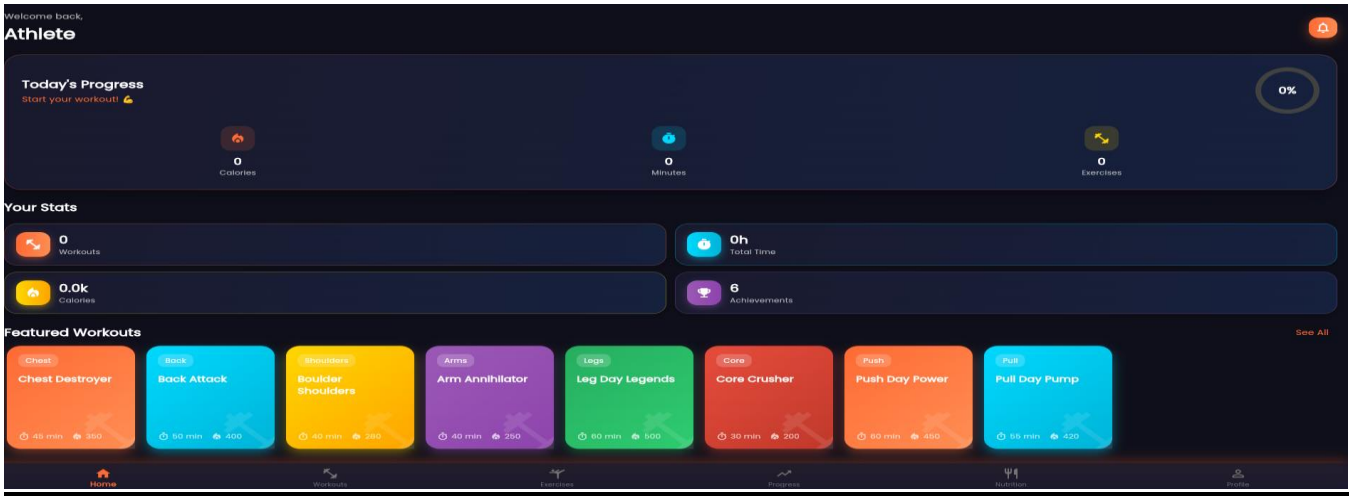
The screen provides users with clear options to either sign in with an existing account or create a new one. For quick access, users may also continue as a guest with limited app privileges.



Home Screen

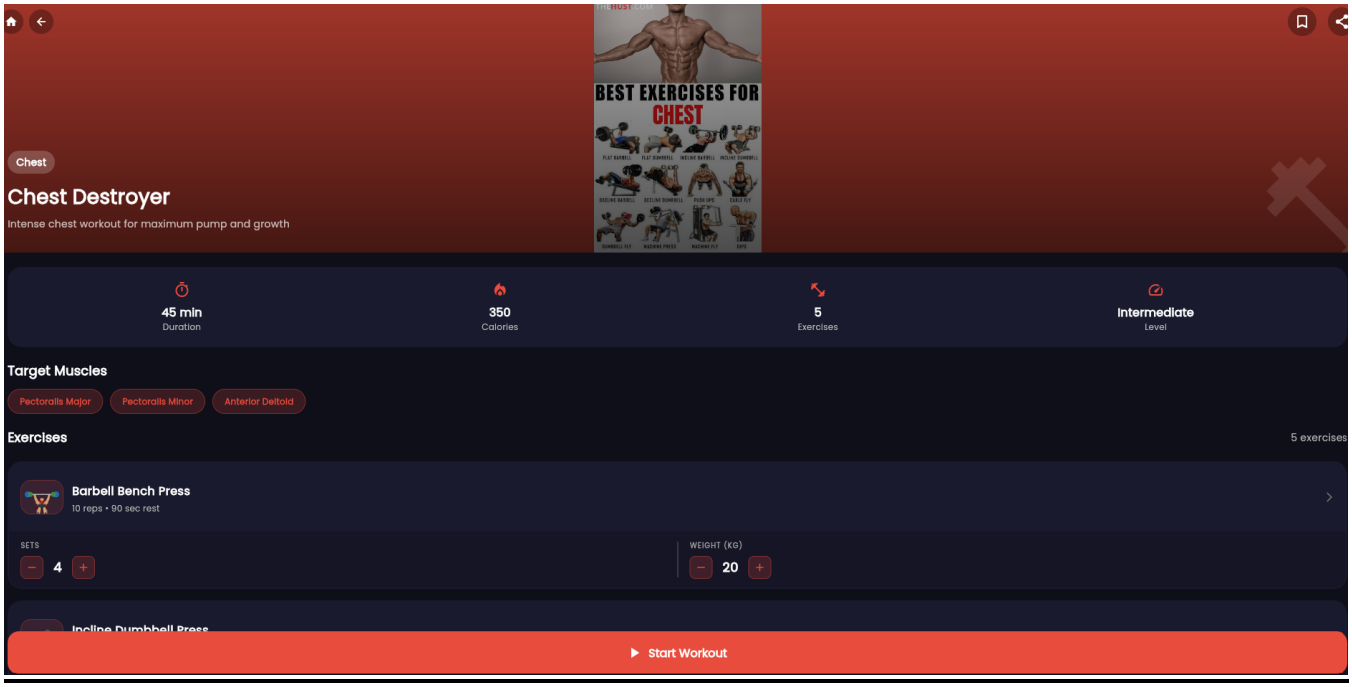
The home screen displays all key icons, allowing users to seamlessly navigate to detailed exercise pages and view high-quality animated demonstrations. Users can select any exercise within a group and record their preferred repetitions, sets, and weight to monitor progress accurately. Additionally, the bottom

navigation bar provides quick access to the app's extended features and tools.



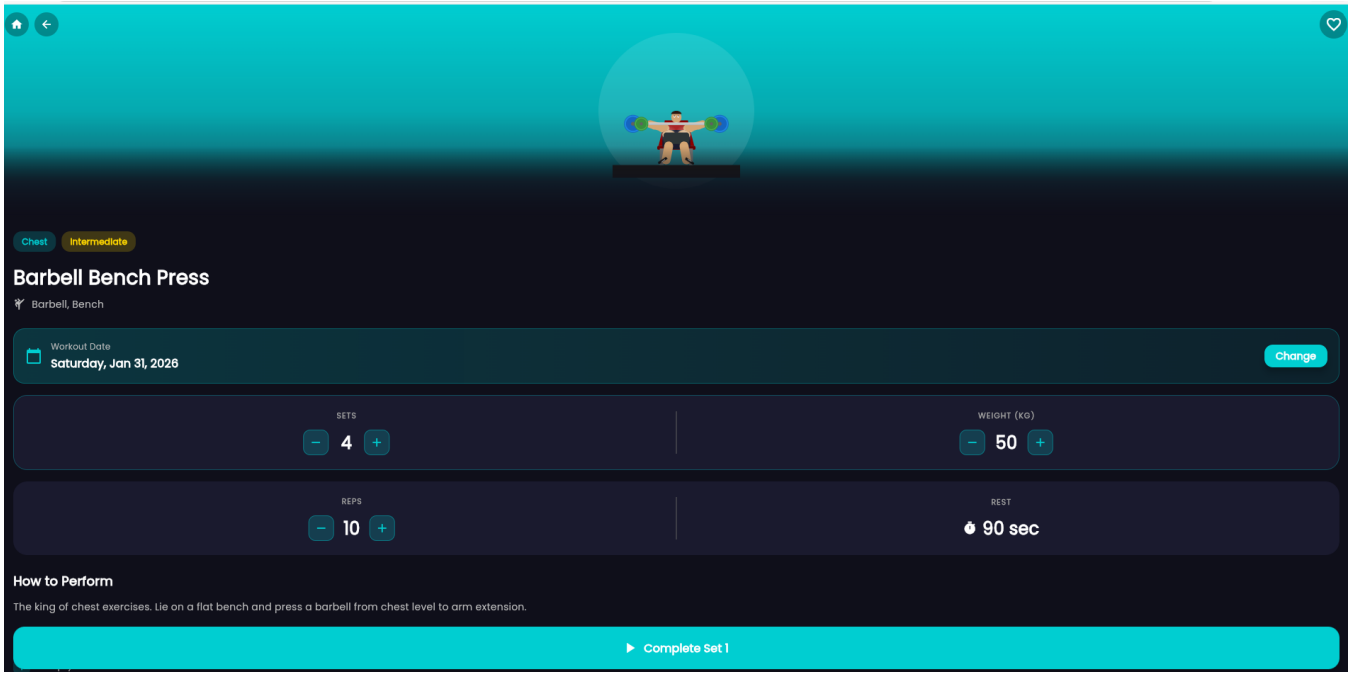
Detail Workout Screen

The detailed workout screen is designed to allow users to select from a range of targeted workouts and seamlessly navigate to related sub-screens. This layout supports efficient exploration of workout options and enhances the overall training workflow.



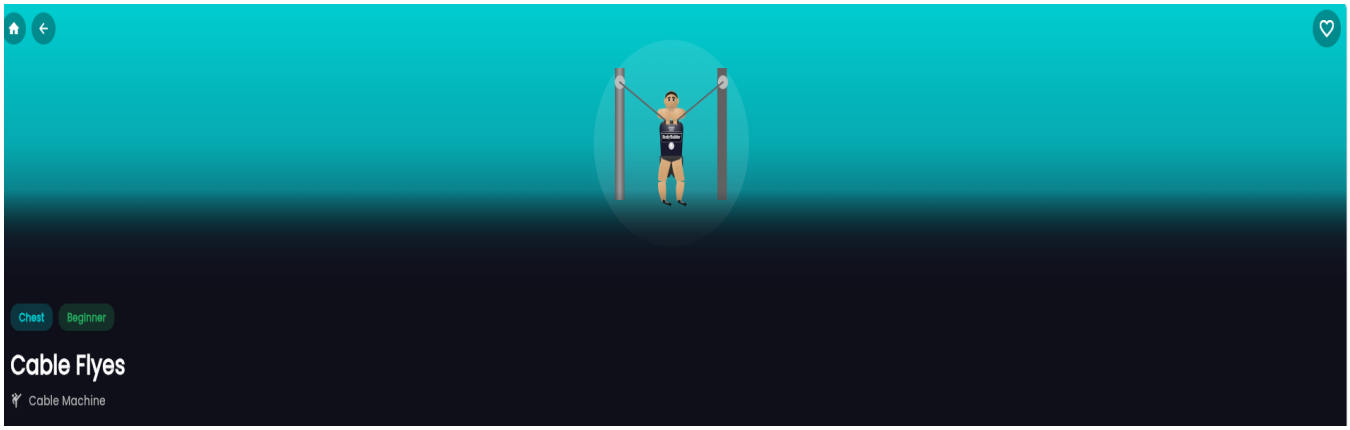
Specific workout Screen

This screen features a guided animation demonstrating proper form, enabling users—especially first-time learners—to confidently mirror each exercise. Users can also input their desired repetitions, sets, and weight to accurately log and track their workout for the selected day.



This screen showcases a chest-focused workout featuring an animated demonstration of a man performing a cable-pull movement. The athlete is shown wearing a gym jersey labeled ‘Body Builder,’ with a stylized ‘B’ logo displayed on the T-shirt. This visual guidance helps users understand proper

form and execution for the exercise.



Sample codes for Gym T-Shirt and Logo

Sample code snippets are provided to demonstrate the correct implementation of required libraries, features, and functionalities. Each example is thoroughly tested to ensure error-free performance and seamless integration within the app. These code samples support the delivery of smooth animations and exercise movements, enabling users to incorporate specific workouts into their routines and work toward their long-term fitness goals.

```
1331
1332    // Gym Shirt name/logo with border - positioned lower to not block hand movement
1333    final textPainter = TextPainter(
1334      text: TextSpan(
1335        text: 'Body Builder',
1336        style: TextStyle(
1337          color: const Color.fromARGB(255, 255, 255, 255).withOpacity(0.8),
1338          fontSize: 5 * scale,
1339          fontWeight: FontWeight.bold,
1340          shadows: [
1341            Shadow(
1342              color: Colors.black.withOpacity(0.6),
1343              offset: const Offset(1, 1),
1344              blurRadius: 2,
1345            ), // Shadow
1346          ],
1347        ), // TextStyle
1348      ), // TextSpan
1349      textDirection: TextDirection.ltr,
1350    ); // TextPainter
1351    textPainter.layout();
1352
1353    // Calculate text position - moved lower (0.25 instead of 0.12)
1354    final textX = center.dx - textPainter.width / 2;
1355    final textY = center.dy + height * 0.28;
1356
1357    // Draw border rectangle around the text
1358    final borderPadding = 2 * scale;
1359    final borderRect = RRect.fromRectAndRadius(
1360      Rect.fromLTWH(
1361        textX - borderPadding,
1362        textY - borderPadding - 2 * scale,
1363        textPainter.width + borderPadding * 2,
1364        textPainter.height + borderPadding * 2 + 2 * scale,
1365      ), // Rect.fromLTWH
1366      Radius.circular(2 * scale),
```

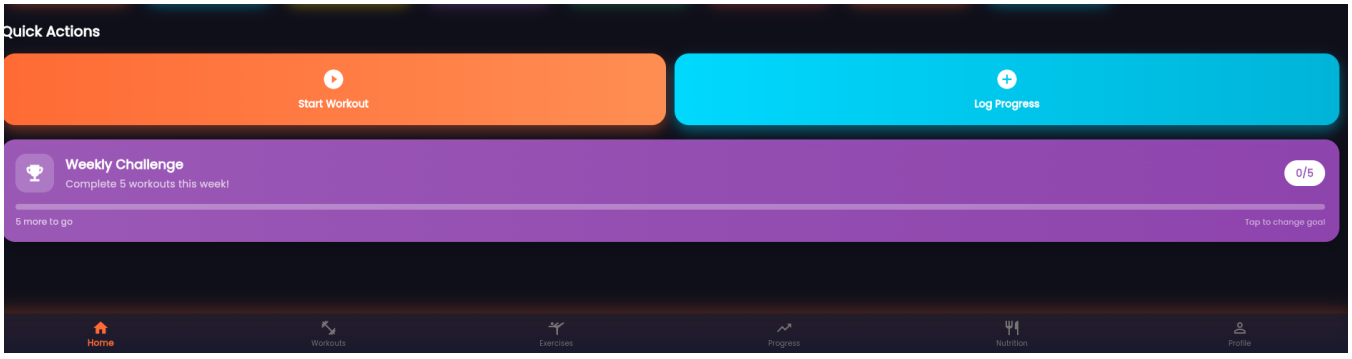
```
      textPainter.height + borderPadding * 2 + 2 * scale,
    ), // Rect.fromLTWH
    Radius.circular(2 * scale),
  ); // RRect.fromRectAndRadius

  // Draw border line above text
  canvas.drawLine(
    Offset(textX - borderPadding, textY - borderPadding),
    Offset(textX + textPainter.width + borderPadding, textY - borderPadding),
    Paint()
      ..color = Colors.white.withOpacity(0.7)
      ..strokeWidth = 1 * scale,
  );

  // Draw border background
  canvas.drawRRect(
    borderRect,
    Paint()..color = Colors.black.withOpacity(0.3),
  );

  // Draw border outline
  canvas.drawRRect(
    borderRect,
    Paint()
      ..color = Colors.white.withOpacity(0.5)
      ..style = PaintingStyle.stroke
      ..strokeWidth = 0.5 * scale,
  );

  // Draw text below the border line
  textPainter.paint(canvas, Offset(textX, textY));

  // Draw "B" circle below the Body Builder text
  final bRadius = 4 * scale;
  final bCenter = Offset(center.dx, textY + textPainter.height + bRadius + 3 * scale);

  // Circle background
  canvas.drawCircle(
    bCenter,
    bRadius,
```

```
  // Circle background
  canvas.drawCircle(
    bCenter,
    bRadius,
    Paint()..color = primaryColor.withOpacity(0.8),
  );

  // Circle border
  canvas.drawCircle(
    bCenter,
    bRadius,
    Paint()
      ..color = Colors.white.withOpacity(0.6)
      ..style = PaintingStyle.stroke
      ..strokeWidth = 1 * scale,
  );

  // Draw "B" letter
  final bTextPainter = TextPainter(
    text: TextSpan(
      text: 'B',
      style: TextStyle(
        color: Colors.white,
        fontSize: 5 * scale,
        fontWeight: FontWeight.bold,
      ), // TextStyle
    ), // TextSpan
    textDirection: TextDirection.ltr,
  ); // TextPainter
  bTextPainter.layout();
  bTextPainter.paint(
    canvas,
    Offset(bCenter.dx - bTextPainter.width / 2, bCenter.dy - bTextPainter.height / 2),
  );
}

// Draw a circle with "B" below the Body Builder text
void drawBrandCircle(Canvas canvas, Offset center, double radius, double scale) {
  // Circle background
  final circlePaint = Paint()
    ..color = primaryColor.withOpacity(0.8)
```

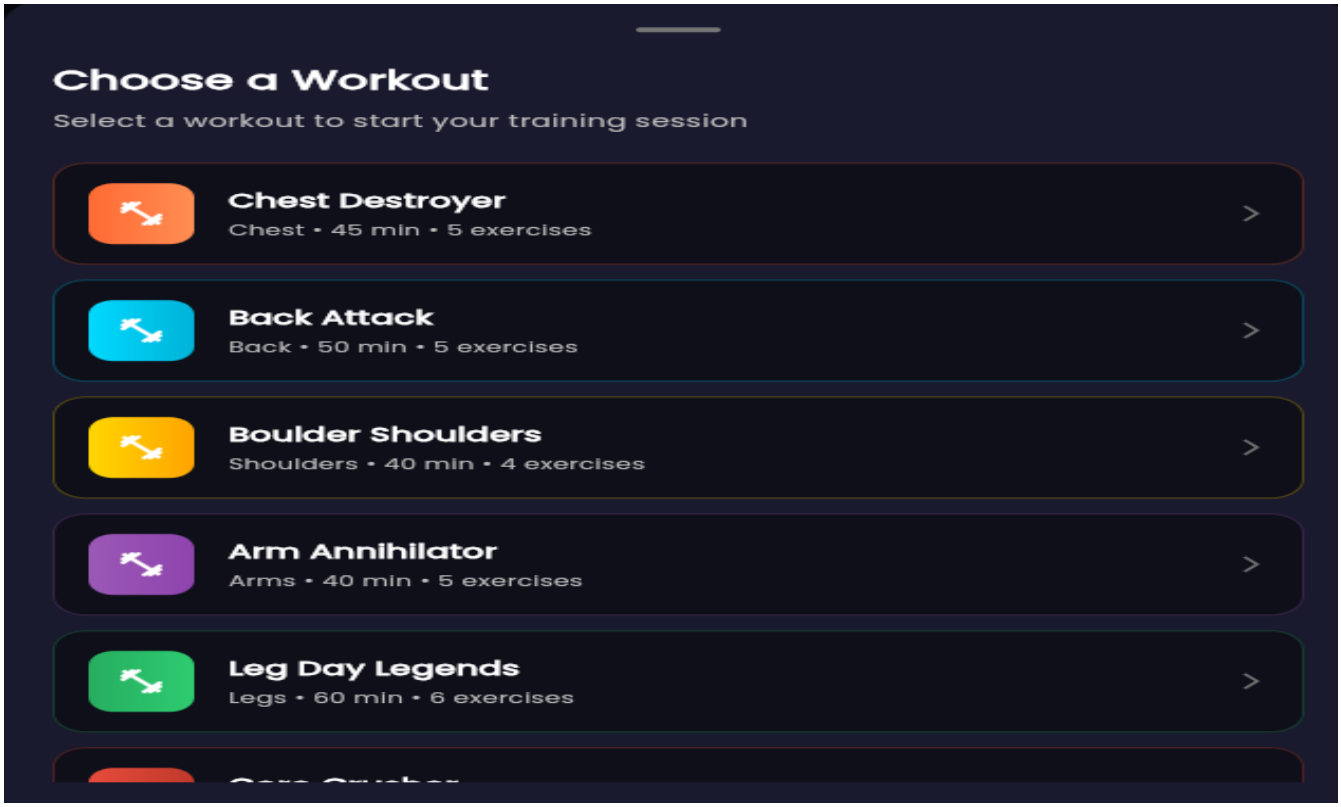
```
    ..color = primaryColor.withOpacity(0.8)
    ..style = PaintingStyle.fill;
  canvas.drawCircle(center, radius, circlePaint);

  // Circle border
  final borderPaint = Paint()
    ..color = Colors.white.withOpacity(0.6)
    ..style = PaintingStyle.stroke
    ..strokeWidth = 1 * scale;
  canvas.drawCircle(center, radius, borderPaint);

  // "B" letter inside the circle
  final textPainter = TextPainter(
    text: TextSpan(
      text: 'B',
      style: TextStyle(
        color: Colors.white,
        fontSize: radius * 1.2,
        fontWeight: FontWeight.bold,
      ), // TextStyle
    ), // TextSpan
    textDirection: TextDirection.ltr,
  ); // TextPainter
  textPainter.layout();
  textPainter.paint(
    canvas,
    Offset(center.dx - textPainter.width / 2, center.dy - textPainter.height / 2),
  );
}
```

Quick Action Screen

The Quick Action section, integrated within the home screen, provides users with immediate access to start a workout and efficiently record the necessary details to track their progress. This streamlined design promotes faster interaction, supports accurate logging, and contributes to a more intuitive overall user experience.

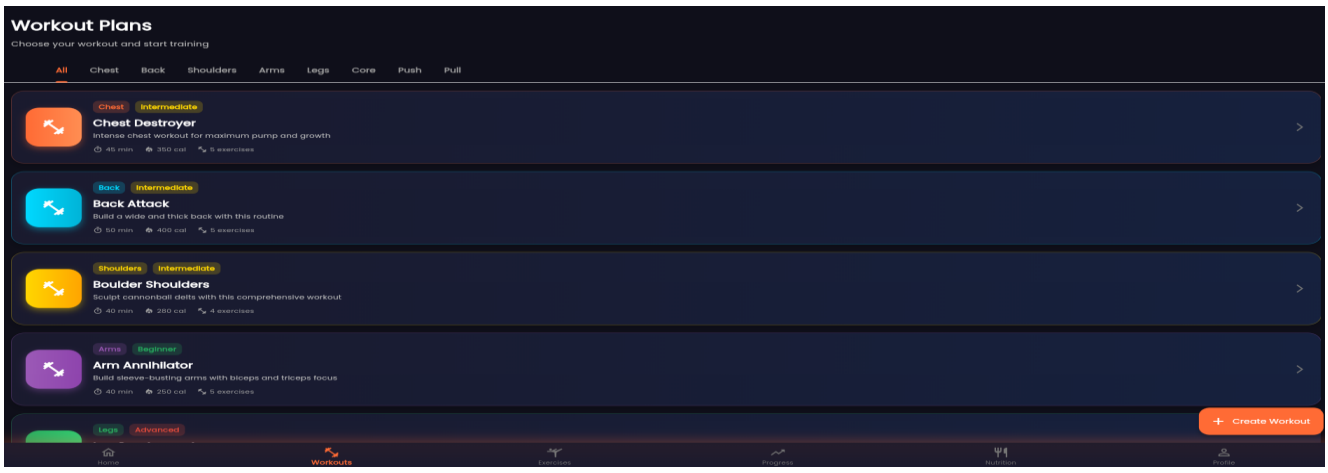


Users can quickly access and select their preferred workouts or training sessions, enabling seamless navigation to additional app features and tools.



Workout Screen

Users can also access their workout plans through the Workout tab located in the bottom navigation bar, ensuring quick and intuitive navigation to personalized training routines.



Exercise Menu Screen

Users can access the exercise library and find the specific workouts.

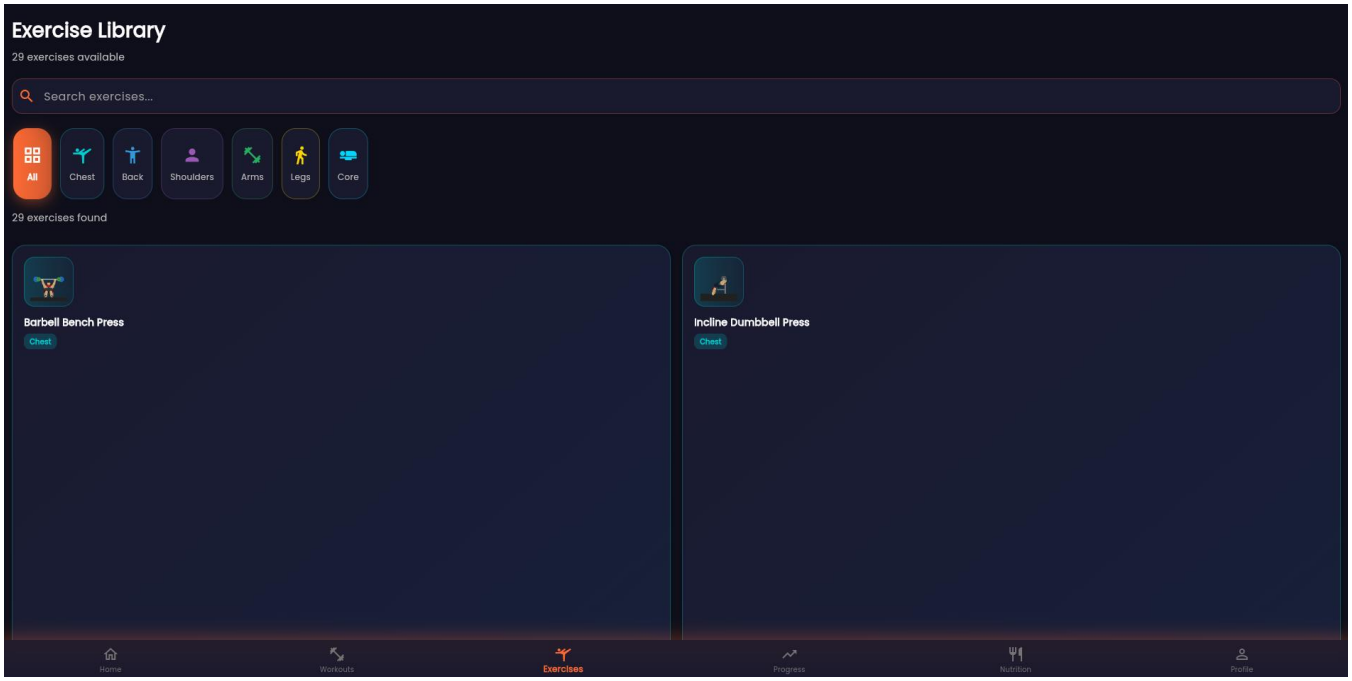
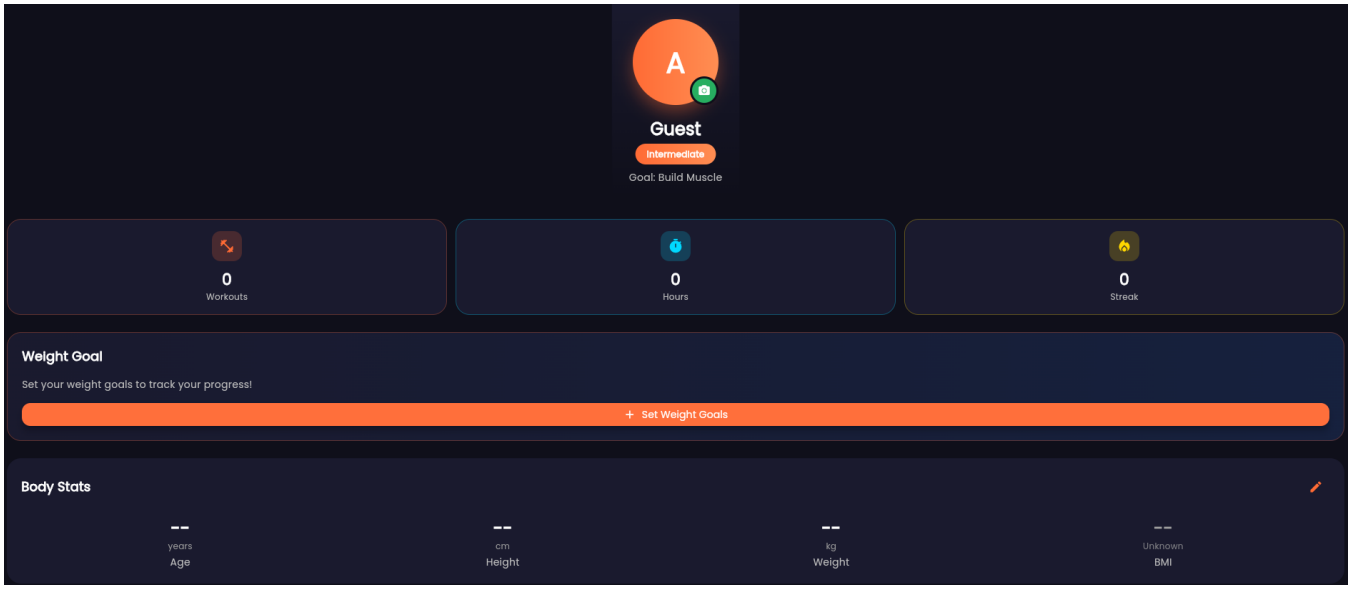


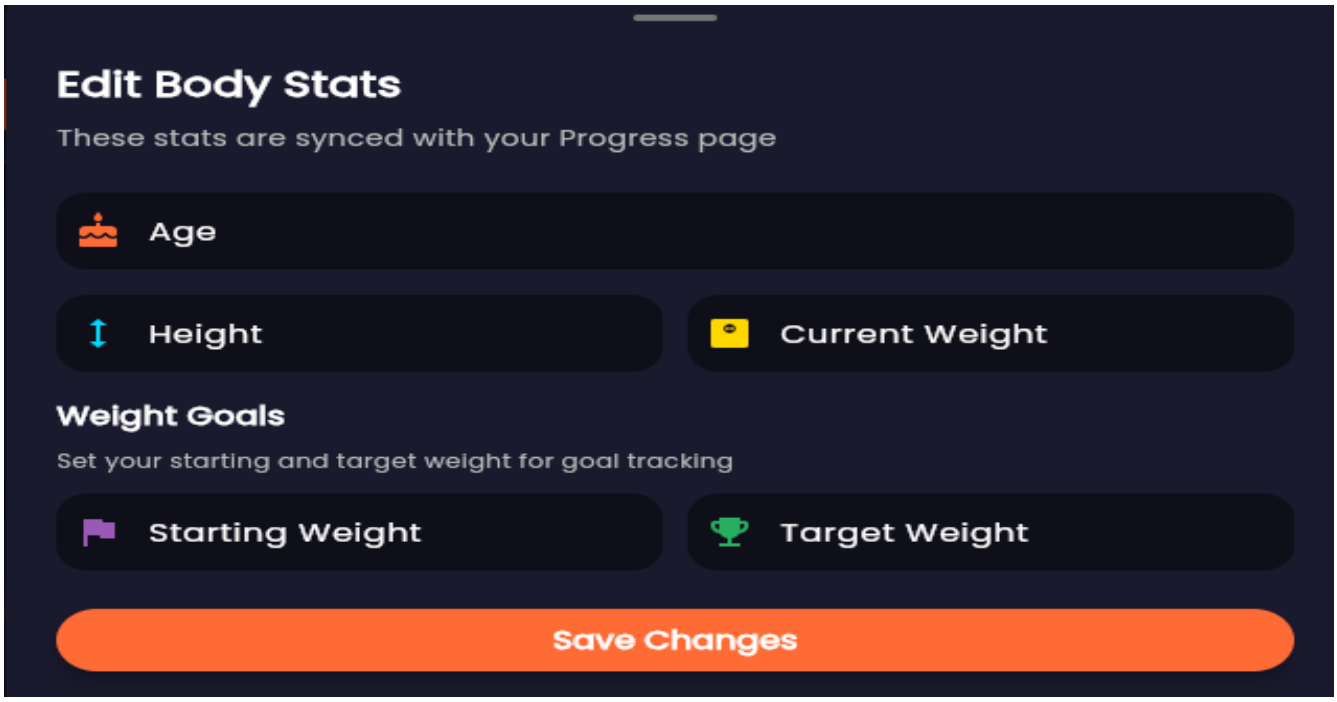
Exhibit II

In this context, Google Fonts library is implemented to provide clear, legible typography across all devices. Their consistent rendering enhances readability and supports users with visual impairments, ensuring that labels and interface elements remain accessible to a wide range of audiences. Please see

below the snapshots for more details.



Clear, well-defined typography and labels ensure that users can easily view the interface and accurately enter their body statistics.



The customized workout screen enables users to build their own workout routines by selecting from a curated list of exercises. The interface is designed with clarity and precision, presenting all essential

information in an organized manner. This ensures that users can navigate the screen effortlessly and make informed selections with confidence.

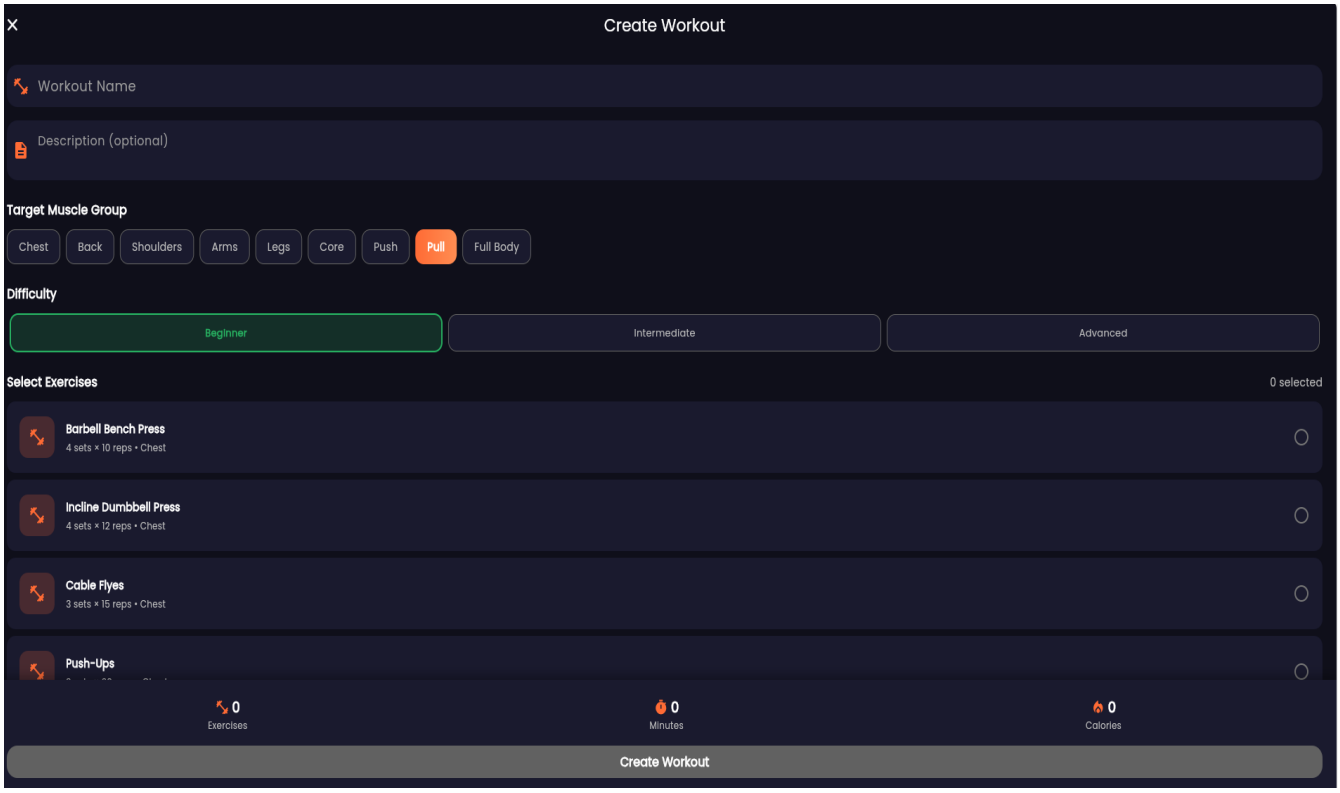
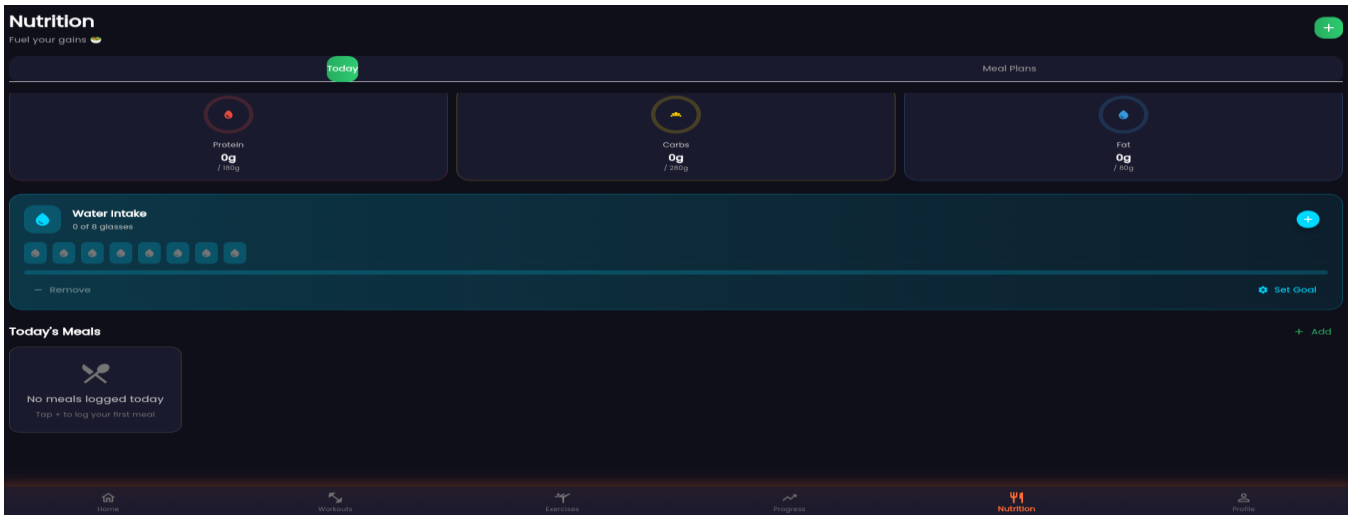
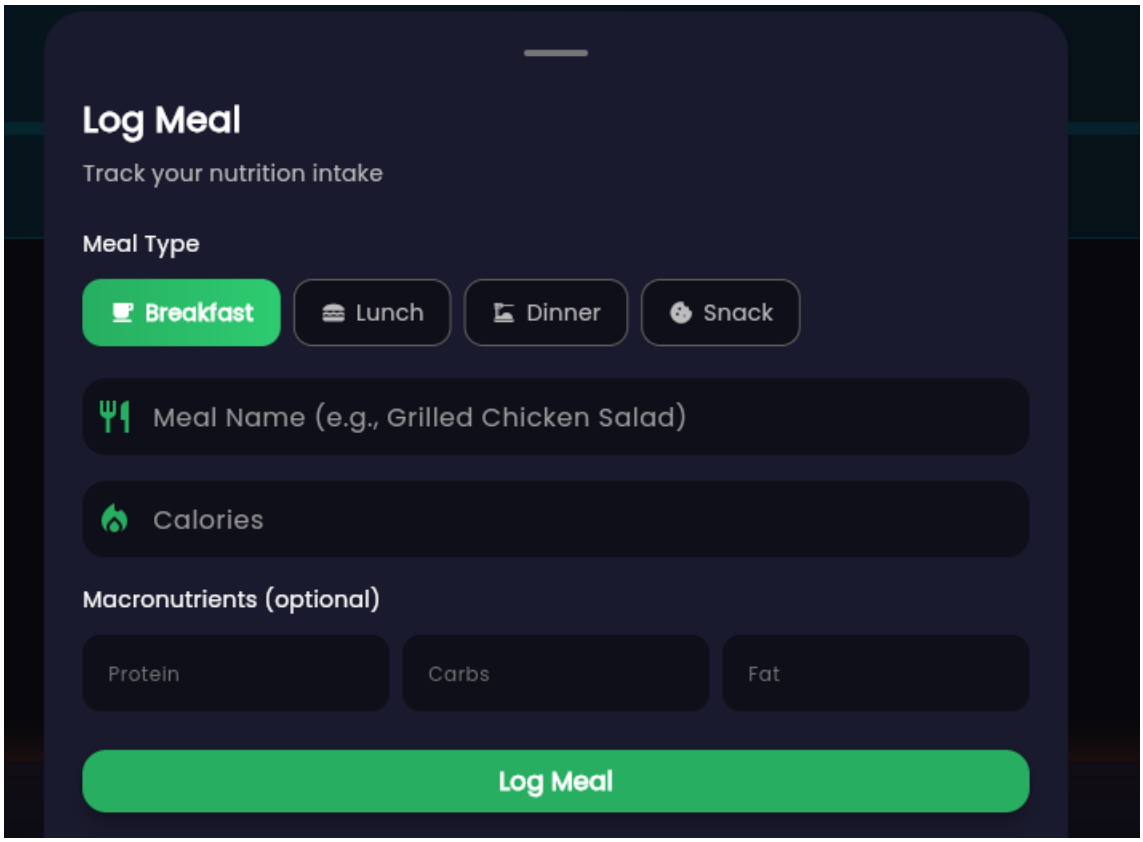


Exhibit III

Within the Nutrition tab, users can log their meal plans and track their daily water intake. The layout is structured to support quick entry and clear visibility, helping users manage their nutritional habits with ease.



The meal-logging screen is structured around key daily categories—breakfast, lunch, dinner, and snacks—allowing users to record meal names and corresponding calorie intake with clarity and ease.



,

Exhibit IV

During the app's development, multiple databases were implemented using the SQLite library to securely store and manage user-related information. These include the user table, workout log table, exercise log table, progress entries table, meal log table, user settings table, and user operations table. The following snippet illustrates the structure of these databases.

```dart
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';
import 'dart:async';

/// Database service singleton for SQLite operations
///
/// Provides persistent storage for all app data on mobile/desktop platforms.
/// Uses lazy initialization to create database on first access.
///
/// Note: For web compatibility, use AuthService and ProgressService
/// which use SharedPreferences instead.
class DatabaseService {
  // ==========================================
  // SINGLETON PATTERN
  // ==========================================

  /// Single instance of the database service
  static final DatabaseService _instance = DatabaseService._internal();

  /// Cached database reference
  static Database? _database;

  /// Factory constructor returns singleton instance
  factory DatabaseService() => _instance;

  /// Private internal constructor
  DatabaseService._internal();

  // ==========================================
  // DATABASE INITIALIZATION
  // ==========================================

  /// Get database instance (creates if doesn't exist)
  ///
  /// Uses lazy initialization pattern - database is only created
```

The following code snippet illustrates the structure of the user-related database tables. These tables store essential information such as email, password, body measurements, height, and other profile attributes required to support personalized functionality within the app.

```dart
/// Calls _onCreate to set up table schema on first run.
Future<Database> _initDatabase() async {
  // Get platform-appropriate database path
  String path = join(await getDatabasesPath(), 'ironforge.db');
  return await openDatabase(
    path,
    version: 1,
    onCreate: _onCreate,
  );
}

/// Create all database tables
///
/// Called once when database is first created.
/// Sets up complete schema for all app features.
Future<void> _onCreate(Database db, int version) async {
  // ----------------------------------------
  // USERS TABLE
  // Stores user account and profile data
  // ----------------------------------------
  await db.execute('''
    CREATE TABLE users (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      email TEXT UNIQUE NOT NULL,
      password TEXT NOT NULL,
      firstName TEXT NOT NULL,
      lastName TEXT NOT NULL,
      profileImage TEXT,
      height REAL,
      weight REAL,
      age INTEGER,
      fitnessGoal TEXT,
      experienceLevel TEXT,
      createdAt TEXT NOT NULL,
      lastLogin TEXT
    )
  ''');

  // ----------------------------------------
  // WORKOUT LOGS TABLE
```

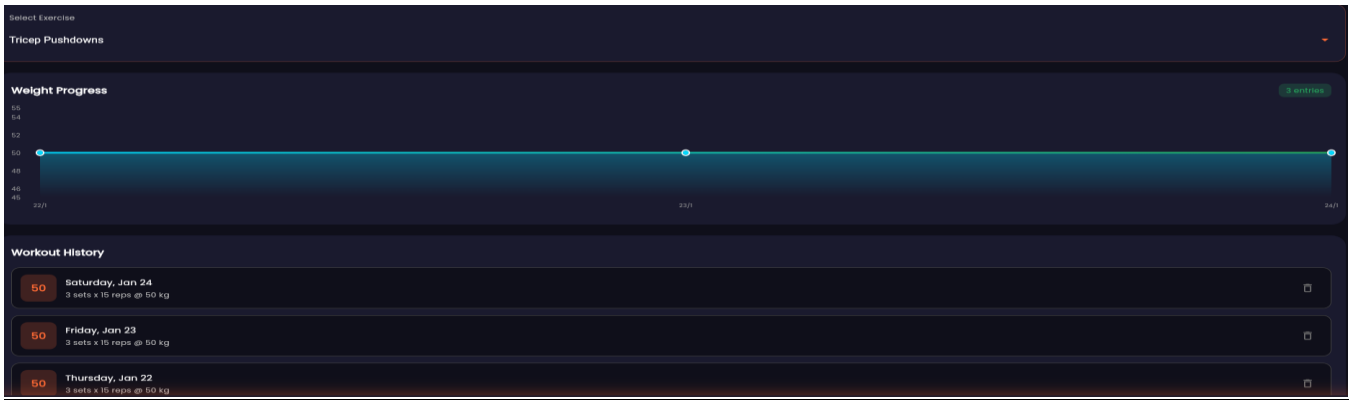
A JSON-based data structure is also implemented to capture detailed workout information, including exercise type, date, weight, repetitions, and sets. This structured format supports efficient data collection and enables the generation of progress charts for visual analysis. The following snippet from exercise_log_services.dart demonstrates the implementation.

```dart
import 'package:shared_preferences/shared_preferences.dart';
import 'dart:convert';

/// Data model representing a single exercise log entry
///
/// Captures all details about an exercise performed during a workout:
/// - Exercise identification (name, muscle group)
/// - Performance metrics (sets, reps, weight)
/// - Temporal data (date performed)
///
/// Each entry is uniquely identified by exercise name + date combination.
/// Only one entry per exercise per day is stored (newer updates replace older).
class ExerciseLogEntry {
  final String id;
  final String exerciseName;
  final String muscleGroup;
  final DateTime date;
  final int sets;
  final int reps;
  final double weight;
  final String notes;

  ExerciseLogEntry({
    required this.id,
    required this.exerciseName,
    required this.muscleGroup,
    required this.date,
    required this.sets,
    required this.reps,
    required this.weight,
    this.notes = '',
  });

  /// Unique key for date-based deduplication (one entry per exercise per day)
  String get dateKey => '${exerciseName}_${date.year}-${date.month.toString().padLeft(2, '0')}-${date.day.toString().padLeft(2, '0')}';

  Map<String, dynamic> toJson() => {
    'id': id,
    'exerciseName': exerciseName,
    'muscleGroup': muscleGroup,
    'date': date.toIso8601String(),
    'sets': sets,
```

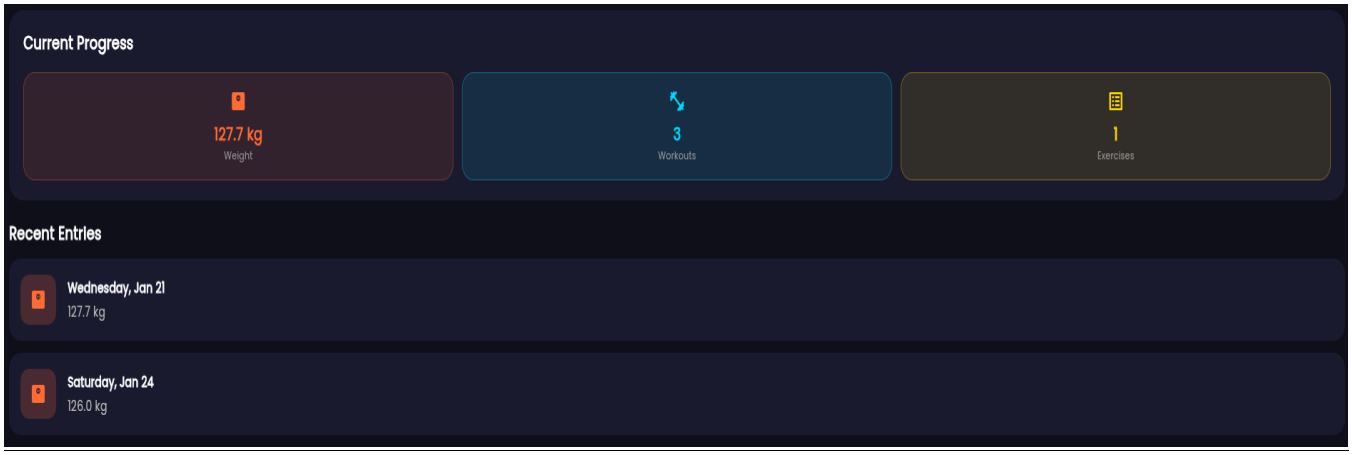
```dart
/// Unique key for date-based deduplication (one entry per exercise per day)
String get dateKey => '${exerciseName}_${date.year}-${date.month.toString().padLeft(2, '0')}-${date.day.toString().padLeft(2, '0')}';

Map<String, dynamic> toJson() => {
  'id': id,
  'exerciseName': exerciseName,
  'muscleGroup': muscleGroup,
  'date': date.toIso8601String(),
  'sets': sets,
  'reps': reps,
  'weight': weight,
  'notes': notes,
};

factory ExerciseLogEntry.fromJson(Map<String, dynamic> json) => ExerciseLogEntry(
  id: json['id'] as String,
  exerciseName: json['exerciseName'] as String,
  muscleGroup: json['muscleGroup'] as String? ?? 'Other',
  date: DateTime.parse(json['date'] as String),
  sets: json['sets'] as int,
  reps: json['reps'] as int,
  weight: (json['weight'] as num).toDouble(),
  notes: json['notes'] as String? ?? '',
);

ExerciseLogEntry copyWith({
  String? id,
  String? exerciseName,
  String? muscleGroup,
  DateTime? date,
  int? sets,
  int? reps,
  double? weight,
  String? notes,
}) {
  return ExerciseLogEntry(
    id: id ?? this.id,
    exerciseName: exerciseName ?? this.exerciseName,
    muscleGroup: muscleGroup ?? this.muscleGroup,
    date: date ?? this.date,
```

Exhibit V

The app incorporates multiple progression charts to provide clear, data-driven visualizations of the user's fitness journey. These charts allow users to monitor changes in their weight, track performance trends, and evaluate their progress toward daily and long-term goals. By presenting information in an intuitive visual format, the system supports informed decision-making and helps users stay motivated as they work toward their health and fitness objectives.



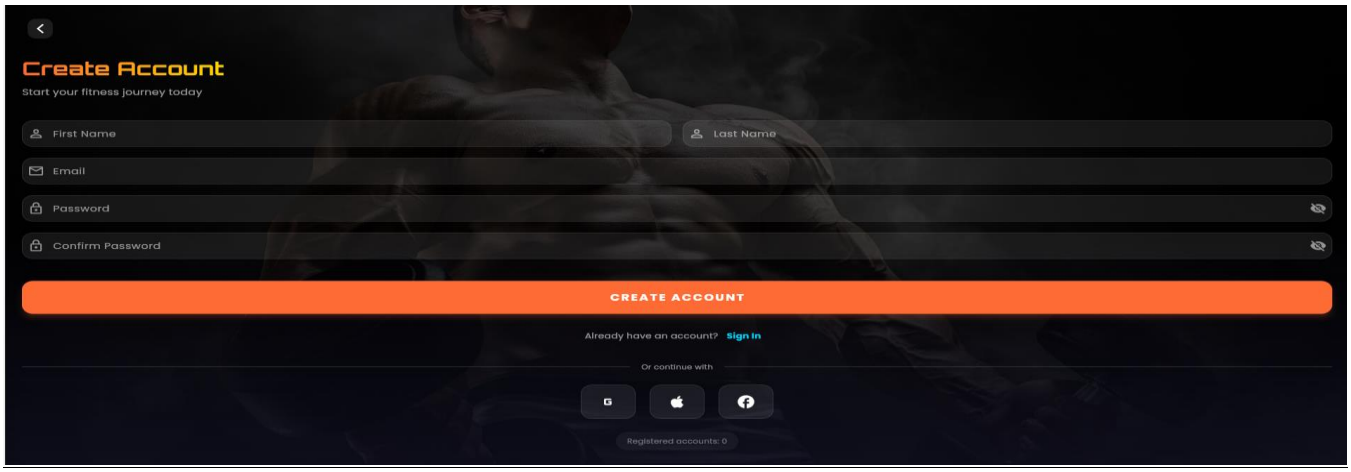
The main progress screen displays the total number of workouts completed for each specific exercise, enabling users to monitor their weight-lifting progression over time. To generate meaningful chart-based insights, users must have multiple entries recorded across different days; otherwise, the visualization will remain empty until sufficient data is available.



When users record multiple weight entries across different dates, the progress tracker automatically highlights changes over time. This visual comparison supports users in monitoring their weight-loss goals and assessing their progress toward personal targets.

Exhibit VI

Contemporary mobile applications frequently incorporate social authentication methods—such as Facebook, Apple, and Google—to provide users with a fast and secure account-creation experience. In this app, integrated social login functionality allows users to authenticate through their preferred platform, with all credentials encrypted and safeguarded against unauthorized access. This streamlined onboarding process enables users to transition smoothly into the app's core features, allowing them to concentrate on their health and fitness goals, including structured meal planning, consistent water-intake tracking, and personalized workout routines. The following section presents the user account creation and login interface.



The following snippet demonstrates the implementation of Google Sign-In authorization, showcasing how user credentials are securely authenticated and managed within the application.

```dart
Future<void> _signInWithGoogle() async {
  setState(() {
    _isLoading = true;
    _errorMessage = null;
  });

  try {
    final authService = AuthService();
    final result = await authService.signInWithGoogle();

    if (!mounted) return;

    if (result['success'] == true) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Welcome, ${authService.userFullName}!'),
          backgroundColor: Colors.green,
        ), // SnackBar
      );
      Navigator.of(context).pushReplacementNamed('/home');
    } else {
      setState(() {
        _errorMessage = result['error'] ?? 'Google sign-in failed';
      });
    }
  } catch (e) {
    if (mounted) {
      setState(() {
        _errorMessage = 'Google sign-in error: ${e.toString()}';
      });
    }
  } finally {
    if (mounted) {
      setState(() {
        _isLoading = false;
      });
    }
  }
}
```

The following snippet demonstrates the implementation of Apple Sign-In authorization, showcasing how user credentials are securely authenticated and managed within the application.

```
Future<void> _signInWithApple() async {
  setState(() {
    _isLoading = true;
    _errorMessage = null;
  });

  try {
    final authService = AuthService();
    final result = await authService.signInWithApple();

    if (!mounted) return;

    if (result['success'] == true) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Welcome, ${authService.userFullName}!'),
          backgroundColor: Colors.green,
        ), // SnackBar
      );
      Navigator.of(context).pushReplacementNamed('/home');
    } else {
      setState(() {
        _errorMessage = result['error'] ?? 'Apple sign-in failed';
      });
    }
  } catch (e) {
    if (mounted) {
      setState(() {
        _errorMessage = 'Apple sign-in error: ${e.toString()}';
      });
    }
  } finally {
    if (mounted) {
      setState(() {
        _isLoading = false;
      });
    }
  }
}
```

The following snippet demonstrates the implementation of Facebook Sign-In authorization, showcasing how user credentials are securely authenticated and managed within the application.

```
Future<void> _signInWithFacebook() async {
  setState(() {
    _isLoading = true;
    _errorMessage = null;
  });

  try {
    final authService = AuthService();
    final result = await authService.signInWithFacebook();

    if (!mounted) return;

    if (result['success'] == true) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Welcome, ${authService.userFullName}!'),
          backgroundColor: Colors.green,
        ), // SnackBar
      );
      Navigator.of(context).pushReplacementNamed('/home');
    } else {
      setState(() {
        _errorMessage = result['error'] ?? 'Facebook sign-in failed';
      });
    }
  } catch (e) {
    if (mounted) {
      setState(() {
        _errorMessage = 'Facebook sign-in error: ${e.toString()}';
      });
    }
  } finally {
    if (mounted) {
      setState(() {
        _isLoading = false;
      });
    }
  }
}
```

The following code snippet is taken from the *authorization_service.dart* file, which manages user account creation and the handling of shared preferences associated with social media authentication. In addition to supporting third-party login methods, this service enables users to reset their passwords and update their personal information as needed.

```dart
//
// Features:
// - User registration (sign up) with email/password
// - User authentication (sign in)
// - Social login (Google, Apple, Facebook)
// - Session persistence using SharedPreferences
// - Password reset functionality
// - Profile management
//
// Storage:
// - Uses SharedPreferences for cross-platform persistence (web, mobile, desktop)
// - User data stored as JSON with email as key
// - Session data stored separately for quick auth checks
//
// Security Notes:
// - Passwords are stored in plain text (for demo purposes)
// - Production apps should use proper hashing (bcrypt, argon2)
// - Consider Firebase Auth or similar for production
// ============================================================================

import 'package:shared_preferences/shared_preferences.dart';
import 'package:google_sign_in/google_sign_in.dart';
import 'package:sign_in_with_apple/sign_in_with_apple.dart';
import 'package:flutter_facebook_auth/flutter_facebook_auth.dart';
import 'dart:convert';

/// Authentication service singleton
///
/// Manages user accounts, sessions, and authentication state.
/// Uses the singleton pattern to ensure consistent state across the app.
///
/// Usage:
/// ```dart
```