Technical Specification for Workout Tracking Platform

User Stories/User Requirements:

1.1 User Account Creation: The user will be able to create an account by providing their email address and password. The user will also add personal information in order to assist the api in creating a recommended workout plan.

1.2 Fitness Goals: Users will be able to set fitness goals, such as losing weight, gaining muscle mass, or increasing endurance.

1.3 Logging Workouts: Users will be able to log their workouts, including the exercises performed, sets, reps, weights, and duration.

1.4 Recommendations: The platform will provide users with recommendations for workouts based on their fitness goals, previous activity and their user information.

Stretch goal:

1.5 Social Features: Users will be able to connect with friends and join groups to share their progress and keep each other motivated.

Endpoints:

2.1 User Account Creation:

POST /users

This endpoint takes the user data type and adds a new user to the database.

The user datatype is in the following structure:

{

User_id: generated user_id,
Lbs: weight in pounds,
Name: name of fighter,
Log_id: id pointing to the user's workout log,
Plan_id: id pointing to the user's workout plan,
Inches: user's height in inches,

Sport: optional sport that the user is trying to achieve these goals for (will allow us to specify exercises)
}

2.2 Fitness Goals:

GET /goals
This endpoint would return a list of the user's goals. The output would be formatted as so:

{
User_id: the user id of the goal,
User_name: the name of the user whose goal this is,
Description: a description of the goal (this depends on how we structure the goal intake and data processing for workouts)

}

GET /goals/plan
This endpoint returns the workout plan based on the goals of a user. The output would be formatted as so:
{
Plan_id: the id of the workout plan,
Goal_id: the id of the goal(s) associated with it,
User_id: the id of the user the plan is associated with,
Workout_id: the id(s) of the workout the goal is leading the user to
}

POST /goals
This endpoint would post the goal. The post would be in the following structure:

{
user_id: the user id of the goal,
plan_id: the id of the plan,
goal_id: the id of the goal,
Description: varchar description of the goal

}

PUT /goals/:goal_id
This endpoint allows the user to update their goal. Following the update of the goal the api will re-adjust the plan_id it is pointing to.

{
    Goal_id: the id of the goal to update
    User_id: the user id for the goal
    Description: the varchar string of what the goal is being updated to
}

DELETE /goals/:goal_id
This endpoint deletes the goal to its associated id.

2.3 Logging Workouts:

GET /workouts
This endpoint will return the user's workout log
{
    User_id: the id of the log's user,
    Workouts: the list of workouts the user has logged

    The list would be structured as follows:
    {
        Workout_id: id of the workout,
        Time_posted: time the workout was posted,
        Feedback: the feedback the user provided for the workout
    }
}

POST /workouts
This endpoint would allow the user to create a workout for their log

{
User_id: the id of the user who's log this is being added to,
Log_id: the log that the workout is being added to,
Workout_id: the id of the workout the user wants to add,
Feedback: the feedback for the workout
}

PUT /workouts/:workout_id
This endpoint would allow the user to edit the workout for their log
{
Log_id: the id of the log the user wants to edit,
Workout_id: the id of the workout the user wants to edit,
Feedback: the varchar string of feedback the user provides
}

DELETE /workouts/:workout_id
This endpoint would allow the user to delete the workout for their log

2.4 Recommendations:

GET /recommendations
This endpoint would list all of the workout plans for the associated user. The output would be in the following format:

{
User_id: The id of the user who's workouts they are,
Plans: A list of workout plan's relating to the user and it's goals.
}

2.5 Social Features: Social features will be expanded upon in v2.

Detailed Descriptions of Edge Cases and Transaction Flows:
3.1 User Account Creation: The endpoint should check if the provided email address is already in use. If it is, the endpoint should return an error message and prevent account creation.

3.2 Fitness Goals: When creating or updating a fitness goal, the endpoint should check that the user is authorized to perform the action. If the user is not authorized, the endpoint should return an error message and prevent the action. The endpoint should also check that the provided goal data is valid, including the goal type, target value, and target date.

3.3 Logging Workouts: When creating or updating a workout, the endpoint should check that the user is authorized to perform the action. If the user is not authorized, the endpoint should return an error message and prevent the action. The endpoint should also check that the provided workout data is valid, including the exercise type, sets, reps, weights, and duration.

3.4 Recommendations: The recommendations endpoint should use a recommendation algorithm based on the user's fitness goals and previous activity. If the user has not logged enough data, the endpoint should return a message indicating that more data is needed.

Stretch Goal:

3.5 Social Features: When creating or updating a friend or group, the endpoint should check that the user is authorized to perform the action. If the user is not authorized, the endpoint should return an error message and prevent the action. The endpoint should also check that the provided friend or group data is valid, including the friend or group name and user ids.

Additional Features:
4.1 Data Validation: All endpoints should perform data validation to ensure that the provided data is in the correct format and meets any constraints.

4.2 Error Handling: All endpoints should return appropriate error messages if an error occurs, including the HTTP status code and a description of the error.

4.3 Authentication: The API should use authentication to ensure that only authorized users can access or modify data.

4.4 Authorization: The API should use authorization to ensure that users can only access or modify their own data.

4.5 Security: The API should use HTTPS to ensure that all data is transmitted securely.

4.6 Scalability: The database should be designed for scalability to handle a large number of users and data over time.