# OsoFit

A software Engineering I project by Connor Griffin, Dannis Wu, Jack Fontenot, Mason Baxter, Savannah Johnson, and Ethan Rogers

# Table of Contents

(Screenshot of word pdf of template)

# OsoFit Vision Document

Version 1.0 | Date: 14 September 2025

## 1. Introduction

The purpose of this Vision document is to define the high-level needs, features, and objectives of OsoFit, a health and fitness tracking application designed to help users monitor daily wellness, set goals, and engage with trainers and peers. The document captures the motivation behind the product and outlines its major capabilities.

### *References*

- Group Project Deliverable 7, OsoFit (Sept 2025)
- Visual Paradigm Diagrams (Domain Model, SSDs)
- GitHub Repository: https://github.com/jaackfont05/OsoFit

## 2. Positioning

### *Problem Statement*

The problem of inconsistent health tracking and structured goal monitoring affects both individuals who want a better tool to manage their lifestyle and trainers who need easier access to clients. The impact of this problem is unstructured routines that lack guidance and accountability. A successful solution would provide integrated health, nutrition, and exercise tracking, personalized goal setting and progress reporting, and trainer-led classes with social accountability.

### *Product Position Statement*

For health-focused individuals and trainers who need a unified health and fitness management platform, the OsoFit App is a mobile application that allows users to track health metrics, manage workouts, and set personalized goals. Unlike other fitness trackers that isolate these features, our product integrates tracking, social interaction, and trainer functionality into one cohesive system.

## 3. Stakeholder and User Descriptions

### *Stakeholder Summary*

| Name | Description | Responsibilities |
|---|---|---|
| General Users | Individuals seeking to improve fitness | Log health data, view progress, follow plans |
| Trainers | Certified professionals | Create exercise plans, lead classes, monitor clients |
| Admins | System administrators | Manage accounts, ensure security, handle support |

### *User Environment*

Accessible via mobile and desktop; designed for quick daily interaction and clear dashboards. Requires internet connectivity for data syncing.

### *Summary of Key Stakeholder or User Needs*

| Need | Priority | Current Solution | Proposed Solution |
|---|---|---|---|
| Track all health metrics | High | Multiple separate trackers | Unified OsoFit platform |
| Social accountability | Medium | Social media or none | Friend/group systems |
| Trainer-client integration | High | Manual messaging | Built-in trainer tools |

## 4. Product Overview

### *Product Perspective*

OsoFit is a standalone mobile and web application designed to synchronize user data through secure cloud storage. It interacts with standard APIs for authentication and optional wearable device integration.

### *Assumptions and Dependencies*

- Users have access to smartphones or computers.
- Requires stable internet connection.
- Depends on secure authentication and database infrastructure.

## 5. Product Features

- User Management: Create, log in, reset passwords.
- Health Tracking: Record calories, weight, sleep, and water intake.
- Workout Logging: Save workout sessions and monitor progress.
- Goals and Reporting: Set weight/sleep goals, view analytics.
- Trainer Tools: Create classes, self-paced or scheduled plans.
- Class Management: Register, track attendance, manage sessions.
- Social Features: Add friends, create groups, send requests.
- Injury Logging: Record injuries and adjust plans accordingly.

## 6. Other Product Requirements

- Platform: Mobile (Android/iOS) and Web.
- Performance: Real-time syncing between users and trainers.
- Security: Encrypted user data, password recovery.
- Scalability: Supports growing user base and integration with wearable APIs.
- Usability: Intuitive interface suitable for all experience levels.
- Documentation: Online help and tutorials integrated into the app.

Use Cases:

**Jack Fontenot**
1. Fully-dressed

**ID** UC Change User/Trainer Password
**Scope** Login system
**Level** User goal
**Stakeholders and interests**
User
- Person that forgot or needs to change their password to track their fitness goals.

Trainer
- Person that forgot or needs to change their password to assist users.

System admin
- Person responsible for login system run

**Precondition:** Password change is requested.

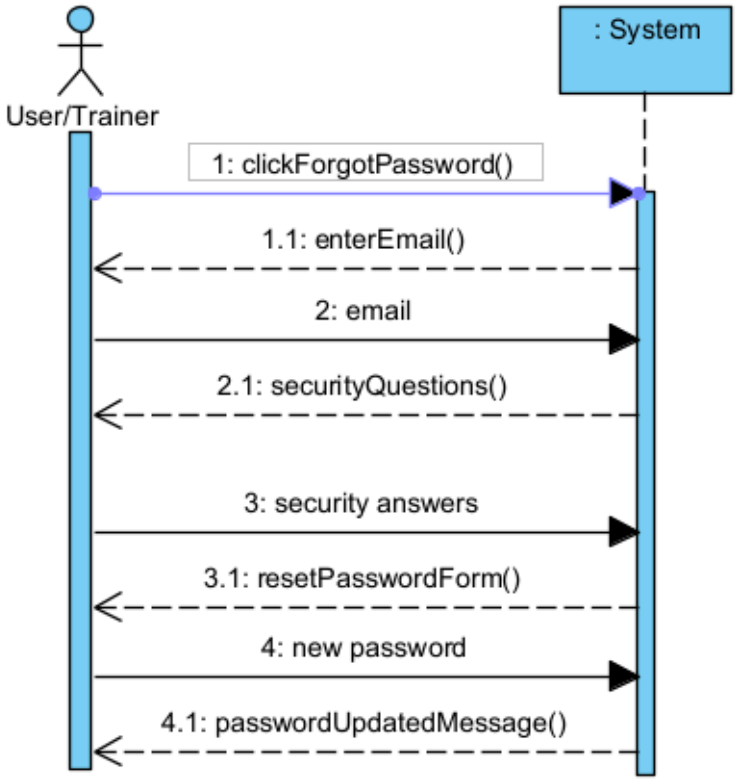
**Postcondition:** System updates the account's password.

**Main success scenario:**

1. User/trainer forgets their password
2. User/trainer clicks "forgot password?"
3. User/trainer enters their security questions answers
4. User/trainer clicks the confirm
5. User/trainer enters the updated password into the "new password" box
6. User/trainer clicks "update password"
7. System saves the account's new password
8. User/trainer logs in with new password

**Alternate paths:**
a.* anytime the process does not work
1. User/trainer emails user support
2. Admin reads user/trainer's email and manually resets password

**Contract CO2**: Change User/Trainer password
**Cross-references:** none
**Pre-Condition**: User/Trainer has an account and has forgotten their password
**Post-Conditions:**
- uPass was changed (attribute modification)




2. Fully-dressed

**ID** UC Logging Calories and Nutritional Macros
**Scope** Nutrition tracking system
**Level** user goal
**Stakeholders and interests**
User

- Person that is interested in logging their calories and macros

Trainer

- Person that is interested in checking the diet of their clients

System admin

- Person responsible for nutrition tracking system run

**Precondition:** User has eaten a meal

**Postcondition:** Meal is saved and nutritional info updated

**Main success scenario:**

1. User eats a meal
2. User logs in to the app
3. User clicks “log meal”
4. User enters the name of the meal
5. User enters the number of calories and amount of fat, carbs, and protein in grams of the meal
6. User clicks “confirm meal”
7. System saves all the info of the meal
8. System adds meal to the user’s previous meals list
9. System adds the amount of calories and macros towards their daily goals of each
10. Trainer can view the user’s meal

**Alternate paths:**

a.* anytime the app does not respond

1. User/trainer restarts the app

2.a If user cannot login to the app

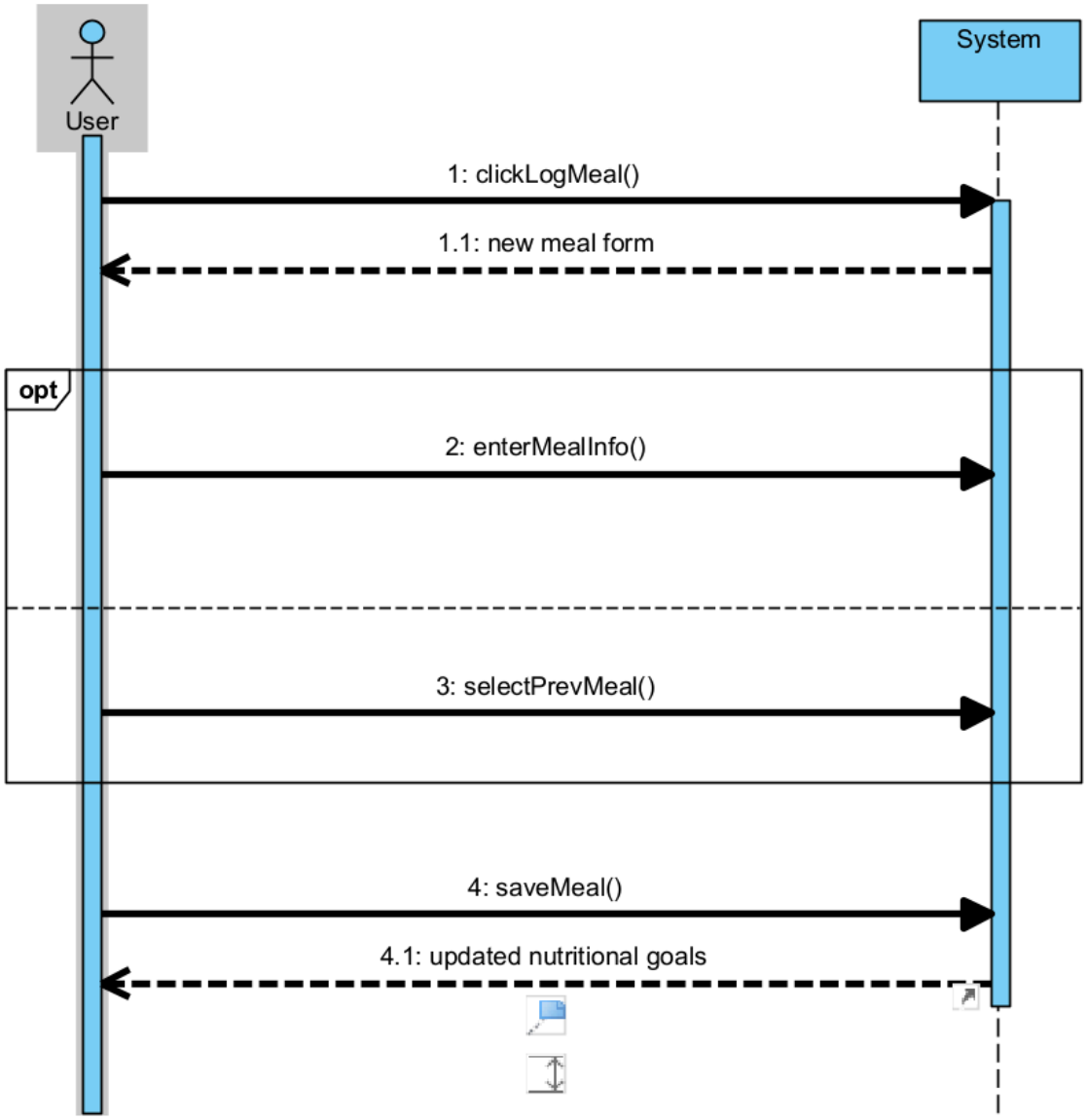
1. User resets their password

4.a If user ate a meal that was previously logged

1. User finds the meal in their previous meals list
2. User selects the meal
3. System saves all the info of the meal

5.a if user does not enter calories, fat, carbs, or protein,

1. The system updates according to the fields entered




**Contract CO2**: enterMeal
**Operation:** enterMeal(calories: integer)
**Pre-conditions**: User has eaten a meal
**Post-conditions:**

- A Meal instance uMeal was created (instance creation)
- uMeal was associated with the User. (association formed)
- uMeal.calories was set to calories. (attribute modification)

3. Brief

**ID** UC Create an Exercise

A user needs to create an exercise. The user will login to the system and then click the “Add Exercise” button. The user will then enter the exercise name, equipment needed, sets, and reps. The user clicks the “Create Exercise” button, and the system saves the exercise. The system notifies the user that the exercise has been saved.

- **Savannah Johnson:**

1. Fully Dressed: Logging weight

**ID** UC Tracking weight

**Scope** Health & Fitness System

**Level** User goal

**Stakeholders and Interests**

General User (User) - A user wants to track and log their weight over time

Trainer - Creates and manages weight logs and fitness plans, monitors progress

Admin - Ensures content follows app rules, policies, and maintains privacy

System Manager/Maintainer - Ensures the application runs smoothly, stores and updates data efficiently.

**Precondition:** User profile is created and the user is logged in

**Postcondition:** Weight progress is logged and saved. The user’s weight history log is updated with the newly created weight entry data. The user’s weight goals are updated accordingly.

**Main Success Scenario:**

1. User wants to log their current weight
2. User clicks “Record Weight”
3. System prompts the user to input their weight and specified unit of measurement
4. User enters date for the weight input
5. User enters weight
6. User clicks “Save Weight”
7. System validates fields, ensuring correct units and valid weight and date ranges
8. System stores the new weight entry in the user’s history
9. User views their updated progress and weight history

**Alternate Paths:**

a.* System does not respond

    1. An error message is displayed to the user asking them to restart the entry process

3a. A new form can’t be created or loaded

    1. System shows an error message and asks the user to refresh the page or try again later.

7a. The user entered an invalid unit measurement

    1. System shows an error message and asks user to change the unit

7b. The user entered an invalid range for weight

1. System shows an error message and asks user to enter a value greater than 0

7c. The user entered an invalid date

1. System shows an error message and asks user to enter a correct date where months are between 0 and 12, days are between 1 and 31, and year is 4 digits.

7d. The user left a required field blank

1. System shows an error message and asks user to enter a value in the fields

8a. System is unable to save entry

1. System shows an error message and asks user to try again

9a. System is unable to update the user's weight

1. System sends error report to system manager
2. System warns the user that their progress may not have saved
2. System shows error message to user, informs them that goal progress might not be accurate until a system manager can correct the issue

**Operation Contracts:**

recordNewWeight()

Operation: recordNewWeight()

Cross References: None

Precondition: User profile has been created. The user is logged in.

Postcondition: A new weight entry form is created and displayed for the user to fill out

saveWeight()

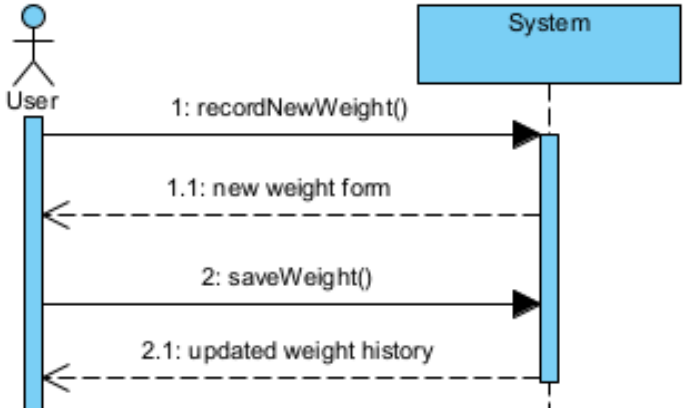
Operation: saveSleep(Date: date, Weight: double, Unit: string)

Cross References: None

Precondition: User has filled out the form with correct values for unit, date, and weight.

Postcondition:

- A weightRecord is created for the user
- User's weight goals have their progress updated
- The weight entry is stored in user's weight history




2. Fully Dressed: Set weight goal

**ID** UC Setting Weight Goal

**Scope** Health & Fitness System
**Level** User Goal
**Stakeholders and Interests**
General User (User) - Wants to track and set weight goals over time
Trainer - Creates and manages exercise plans
Admin - Ensures content follows app rules and policies
System Manager/Maintainer - Person responsible for ensuring the application runs smoothly
**Precondition:** User profile has successfully been created.
**Postcondition:** New weight goal is saved in the system. This goal's progress is updated. The user can view their weight goal and monitor progress of the weight goal.
**Main Success Scenario:**
1. User wants to create or view a weight goal
2. User clicks "Weight Goals"
3. System displays the user's current weight goal(s)
4. User clicks "Create Weight Goal"
5. System displays weight goal form
6. User inputs a weight and unit of measurement for the goal
7. User enters a target date for the goal (optional)
8. User clicks "Save New Goal"
9. System validates weight range, valid date, and valid unit of measurement
10. If fields are correct, the system stores the weight goal and updates the user's weight goal history.
11. The user can view weight goal history and progress toward the current weight goal.

**Alternate Paths:**
2a. User's sleep goals page can't be loaded or displayed properly
    1. An error message is displayed to the user notifying them that there has been an error and to refresh the page or to try again later.
    2. System sends error message to the system manager
6a. The user entered an invalid unit
    1. System displays an error message and asks user to enter a valid unit
6b. The user entered an invalid weight range
    1. System shows an error message and asks user to enter a positive weight goal
6c. The user entered an invalid date
    1. System shows an error message and asks user to enter a correct date that is in the future, where the values are within correct range: months are between 0 and 12, days are between 1 and 31, and year is 4 digits
10a. The system is unable to store the weight goal
    1. System sends error report to system manager
    2. System warns the user that their progress may not have saved

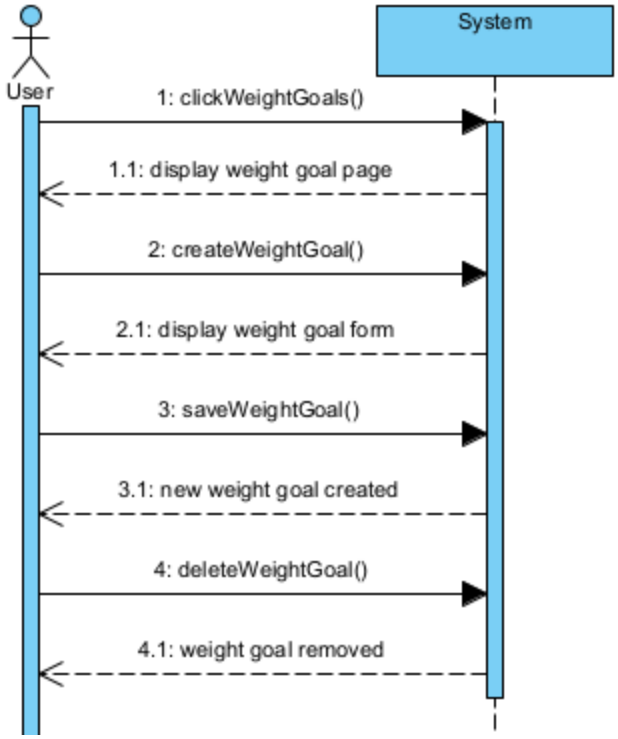
**Operation Contracts:**

clickWeightGoals:
Operation: clickWeightGoals()
Cross References: None
Precondition: User account is created and the user is logged in.
Postcondition: The user is able to view the weight goals page

createWeightGoal:
Operation: createWeightGoal()
Cross References: None
Precondition: User is currently viewing the weight goals page
Postcondition: The user is able to view the weight goal form.

saveWeightGoal:
Operation: saveWeightGoal(TargetWeight: double, Unit: string, TargetDate: date)
Cross References: None
Precondition: User has entered correct values for target weight, unit of measurement, and target date (optional).
Postcondition: A new weight goal is created for the user and updated. The user can view the weight goal and history.

deleteWeightGoal:
Operation: deleteWeightGoal(ID: int)
Cross References: None
Precondition: The specified goal already exists
Postcondition: The specified weight goal is removed from the user's history, and the user's weight goals are updated.




3. Fully Dressed: Tracking hydration

**ID** UC Logging water intake

**Scope** Health & Fitness System

**Level** User goal

**Stakeholders and Interests**

General User (User) - A user wants to track and log their water intake over time

Trainer - Creates and manages hydration logs, monitors progress

Admin - Ensures content follows app rules, policies, and maintains privacy

System Manager/Maintainer - Ensures the application runs smoothly, stores and updates data efficiently.

**Precondition:** User profile is created and the user is logged in

**Postcondition:** Hydration progress is logged and saved. The user's hydration log is updated with the newly created entry data. The user's hydration goals are updated accordingly.

**Main Success Scenario:**

1. User wants to log water intake
2. User clicks "Record Water"
3. System prompts the user to input the amount of water and units (oz, ml, etc.)
4. User enters date for the water input
5. User clicks "Save"
6. System validates fields, ensuring valid units, quantity, and date range
7. System stores and updates the new water intake entry in the user's hydration log
9. User views their updated hydration progress and history

**Alternate Paths:**

a.* System does not respond

1. An error message is displayed to the user asking them to restart the entry process

3a. A new form can't be created or loaded

1. System shows an error message and asks the user to refresh the page or try again later.

6a. The user entered an invalid unit of measurement

1. System shows an error message and asks user to change the unit

6b. The user entered an invalid range for water intake

1. System shows an error message and asks user to enter a value greater than 0

6c. The user entered an invalid date

1. System shows an error message and asks user to enter a correct date where months are between 0 and 12, days are between 1 and 31, and year is 4 digits.

6d. The user left a required field blank

1. System shows an error message and asks user to enter a value in the fields

7a. System is unable to save entry

1. System shows an error message and asks user to try again

7a. System is unable to update the user's hydration

1. System sends error report to system manager
2. System warns the user that their progress may not have saved
2. System shows error message to user, informs them that goal progress might not be accurate until a system manager can correct the issue

**Operation Contracts:**

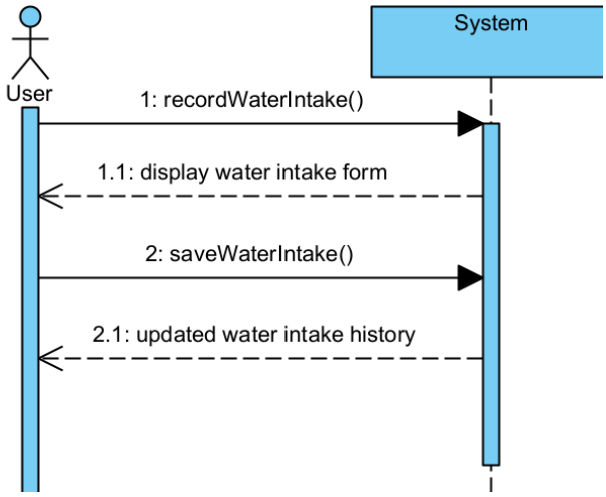
recordWaterIntake()

Operation: recordWaterIntake()
Cross References: None
Precondition: User profile has been created. The user is logged in.
Postcondition: A new water intake entry form is created and displayed for the user to fill out

saveWaterIntake()

Operation: saveWaterIntake(Date: date, Weight: double, Unit: string)
Cross References: None
Precondition: User has filled out the form with correct values for unit, date, and water intake.
Postcondition:

- A waterRecord is created for the user
- User's hydration goals have their progress updated
- The water intake entry is stored in user's water intake history


System
User
1: recordWaterIntake()
1.1: display water intake form
2: saveWaterIntake()
2.1: updated water intake history

**Mason Baxter:**

1.Fully Dressed: Tracking Sleep

**ID** UC Tracking Sleep

**Scope** Health & Fitness System

**Level** User Goal

**Stakeholders and Interests**

General User (User) - Person who is interested in tracking their sleep

Trainer - Person who creates and manages exercise plans/classes

Admin - Person who is responsible for ensuring content follows app rules and policies.

System Manager/Maintainer - Person responsible for ensuring the application runs smoothly

**Precondition:** User profile has been created.

**Postcondition:** Sleep goal progress has been updated and saved. Entry data for sleep is created and saved in the user's sleep history.

**Main Success Scenario:**

1. User wants to log an amount of sleep they had.
2. User clicks "Record Sleep"
3. System prompts user to input the amount of sleep and a rating of the quality of sleep on a scale from one to ten
4. User enters the date(s) when the sleep occurred
5. User enters the amount of sleep
6. (optional) User enters the quality of the sleep from one to ten
7. User clicks "Save Sleep"
8. System validates all fields
9. System creates a new sleep entry in the user's history
10. System updates any of the user's sleep goals with this new information

**Alternate Paths:**

a.* Anytime system does not respond

    1. An error message is displayed to the user asking them to restart the entry process

3a. A new form can't be created or loaded

    1. System shows an error message, asks the user to refresh the page or try again later.

8a. The user entered a value that can't be validated

    1. System shows an error message, asks the user to change the entry amount or try again

8b. The user entered a value that is out of the given range for quality of sleep

    1. System shows an error message, asks the user to re-enter a value between one and ten

8c. The user left a required field blank

    1. System shows an error message, asks the user to enter a value in the required fields

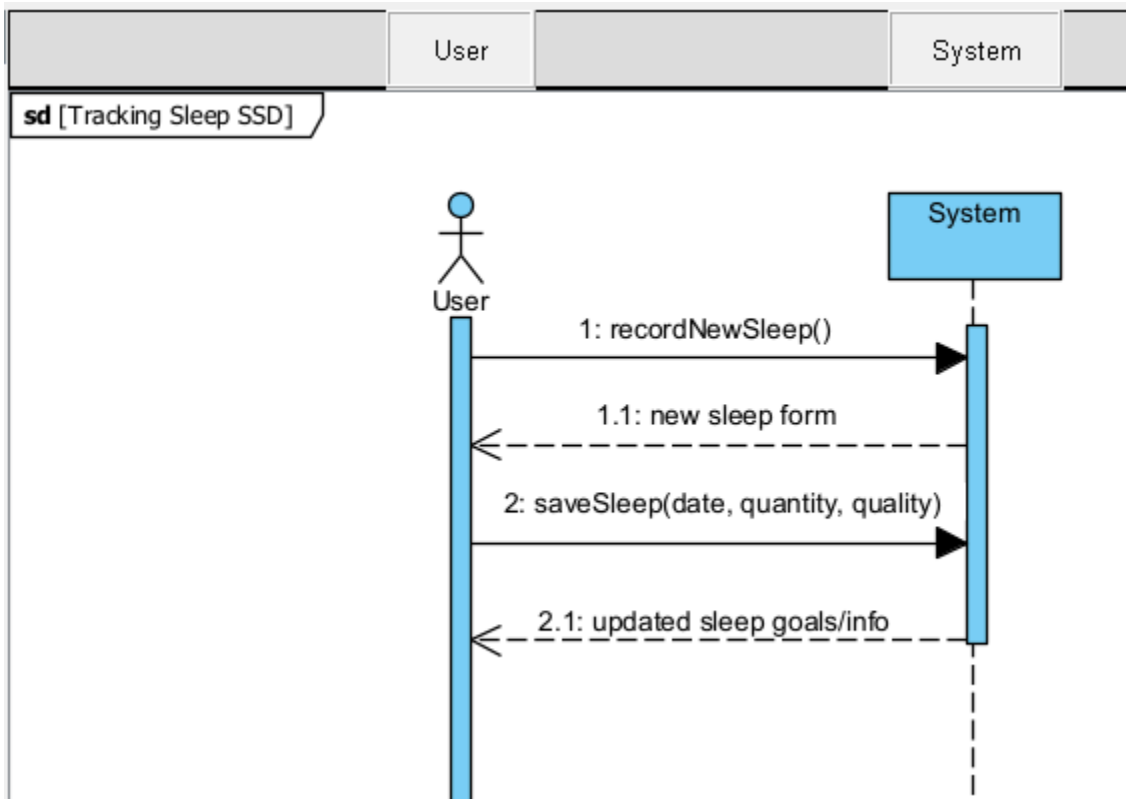
9a. System is unable to save or create a new entry

    1. System shows an error message, asks user to try again

10a. System is unable to update the user's sleep goals

    1. System sends error message to system manager
    2. System shows error message to user, informs them that goal progress might not be accurate until a system manager can correct the issue

Screenshot of SSD:




Operation Contracts:
recordNewSleep()
Operation: recordNewSleep()
Cross References: None
Precondition: User profile has been created. The user is on their dashboard.
Postcondition: A new sleep form is created and displayed for the user to fill out

saveSleep()
Operation: saveSleep(Date: date, Quantity: integer, Quality: integer)
Cross References: None
Precondition: User has filled out the form with correct values for date, and quantity.
Postcondition:

- A sleepRecord has been created for the user
- User's sleep goals have their progress updated if this sleepRecord's date, quality, or quantity is within the range

2. Fully Dressed: Setting Sleep Goals

**ID** UC Setting Sleep Goals

**Scope** Health & Fitness System

**Level** User Goal

**Stakeholders and Interests**

General User (User) - Person who is interested in tracking and setting goals for their sleep.

Trainer - Person who creates and manages exercise plans/classes

Admin - Person who is responsible for ensuring content follows app rules and policies

System Manager/Maintainer - Person responsible for ensuring the application runs smoothly

**Precondition:** User profile has successfully been created.

**Postcondition:** New sleep goal is saved in the system. This goal's progress is updated based upon the user's previous sleep entries that fit within the goal's ranges.

**Main Success Scenario:**

1. User wants to create a new sleep goal
2. User clicks "Sleep Goals"
3. System displays the user's created sleep goals
4. User clicks "Create New Sleep Goal"
5. System shows sleep goal form
6. User inputs a name for the goal
7. User selects if the goal is a daily, weekly, or monthly goal.
8. (optional) User selects a start date for the goal.
9. User inputs the amount of sleep they want to reach in the given timeframe
10. (optional) User inputs the minimum quality of sleep that will count towards this goal
11. User clicks "Confirm New Goal"
12. System validates all the fields
13. System creates a new sleep goal for the user with the given information and constraints
14. System updates the newly created sleep goal with the user's previous sleep records

**Alternate Paths:**

a.* Anytime system does not respond

    1. An error message is displayed to the user asking them to restart the goal creation process

3a. User's sleep goals page can't be loaded or displayed properly

    1. An error message is displayed to the user notifying them that there has been an error and to refresh the page or to try again later.
    2. System sends error message to the system manager

5a. A new form can't be created or loaded

    1. System shows an error message, asks the user to refresh the page or try again later

8a. The user did not set a start date for the goal

    1. System sets the start date for the goal to the default start date (the date the user created the goal).

10a. The user does not set a minimum quality of sleep that will count

    1. System sets the minimum quality of sleep for the goal to the default value of one.

12a. The user did not complete all required fields

    1. System shows an error message, asks the user to enter a value in the required fields

12b. The user entered a value that can't be validated

1. System shows an error message, asks the user to either change the entry or try again

12c. The user entered a minimum quality of sleep value outside the given range for quality of sleep

1. System shows an error message, asks the user to re-enter a value between one and ten

12d. Policy Keywords Found in Name

1. System shows an error message, asks user to input a different name

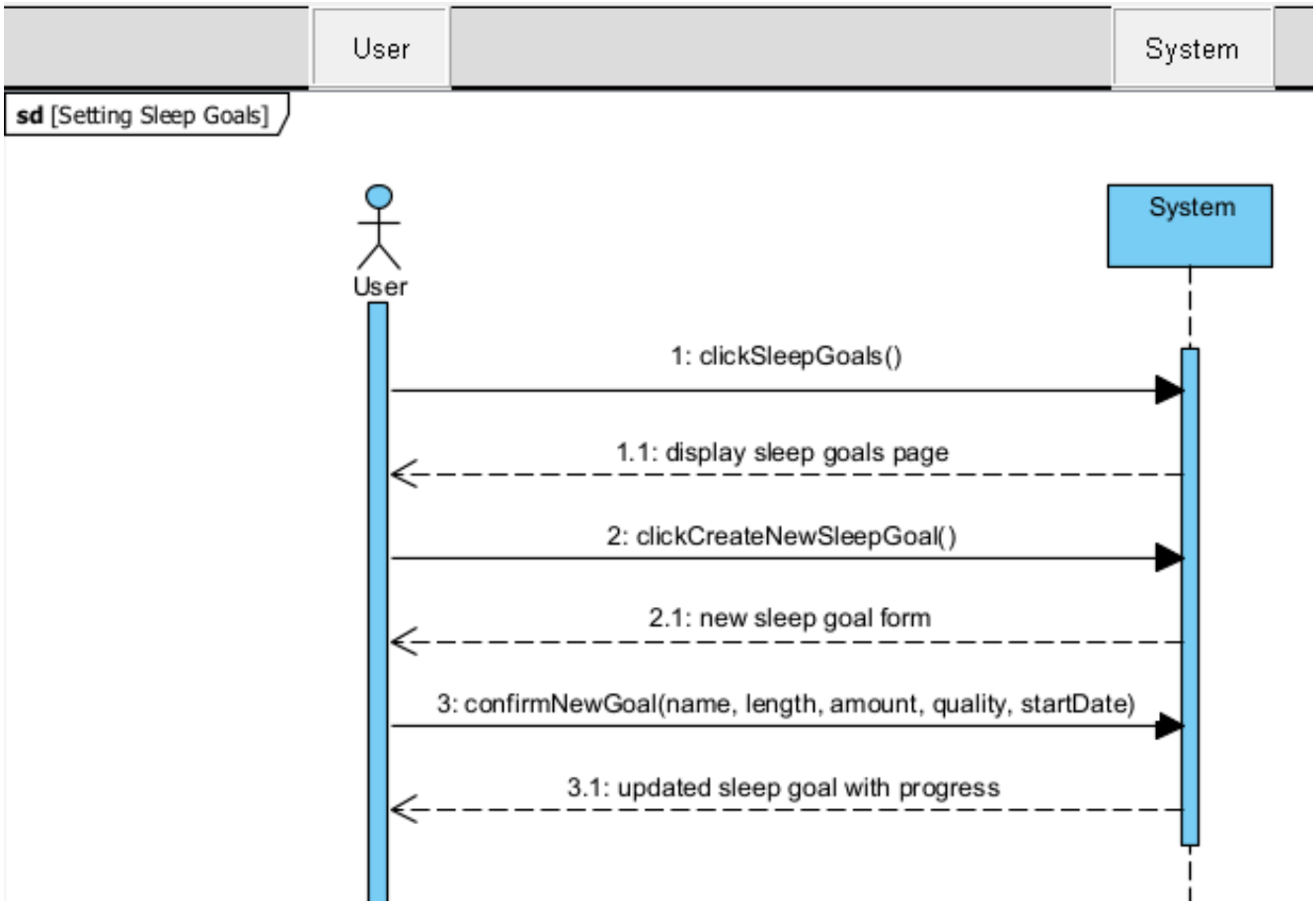
13a. System is unable to save or create a new entry

1. System shows an error message, asks user to try again

14a. System is unable to update the user's sleep goal

1. System sends error message to system manager
2. System shows an error message to the user, informing them that goal progress might not be accurate until a system manager can correct the issue.

Screenshot of SSD:



Operation Contracts:
clickSleepGoals:
Operation: clickSleepGoals()
Cross References: None
Precondition: User profile has already been created. The user is on their dashboard.
Postcondition: None

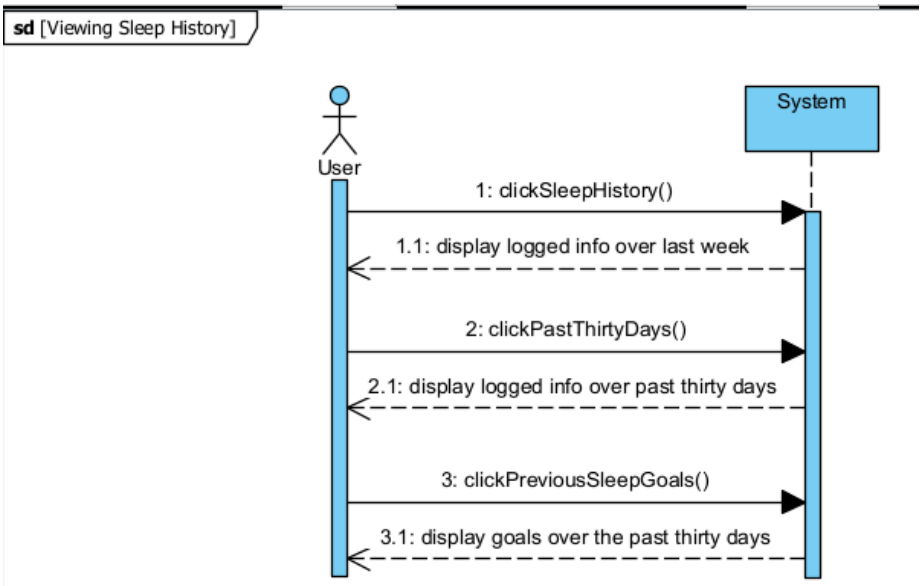
clickCreateNewSleepGoal:
Operation: clickCreateNewSleepGoal()
Cross References: None
Precondition: User is on the sleep goals page
Postcondition: A new sleep goal form is created for the user to fill out

confirmNewGoal:
Operation: confirmNewGoal(name: string, length: integer, amount: integer, quality: integer, startDate: date)
Cross References: None
Precondition: User has filled out a sleep goals form with correct values for name, length, and amount
Postcondition: A sleepGoal has been created for the user and updated according to the user's sleepRecords

3. Fully Dressed: Viewing Sleep History

**ID** UC Viewing Sleep History and Goals Over the Past Month
**Scope** Health & Fitness System
**Level** User Goal
**Stakeholders and Interests**
General User (User) - Person who is interested in tracking and setting goals for their sleep
Trainer - Person who creates and manages exercise plans/classes
Admin - Person who is responsible for ensuring content follows app rules and policies
System Manager/Maintainer - Person responsible for ensuring the application runs smoothly
**Precondition:** User profile has successfully been created.
**Postcondition:** None
**Main Success Scenario:**
1. User wants to view their logged sleep history
2. User clicks “Sleep History”
3. System displays the user’s previously logged sleep information over the past week
4. User clicks “Past 30 Days”
5. System displays the user’s previously logged sleep information over the past 30 days
6. User clicks “Previous Sleep Goals”
7. System displays the user’s sleep goals that had occurred during the past 30 days
**Alternate Paths:**
a.* Anytime system does not respond
    1. An error message is displayed to the user asking them to refresh the page.
3a. The user's sleep history can’t be loaded or displayed properly for the past week.
    1. An error message is displayed to the user notifying them that there has been an error and to refresh the page or to try again later.
5a. The user’s sleep history can’t be loaded or displayed properly for the previous thirty days
    1. An error message is displayed to the user notifying them that there has been an error and to refresh the page or to try again later.
7a. The user’s sleep goals for the past thirty days can’t be loaded or displayed
    1. An error message is displayed to the user notifying them that there has been an error and to refresh the page or to try again later.

Screenshot of SSD:



Operation Contracts:
clickSleepHistory:
Operation: clickSleepHistory()
Cross References: None
Precondition: User profile has been created. The user is on their dashboard.
Postcondition: None

clickPastThirtyDays:
Operation: clickPastThirtyDays()
Cross References: None
Precondition: User is on their sleep history page.
Postcondition: viewSleepTimeline is set to thirty days.

clickPreviousSleepGoals:
Operation: clickPreviousSleepGoals()
Cross References: None
Precondition: User is on their sleep history page.
Post Condition: None

**Ethan Rogers:**

1. Fully Dressed: Sending a Friend Request

**ID** UC Send Friend Request

**Scope** Social

**Level** Friends

**Stakeholders and Interests**

General User (user): someone who wants to connect with other people virtually

System Maintainer: person responsible for making sure that the social menus work

**Precondition:** User profile has already been created

**Postcondition:** none

**Main Success Scenario:**

1. User wants to send friend request
2. User clicks on “Social” menu
3. User navigates to “Friends” section
4. User clicks on “Add Friend” button
5. User submits the username of the person they want to friend
6. System notifies the other user of the friend request

**Alternate Paths:**

a. System does not respond

1. An error message is displayed to the user asking them to refresh the app

3a. The user’s friends list could not be loaded

1. An error message is displayed that there was an error loading their friends and to reload the app to try again

5a. The submitted username is invalid

1. An error message is displayed saying that the specified user doesn’t exist and to try another name

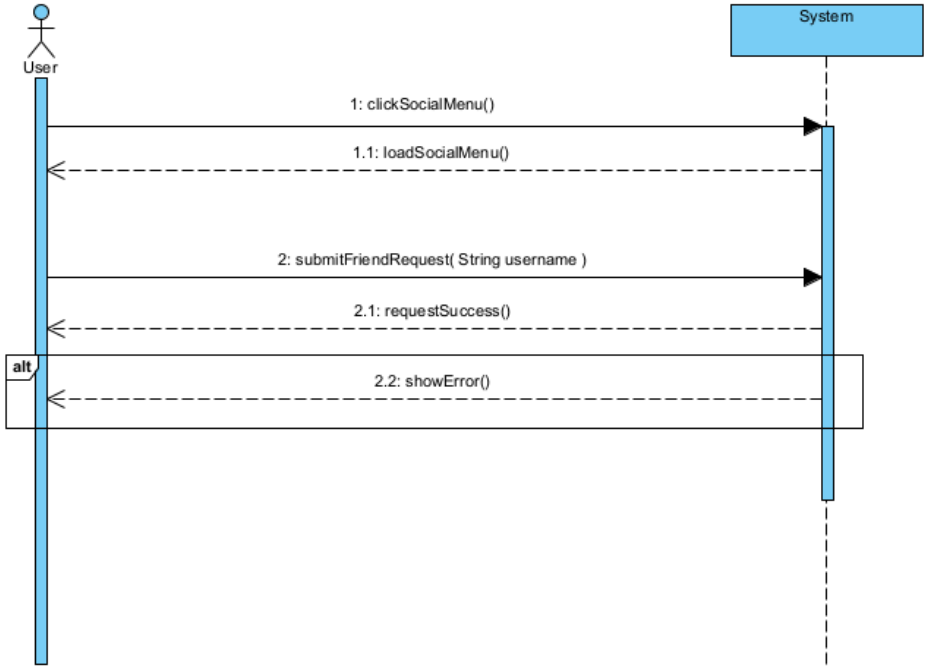
6a. The system does not receive the request

1. An error message is displayed telling the user to try again

6b. The system does not notify the other user

1. The other user will not be aware of the friend request until they open the app

**SSD:**




**Operation Contracts:**

clickSocialMenu:

Operation: clickSocialMenu()
CrossReferences: None
Precondition: The profile has been created and the user is on the dashboard
Postcondition: none

clickAddFriend:

Operation: clickAddFriend()
CrossReferences: none
Precondition: user is already in the social menu
Postcondition: none

submitFriendRequest:

Operation: submitFriendRequest(String username)
CrossReferences: none
Precondition: user has clicked add a friend button
Postcondition: the system will notify the other user of the friend request

2. Fully Dressed: Accepting a Friend Request

**ID** UC Accept Friend Request

**Scope** Social

**Level** Friends

**Stakeholders and Interests**

General User (user): someone who wants to connect with other people virtually

System Maintainer: person responsible for making sure that the social menus work

**Precondition:** User profile has already been created. User has received a friend request.

**Postcondition:** none

**Main Success Scenario:**

1. User wants to accept friend request
2. User clicks on “Social” menu
3. User navigates to “Friends” section
4. User clicks the “Incoming Friend Request” button
5. User accepts the request of the person they want to be friends with
6. System records the friendship between the two users. System notifies the other user that the friend request has been accepted.

**Alternate Paths:**

a. System does not respond

    1. An error message is displayed to the user asking them to refresh the app

3a. The user’s friends list could not be loaded

    1. An error message is displayed that there was an error loading their friends and to reload the app to try again

4a. The user’s incoming friend request failed to load

    1. An error message is displayed saying that their friend requests could not be displayed and to try restarting the app.

5a. The user rejects the friend request

    1. The system deletes the request

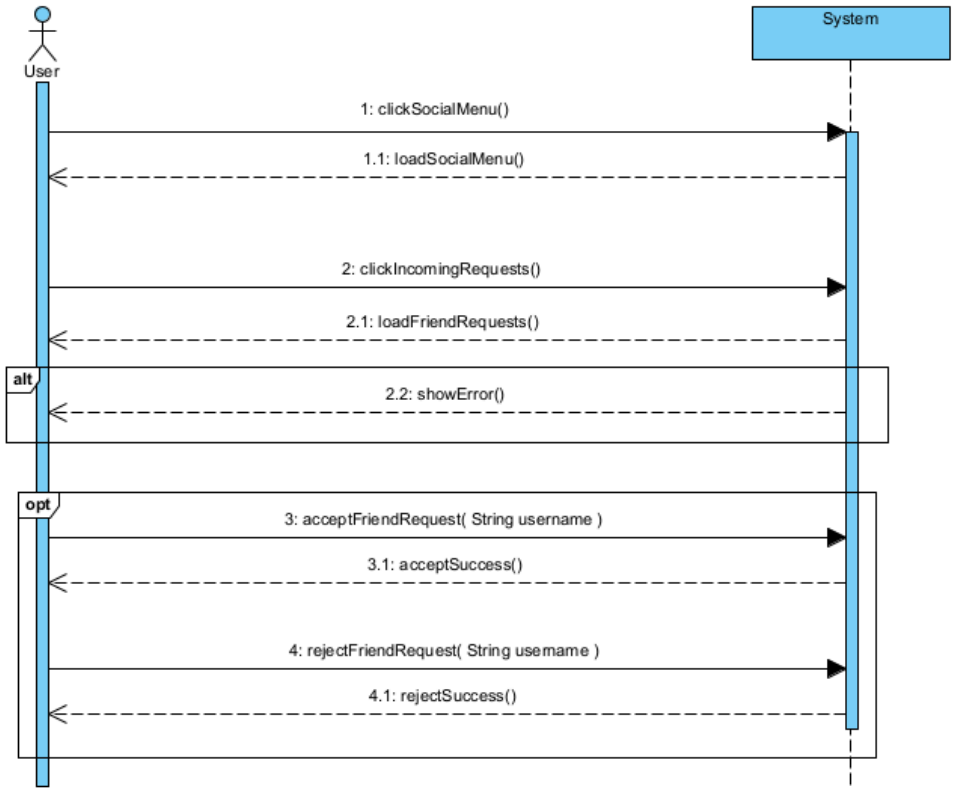
6a. The system does not receive the accept request

    1. An error message is displayed telling the user to try again

6b. The system does not notify the other user

    1. The other user will not be aware of the accepted friend request until they open the app

**SSD:**




**Operation Contracts:**

clickSocialMenu:

Operation: clickSocialMenu()
CrossReferences: None
Precondition: The profile has been created and the user is on the dashboard
Postcondition: none

clickIncomingRequests:

Operation: clickIncomingRequests()
CrossReferences: none
Precondition: user is already in the social menu, the user must have at least 1 incoming friend request
Postcondition: the app must load all of the incoming friend requests

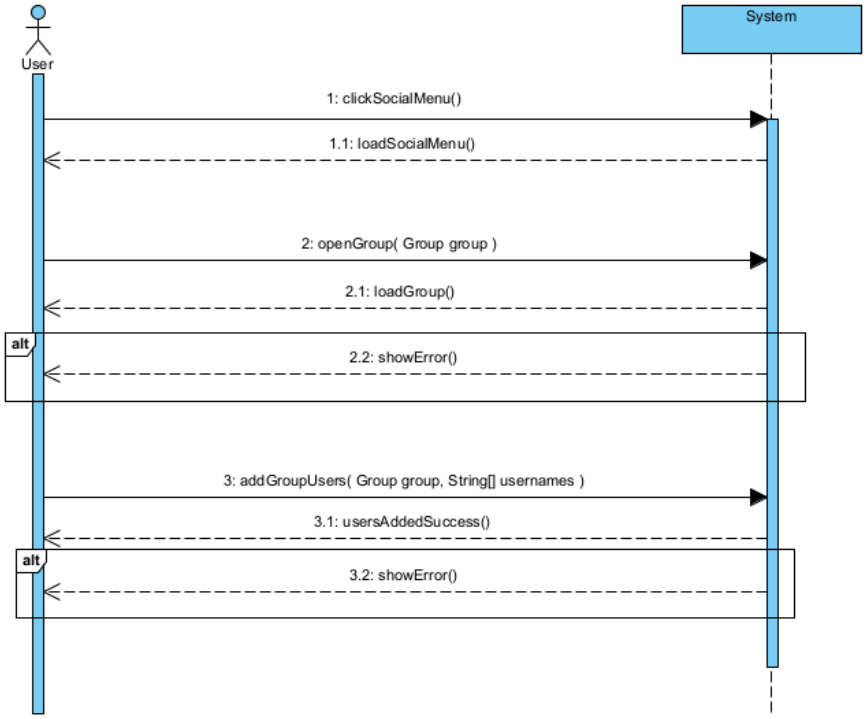
acceptFriendRequest:

Operation: submitFriendRequest(String username)
CrossReferences: none
Precondition: user has clicked show incoming friend requests button
Postcondition: the system will record the friendship between the two users

rejectFriendRequest:
Operation: rejectFriendRequest(String username)
CrossReferences: none
Precondition: user has clicked show incoming friend requests button
Postcondition: none

3. Fully Dressed: Adding Users to Group
**ID** UC Add Group Users
**Scope** Social
**Level** Groups
**Stakeholders and Interests**
General User (user): someone who wants to connect with other people virtually
System Maintainer: person responsible for making sure that the social menus work
**Precondition:** User profile has already been created. User has received a friend request.
**Postcondition:** none
**Main Success Scenario:**
1. User wants to accept friend request
2. User clicks on “Social” menu
3. User navigates to “Groups” section
4. User clicks on the Group they want to add users to
5. User clicks on “Add members”
6. User selects users from their friends list to add to the group and clicks submit
7. System adds those users to the group in the data base
**Alternate Paths:**
a. System does not respond
    1. An error message is displayed to the user asking them to refresh the app
3a. The user’s groups list could not be loaded
    1. An error message is displayed that there was an error loading their groups and to reload the app to try again
5a. The user rejects the friend request
    1. The system deletes the request
7a. The system does not receive the request
    1. An error message is displayed telling the user to try again

**SSD:**




**Operation Contracts:**

clickSocialMenu:
Operation: clickSocialMenu()
CrossReferences: None
Precondition: The profile has been created and the user is on the dashboard
Postcondition: none

openGroup:
Operation: openGroup( Group group )
CrossReferences: none
Precondition: user is already in the social menu
Postcondition: none

clickAddUsers:
Operation: clickAddUsers()
CrossReferences: None
Precondition: the user is in the group menu
Postcondition: none

addGroupUsers:
Operation: addGroupUsers( Group group, String[] usernames )
CrossReferences: none

Precondition: user is already in the group menu and has input as many of their friends that are not in group as they want
Postcondition: it will be recorded that each of the users in usernames will be added to the group

**Dannis Wu:**
1. Full-dressed: Create a Self-Paced Exercise Plan

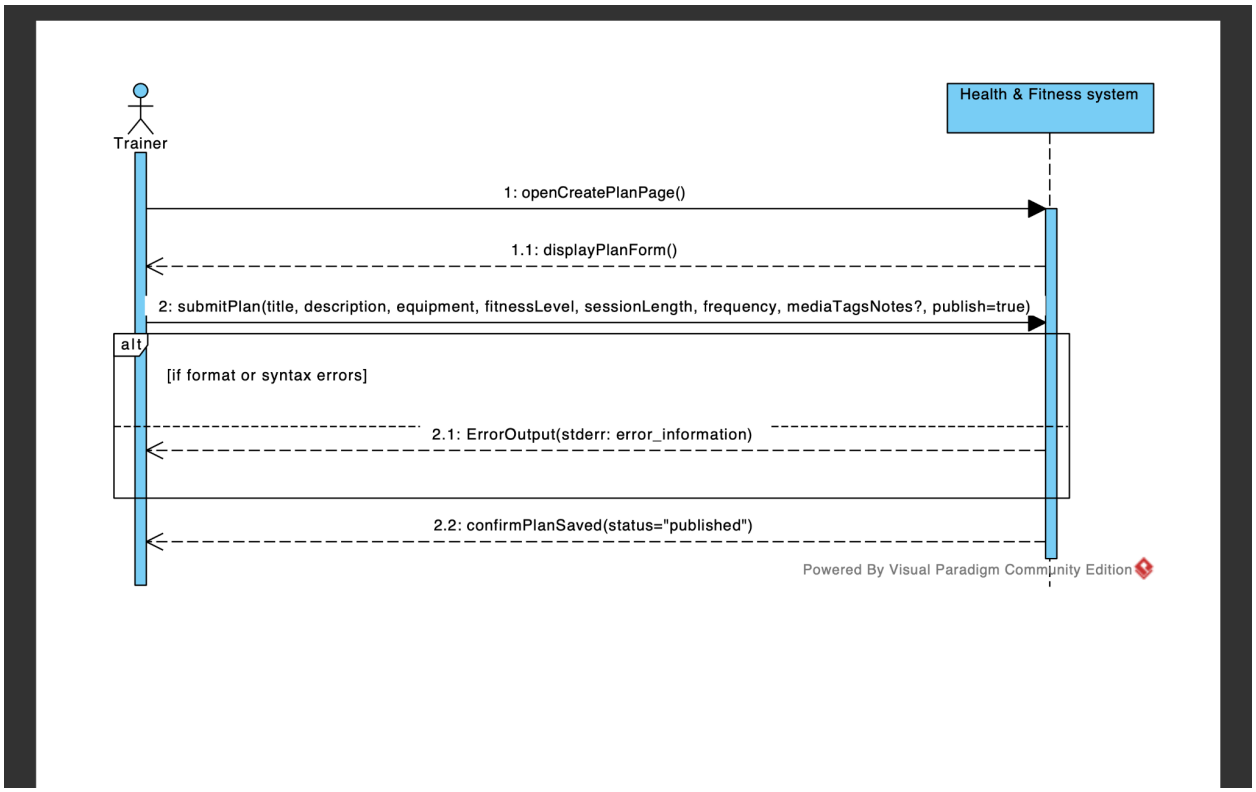
**ID** UC Create a Self-paced Plan
**Scope** Health & Fitness system
**Level** user goal
**Stakeholders and interests**
Customer (General user) — person that uses plans to exercise safely.
Trainer — person who creates and manages exercise plans/classes.
Admin — person checking content follows policy.
System maintainer — person responsible for application run.

**Precondition:** Trainer is authenticated.
**Postcondition:** Plan is saved (draft or published).

**Main success scenario:**

1. trainer wants to create a self-paced plan.
2. trainer opens **Create plan** page.
3. system shows plan form.
4. trainer fills basic info: title and description.
5. trainer selects required equipment.
6. trainer selects recommended fitness level.
7. trainer enters average session length.
8. trainer enters suggested frequency (e.g., 3 times per week).
9. (optional) trainer adds media/tags/safety notes.
10. trainer chooses **Publish**.
11. system validates all fields.
12. system saves plan and confirms success.

**Alternate paths:**
a.* anytime system does not respond — system autosaves draft; trainer will try again.
3.a form cannot load — system shows error; trainer will refresh or come back later.
11.a some required field is empty or invalid — system highlights fields; trainer fixes and goes to step 10.
10.a trainer chooses **Save as draft** — system saves draft; plan is not visible to users.
11.b duplicate title detected — system suggests rename; trainer picks a new title and goes to step 10.
11.c policy keywords found — system marks **Pending review**; trainer is notified.
12.a network failure on save — system keeps local draft and shows warning.




**Contract CO1: createSelfPacedPlan**

**Operation:**
createSelfPacedPlan

**Pre-condition:**
Trainer is logged in and has opened the create plan page.

**Post-conditions:**

- A new Plan was created.

- The Plan was linked to the Trainer.
- The Plan title was set.
- The Plan description was set.
- The Plan equipment was set.
- The Plan fitness level was set.
- The Plan session length was set.
- The Plan frequency was set.
- The Plan notes or media were set if provided.
- The Plan status was set to Published if valid, or Draft if saved as draft.
- If there are errors, the system shows a message and does not save.

2. Fully-dressed: Create and Register a Scheduled Class

**ID** UC Create and Register a Scheduled Class
**Scope** Health & Fitness system
**Level** user goal
**Stakeholders and interests**
Customer (General user) — person that wants clear time, level, prerequisites, and seats.
Trainer — person who schedules, hosts, and manages the class.
Admin — person ensuring limits and policies are respected.
System maintainer — person responsible for application run.

**Precondition:** Trainer is authenticated.
**Postcondition:** Class is saved (draft or published) with schedule and capacity.

**Main success scenario:**

1. trainer wants to create a scheduled class.
2. trainer opens **Create class** page.
3. system shows class form.
4. (optional) trainer links an existing plan as curriculum.
5. trainer fills title and description.
6. trainer sets date/time, days of week, session length, and number of weeks.

7. trainer selects delivery mode (in-person location or online link).

8. trainer sets enrollment limit (max participants).

9. trainer sets recommended fitness level and prerequisites (recommended or required).

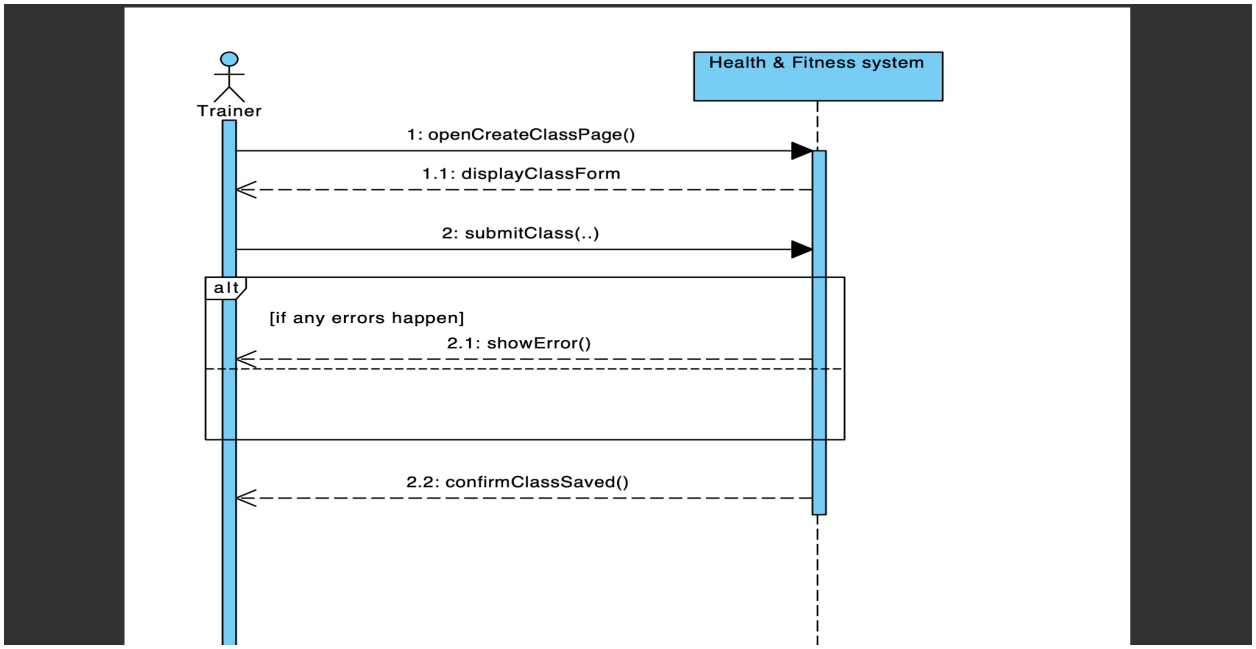
10. trainer lists required equipment.

11. trainer chooses **Publish**.

12. system validates fields and checks for schedule conflict with trainer's other classes.

13. system saves class (and recurrence), opens registration, and confirms success.

**Alternate paths:**
a.* anytime system does not respond — system autosaves draft; trainer will try again.
7.a online link cannot be generated — system asks trainer to paste a link; continue to step 11.
8.a enrollment limit missing — system applies default; trainer may edit before step 11.
9.a prerequisites marked "required" but empty — system blocks publish; trainer fills and goes to step 11.
12.a conflict with another class — system shows conflict list; trainer edits schedule and returns to step 11.
11.a trainer chooses **Save as draft** — class saved as draft; not visible for registration.
12.b policy issue detected — class set to **Pending review**; trainer notified.
13.a storage/network error — keep local draft and show warning.



**Contract CO2: submitClass**

**Operation:**
submitClass

**Pre-condition:**
Trainer is logged in and has opened the Create Class page.

**Post-conditions:**

- A new Class was created.
- The Class was linked to the Trainer.
- The Class title and description were set.
- The schedule (date, time, days, weeks) was set.
- The delivery mode (in-person or online) was set.
- The enrollment limit was set.
- The fitness level and prerequisites were set.
- The required equipment was stored.
- If publish was chosen and validation passed, Class status was Published.
- If trainer chose Save as Draft, Class status was Draft.
- If errors happen, the system shows an error and does not save.

3. Full-dressed: Host a Scheduled Class Session

**ID** UC Host a Scheduled Class Session
**Scope** Health & Fitness system
**Level** user goal

**Stakeholders and interests**
Customer (General user) — person attending a class, wants fair admission and accurate attendance tracking.
Trainer — person hosting the class, wants to run session smoothly, admit participants, and track attendance.
Admin — person monitoring policy compliance and participant disputes.
System maintainer — person responsible for system uptime and correct roster/attendance functions.

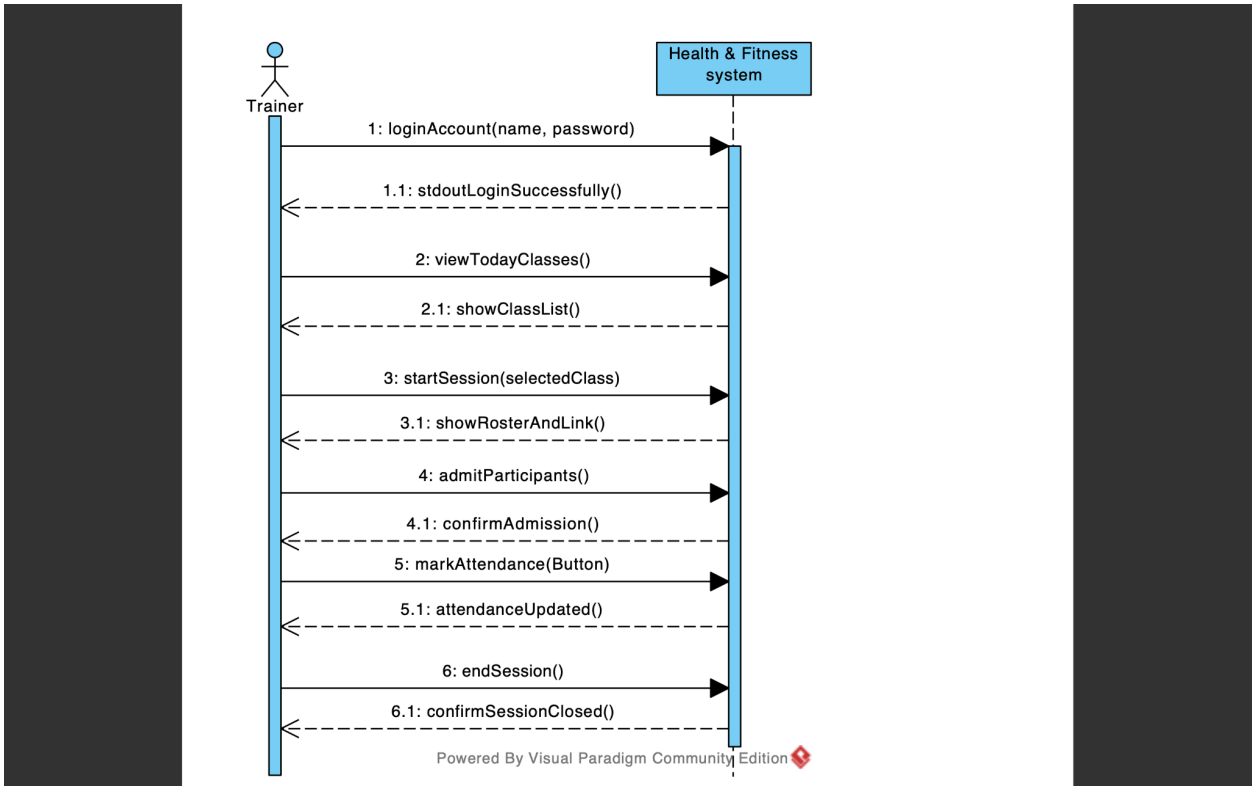
**Precondition:** Trainer is authenticated and has an active scheduled class today.
**Postcondition:** Session is closed; attendance, duration, and notes are stored; statistics are updated.

**Main success scenario:**

1. trainer logs in to the system.
2. trainer opens Today's Classes page.

3. system displays the trainer's scheduled classes for the day.
4. trainer selects the desired class.
5. trainer presses Start Session.
6. system opens the roster, attendance controls, and if online, shows the meeting link.
7. participants join the class.
8. trainer checks registration and prerequisites for each participant.
9. system admits eligible participants and shows confirmation.
10. trainer marks attendance as participants join.
11. late participants may join if seats remain; trainer admits them.
12. trainer runs the class session.
13. when the workout ends, trainer presses End Session.
14. system saves attendance, duration, and optional notes.
15. system updates class statistics and closes the session.

**Alternate paths:**
a.* anytime system does not respond — trainer may retry; if failure persists, session may be canceled.
6.a meeting link fails to generate — trainer pastes external link manually; continue to step 7.
9.a participant not registered or missing required prerequisites — system denies admission and shows message.
11.a class is full — late participant placed on waitlist for this session.
12.a no participants join within grace period — trainer cancels session; system notifies registered users.
13.a trainer cancels mid-session due to technical issues — system notifies users and logs partial session.
14.a storage/network failure — attendance and notes kept locally until sync succeeds.




**Contract CO3: hostScheduledClassSession**

**Operation:**
hostScheduledClassSession

**Pre-condition:**
Trainer is logged in and has selected a scheduled class for today.

**Post-conditions:**

- The session was opened for that class.
- The roster and, if online, the meeting link were shown.
- Eligible participants were admitted; ineligible were not (or waitlisted).
- Attendance marks were saved.
- The session was ended and marked closed.
- Duration and optional notes were stored.
- Class statistics were updated.

- **Connor Griffin**

  1. Full-Dressed: Log Workout Session

  ID: UC Log a Workout Session

  Scope: Health and Fitness system

  Level: user goal

  Stakeholders and Interests:

  User: wants quick, accurate logging and immediate dashboard updates.

  Admin: needs data integrity and auditability

  System maintain: needs reliable storage and aggregation

  Precondition: User is authenticated and has access to the workout page

  Postcondition: Workout is saved, dashboard(weekly minutes, calories, goal progress) is updated

  Main success scenario:

  1. user opens the Workout page.
  2. system displays the workout form (type, duration, calories, optional notes, default date/time).
  3. user selects workout type (cardio, strength, yoga, or custom).
  4. user enters duration and calories; optionally notes.
  5. user clicks Save Workout.
  6. system validates required fields and numeric ranges.
  7. system stores the workout and updates dashboard aggregates.
  8. system shows confirmation and refreshed dashboard summary.

  Alternate paths:

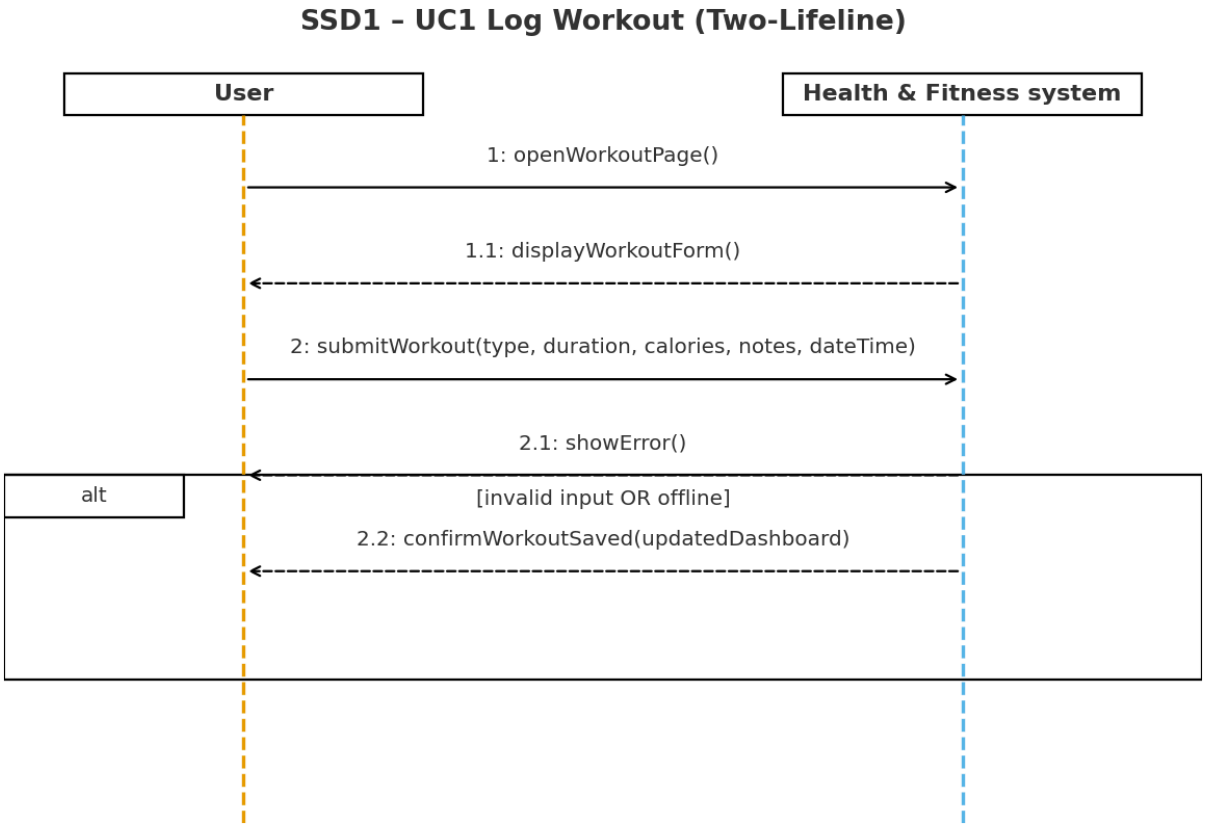
  a.* any time the system is unreachable — show connection error; allow retry or offline queue.

  3.a user leaves workout type empty — system highlights missing field; user fills it and continues.

  6.a invalid values (negative/too large) — system shows validation error; user corrects.

  7.a possible duplicate within short window — system warns; user confirms or cancels

.




Contract OC1: logWorkout
Operation:
logWorkout(userId, workoutId, durationmin, calories, logDateTime)
Pre-conditions:
• userId corresponds to an authenticated user.
• workoutId refers to an existing workout template owned by userId
• durationMin ≥ 0, calories ≥ 0.
Post-conditions (Success):

- A new WorkoutLog is created and persisted with
  - Email = userid
  - Reference to workoutid
  - Duration = durationmin
  - Calories = calories
  - Time_current = logDateTime
  - Finish marked as completed
- User's stats / dashboard aggregates are recalculated or flagged for recalculation
- A confirmation is returned containing the logged workout summary

Exceptions

- If workout id does not exist or does not belong to userid, an error is reported and no log is created
- If any db error occurs during persistence, an error message is returned and no log is created
- If validation fails, a validation error is returned and no log is created

-

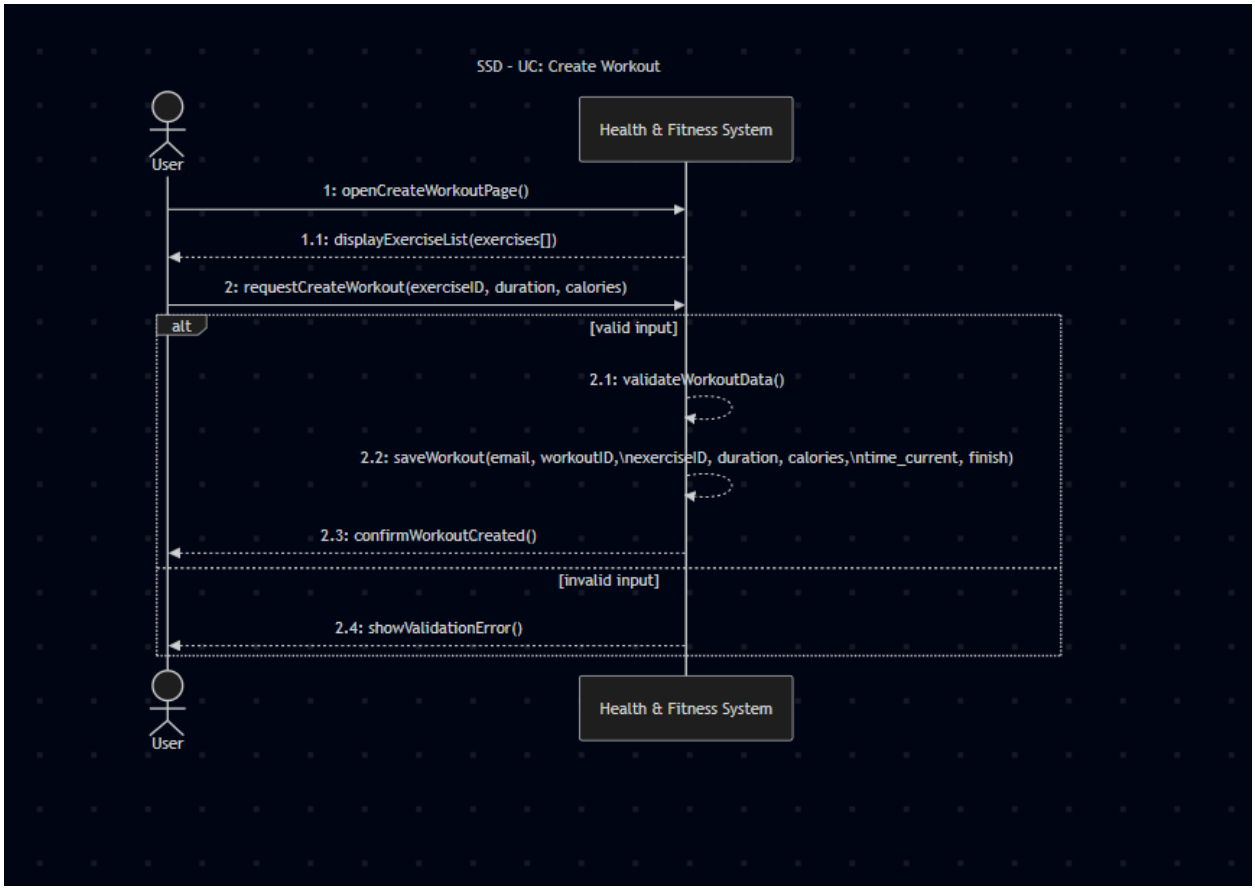
2. Full-Dressed: Create Workout

ID: UC Join Plan
Scope: Health & Fitness system
Level: user goal
Stakeholders and interests:
User: wants to create a reusable workout that combines exercise and parameters
Trainer: Wants users to follow structured workouts .
Admin: Ensures workouts are stored correctly and consistently
Precondition:
- User is authenticated, and at least one exercise exists in the user's exercise library

Postcondition:
- A new workout is created and saved in the system
- The workout becomes available for logging and reuse

Main success scenario:
1. user navigates to Create Workout page
2. System displays workout creation form and available exercises
3.User selects an exercise
4. User enters workout duration and estimated calories burned
5. User confirms workout creation
6. System validates input values .
7. System stores the workout in the database
8. System confirms successful workout creation

Alternate Paths:
- 4a: user enters invalid duration or calories
    - System displays validation error and requests correction
- 7a: Database write fails
    - System displays error message and aborts creation

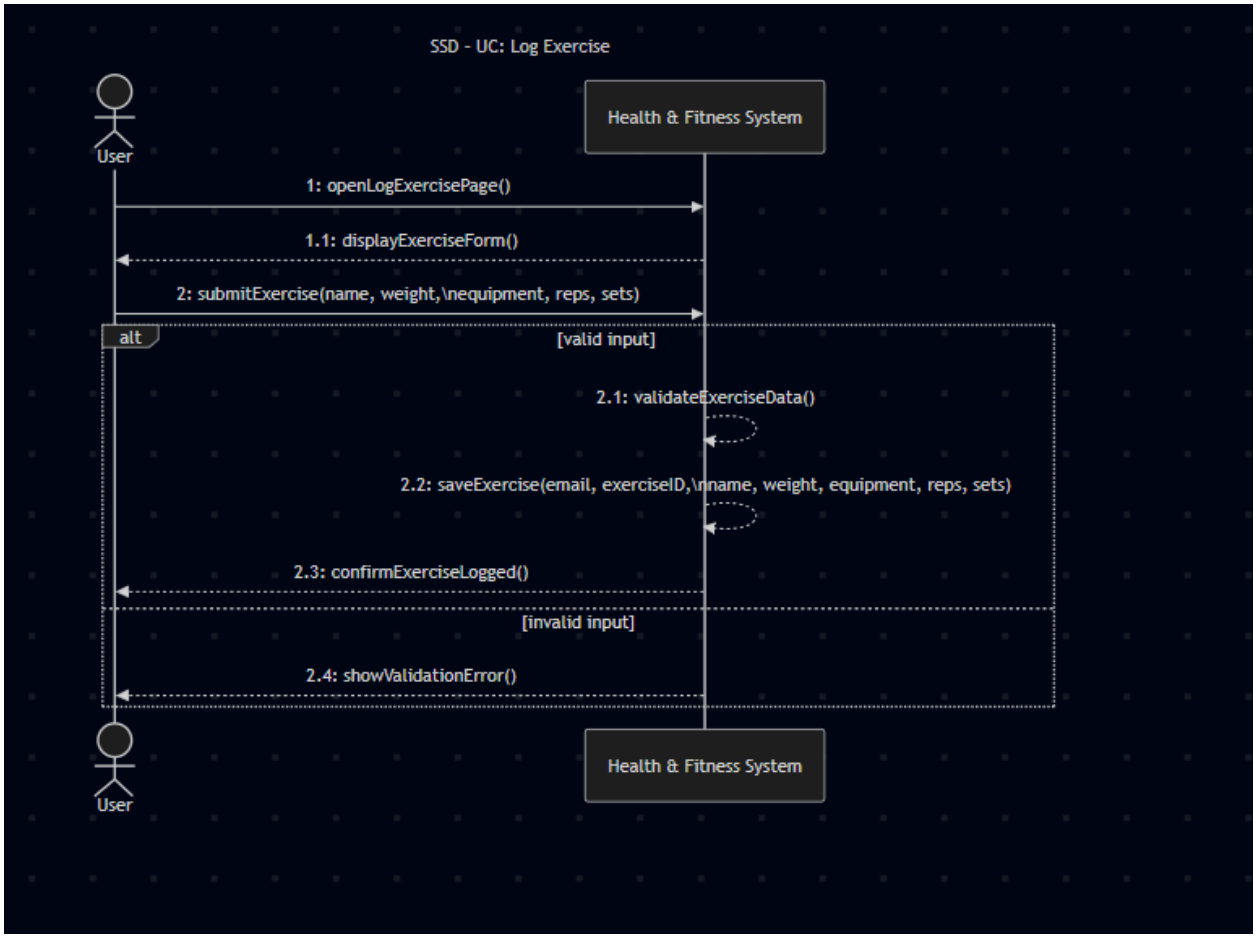



Contract OC2: createWorkout
Operation:
createWorkout(userId, type, durationMin, calories, notes, dateTime)
Pre-conditions:
• userId corresponds to an authenticated user.
• type ∈ {cardio, strength, yoga, custom}.
• durationMin ≥ 0, calories ≥ 0. Post-conditions (Success):
• A new Workout is created and persisted.
• Dashboard aggregates for the user are recalculated (or flagged for recalculation).
• Confirmation is returned with workoutId.
Exceptions:
• ValidationError for missing/invalid fields.
• DuplicateWarning if similar workout exists within a brief threshold window (optional).

3. Full-Dressed: Log Exercise

ID: UC Log Exercise
Scope: Health & Fitness system
Level: User goal
Stakeholders and interests:
User: wants to track individual exercises accurately
Admin: Wants insight into client exercise performance
System maintainer: Ensures correct exercise records
Precondition: User is authenticated
Postcondition: Exercise entry is saved and associated with the user
Main success scenario:
1. User navigates to add exercise page
2. System displays exercise input form
3. User enters exercise name, sets, reps, weight, and equipment
4. User submits the exercise
5. System validates required fields
6. System stores exercise in the database
7. System confirms exercise creation
Alternate paths:
3.a: Required fields missing
- System highlights missing fields and prompts correction

6.a: Database error
- System displays error message and aborts save




OC: Log Exercise
Operation: LogExercise(userid, exerciseName, weightLb, sets, reps, type, logDateTime)

Preconditions:

- User is authenticated
- Exercise name is non empty
- Weightlb >0, sets >0, reps >0
- logDateTime is valid date time

Postconditions:

- A new exercise record is created and persisted with:
    - Email = userID
    - Name = exercise name
    - Weight_pounds = weightLb
    - Sets, reps
    - Type
    - Time_current = LogDateTime

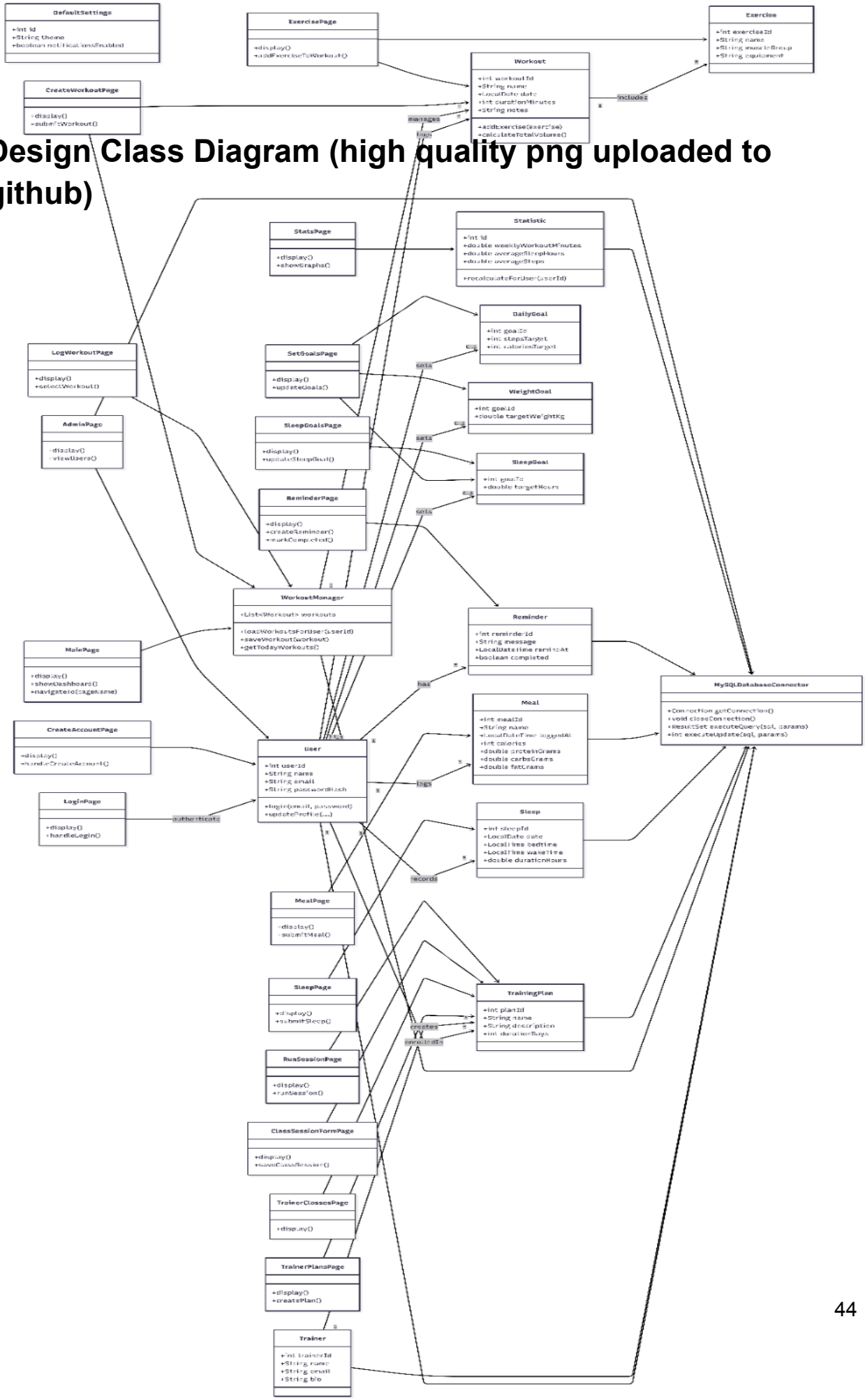
- User’s strength related stats are recalculated or flagged for recalculation
- Confirmation is returned with the new exerciseID or logged exercise summary

Exceptions:

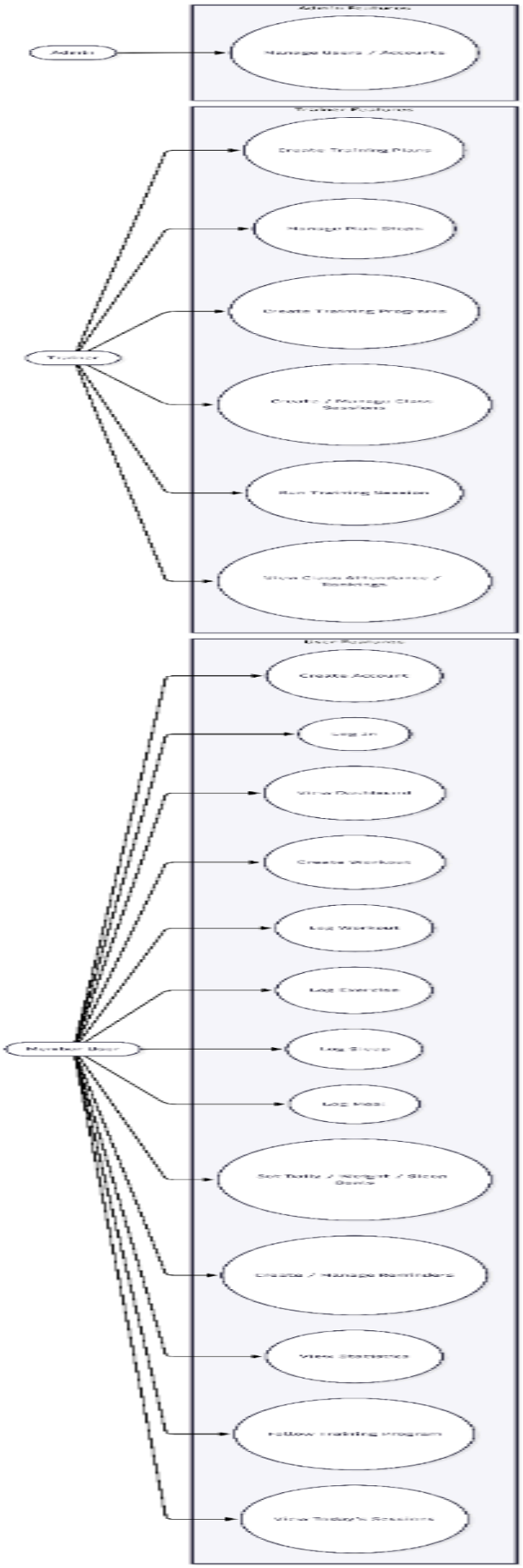
- If mandatory fields are missing/ invalid, a validation error is shown and no exercise is logged
- If a DB error occurs while inserting the exercise, an error

**Hours Worked:**
Jack - 45 hours
Ethan - 1 hour
Dannis - 35 hours
Savannah - 5 hours
Mason - 25 hours
Connor - 45 hours

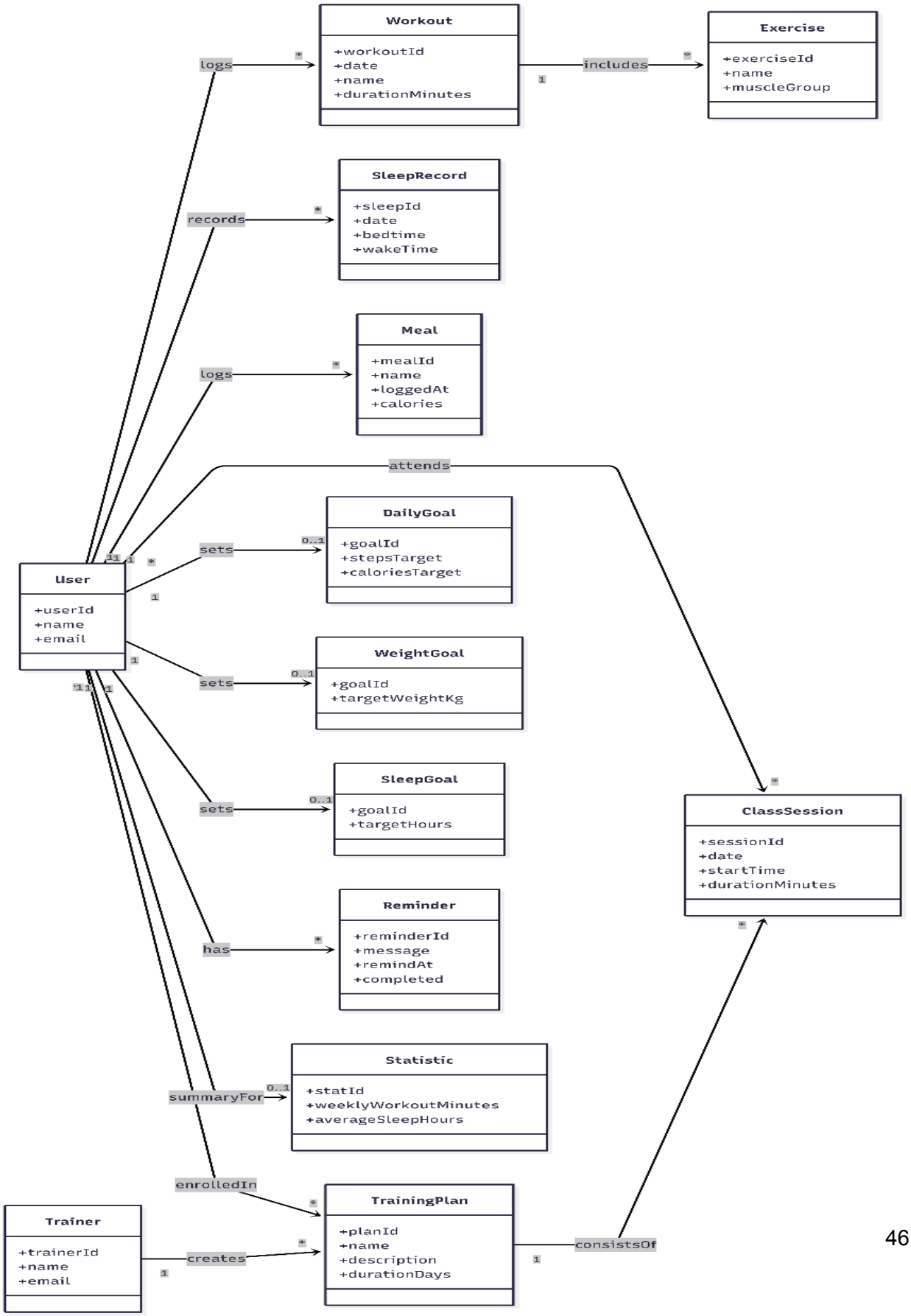
# Design Class Diagram (high quality png uploaded to github)



## Use Case Diagram (high quality png in github repo)



# Domain Model (high quality version in github)



## Summary of JUNIT

For Iteration 3 we added JUnit tests to cover the core logic involved in creating and logging exercises and workouts, plus basic validation of user input and database persistence.

- Test focus:
  - Validation of exercise creation (name, sets, reps, weight)
  - Correct linkage of an exercise to a workout
  - Handling of invalid inputs (empty name, negative or zero values)
  - Proper interaction with the database layer through MySQLDatabaseConnector
- Main test classes:
  - WorkoutManagerTest - tests logExercise and saveWorkout logic
  - ExerciseTest - tests creation of Exercise objects and derived values
- Coverage Highlights:
  - At least one passing test for each normal CreatExercise scenario
  - Multiple tests for invalid input paths (missing name, non-positive sets/reps/weight)
  - Database write failures are handled by returning an error instead of crashing the UI
- Test Case template:

**Test Case Template**

| TC ID | scenerio/condition | Test inputs | Expected result |
|---|---|---|---|
| CE-01 | Create exercise w/valid data | Userid = 1, exercise name= ‘bench”, set 5, reps = 6, weight = 80 | New exercise object created linked to selected workout. Record saved to db and method returns success |
| CE-02 | Missing exercise name | Same as ce-01 with name ==”” (empty string) | Validation fails, no exercise is saved, method returns failure, UI shows “exercise name is required” |
| CE-03 | Non-positive sets | Sets == 0 | Validation fails, no |

| Test Case Template | | | |
|---|---|---|---|
| Tт TC ID | Tт scenerio/condition | Test inputs | Expected result |
| | | | exercise is saved, failure/validation error, UI shows sets must be greater than 0 |
| CE-04 | Non-positive reps | Reps == -5 | Validation fails, no exercise is saved, method returns failure/validation error, UI shows “ weight cannot be negative” |

# Installation and Set Up

- Email jack_fontenot1@baylor.edu your IP address (v4), so you can connect to the database. If your IP address is not whitelisted, you will not be able to connect to the database and the app will not work.
    - If your email receives no response, text 985-791-8555
- If the jar file on the GitHub is not working or running slow, then your IP address has not been whitelisted.
- An admin and a set up trainer login can be found at the bottom of the User Manual.

# User Manual

1. Create an account
2. Fill in fields and select the type of account desired, user or trainer (an admin login and a trainer with preloaded data is provided at the bottom of this manual)
3. Login into the created account
4. Go to the Set Goals page and set your personal health goals
5. Go to the Sleep Goals page and set your personal sleep goals
6. Exercises and workouts can be created in their respective create pages
7. Workouts can be logged in the Log Workout page, but they must be created first
8. Meals can be created and logged in the Log Meal Page
9. Classes can be viewed and joined in the Class Page
10. Personal stats can viewed and created in the Stats Page
11. Reminders can be set and viewed in the Reminders Page

**Trainer Section**

1. Programs can be managed, created, and viewed on the Trainer Programs Page.
2. Sessions scheduled for the current day can be viewed and started on the Sessions Today Page.
3. The Preview Search page allows trainers to see all classes and plans

**Admin Section**

1. All accounts can be viewed and have their passwords changed from the dropdown box in the center
2. Accounts can be created through the Create Account button

**Admin login:** admin@gmail.com, 123
**Trainer login:** xiangyu_wu2@baylor.edu, 12345

# Member contribution summary

1. Connor Griffin: responsible for all workout related use cases. Created all UIs for log workout, create workout and log exercise. Also helped create other UIs including Main page and sleep page. Submitted and worked heavily on all group project deliverables throughout the course of the project. Organized and assigned all work to group members, completed work that was not fulfilled by other group members. Helped connect logic to database for other UI functions including stats page.
2. Jack Fontenot: Responsible for database setup and github repository librarian. Also responsible for all login capabilities and the UIs for those given features. Responsible for all admin features, UIs, and database logic. Added majority of buttons and backend logic for page creation from menu bar. Created the wireframes that the project GUI was modeled upon. Created skeleton UIs for user side pages to accelerate and guide other member's productivity. Oversaw the database logic for all user functionalities.
3. Mason Baxter: Responsible for all sleep use cases and UIs. Fully implemented the sleep cases and connected logic to database. Also helped with implementation of main page and connecting back end logic for main page. Also fully implemented the reminders page and updated logic and UI for that use case.
4. Dannis Wu: Fully implemented trainer use cases and implemented and connected all back end logic for group creation for trainer as well as the UI implementation. Created the menu bar for the user side. Worked alongside Jack to create SQL tables and database logic.
5. Savannah Johnson: Made updates to stats page and weight documentation.
6. Ethan Rogers: uploaded group.java document

## Group Meeting Summary

Met multiple times and discussed roles for the project and our individual progress. We also kept in contact via an iMessage group chat which every member of the group had access to and we kept in constant communication throughout the period in discussing what had to be done as well as the expectations for all group members. In our meetings we were able to flush out different issues we were having especially with database connection and various other use cases that we found to give us trouble. The most useful tool for use was the intellij live coding feature that allowed all of us to edit code simultaneously without having to worry about constant merging and coding over each other. Overall, these meetings were helpful and allowed us to keep on track as a group and make the progress that we needed to in an efficient and straightforward manner. All group members were aware when meetings were occurring and the work that they were responsible for although attendance and effort varied as shown in contributions and github commitments.

## Bonus Features

- Retro UI (consistent throughout entire application)
- Database hosted on Google Cloud SQL
- A user can reset their password by answering their security questions, rather than having to have an admin reset their password
- A trainer can update a class before, during, and after completion