BILKENT UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CS 353 - Database Systems

Section 2 | Group 27

Design Report

24/03/2024

Yassin Younis	22101310
Muhammed Shayan Usman	22101343
Zahaab Khawaja	22101038
Ege Mehmet Kayaselçuk	22003416
Murat Ertan	22003067

1. Introduction **3**
2. Revised Functional Requirements **3**
3. Revised ER Diagram **4**
4. Table Schemas **5**
5. UIs and SQL Queries of Functionalities **14**
5.1. Common Functionality-1: User Management 14
5.1.1. Login Validation: 14
5.1.2. Create New User: 15
5.2. Common Functionality-2 (EXTRA): Leaderboard 16
5.2.1. List Top Performers for a Metric: 16
5.2.2. Update Leaderboard: 17
5.3. Trainer-Specific Functionality: Creating Workout Programs 18
6. Implementation Plan **23**

1. Introduction

In the era of digital health and wellness, there is a considerable need for an easy-to-use app that you can use to monitor, plan, and enhance your fitness activities. Our Fitness Tracker and Trainer project is designed to meet the growing demands of fitness enthusiasts and professional trainers for a comprehensive, user-friendly, and secure system to track and improve health and fitness on their journey of self-improvement. This proposal outlines the application system and its database usage and defines the requirements and limitations of such a system.

Additionally, we have separately listed the functional and non-functional requirements of the project. The functional requirements are the functions provided by the system, and they define the interaction between the system and the user based on the inputs provided. The non-functional requirements focus on meeting the user's expectations of how the system should behave and specify the measures that will ensure that the system will operate functionally and how it will handle incidents.

Lastly, we will also mention the limitations of our system and how and why we found workarounds to problems that arose while designing the app.

2. Revised Functional Requirements

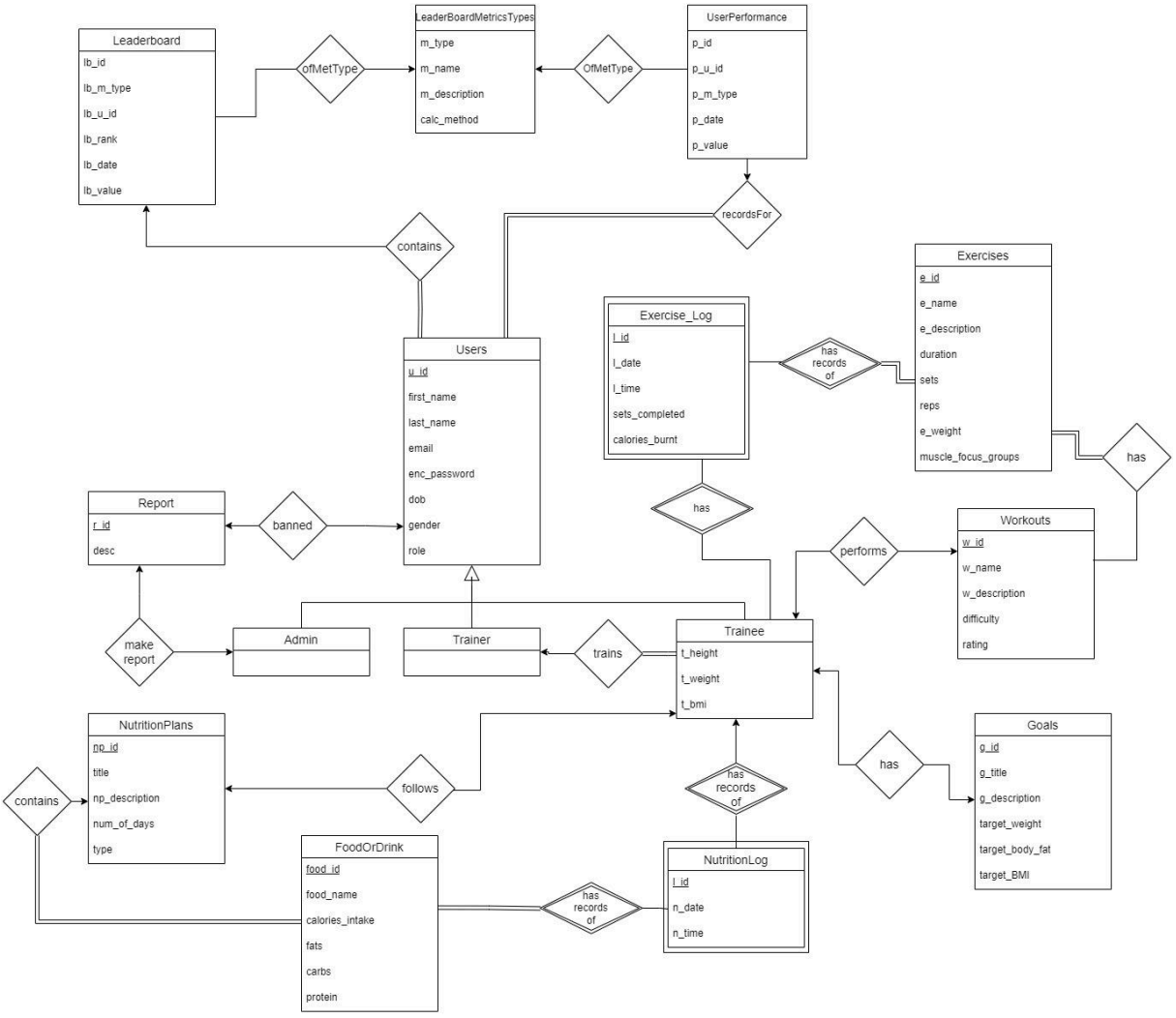
The system must meet the following requirements to ensure its effectiveness and reliability:

- User Authentication and Authorization: Secure login mechanisms and role-based access control to protect user information and privacy.
- Data Integrity and Security: Use encryption and secure data handling practices to safeguard health data and personal information.
- Responsive Interface: A user-friendly interface that adjusts to various devices and screen sizes, enhancing accessibility and usability.
- Real-Time Updates: Ability to handle real-time data input and updates for progress tracking and feedback.
- Analytics and Reporting: Advanced analytics for personalised recommendations and generating reports for users and administrators.

3. Revised ER Diagram

The following changes have been made:

- Same naming for terms in er diagram - all terms are now distinct
- Daily food is redundant - Removed Daily food table
- No need for primary keys for trainer and admin - removed
- Additional scenario in ER diagram - Added leaderboard as extra scenario



4. Table Schemas

Note: Primary keys are underlined, Foreign keys are given as FK: … , and Candidate keys are explicitly mentioned

Users(u_id, first_name, last_name, email, enc_password, dob, gender, role)
Candidate key: email

```
CREATE TABLE Users (
   u_id            INT PRIMARY KEY,
   first_name      VARCHAR(255),
   last_name       VARCHAR(255),
   email           VARCHAR(255) UNIQUE NOT NULL,
   enc_password    VARCHAR(255) NOT NULL,
   dob             DATE,
   gender          VARCHAR(50),
   role            VARCHAR(50),
   CHECK (usertype IN ('ADMIN', 'TRAINEE', 'TRAINER'));
);
```

Goal(g_id, g_title, g_description, target_weight, target_body_fat, target_BMI, u_id)
FK: u_id references Users

```
CREATE TABLE Goal (
   g_id            INT PRIMARY KEY,
   g_title         VARCHAR(255),
   g_description   VARCHAR(80),
   target_weight   DECIMAL(10, 2),
   target_body_fat DECIMAL(5, 2),
   target_BMI      DECIMAL(5, 2),
   u_id            INT,
   FOREIGN KEY (u_id) REFERENCES Users(u_id)
);
```

Trains(trainer_id, trainee_id)
FK: trainer_id references u_id from User

FK: trainee_id references u_id from User

```
CREATE TABLE Trains (
   trainer_id          INT,
   trainee_id          INT,
   PRIMARY KEY(trainer_id, trainee_id),
   FOREIGN KEY (trainer_id) REFERENCES Users(u_id),
   FOREIGN KEY (trainee_id) REFERENCES Users(u_id)
);
```

PerformsWorkout(u_id, w_id, w_status)

FK: u_id references User

FK: w_id references Workouts

```
CREATE TABLE PerformsWorkout (
   u_id                INT,
   w_id                INT,
   w_status            VARCHAR(255),
   PRIMARY KEY(u_id, w_id),
   FOREIGN KEY (u_id) REFERENCES Users(u_id),
   FOREIGN KEY (w_id) REFERENCES Workouts(w_id)
);
```

HasGoal(u_id, g_id, g_status)

FK: u_id references User

FK: g_id references Goals

```
CREATE TABLE HasGoal (
   u_id            INT,
   g_id            INT,
   g_status        VARCHAR(255),
   PRIMARY KEY(u_id, g_id),
   FOREIGN KEY (u_id) REFERENCES Users(u_id),
   FOREIGN KEY (g_id) REFERENCES Goal(g_id)
);
```

Exercises(e_id, e_name, e_description, duration, sets, reps, e_weight, focus_groups)

```
CREATE TABLE Exercises (
   e_id            INT PRIMARY KEY,
   e_name          VARCHAR(255),
   e_description   VARCHAR(80),
   duration        INT,
   sets            INT,
   reps            INT,
   e_weight        DECIMAL(10, 2),
   focus_groups    VARCHAR(80)
);
```

Workouts(w_id, w_name, w_description, difficulty, rating)

```
CREATE TABLE Workouts (
   w_id              INT PRIMARY KEY,
   w_name            VARCHAR(255),
   w_description     VARCHAR(80),
   difficulty        INT,
   rating            DECIMAL(3, 2)
);
```

Exercise_Log(l_id, u_id, date, time, sets_completed, calories_burnt, e_id, w_id)

FK: u_id references Users

FK: e_id references Exercises

FK: w_id references Workouts

```
CREATE TABLE Exercise_Log (
   l_id              INT PRIMARY KEY,
   u_id              INT,
   date              DATE,
   time              TIME,
   sets_completed    INT,
   calories_burnt    INT,
   e_id              INT,
   w_id              INT,
   FOREIGN KEY (u_id) REFERENCES Users(u_id),
   FOREIGN KEY (e_id) REFERENCES Exercises(e_id),
   FOREIGN KEY (w_id) REFERENCES Workouts(w_id)
);
```

Report(r_id, desc)

FK: reported references Users

```
CREATE TABLE Report (
   r_id        INT PRIMARY KEY,
   desc        VARCHAR(80)
);
```

AdminBans(admin_id, u_id, r_id)

FK: admin_id references u_id from Users

FK: u_id references Users

FK: r_id references Report

```
CREATE TABLE AdminBans (
   admin_id  INT,
   u_id        INT,
   r_id        INT,
   PRIMARY KEY(admin_id, u_id, r_id),
   FOREIGN KEY (admin_id) REFERENCES Users(u_id),
   FOREIGN KEY (u_id) REFERENCES Users(u_id),
   FOREIGN KEY (r_id) REFERENCES Report(r_id)
);
```

NutritionPlans(np_id, title, np_description, num_of_days, type)

```
CREATE TABLE NutritionPlans (
   np_id              INT PRIMARY KEY,
   title              VARCHAR(255),
   np_description  VARCHAR(80),
   num_of_days     INT,
   type               VARCHAR(50)
);
```

FoodOrDrink(food_id, food_name, calories_intake, fats, carbs, protein)

```
CREATE TABLE FoodOrDrink (
    food_id             INT PRIMARY KEY,
    food_name           VARCHAR(255),
    calories_intake     INT,
    fats                DECIMAL(10, 2),
    carbs               DECIMAL(10, 2),
    protein             DECIMAL(10, 2)
);
```

TraineeFollowsNP(u_id, np_id)

FK: u_id references Users

FK: np_id references NutritionPlans

```
CREATE TABLE TraineeFollowsNP (
    u_id                INT,
    np_id               INT,
    PRIMARY KEY(u_id, np_id),
    FOREIGN KEY (u_id) REFERENCES Users(u_id),
    FOREIGN KEY (np_id) REFERENCES NutritionPlans(np_id)
);
```

NutritionLog(l_id, n_date, n_time, u_id, np_id)

FK: u_id references Users

FK: np_id references NutritionPlans

```
CREATE TABLE NutritionLog (
   l_id              INT PRIMARY KEY,
   n_date            DATE,
   n_time            TIME,
   u_id              INT,
   np_id             INT,
   FOREIGN KEY (u_id) REFERENCES Users(u_id),
   FOREIGN KEY (np_id) REFERENCES NutritionPlans(np_id)
);
```

NutritionPlanContents(np_id, food_id)

FK: np_id references NutritionPlans

FK: food_id references FoodOrDrink

```
CREATE TABLE NutritionPlanContents (
   np_id             INT,
   food_id           INT,
   PRIMARY KEY(np_id, food_id),
   FOREIGN KEY (np_id) REFERENCES NutritionPlans(np_id),
   FOREIGN KEY (food_id) REFERENCES FoodOrDrink(food_id)
);
```

LeaderboardMetricTypes(m_type, m_name, m_description, calc_method)

```
CREATE TABLE LeaderboardMetric (
    m_type          INT PRIMARY KEY,
    m_name          VARCHAR(255) NOT NULL,
    m_description   VARCHAR(80),
    calc_method     VARCHAR(80)
);
```

UserPerformance(p_id, p_u_id, p_m_type, p_value, p_date)

FK: p_u_id references u_id from Users

FK: p_m_type references m_type from LeaderboardMetricTypes

```
CREATE TABLE UserPerformance (
    p_id            INT PRIMARY KEY,
    p_u_id          INT,
    p_m_type        INT,
    p_value         DECIMAL(10, 2),
    p_date          DATE,
    FOREIGN KEY (p_u_id) REFERENCES Users(u_id),
    FOREIGN KEY (p_m_type) REFERENCES
LeaderboardMetricTypes(metric_id)
);
```

Leaderboard(lb_id, lb_m_type, lb_u_id, lb_rank, lb_value, lb_date)

FK: lb_m_type references m_type from LeaderboardMetricTypes

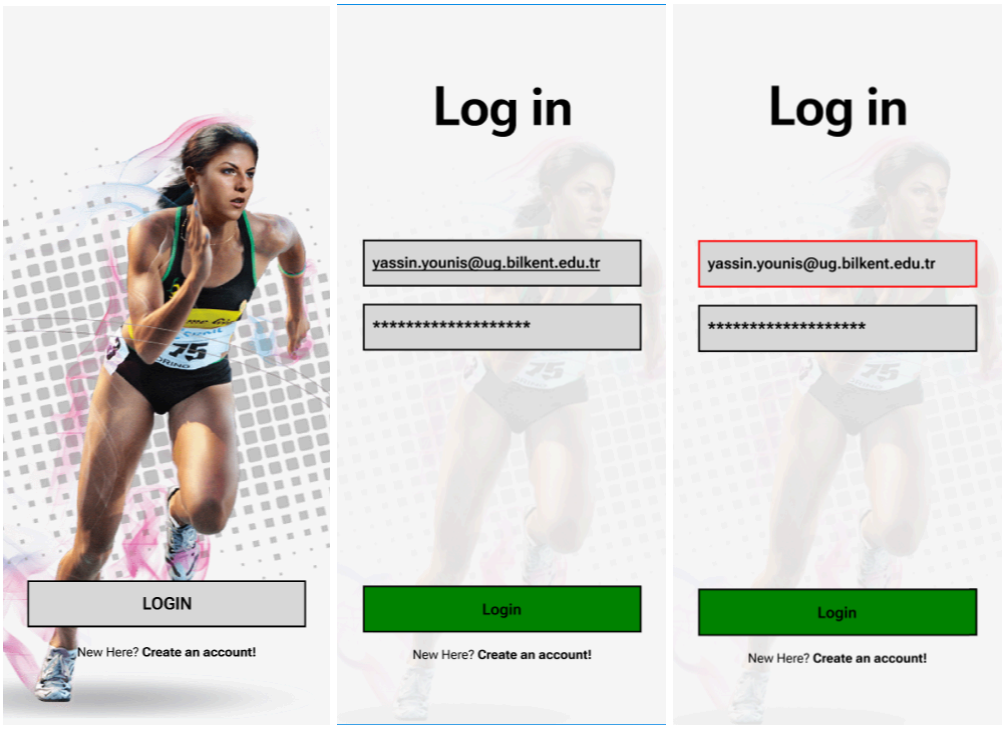
FK: lb_u_id references u_id from Users

```
CREATE TABLE Leaderboard (
    lb_id           INT PRIMARY KEY,
    lb_m_type       INT,
    lb_u_id         INT,
    lb_ rank        INT,
    lb_value        DECIMAL(10, 2),
    lb_date         DATE,
    FOREIGN KEY (lb_m_type) REFERENCES
LeaderboardMetricTypes(metric_id),
    FOREIGN KEY (lb_u_id) REFERENCES Users(u_id)
);
```

5. UIs and SQL Queries of Functionalities

5.1. Common Functionality-1: User Management

5.1.1. Login Validation:

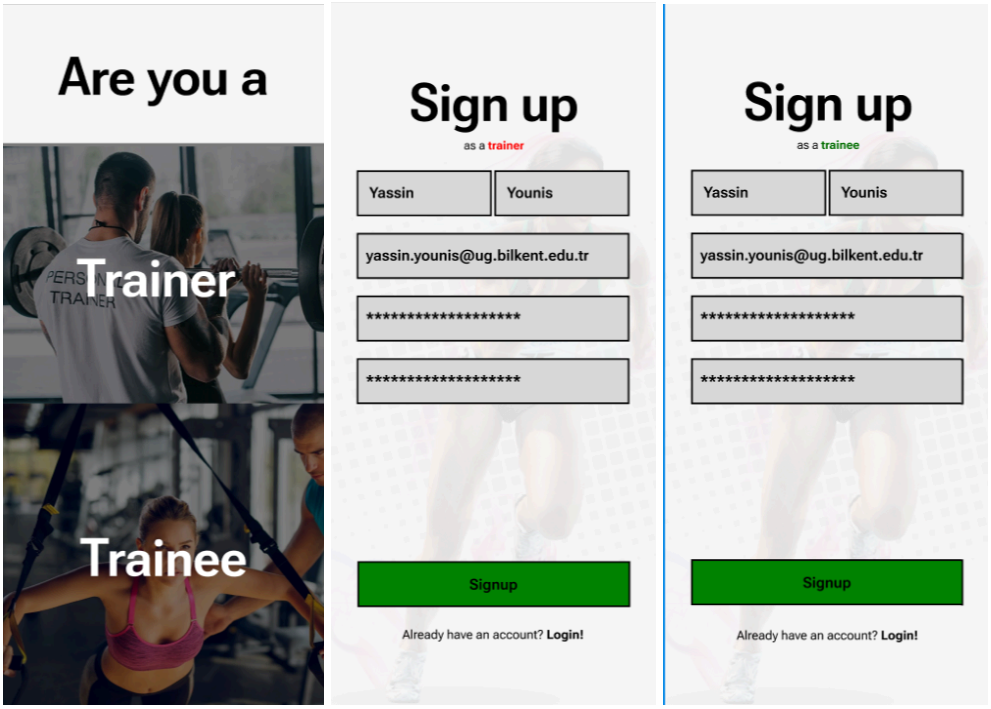


Check if the user's credentials exist in the database.

SELECT * FROM User

WHERE username = name_input AND password = password_input;

5.1.2. Create New User:



Generate a unique user ID by getting all IDs, then create a new user record.

1. Get all IDs to create a unique ID:

SELECT user_id FROM User;

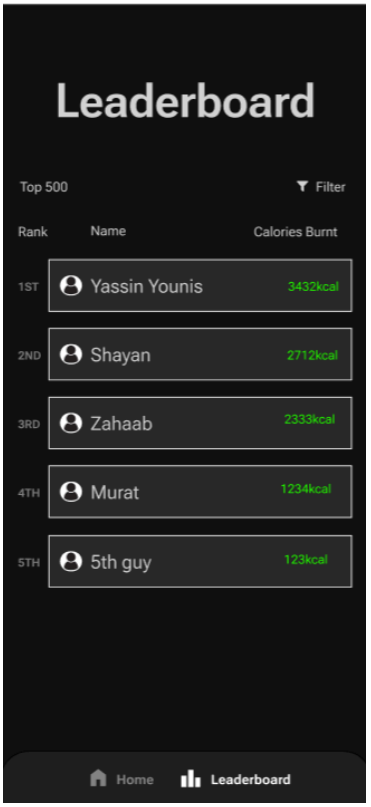
2. Create user:

INSERT INTO User VALUES

(newID, username_input, email_input, name_input, password_input, specified_user_type);

5.2. Common Functionality-2 (EXTRA): Leaderboard

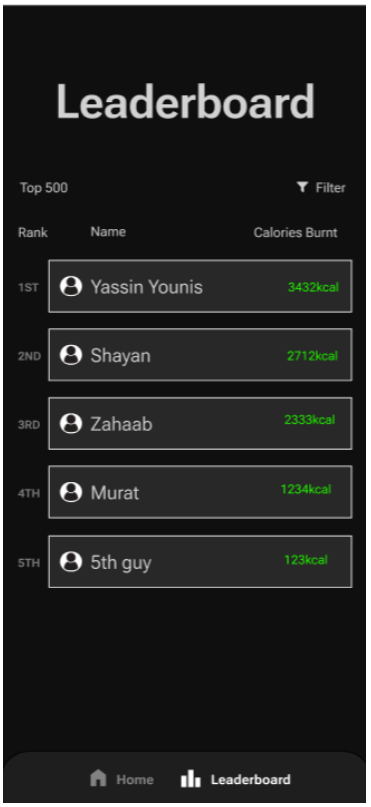
5.2.1. List Top Performers for a Metric:



To display the top performers for a given metric on the leaderboard.

```
SELECT lb_u_id, lb_rank, lb_value, lb_date
FROM Leaderboard
WHERE lb_m_type = specified_metric_type
ORDER BY lb_rank ASC
LIMIT specified_limit;
```

5.2.2. Update Leaderboard:



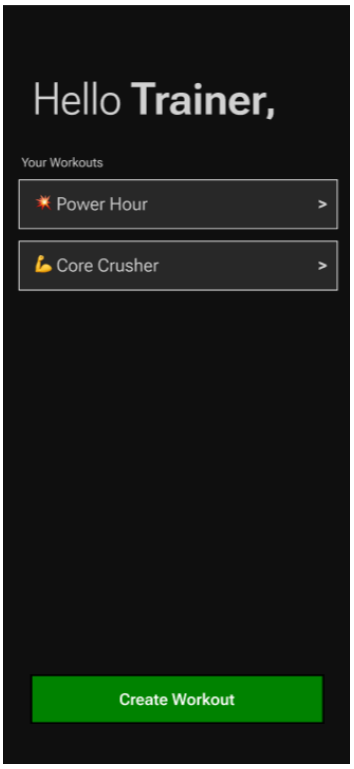
After a user completes a workout or achieves a new metric, update the leaderboard accordingly.

```
INSERT INTO UserPerformance (p_u_id, p_m_type, p_value, p_date)
VALUES (specified_user_id, specified_metric_type, calculated_value,
current_date);

WITH RankedPerformance AS (
   SELECT p_u_id, p_m_type, p_value, p_date,
       RANK() OVER (PARTITION BY p_m_type ORDER BY p_value
DESC) AS lb_rank
   FROM UserPerformance
   WHERE p_m_type = specified_metric_type
)
UPDATE Leaderboard
SET lb_rank = RankedPerformance.lb_rank,
   lb_value = RankedPerformance.p_value,
   lb_date = RankedPerformance.p_date
```

```
FROM RankedPerformance
WHERE Leaderboard.lb_u_id = RankedPerformance.p_u_id
AND Leaderboard.lb_m_type = specified_metric_type;
```

5.3. Trainer-Specific Functionality: Creating Workout Programs



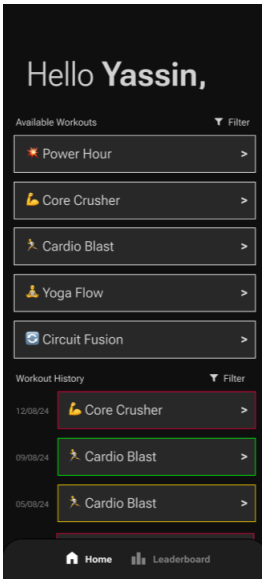
a. List all available workout types.

```
SELECT * FROM Workouts;
```

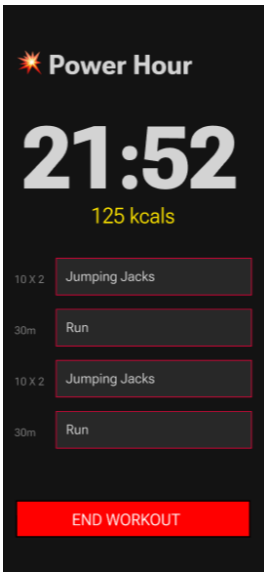
b. Apply filters if necessary (workout intensity, duration, equipment needed, etc.)

```
SELECT * FROM Workouts
WHERE difficulty = 'desired_intensity'
AND duration <= 'maximum_duration_minutes'
AND equipment_needed = 'specific_equipment';
```

c. User selects a workout:

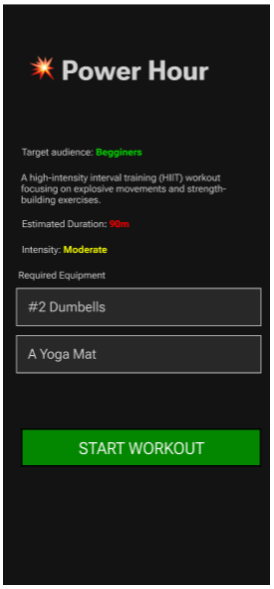


i. The system displays the details of the workout.



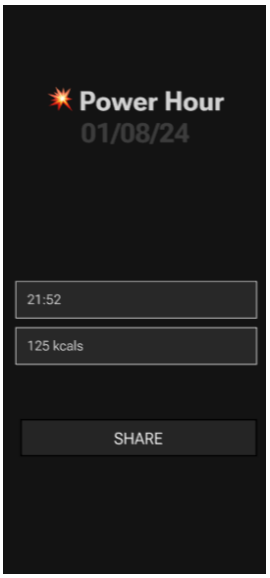

```
SELECT * FROM Workouts
WHERE w_id = ‘selected_workout_id';
```

ii. The user starts the workout session.



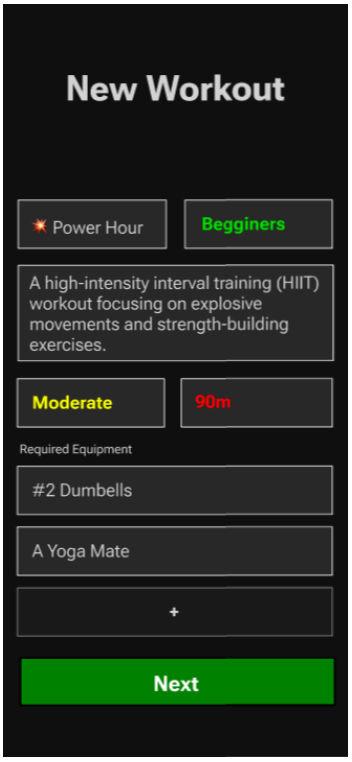
INSERT INTO Exercise_Log(u_id, w_id, date, time, sets_completed, calories_burnt)
VALUES ('user_id', 'selected_workout_id', CURRENT_DATE, CURRENT_TIME, 0, 0);

iii. The system tracks the duration, calories burned, and other relevant metrics + iv. Upon completion, the workout session is added to the user's workout history.



UPDATE Exercise_Log
SET duration = 'workout_duration', calories_burnt = 'calories_burned'
WHERE l_id = 'log_id' AND u_id = 'user_id';

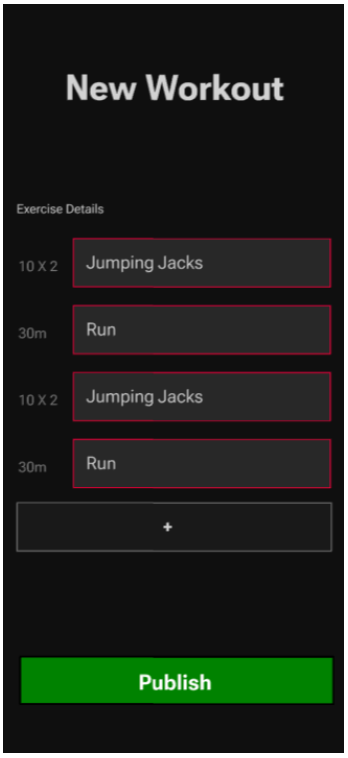
d. The trainer creates a workout program:



i. Specify the details of the workout program + ii. Specify the title of the workout + iii. Select the target audience (e.g., beginners, intermediate, advanced) + iv. Select the required equipment + v. Enter the description and objectives of the workout + vi. Set the estimated duration and intensity.

```
INSERT INTO WorkoutPrograms(w_id, w_name, w_description,
difficulty, target_audience, equipment_needed,
estimated_duration, intensity)
VALUES (DEFAULT, 'program_name', 'program_description',
'difficulty_level', 'target_audience', 'equipment_needed',
'duration', 'intensity');
```

vii. Publish the workout so that users can list it.



UPDATE Workouts

SET is_published = TRUE

WHERE w_id = 'newly_created_workout_id';

6. Implementation Plan

PostgreSQL, which provides powerful data store and management capabilities, will be used as the backend database management system for the fitness tracker database project. PostgreSQL will be a great tool for managing the variety of data produced by our fitness app thanks to its dependability and scalability. As for the frontend, we will use React to provide a responsive and dynamic user interface, guaranteeing a captivating user experience.

The project intends to provide a smooth and user-friendly fitness tracking app that efficiently gathers, saves, and presents user data for monitoring and analysis. Tasks pertaining to front-end interface development, database schema design, back-end development, and iterative design revisions to improve the user experience will all be included in the implementation plan.