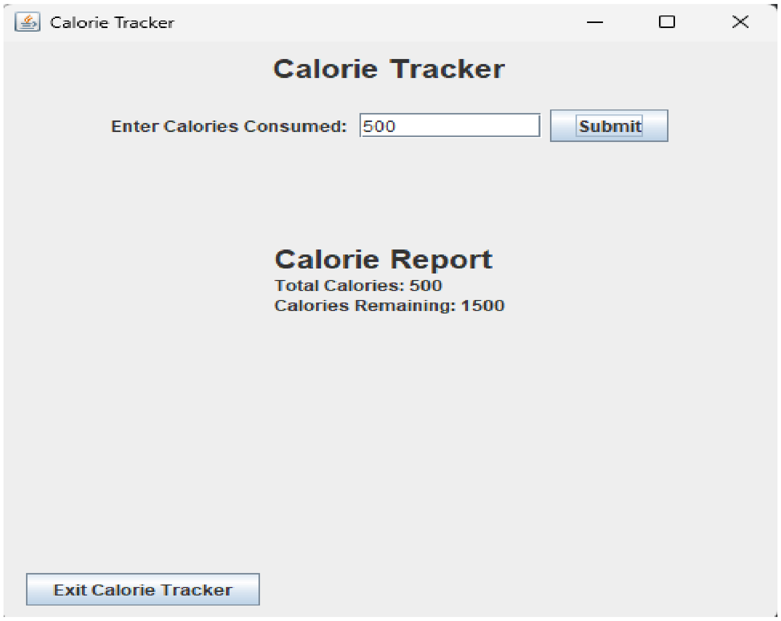
MyFitness

Implementation Status: Our group has made good progress so far in implementing our project. We have a log in page that is functional for giving access to the website as well as signing up new users. We have also implemented multiple use cases and their user interfaces. These use cases are mostly separate right now and our next steps will be joining everything together with correct functionality.

Main Roadblocks: The biggest roadblocks we are facing currently are mainly time related. We are trying to get all of our code implemented and functioning all together which is difficult to do with all group members having different schedules. We have also had some trouble figuring out how to combine code efficiently but we are working through those issues.

UIs:



Calorie Tracker

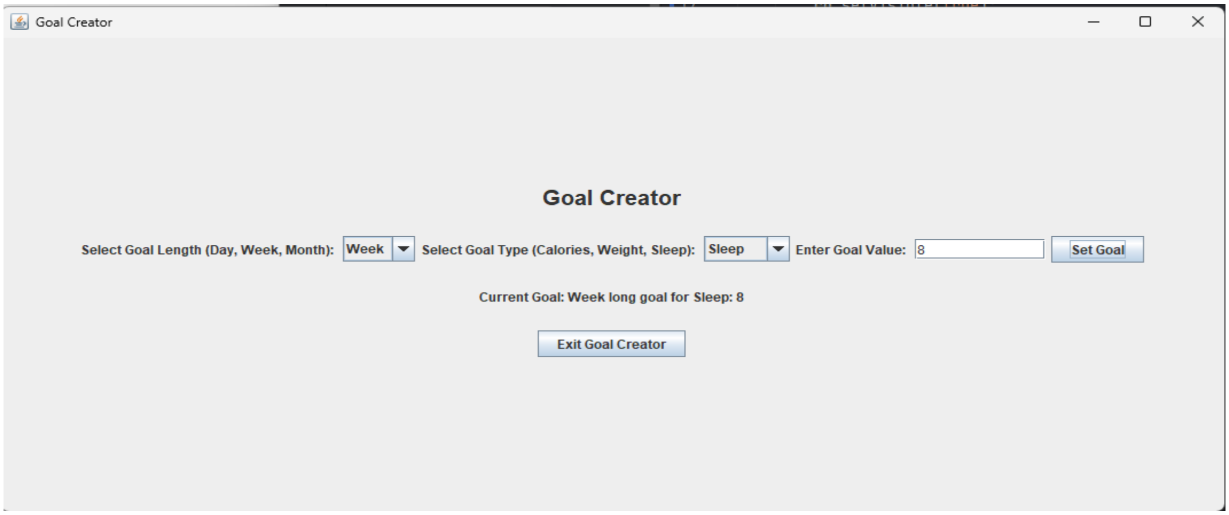
Enter Calories Consumed: 500

Calorie Report

Total Calories: 500

Calories Remaining: 1500

Tracking Calories (Use Case #7) implemented by Ryan Meador and connected with application layer.



Goal Creator

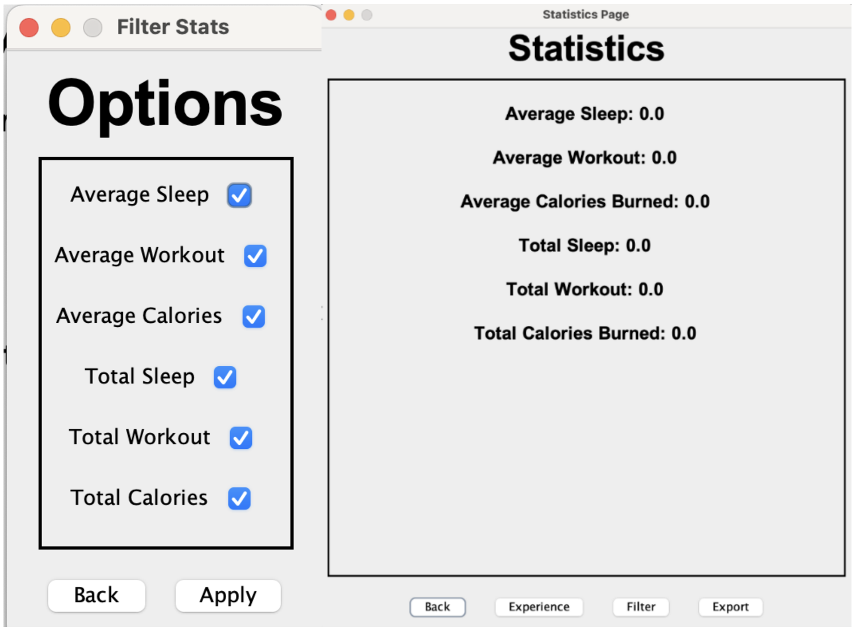
Select Goal Length (Day, Week, Month): Week

Select Goal Type (Calories, Weight, Sleep): Sleep

Enter Goal Value: 8

Current Goal: Week long goal for Sleep: 8

Setting Goals (Use Case #8) implemented by Ryan Meador and connected with application layer.



Average Sleep ☑

Average Workout ☑

Average Calories ☑

Total Sleep ☑

Total Workout ☑

Total Calories ☑

Average Sleep: 0.0

Average Workout: 0.0

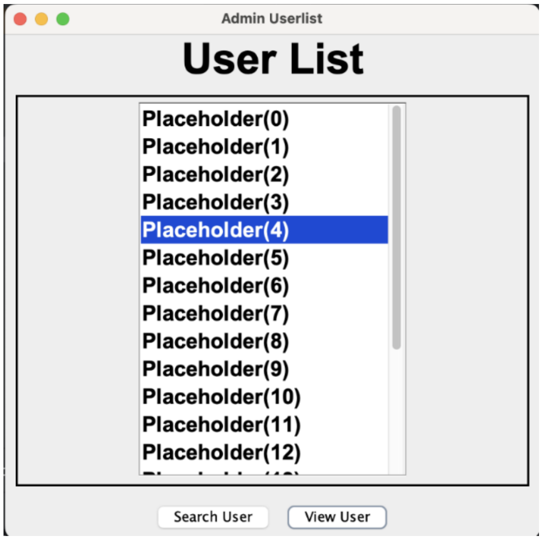
Average Calories Burned: 0.0

Total Sleep: 0.0

Total Workout: 0.0

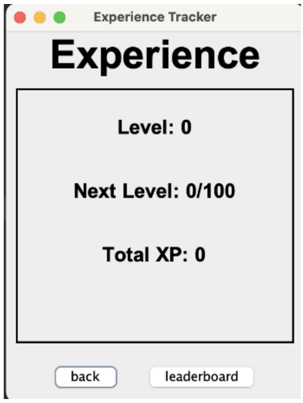
Total Calories Burned: 0.0

Statistics (Use Case #13) implemented by Larry O'Connor but not connected with application layer.



Placeholder(0)
Placeholder(1)
Placeholder(2)
Placeholder(3)
Placeholder(4)
Placeholder(5)
Placeholder(6)
Placeholder(7)
Placeholder(8)
Placeholder(9)
Placeholder(10)
Placeholder(11)
Placeholder(12)

Admin Management (Use Case #14) implemented by Larry O'Connor but not connected with application layer.

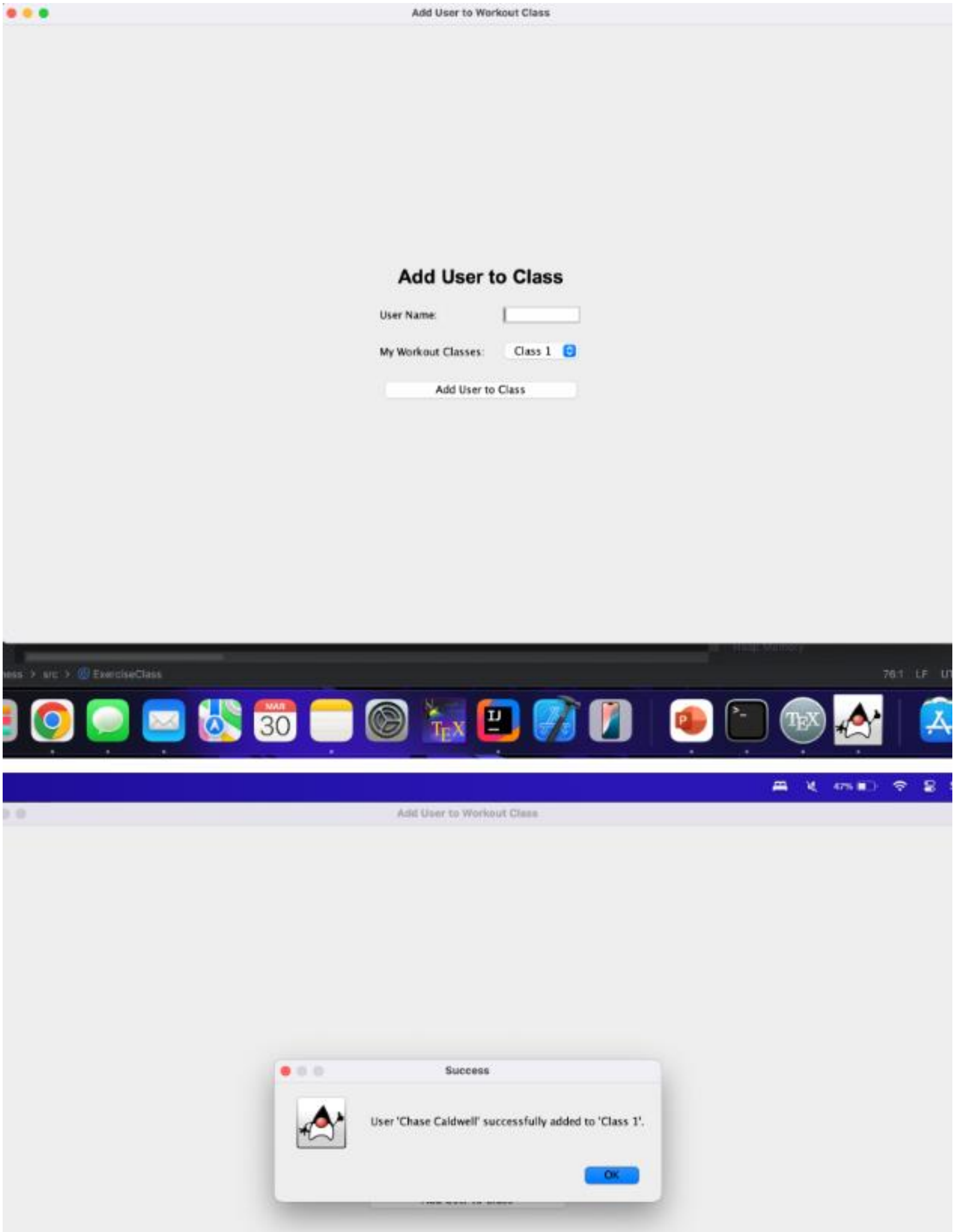


Level: 0

Next Level: 0/100

Total XP: 0

XP System (Use Case #15) implemented by Larry O'Connor but not connected with application layer.



Add User to Workout Class

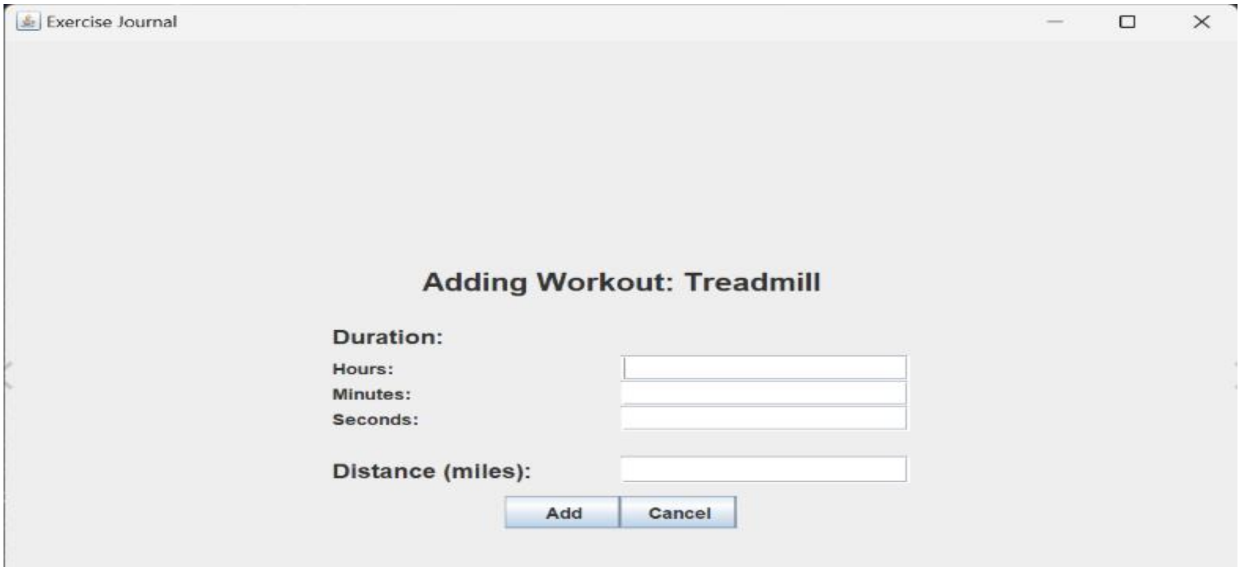
Add User to Class

User Name:

My Workout Classes: Class 1

User 'Chase Caldwell' successfully added to 'Class 1'.

Trainer Adds to Class (Use Case #4) implemented by Chase Caldwell and connected with application layer.



Exercise Journal

Adding Workout: Treadmill

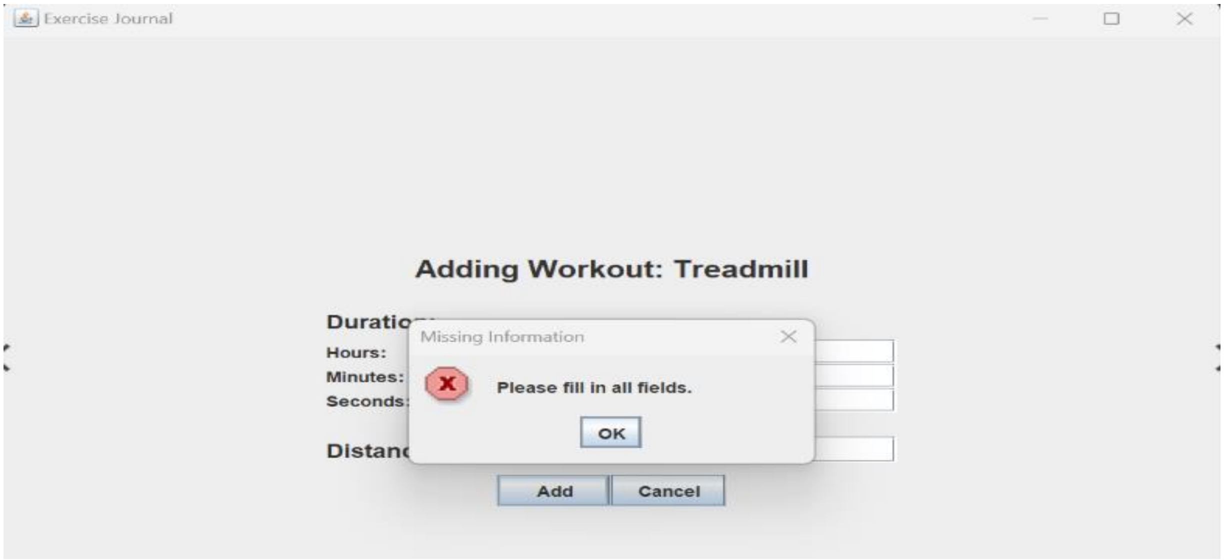
Duration:

Hours:

Minutes:

Seconds:

Distance (miles):

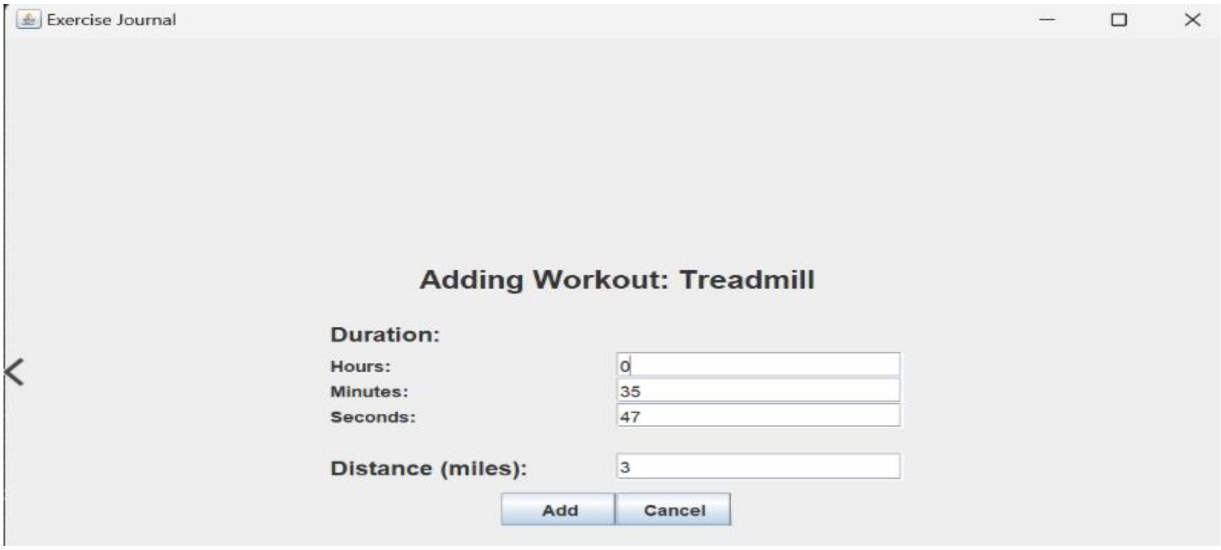


Exercise Journal

Adding Workout: Treadmill

Missing Information

Please fill in all fields.



Exercise Journal

Adding Workout: Treadmill

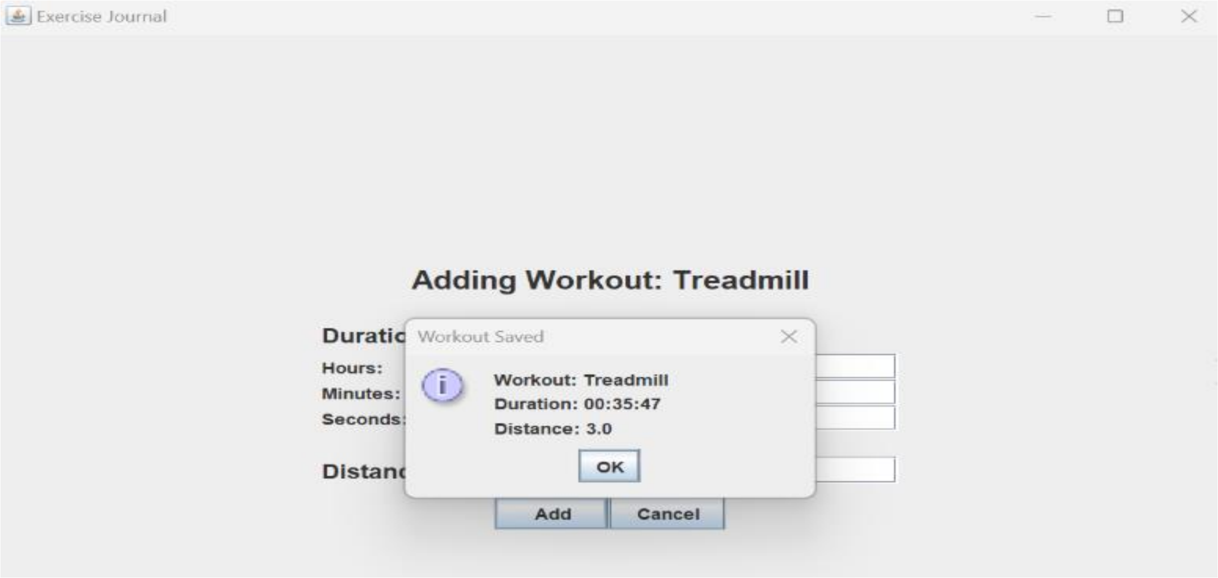
Duration:

Hours: 0

Minutes: 35

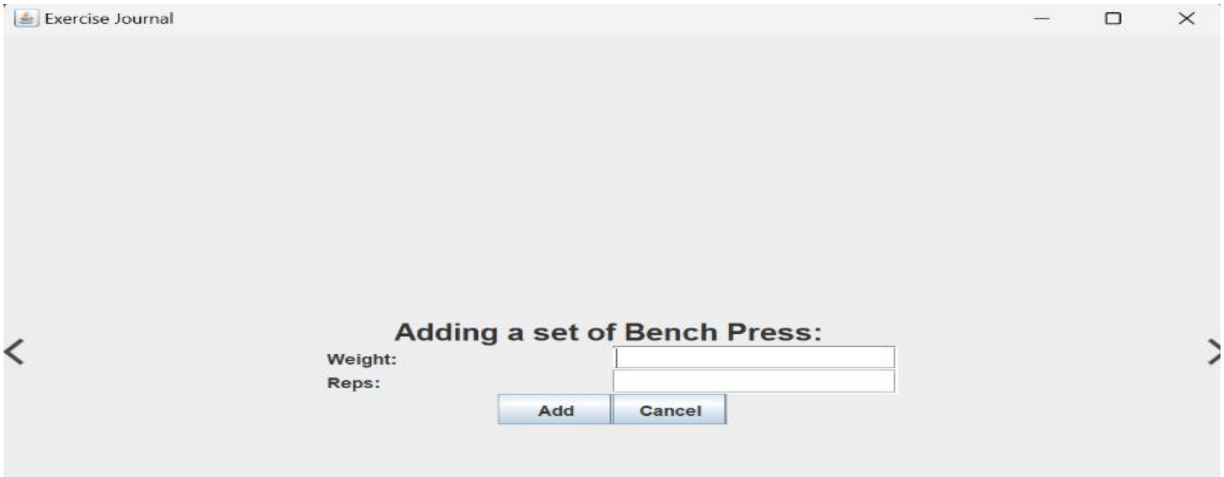
Seconds: 47

Distance (miles): 3

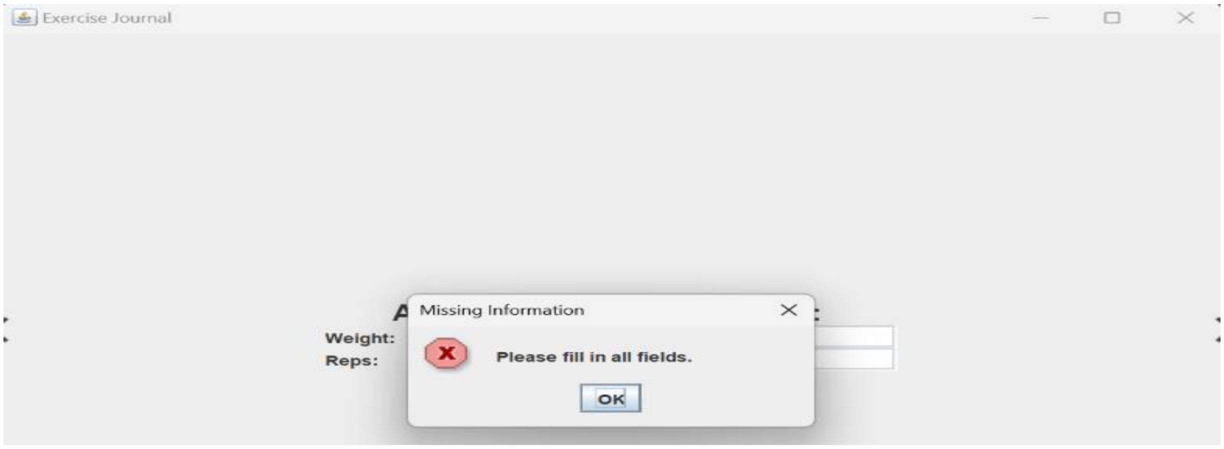


Adding Workout: Treadmill

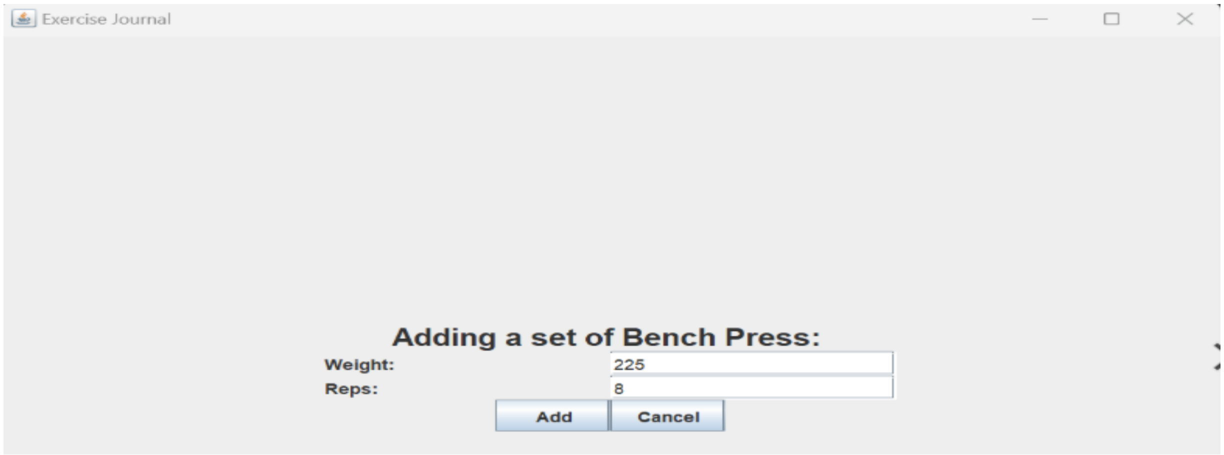
Workout: Treadmill
Duration: 00:35:47
Distance: 3.0



Adding a set of Bench Press:



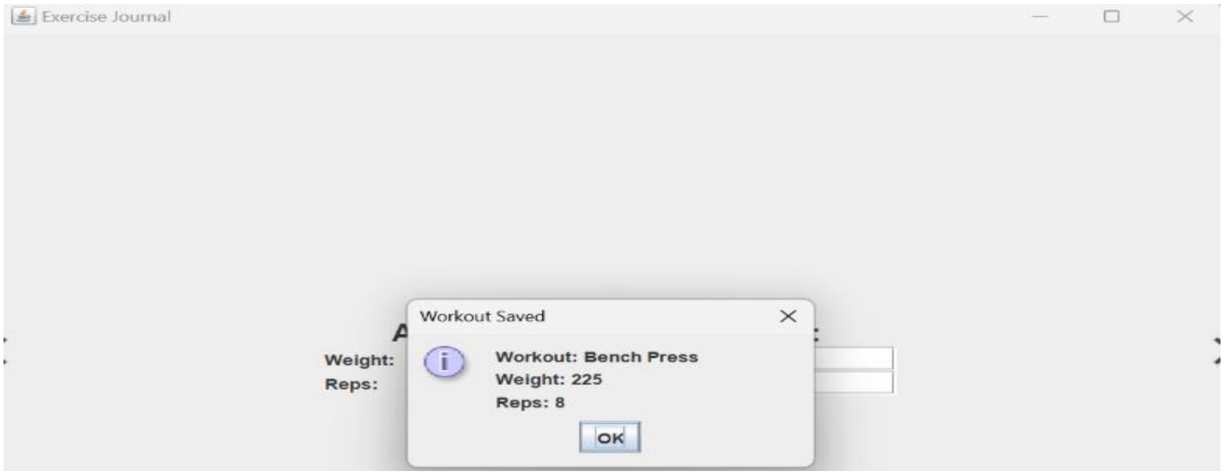
Please fill in all fields.



Adding a set of Bench Press:

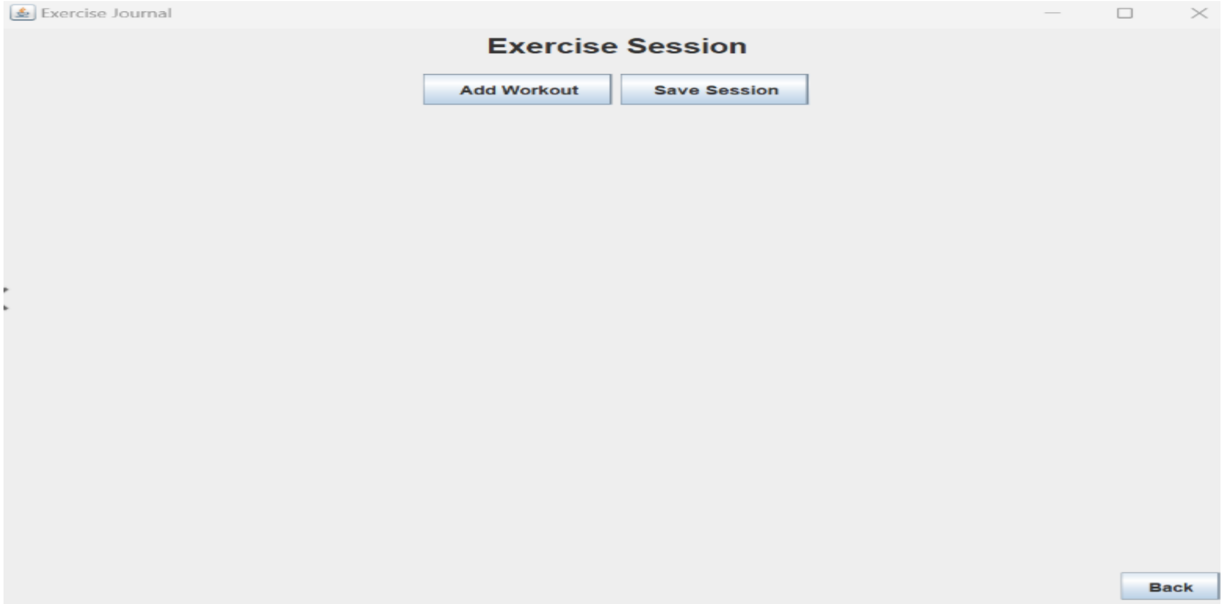
Weight: 225

Reps: 8

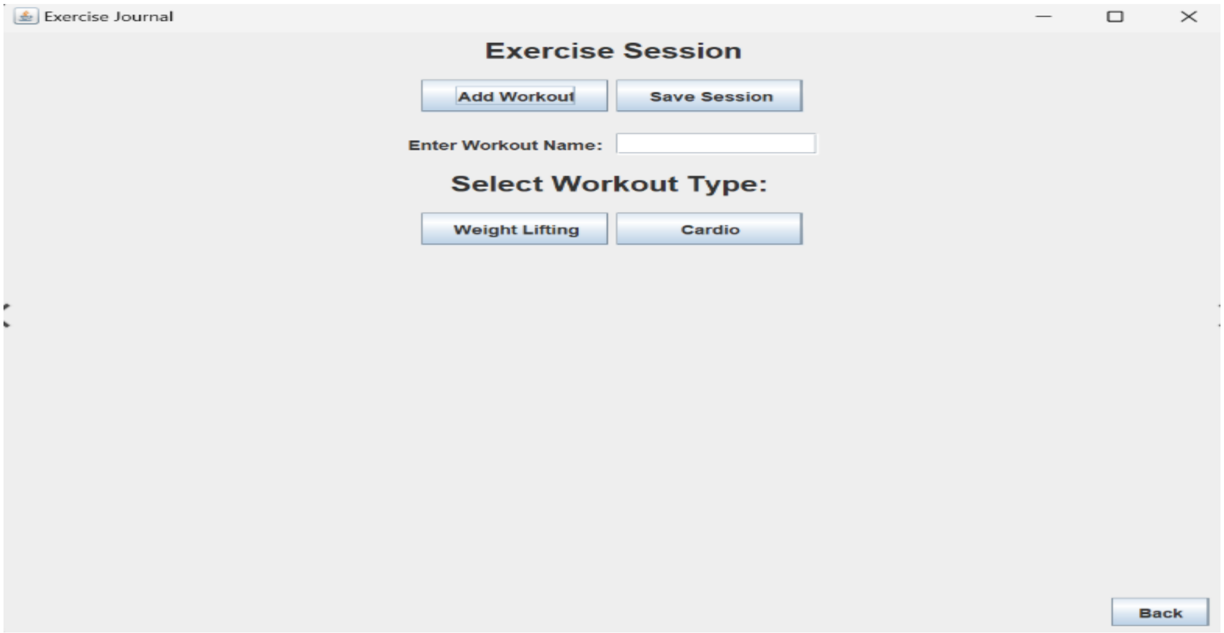


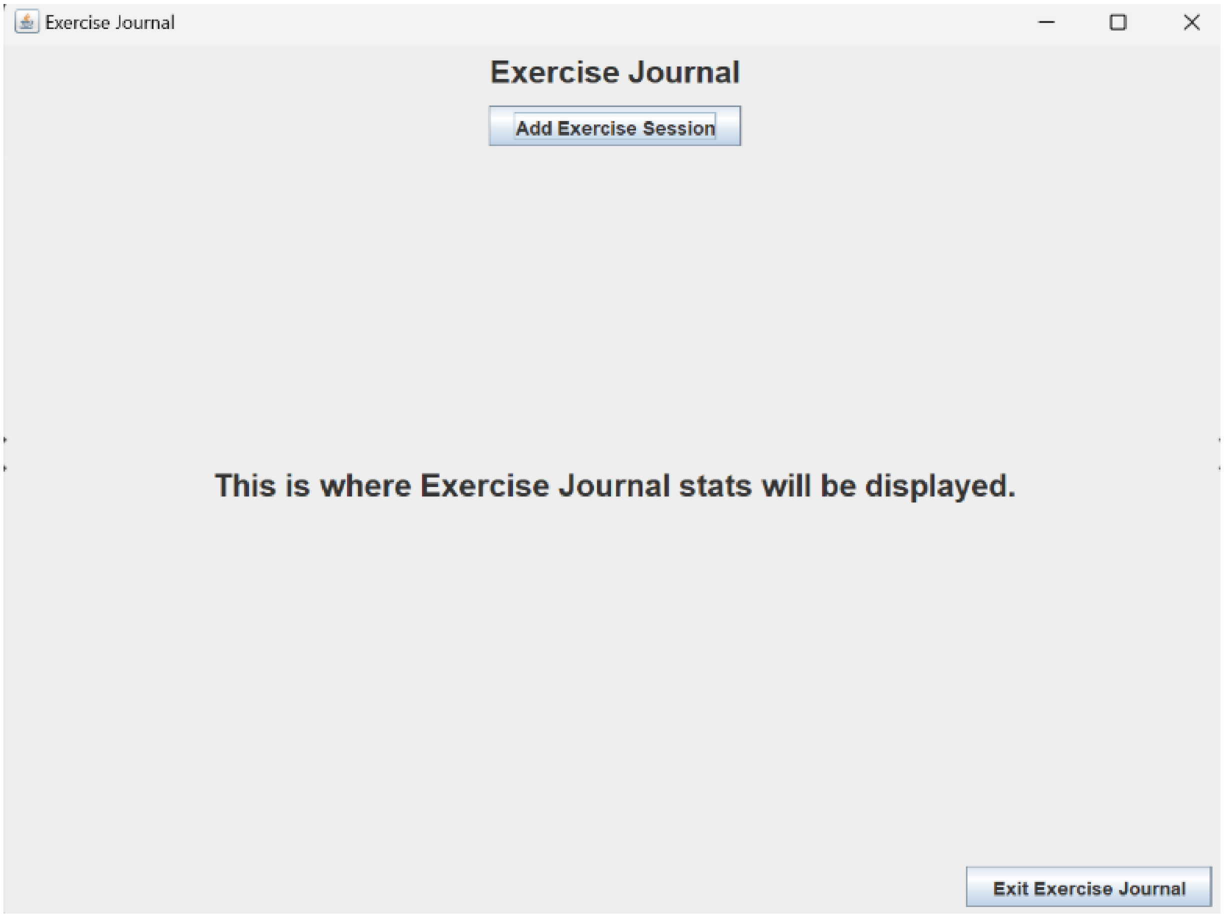
Workout Saved

Workout: Bench Press
Weight: 225
Reps: 8



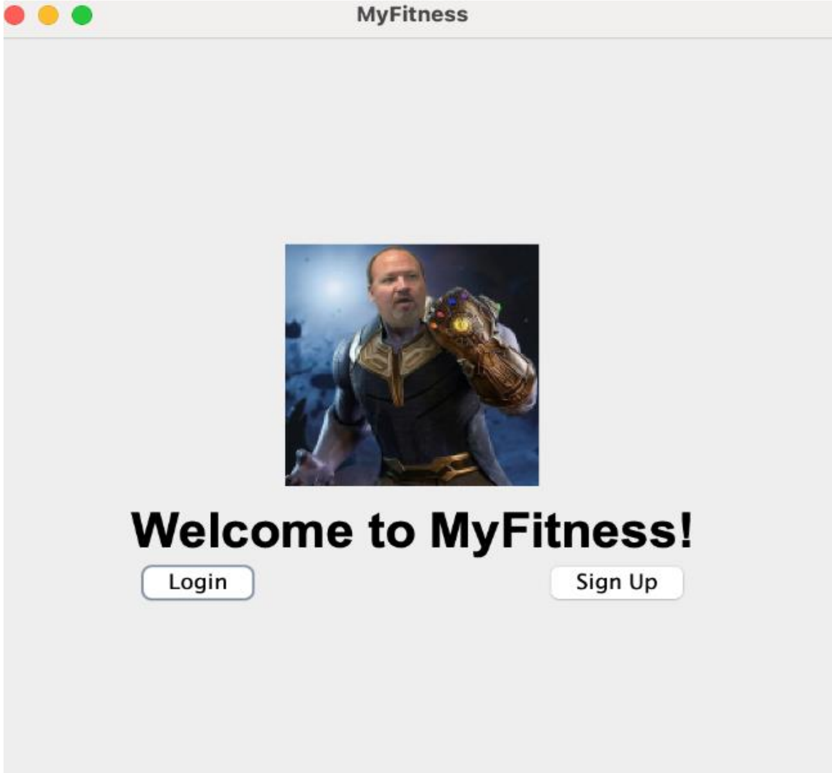
Exercise Session



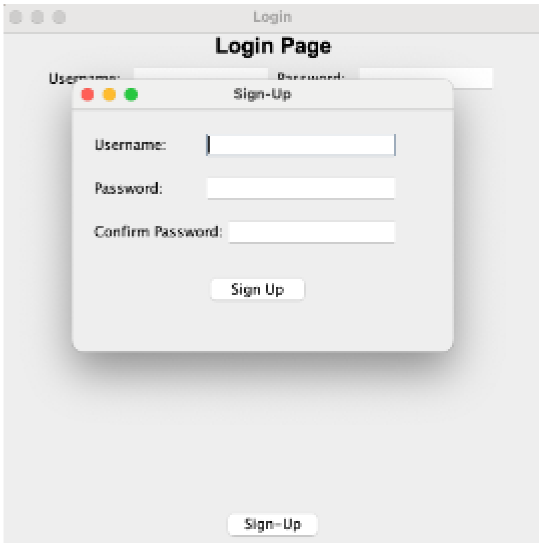


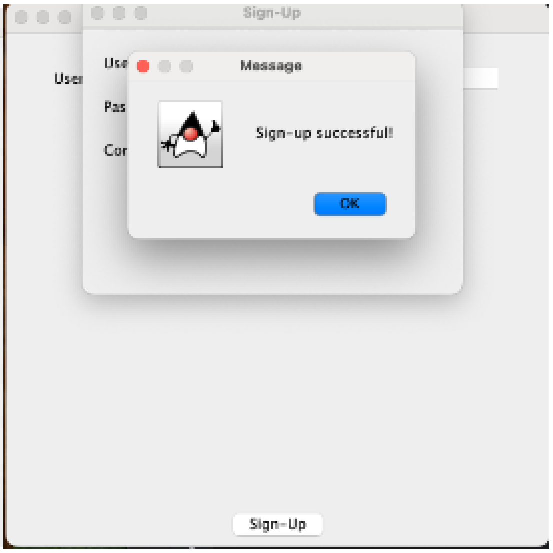
This is where Exercise Journal stats will be displayed.

Exercise Journal (Use Case #1) implemented by Jeremy Huggins and connected with application layer.



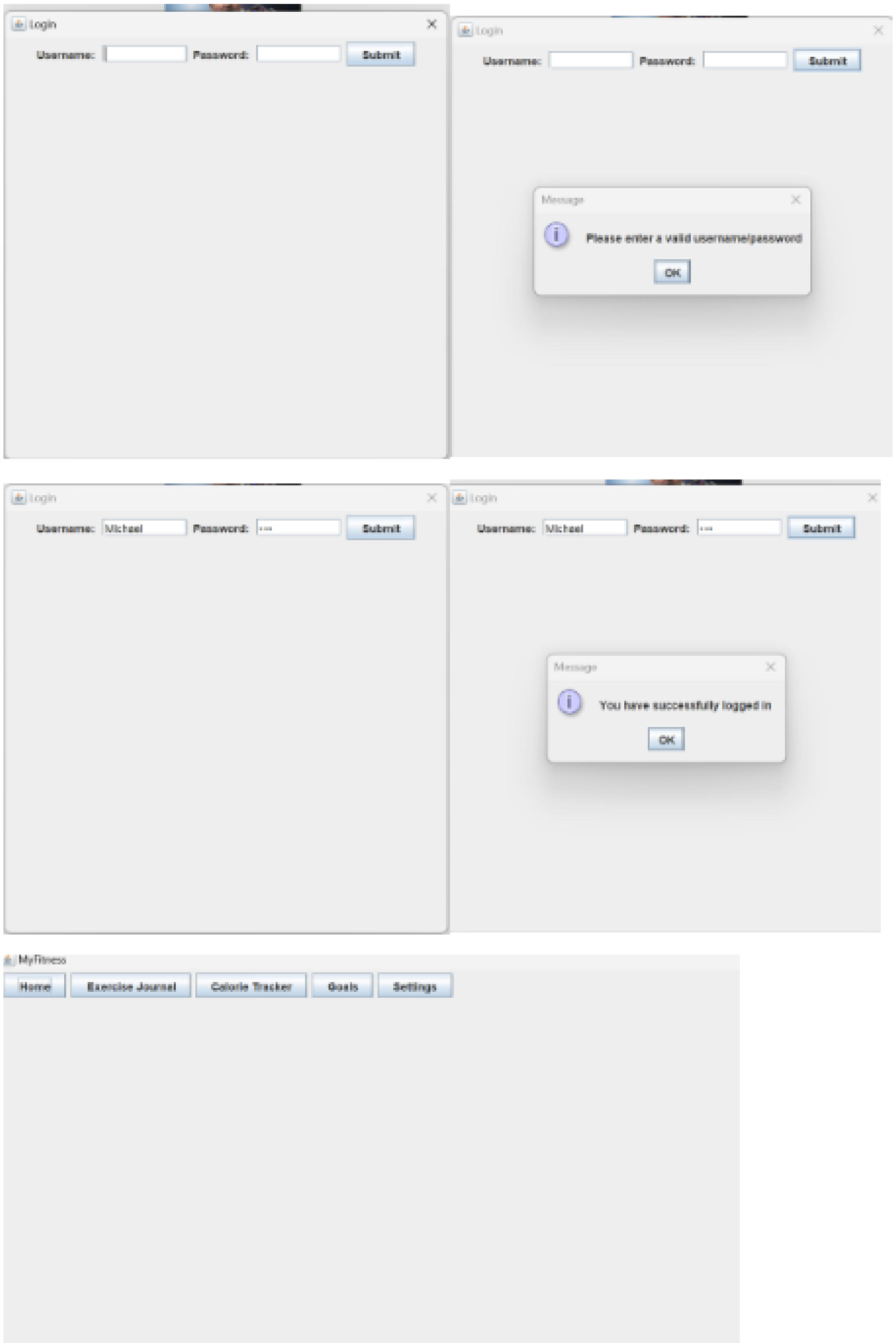
Welcome to MyFitness!





Sign-up successful!

Sign Up implemented by Dannis Wu and connected with application layer.



Please enter a valid username/password

You have successfully logged in

Log in implemented by Michael Pressman and connected with application layer

Use Cases:

Jeremy Huggins Use Cases:

1.Exercise Journal - Use Case: Fully Dressed Author: Jeremy Huggins

ID: user logs a workout in the exercise journal
Scope: Health & Fitness App - Exercise Journal
Level: user goal

Stakeholders and Interests:
General User

- person that is using the app and wants to track their workouts Trainer
- can see a general user's daily workouts and statistics (if general user is enrolled in a trainer led program) System Maintainer
- person responsible for ensuring smooth operation of the app and resolving technical issues preventing the app from running

Precondition: user has an account on the app

Postcondition: workouts are logged and analyzed for statistical information

Main Success Scenario:

1. General User wants to track their most recent or current workout session
2. Opens the Exercise Journal
3. Logs an exercise session's workouts/duration/repetitions
4. System displays the workout session, the user's progress in each workout and their goals
5. User exits the Exercise Journal

Alternate Paths:

a.* anytime the system does not respond

1. system maintainer will restart the application

3.a If the user logs incorrect exercise data

1. The user will have the option to edit the exercise session's data after it has been logged.

4.a The application fails to display information

1. The user contacts the system maintainer
2. System maintainer restarts the application

2.Exercise Planner - Use Case: Fully Dressed Author: Jeremy Huggins

ID: user plans a workout in the exercise planner
Scope: Health & Fitness App - Exercise Planner
Level: user goal

Stakeholders and Interests:
General User

- person that is using the app and wants to plan out their future workouts

Trainer

- can see a general user's workout plan (if general user is enrolled in a trainer led program)

System Maintainer

- person responsible for ensuring smooth operation of the app and resolving technical issues preventing the app from running

Precondition: user has an account on the app

Postcondition: future workouts are planned and set as goals

Main Success Scenario:

1. General User wants to plan a workout routine
2. Opens the Exercise Planner
3. Sets a workout routine with intended workouts and goals for duration/repetitions
4. System displays the exercise plan and their goals
5. User exits the Exercise Planner

Alternate Paths:

a.* anytime the system does not respond

1. system maintainer will restart the application

3.a If the user wishes to have multiple exercises plans

1. The user will have the option to save multiple exercise plans and switch between them at any time.

3.b If the user wishes to modify an existing exercise plan

1. The user will have the option to edit a workout routine
2. The user can then set that workout routine to be their main exercise plan

4.a The application fails to display information

1. The user contacts the system maintainer
2. System maintainer restarts the application

3.Body Metrics Tracker - Use Case: Fully Dressed Author: Jeremy Huggins ID: user logs and tracks changes in their body measurements
Scope: Health & Fitness App – Body Metrics Tracker
Level: user goal

Stakeholders and Interests:
General User

- person that is using the app and wants to keep track of their body measurements

Trainer

- can see a general user's body measurements (if general user is enrolled in a trainer led program)

System Maintainer

- person responsible for ensuring smooth operation of the app and resolving technical issues preventing the app from running

Precondition: user has an account on the app

Postcondition: the user has a log of their body measurements

Main Success Scenario:
1. General User wants to track their body measurements
2. Opens the Body Measurements Tracker
3. Enters a data point for the body measurements they intend to track
4. System displays the data pint and changes in their body measurements
5. User exits the Body Metrics Tracker

Alternate Paths:

a.* anytime the system does not respond

1. system maintainer will restart the application

3.a If the user accidently enters an incorrect data point

    1. The user deletes the data point

    2. The user adds a correct data point to replace it

4.a The application fails to display information

1. The user contacts the system maintainer

2. System maintainer restarts the application

Michael Pressman Use Cases:

4.Sleep Habits Tracker - Fully Dressed Author: Michael Pressman

ID: Customer tracks sleep

Scope: Apps sleep tracking system
Level: user goal

Stakeholders and Interests:
Customer - a person who is using the app and wants to track their sleep
Trainer - can see customers sleep-tracked (if turned on)
System Maintainer- the person responsible for keeping the app running

Precondition: The user has an account on the app

Postcondition: Sleep is inputted and tracked

Main Success Scenario:

1. Customer wants to track the sleep from the previous night
2. Opens and logs in to the application
3. Opens the sleep tracker
4. Enters the start of when they fell asleep and the time they woke up
5. System displays the total amount of sleep in hours and minutes the user slept, compare it to their goal for the amount of sleep per night, then either congratulates the user for reaching their goal, or tells the user how much more sleep they needed to reach their goal.
6. User exits the sleep tracker
7. User logs off the app

Alternate Paths:
a - sleep tracking system does not respond

1. System maintainer will restart application

b - application does not allow user to log in

1. User receives error message and tries again
2. User emails customer support
3. System will request user input again

c - user enters invalid time of sleep either the start of sleeping or when they woke up

1. System will present error message
2. System will request user input again

d - application is not presenting requested information

1. User restarts the application
2. User contacts system maintainer
3. System maintainer restarts application

5.) Leaderboard Use Case - Fully Dressed (written by Michael Pressman)

ID: View and Interact with Leaderboard

Scope: App's leaderboard system

Level: User goal

Stakeholders and Interests:

Customer: Wants to see their ranking and compare progress with others.

Trainer: May want to monitor their clients' progress (if allowed).

System Maintainer: Ensures the leaderboard functions correctly and updates rankings.

Precondition:

- User has an account on the app and is logged into app - Leaderboard is enabled and populated with user data.

Postcondition:

- User has viewed the leaderboard and can see rankings.
- If applicable, user has interacted with leaderboard features (e.g., filtering results).

Main Success Scenario:

1. User opens and logs into the app
2. User navigates to the leaderboard section
3. System fetches and displays the leaderboard rankings
4. User views rankings and can scroll through different users.
5. User applies filters (e.g., by time period, specific workout category, friends)
6. System updates leaderboard based on selected filters.
7. User selects a profile to view more details.
8. System displays selected user's public fitness stats.
9. User exits the leaderboard section.
10. User logs out of the app.

Alternate Paths:

a – Leaderboard fails to load:

1. System displays an error message.
2. User tries refreshing the leaderboard.

3. If the issue persists, the user contacts customer support.

b – Filters do not work properly:

1. System displays an error message
2. User resets filters and tries again.
3. If issue persists, the user contacts customer support.

c – User profile details do not load properly:

1. system will present error message
2. User returns to the leaderboard and selects a different profile
3. If issue persists, the user contacts the system maintainer d – User wants to report incorrect data:

1. user selects the option to report an issue
2. System notifies the app's support team.
3. Support team reviews and corrects the data if needed

6.) Friend system Use Case- Fully Dressed (written by Michael Pressman)

ID: Add and manage friends

Scope: App's friend system

Level: user goal

Stakeholders and Interests:

User (Customer): Wants to add, remove, and interact with friends.

Trainer: May want to connect with clients for tracking progress (if permitted).

System Maintainer: Ensures the friend system works correctly and updates connections.

Precondition:

- User has an account and is logged into the app. - Friend system is enabled and available.

Postcondition:

- User has added, removed, or interacted with a friend in the app.

Main Success Scenario:

1. User logs into the app

2. User navigates to the friend system section.
3. User searches for a friend by username or suggested contacts.
4. System displays matching users or suggested friends.
5. User selects a person and sends a friend request.
6. System notifies the recipient of the friend request.
7. Recipient accepts the request.
8. System updates both users' friend lists.
9. User can now view their friend's activity (if privacy settings allow).
10. User exits the friend system section.

Alternate Paths:

a – Friend request is not sent due to system error:

1. System displays an error message
2. User retries sending the request
3. If the issue persists, user contacts customer support. b – Recipient declines the request

1. System notifies the sender that the request was declined
2. User can send another request later or move on c – user wants to remove a friend:

1. User navigates to their friend list
2. User selects a friend and chooses the "Remove Friend" option.
3. System removes the friend from both users' lists.
4. If applicable, system adjust privacy settings accordingly d – User receives a friend request but does not recognize the sender:

1. User checks the sender's profile.
2. User decided to either accept, ignore, or block the request.
3. If blocked, system prevents further friend request from that user.

e – Friend activity is not updating correctly:

1. User refreshes the friend system section.
2. If still incorrect, user contacts system maintainer.

Ryan Meador Use Cases

7.Tracking Calories Use Case - Fully Dressed (written by Ryan Meador)

ID: Customer tracks calories

Scope: App's calorie tracking system

Level: user goal

Stakeholders and Interests:

Customer - person that is using the app and wants to track their calories

Trainer - can see customers calories tracked (if turned on)

System Maintainer - person responsible for keeping the app running

Precondition: User has an account on the app

Postcondition: Calories are input and tracked

Main Success Scenario:

1. Customer wants to track the calories from their most recent meal
2. Opens and logs in to the application
3. Opens the calorie tracker
4. Enters amount of calories consumed
5. A new calorie report is created
6. If the user has goals, the system takes in the new calorie report and updates said goals
7. System subtracts calories from the report from total left for the day
8. System displays total calorie intake, the users goals, and how many calories can still be consumed while hitting their goal
9. User exits the calorie tracker
10. User logs off the app

Alternate Paths: a - calorie tracking system does not respond

1. system maintainer will restart application b - application does not allow user to log in

1. user emails customer support
2. system maintainer can manually log user in c - user enters invalid number for calories

1. system will present error message
2. system will request user input again d - application is not presenting requested information

1. user contacts system maintainer

2. system maintainer restarts application

8.Setting Goals Use Case - Fully Dressed (written by Ryan Meador)

ID: Customer sets fitness goals

Scope: App's goal setting system

Level: User goal

Stakeholders and Interests:

Customer - person that is using the app and wants to set fitness goals

Trainer - can see a customer’s goals (if turned on)

System Maintainer - person responsible for keeping the app running

Precondition: User has an account on the app

Postcondition: User has daily, weekly, or monthly goals set for: calories, weight, and sleep

Main Success Scenario:

11. Customer wants to set up a daily, weekly, or monthly goal for a trackable value
12. Opens the goal setting system
13. Selects the length of the goal they want to set (i.e. daily, weekly, or monthly)
14. Selects the variable they want to have a goal for (i.e. calories, weight, or sleep)
15. Enters their specific goal
16. A new goal is created
17. Customer's trainer is notified of their new goal
18. Customer is returned to the main goal screen and can continue to add goals

Alternate Paths:

a - goal setting system does not respond 1. system
maintainer will restart application b - application
does not allow user to log in

1. user emails customer support
2. system maintainer can manually log user in c - user does not have a trainer 1. no trainer is notified d - application is not presenting requested information

1. user contacts system maintainer
2. system maintainer restarts application

9.) Setting Reminders Use Case - Fully Dressed (written by Ryan Meador)

ID: Customer sets daily reminders

Scope: App's reminder system

Level: User reminder

Stakeholders and Interests:

Customer - person that is using the app and wants to set reminders

System Maintainer - person responsible for keeping the app running

Precondition: User has an account on the app

Postcondition: User has reminders set for daily tasks

Main Success Scenario:

1. Customer wants to set a reminder for one of their daily tasks
2. Opens the reminder creation system
3. Select the task they want to have a reminder for
4. Select the time of day they want to be reminded
5. A new reminder is created
6. Customer is returned to the main reminder screen and can continue to add new reminders

Alternate Paths:

a - reminder system does not respond

1. system maintainer will restart application b - application does not allow user to log in

1. user emails customer support
2. system maintainer can manually log user in c - application is not presenting requested information

1. user contacts system maintainer
2. system maintainer restarts application

Chase Caldwell Use Cases:

10.) Trainer Adds to Class Use Case - Fully Dressed Author: Chase Caldwell

ID: Trainer Adds User to Workout Class

Scope: Health & Fitness App – Trainer Class Management

Level: User Goal

Stakeholders and Interests:

- Trainer – Needs to create, manage, and update a workout class, including scheduling and participant management.
- Participants – Need to be added or removed from classes based on attendance.
- System Maintainer – Responsible for ensuring smooth operation and resolving technical issues.

Precondition:

- Trainer has an account on the app and is logged in.

Postcondition:

- A workout class is created and updated with participants and a schedule.
- Participants are successfully added or removed.
- Changes are saved in the system.

Main Success Scenario:

1. Trainer wants to create and manage a workout class.
2. Opens and logs into the application.
3. Navigates to the Class Management section.
4. Selects Create New Class and enters a class name.
5. Adds participants by selecting Add Participants and inputting their details.
6. System saves participant information and updates the class roster.
7. Trainer logs off the app after successfully managing the class.

Alternate Paths:

a - System does not respond to a request

1. Trainer retries the action.
2. If the issue persists, the trainer contacts the system maintainer.

b - Trainer enters an invalid class name

1. System displays an error message.
2. Trainer enters a new class name. c - Trainer enters invalid participant details

1. System presents an error message.
2. Trainer corrects the information and retries.

d - Application does not allow trainer to log in

1. Trainer contacts customer support.
2. System maintainer manually resets access if necessary.

11.) Challenge Use Case (written by Chase Caldwell)

ID: User sends a challenge to another user

Scope: Health & Fitness App – User Social Management

Level: User Goal

Stakeholders and Interests:

- Trainer – Needs to see challenges in order to engage with users.
- Participants – Needs to send challenges in order to interact with friends on app.
- System Maintainer – Responsible for ensuring smooth operation and resolving technical issues.

Precondition:

- User has an account on the app and is logged in.
- User is friends with the user they want to challenge Postcondition:
- User receives a challenge from their friend
- Challenge countdown is started
- Changes are saved in the system.

Main Success Scenario:

1. User wants to send a challenge to a friend 2.
Opens and logs into the application.
3. Navigates to the Friend Management section.
4. Selects the friend they desire.
5. Clicks on send challenge.
6. User creates their desired challenge
7. System confirms the challenge was sent.
8. User logs off the app after successfully challenging a friend.

Alternate Paths: a - System does not
respond to a request

1. User retries the action.
2. If the issue persists, the user contacts the system maintainer.

b - Application does not allow user to log in

1. User contacts customer support.
2. System maintainer manually resets access if necessary.

12.) Fitness Group Use Case (written by Chase Caldwell)

ID: User joins a fitness group with other users

Scope: Health & Fitness App – User Social Management

Level: User Goal

Stakeholders and Interests:

- Trainer – Need to have ability to join fitness and social groups.
- Participants – Need to have ability to join fitness and social groups.
- System Maintainer – Responsible for ensuring smooth operation and resolving technical issues.

Precondition:

- User has an account on the app and is logged in.

Postcondition:

- User has joined the fitness group
- Group is updated and saved in the system.

Main Success Scenario:

1. User wants to join a fitness group
2. User logs in to the app
3. User goes to social section
4. User joins a fitness group
5. User logs off Alternate Paths:

a - System does not respond to a request

3. User retries the action.
4. If the issue persists, the user contacts the system maintainer.

b - Application does not allow user to log in

3. User contacts customer support.
4. System maintainer manually resets access if necessary.

<u>Larry O'Connor Use Cases:</u>

13.Statistics Use Case - Fully Dressed (written by Larry O'Connor)

ID: User Views and Exports Statistics

Scope: Apps statistics tracking system Level:
user goal

Stakeholders and Interests:

User - Wants to view, filter, and export their workout and class statistics.

Trainer - can see user statistics (if user has sharing is enabled). System Maintainer - person responsible for keeping the app running Precondition:

- User has an account on the app.
- User is logged into the system
- System has recorded workout data Postcondition:
- Statistics are retrieved and displayed correctly.

Main Success Scenario:

1. User wants to view their overall workout and class statistics.
2. Opens and logs in to the application.
3. Navigates to the Statistics page.
4. System retrieves and displays statistics based on the user's workout history and classes taken.
5. User reviews the generated statistics.
6. User is satisfied with the displayed statistics.
7. User logs off the app.

Alternate Paths:

a - User wants to export statistics

1. User logs into the app
2. Navigates to the Statistics page.
3. Opens the "Select Stats" UI.
4. Modifies the displayed statistics by selecting/deselecting options.
5. System updates the displayed statistics.
6. User presses the “Export Stats” button.
7. System generates an image containing the current statistics view.
8. User downloads the image.
9. User logs out of the app.

b - User wants to filter statistics by date range

1. User opens and logs in to the application.
2. Navigates to the Statistics page.
3. Opens the "Date Selection" UI.
4. Selects a start date and an end date.
5. System retrieves and filters statistics within the specified date range.
6. System displays the filtered statistics.
7. User reviews the displayed statistics.
8. User logs out of the app.

c - Statistics system does not respond

1. System maintainer will restart application d - Export function does not work

1. System displays an error message.
2. System maintainer investigates and resolves the issue. e - User enters invalid date range

1. System displays an error message for invalid date
2. User adjusts the date range

14.Admin Management Use Case - Fully Dressed (written by Larry O'Connor)

ID: Admin Manages Users

Scope: App User Management System

Level: user goal

Stakeholders and Interests:

Admin – Wants to manage user accounts efficiently (reset passwords, and view users).

User – Wants account issues resolved quickly, such as password resets.

Trainer – Could need assistance with account recovery

System Maintainer – person responsible for keeping the app running Precondition:

- The admin has a valid account and is logged into the system.
- There are existing users in the system.

Postcondition:

- Passwords are reset securely if requested.
- Unauthorized users do not gain access.

Main Success Scenario

1. Admin logs in to the app using their login.
2. Admin navigates to the user management page.
3. Admin selects an action from the available options (view users, reset password)
4. System retrieves the relevant user data based on the admin's request.
5. Admin resets user selected user's password
6. System updates the database and confirms the action's success.
7. Admin receives a confirmation message and logs out of the system.

Alternate Paths:

a - Admin Enters Incorrect Login Credentials

- Admin enters an incorrect username or password.
- System displays an error message: "Invalid username or password."
- Admin is prompted to try again.
- After multiple failed attempts, the system locks the account.

b - Admin Tries to View Users, But No Users Exist

1. Admin navigates to the "View Users" UI.
2. System finds no users or trainers in the database.
3. System displays the message "No users found".

c - Admin Views All Users and Filters by Role

1. Admin opens the "View Users" UI.
2. Admin filters users by role (User, Trainer, Admin).
3. System retrieves and displays the filtered list.

15.XP System Use Case - Fully Dressed (written by Larry O'Connor)

ID: User Views and Earns XP

Scope: Apps XP tracking system Level:
user goal

Stakeholders and Interests:

User – Wants to track progress through XP accumulation from various activities.

Trainer – can see XP earned by users

System Maintainer – person responsible for keeping the app running

Precondition:

- User has an account on the app.
- User is logged into the system.

Postcondition:

- XP is retrieved and displayed properly

Main Success Scenario

1. User logs into the app
2. User records an activity that earns XP (calories burned, sleep time, workout length, etc). 3. System calculates XP to add based on entered information 4. System calculates bonus XP to add based on your goals.
5. User views updated XP UI
6. User is satisfied with the displayed progress 7. User logs off the app.

Alternate Paths:

a - User removes workout

- User logs into the app
- User removes previously recorded workout
- System calculates XP to remove based on removed information
- System removes necessary XP • User logs off the app b - XP leaderboard access

- User wants to compare their XP with others.
- User navigates to the leaderboard UI.
- System retrieves and displays XP rankings among users.
- User reviews their displayed rank

- User logs off the app

16.Dannis Wu Use Cases Use Case 1: Tracking Calories

Author: Dannis Wu

ID: User Tracks Calorie Intake

Scope: MyFitness App – Calorie Tracking Level: User Goal

Stakeholders and Interests:

- User – Wants to track daily calorie intake to monitor diet.
- System Maintainer – Ensures data is saved and retrieved reliably.

Preconditions:

- User is logged in and has an account.

Postconditions:

- Calories are added to the user's daily log.
- User sees an updated total and summary.

Main Success Scenario:

1. User opens the MyFitness app.
2. Navigates to the Calorie Tracker.
3. System opens the calorie tracker.
4. User enters calories consumed.
5. System logs the entry and updates the daily total.
6. User requests a calorie summary.
7. System displays calories consumed and remaining.
8. User closes the calorie tracker.

Alternate Paths:

a – Calorie entry invalid (e.g., not a number):

1. System displays an error message.
2. User re-enters the calorie value.

b – Summary requested before any data exists:

1. System notifies user that no data is available.
2. User enters calories before retrying.

17.Use Case 2: Managing Exercise Journal

Author: Dannis Wu

ID: User Logs Exercise Sessions

Scope: MyFitness App – Exercise Journal Level: User Goal

Stakeholders and Interests:

- User – Wants to track workouts and view progress.
- System Maintainer – Keeps journal data persistent and accessible.

Preconditions:

- User is logged in and inside the Exercise Journal module.

Postconditions:

- Workouts are added to the user's log.
- User can review exercise history.

Main Success Scenario:

1. User opens the Exercise Journal.
2. System loads journal interface.
3. User creates a new exercise session.
4. User adds one or more workouts with type and details.
5. System saves workouts under the current session.
6. User views exercise history.
7. System displays past sessions and workout data.
8. User exits the journal.

Alternate Paths:

a – No workouts added yet:

1. System notifies user that no sessions are available to view.
2. User starts a session and logs data.

b – Invalid workout details (e.g., empty name):

1. System prompts user to enter valid details.
2. User corrects and submits again.

18.Use Case 3: Trainer Manages Workout Class

Author: Dannis Wu

ID: Trainer Manages Workout Classes

Scope: MyFitness App – Trainer Management Level: User Goal

Stakeholders and Interests:

- Trainer – Needs to create and manage workout classes.
- Participants – Want to be correctly added to class rosters.
- System Maintainer – Ensures system stores schedule and class data accurately.

Preconditions:

- Trainer is logged into the app.

Postconditions:

- Class created or modified with participants and schedule.

Main Success Scenario:

1. Trainer logs into the app.
2. Opens the Class Manager interface.
3. Creates a new class with a name.
4. Adds participants using their IDs.
5. Updates class schedule with time and details.
6. System saves all data and confirms update.
7. Trainer closes the class manager.

Alternate Paths:

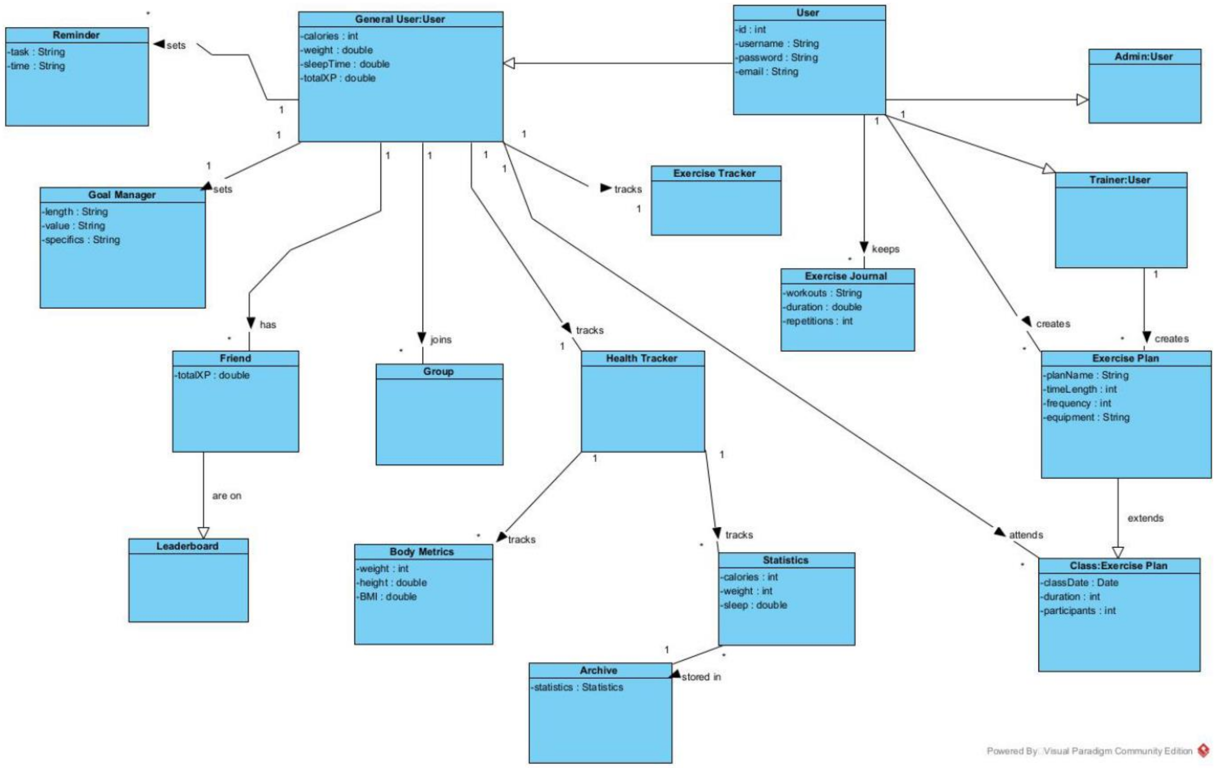
a – Invalid participant ID entered:

1. System displays an error.
2. Trainer re-enters valid ID.

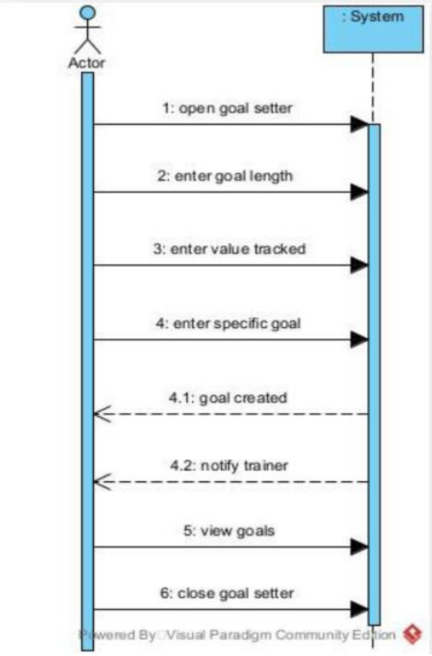
b – Schedule update fails due to missing fields:

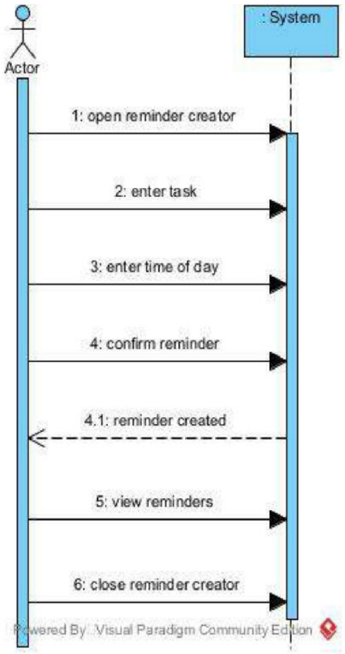
1. System alerts trainer of incomplete details.
2. Trainer corrects and resubmits.

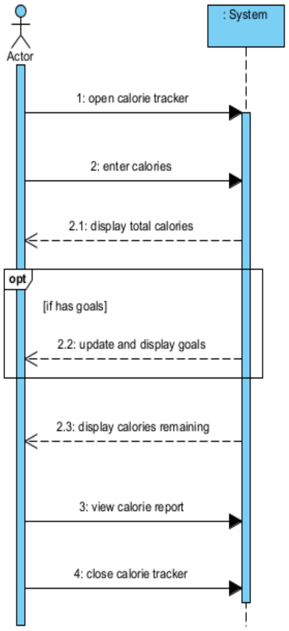
Domain Model:

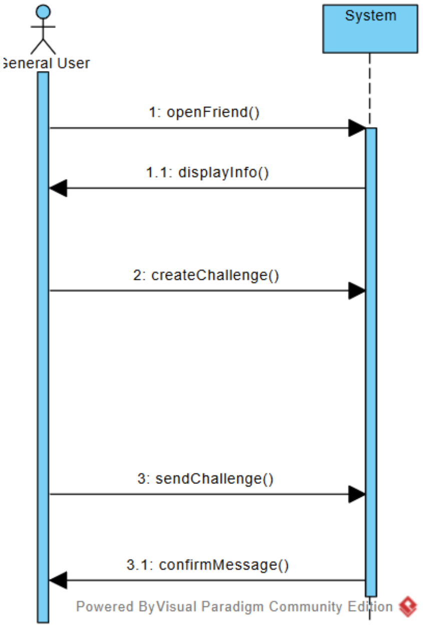


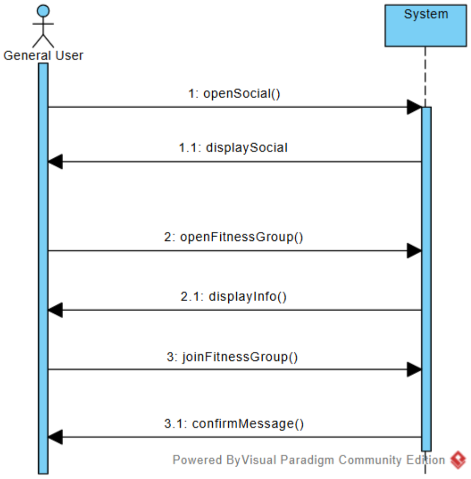
SSDs:

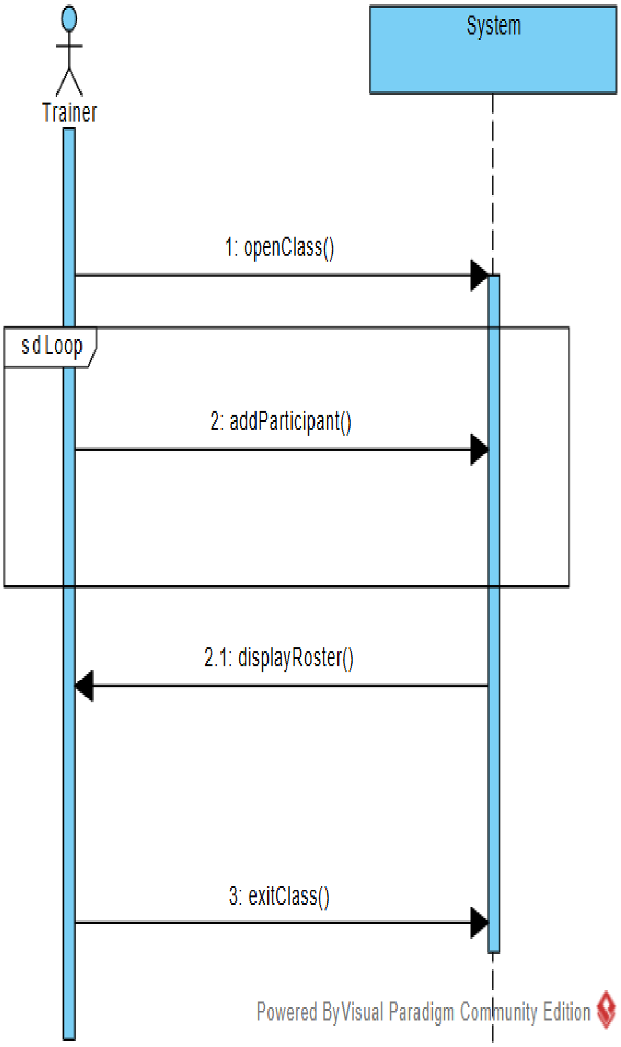

Actor
: System
1: open goal setter
2: enter goal length
3: enter value tracked
4: enter specific goal
4.1: goal created
4.2: notify trainer
5: view goals
6: close goal setter
Powered By Visual Paradigm Community Edition

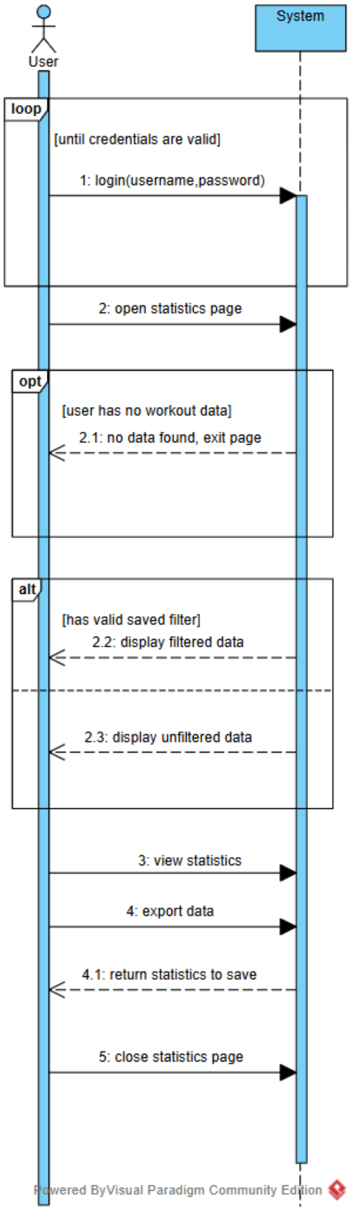

Actor
: System
1: open reminder creator
2: enter task
3: enter time of day
4: confirm reminder
4.1: reminder created
5: view reminders
6: close reminder creator
Powered By Visual Paradigm Community Edition

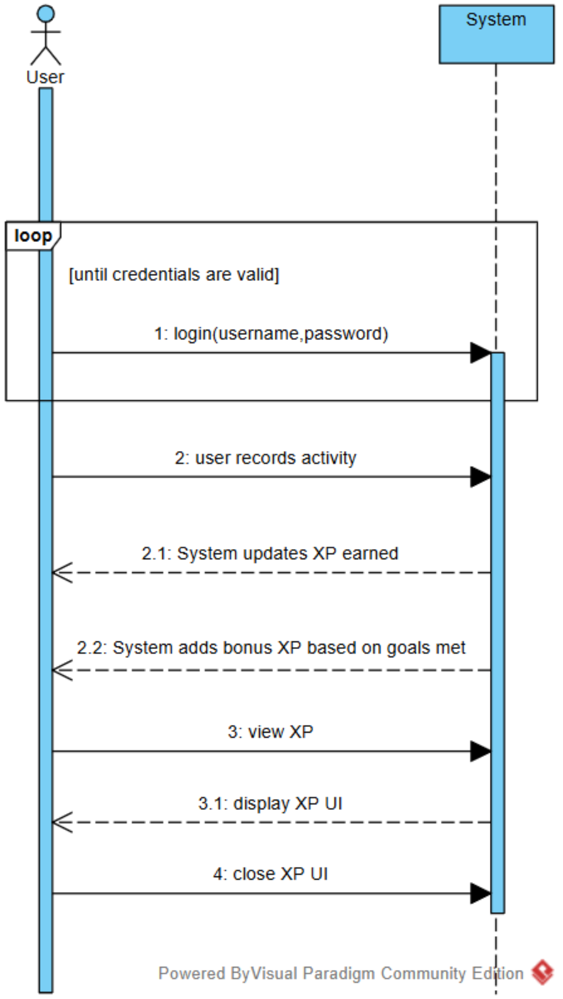

Actor
: System
1: open calorie tracker
2: enter calories
2.1: display total calories
opt
[if has goals]
2.2: update and display goals
2.3: display calories remaining
3: view calorie report
4: close calorie tracker

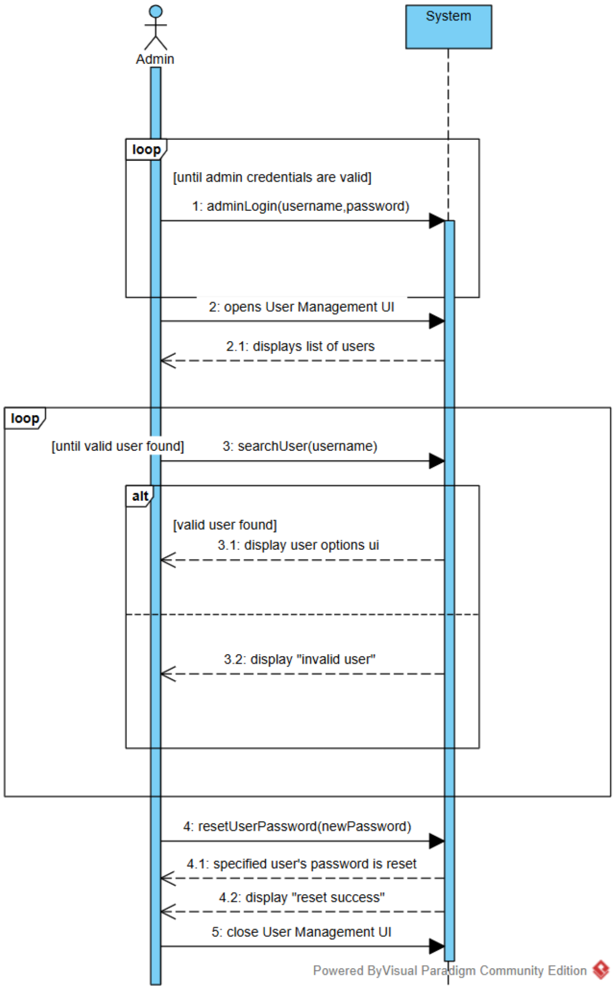

General User
System
1: openFriend()
1.1: displayInfo()
2: createChallenge()
3: sendChallenge()
3.1: confirmMessage()
Powered By Visual Paradigm Community Edition

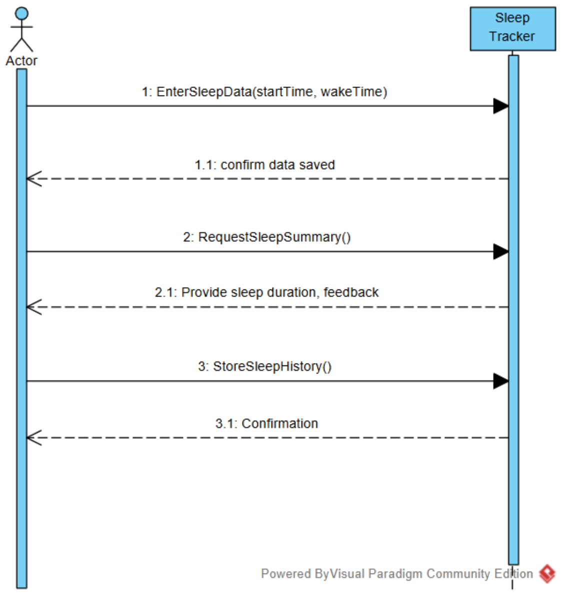

General User
System
1: openSocial()
1.1: displaySocial
2: openFitnessGroup()
2.1: displayInfo()
3: joinFitnessGroup()
3.1: confirmMessage()
Powered By Visual Paradigm Community Edition

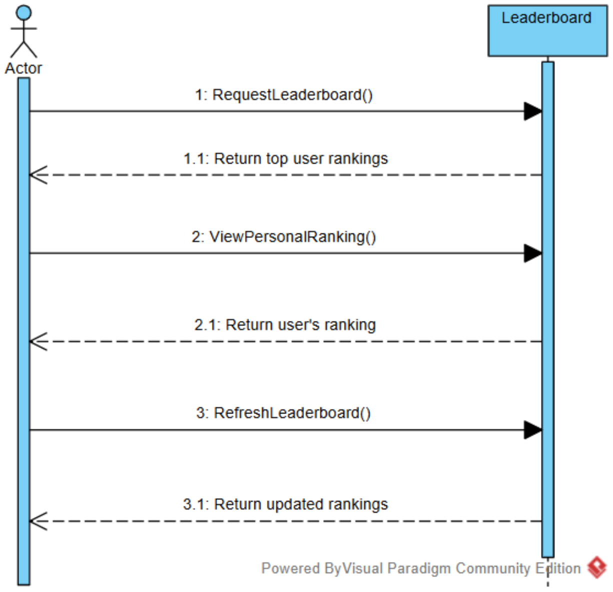

Trainer
System
1: openClass()
sd Loop
2: addParticipant()
2.1: displayRoster()
3: exitClass()
Powered By Visual Paradigm Community Edition

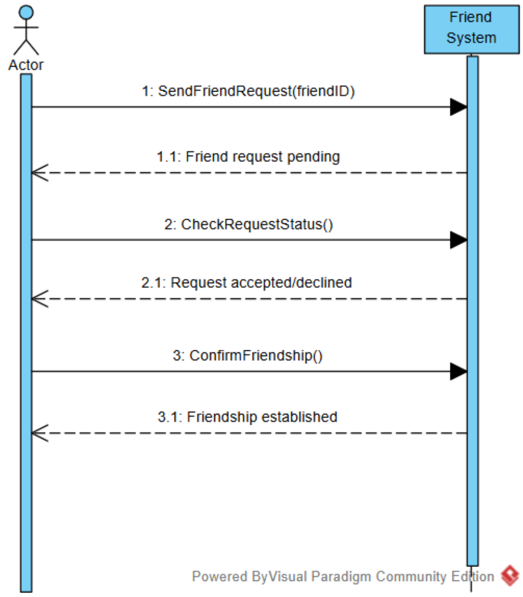

User
System
loop
[until credentials are valid]
1: login(username,password)
2: open statistics page
opt
[user has no workout data]
2.1: no data found, exit page
alt
[has valid saved filter]
2.2: display filtered data
2.3: display unfiltered data
3: view statistics
4: export data
4.1: return statistics to save
5: close statistics page
Powered By Visual Paradigm Community Edition

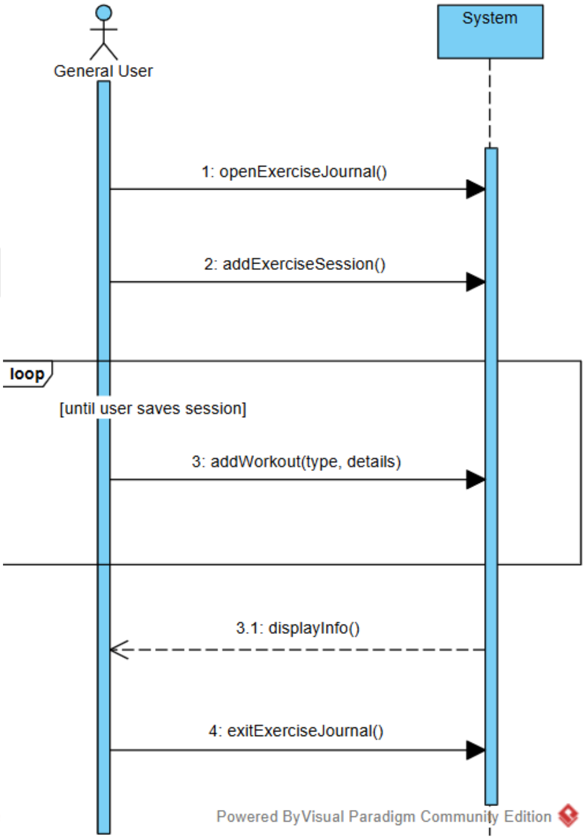

User
System
loop
[until credentials are valid]
1: login(username,password)
2: user records activity
2.1: System updates XP earned
2.2: System adds bonus XP based on goals met
3: view XP
3.1: display XP UI
4: close XP UI
Powered ByVisual Paradigm Community Edition

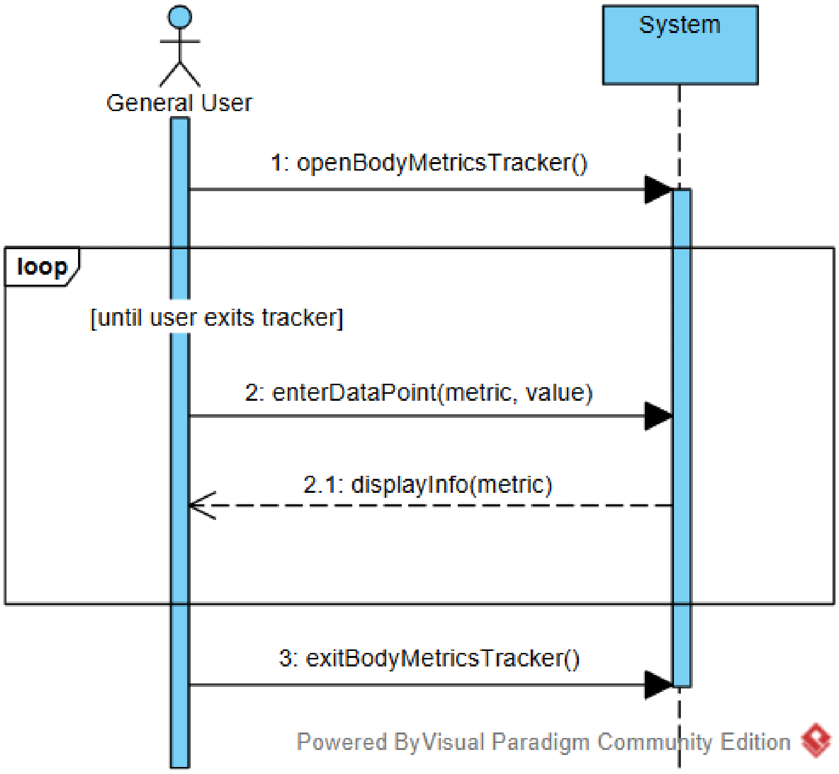

Admin
System
loop
[until admin credentials are valid]
1: adminLogin(username,password)
2: opens User Management UI
2.1: displays list of users
loop
[until valid user found]
3: searchUser(username)
alt
[valid user found]
3.1: display user options ui
3.2: display "invalid user"
4: resetUserPassword(newPassword)
4.1: specified user's password is reset
4.2: display "reset success"
5: close User Management UI
Powered ByVisual Paradigm Community Edition

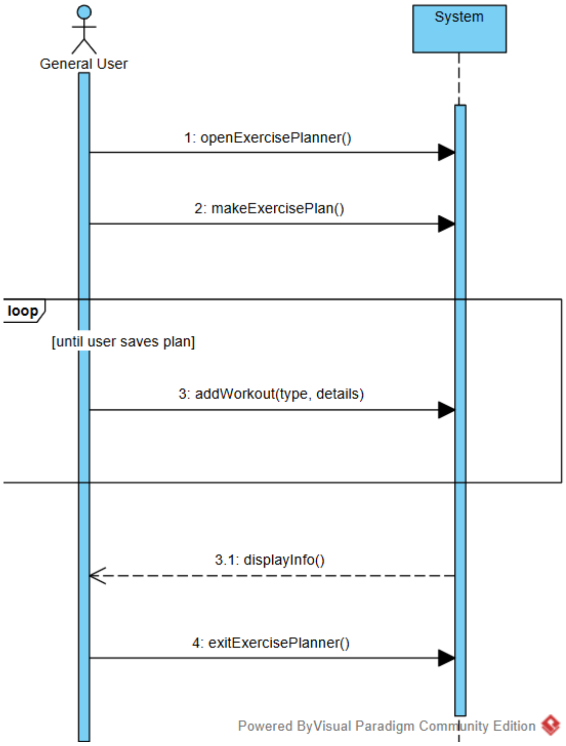

Actor
Sleep Tracker
1: EnterSleepData(startTime, wakeTime)
1.1: confirm data saved
2: RequestSleepSummary()
2.1: Provide sleep duration, feedback
3: StoreSleepHistory()
3.1: Confirmation
Powered ByVisual Paradigm Community Edition

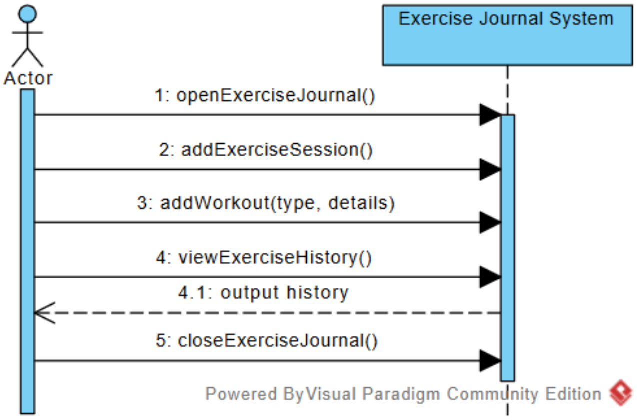

Actor
Leaderboard
1: RequestLeaderboard()
1.1: Return top user rankings
2: ViewPersonalRanking()
2.1: Return user's ranking
3: RefreshLeaderboard()
3.1: Return updated rankings
Powered ByVisual Paradigm Community Edition

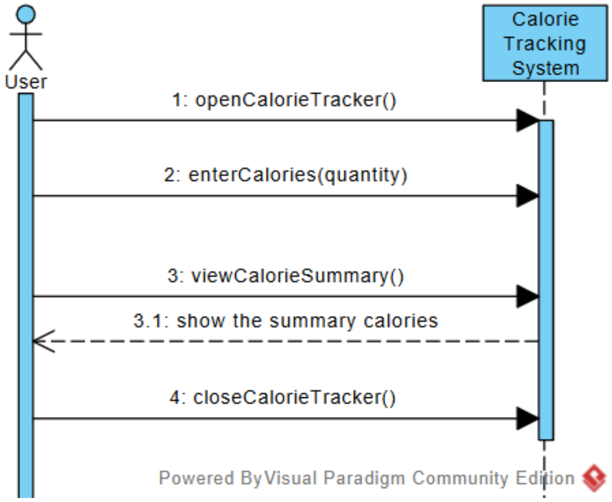

Actor
Friend System
1: SendFriendRequest(friendID)
1.1: Friend request pending
2: CheckRequestStatus()
2.1: Request accepted/declined
3: ConfirmFriendship()
3.1: Friendship established
Powered ByVisual Paradigm Community Edition


General User
System
1: openExerciseJournal()
2: addExerciseSession()
loop
[until user saves session]
3: addWorkout(type, details)
3.1: displayInfo()
4: exitExerciseJournal()
Powered ByVisual Paradigm Community Edition


General User
System
1: openBodyMetricsTracker()
loop
[until user exits tracker]
2: enterDataPoint(metric, value)
2.1: displayInfo(metric)
3: exitBodyMetricsTracker()
Powered ByVisual Paradigm Community Edition


General User
System
1: openExercisePlanner()
2: makeExercisePlan()
loop
[until user saves plan]
3: addWorkout(type, details)
3.1: displayInfo()
4: exitExercisePlanner()
Powered ByVisual Paradigm Community Edition


Actor
Exercise Journal System
1: openExerciseJournal()
2: addExerciseSession()
3: addWorkout(type, details)
4: viewExerciseHistory()
4.1: output history
5: closeExerciseJournal()
Powered ByVisual Paradigm Community Edition


User
Calorie Tracking System
1: openCalorieTracker()
2: enterCalories(quantity)
3: viewCalorieSummary()
3.1: show the summary calories
4: closeCalorieTracker()
Powered ByVisual Paradigm Community Edition

Operation Contracts:

Operation Contracts for Use Case: Tracking Calories (Ryan Meador)

System Operations

- openCalorieTracker()
- enterCalories()
- viewReport()
- closeCalorieTracker()

Contract 1: openCalorieTracker()

Operation: openCalorieTracker()

Cross References: Use Cases: Tracking Calories

Pre-conditions: User has an account

Post-conditions:

- The calorie tracker is opened

Contract 2: enterCalories()

Operation: enterCalories(quantity: double)

Cross References: Use Cases: Tracking Calories

Pre-conditions: User is in the calorie tracker

Post-conditions:

- A new calorie report was created (instance creation)
- Calorie report was associated with goals (association formed)
- Goals were updated (attribute modification)

Contract 3: viewReport()

Operation: viewReport(calorieReport: calorieReport)

Cross References: Use Cases: Tracking Calories

Pre-conditions: User has entered calories, and a calorie report was created

Post-condition:

- User can see total calories (attribute viewing)
- User can see goals (attribute viewing)
- User can see calories left for the day (attribute viewing)

Contract 4: closeCalorieTracker()

Operations: closeCalorieTracker()

Cross References: Use Cases: Tracking Calories

Pre-conditions: User is currently in the calorie tracker

Post-conditions:

- The calorie tracker is closed

Operation Contracts for Use Case: Setting Goals (Ryan Meador)

System Operations

- openGoals()
- enterLength()
- enterValue()
- enterSpecifics()
- viewGoals()
- closeGoals()

Contract 1: openGoals()

Operation: openGoals()

Cross References: Use Cases: Setting Goals

Pre-conditions: User has an account

Post-conditions:

- The goal creator is opened

Contract 2: enterLength()

Operation: enterLength()

Cross References: Use Cases: Setting Goals

Pre-conditions: User is in the goal creator

Post-conditions:

- A new goal is created (instance creation)
- Attribute length is created (attribute modification)

Contract 3: enterValue()

Operation: enterValue()

Cross References: Use Cases: Setting Goals

Pre-conditions: User has selected a goal length

Post-conditions:

- Attribute value is created (attribute modification)

Contract 4: enterSpecifics()

Operation: enterSpecifics()

Cross References: Use Cases: Setting Goals

Pre-conditions: User has selected a goal value

Post-conditions:

- Attribute specifics is created (attribute modification)

Contract 5: viewGoals()

Operation: viewGoals()

Cross References: Use Cases: Setting Goals

Pre-conditions: User has created a goal

Post-conditions:

- User can see all goals (attribute viewing)

Contract 6: closeGoals()

Operation: closeGoals()

Cross References: Use Cases: Setting Goals

Pre-conditions: User is in the goals creator

Post-conditions:

- The goals creator is closed

Operation Contracts for Use Case: Setting Reminders (Ryan Meador)

System Operations

- openReminder()
- enterTask()
- enterTime()
- confirmReminder()

- viewReminder()
- closeReminder()

Contract 1: openReminder()

Operation: openReminder()

Cross References: Use Cases: Setting Reminders

Pre-conditions: User has an account

Post-conditions:

- The reminder creator is opened

Contract 2: enterTask()

Operation: enterTask()

Cross References: Use Cases: Setting Reminders

Pre-conditions: User is in the reminder creator

Post-conditions:

- A new reminder is created (instance creation)
- Attribute task is created (attribute modification)

Contract 3: enterTime()

Operation: enterTime()

Cross References: Use Cases: Setting Reminders

Pre-conditions: User has selected a reminder task

Post-conditions:

- Attribute time is created (attribute modification)

Contract 4: confirmReminder()

Operation: confirmReminder()

Cross References: Use Cases: Setting Reminders

Pre-conditions: User has selected a reminder time

Post-conditions:

- instance of reminder is created (instance creation)

Contract 5: viewReminder()

Operation: viewReminder()

Cross References: Use Cases: Setting Reminders

Pre-conditions: User has created a reminder

Post-conditions:

- User can see all reminders (attribute viewing)

Contract 6: closeReminder()

Operation: closeReminder()

Cross References: Use Cases: Setting Reminders

Pre-conditions: User is in the reminders creator

Post-conditions:

- The reminders creator is closed

Operation Contract - Trainer Adds to Class Use Case systemOperations:

- login()
- newClass()
- openClass()
- addParticipant()
-

Contract 1:login()
Operation:login(String username, String passWord)
Cross References: Trainer Adds to Class Use Case
Preconditions:

- Account exists

Postconditions:

- Can access account (attribute viewing)

Contract 2: newClass()
Operation: newClass())
Cross References: Trainer Adds to Class Use Case
Preconditions:

- Trainer has an account on the app and is logged in.

Postconditions:

- Class is created and is on social feed (instance creation)

Contract 3: openClass()
Operation: openClass(ExerciseClass class)
Cross References: Trainer Adds to Class Use Case
Preconditions:

- Trainer has an account on the app and is logged in.
- Class exists

Postconditions:

- Class is accessed (attribute viewing)

Contract 4: addParticipant()
Operation: addParticipant(Participant)
Cross References: Trainer Adds to Class Use Case
Preconditions:

- Trainer has an account on the app and is logged in.

Postconditions:

- A workout class is created and updated with participants and a schedule. (instance creation)
- Participants are successfully added or removed. (attribute modification)
- Changes are saved in the system. (attribute modification)

Operation Contract - Challenge Use Case systemOperations:

- openFriend()
- createChallenge()
- sendChallenge()

Contract 1: openFriend()
Operation: openFriend(Friend friend)
Cross References: Challenge Use Case
Preconditions:

- User has an account on the app and is logged in.
- Friend exists

Postconditions:

- Friend is accessed (attribute viewing)

Contract 2: createChallenge()
Operation: createChallenge(Exercise workout, Time duration)
Cross References: Challenge Use Case
Preconditions:

- User has an account on the app and is logged in.

Postconditions:

- Challenge saved in system (attribute modification)

Contract 3: sendChallenge()
Operation: createChallenge(User sender, User receiver)
Cross References: Challenge Use Case
Preconditions:

- User has an account on the app and is logged in.
- User has made a challenge

Postconditions:

- User receives a challenge (instance creation)
- Countdown is started (attribute modification)
- Challenge associated with both users (instance creation)

Operation Contract - Fitness Group Use Case systemOperations:

- joinFitnessGroup()

Contract 1: joinFitnessGroup()
Operation: joinFitnessGroup(User user, Group group)

Cross References: Fitness Group Use Case
Preconditions:

- User has an account on the app and is logged in.
- Group exists

Postconditions:

- User has joined the fitness group (instance creation)
- Member count is updated (attribute modification)
- Changes saved in system (attribute modification)

Operation Contracts for Use Case: Admin Manages Users (Larry OConnor)

System Operations:

- adminLogin(username password)
- openUserManagementUI()
- searchUser(username)
- viewUser(username)
- resetUserPassword(newPassword)
- closeUserManagementUI()

Contract 1: adminLogin()

Operation: adminLogin(username password) Cross References:
Use Case: Admin Manages Users
Preconditions:

- Admin is not logged into the system.

Postconditions:

- If credentials are valid, the system grants admin access to the app. (association formed)

Contract 2: openUserManagementUI()

Operation: openUserManagementUI() Cross References: Use
Case: Admin Management
Preconditions:

- Admin is logged into the system.

Postconditions:

The User Management UI is displayed (attribute modification)

Contract 3: searchUser(username)

Operation: searchUser(username)

Cross References: Use Case: Admin Management

Preconditions:

- Admin is in the User Management UI.

Postconditions:

- opens a window allowing the user to filter the list of users based on the input (attribute modification)

- if a filter is entered, the user list will become filtered based on the input (attribute modification)

Contract 4: selectUser(username)

Operation: selectUser(username)

Cross References: Use Case: Admin Management

Preconditions:

- Admin is in the User Management UI.

Postconditions:

- If a user is selected, profile is displayed (attribute modification)

- If a user is not selected, an error message is displayed. (attribute modification)

Contract 5: resetUserPassword(newPassword)

Operation: resetUserPassword(newPassword)

Cross References: Use Case: Admin Management

Preconditions:

- Admin has selected a valid user.

Postconditions:

- The selected user's password is updated in the system. (attribute modification)

- A message is displayed to confirm the password reset. (attribute modification)

Contract 6: closeUserManagementUI()

Operation: closeUserManagementUI() Cross References: Use Case: Admin Management
Preconditions:

- Admin is currently on the User Management UI.

Postconditions:

- User Management UI is closed (attribute modification)

Operation Contracts for Use Case: User Views and Exports Statistics (Larry OConnor)

System Operations:

- openStatisticsPage()
- viewStatistics()
- exportStatistics()
- filterStatistics(startDate endDate filters)
- validateDate(startDate endDate)

closeStatisticsPage()

Contract 1: openStatisticsPage()

Operation: openStatisticsPage()

Cross References: Use Case: User Views and Exports Statistics Preconditions:

- User is logged into the system.

Postconditions:

- The Statistics page is opened. (attribute modification)

Contract 2: viewStatistics()

Operation: viewStatistics()

Cross References: Use Case: User Views and Exports Statistics Preconditions:

- User is on the Statistics page.
- System has recorded workout data.

Postconditions:

- System retrieves and displays statistics (attribute modification)

Contract 3: exportStatistics()

Operation: exportStatistics()

Cross References: Use Case: User Views and Exports Statistics Preconditions:

- User is on the Statistics page.

System has retrieved and displayed statistics.

Postconditions:

- System generates an image containing the currently displayed statistics. (Instance Creation and Deletion)

- User has option to download the generated statistics image. (attribute modification)

Contract 4: filterStatistics(startDate endDate filters)

Operation: filterStatistics(startDate endDate filters) Cross References: Use Case: User Views and Exports Statistics Preconditions:

- User is on the Statistics page.

- System has recorded workout data.

- User selects a valid date range.

Postconditions:

- System filters statistics based on the selected date range and stat filters. (attribute modification)

- Filtered statistics are displayed to the user. (attribute modification)

Contract 5: validateDate(startDate endDate)

Operation: validateDate(startDate endDate)

Cross References: Use Case: User Views and Exports Statistics Preconditions:

- User selects a date range for filtering statistics.

Postconditions:

If the date range is valid, the system proceeds with filtering the statistics. (attribute modification)

- If the date range is invalid, the system displays an error message (attribute modification)

Contract 6: closeStatisticsPage()

Operation: closeStatisticsPage()

Cross References: Use Case: User Views and Exports Statistics Preconditions:

- User is currently on the Statistics page.

Postconditions:

- The Statistics page is closed. (attribute modification)

Operation Contracts for Use Case: XP Tracking System (Larry OConnor)

System Operations:

- updateXPEarned(activityData)

- addBonusXP(userID activityData)

- openXPUI()

- closeXPUI()

Contract 1: updateXPEarned(activityData)

Operation: updateXPEarned(activityData) Cross
References: Use Case: XP Tracking System
Preconditions:

- User has recorded an activity.

Postconditions:

- System calculates XP earned based on activity details. (attribute modification)

- System updates the user's total XP. (attribute modification)

Contract 2: addBonusXP(activityData)

Operation: addBonusXP(activityData) Cross References:
Use Case: XP Tracking System
Preconditions:

- User has recorded an activity. - User's goals have been met.

Postconditions:

- If goals are met, system adds bonus XP. (attribute modification)

Contract 3: openXPUI()

Operation: openXPUI()

Cross References: Use Case: XP Tracking System

Preconditions:

- User is logged into the system.

Postconditions:

- System displays XP UI. (attribute modification)

Contract 4: closeXPUI()

Operation: closeXPUI()

Cross References: Use Case: XP Tracking System

Preconditions:

- User is currently viewing the XP UI.

Postconditions:

- XP UI is closed (attribute modification)

Exercise Journal - Operation Contract Author: Jeremy Huggins

<u>System Operations</u>

- addExerciseSession()
- addWorkout()
- displayInfo()

<u>Contract: addExerciseSession</u>

Operation: addExerciseSession()

Cross References: Use Cases: Exercise Journal.

Pre-conditions: The user is in an instance of openExerciseJournal().

Post-conditions:

- An ExerciseSession instance *exses* was created. (instance creation)
- *exses* was associated with the ExerciseJournal. (association formed)

<u>Contract: addWorkout</u>

Operation: addWorkout(type: WorkoutID, details: String)

Cross References: Use Cases: Exercise Journal.

Pre-conditions: The system is in an instance of addExerciseSession().

Post-conditions:

- A Workout instance *w* was created. (instance creation)
- *w* was associated with the ExerciseSession. (association formed)
- *w.details* was set to details. (attribute modification)

Contract: displayInfo

Operation: displayInfo()

Cross References: Use Cases: Exercise Journal.

Pre-conditions: The user is in an instance of openExerciseJournal().

Post-conditions:

- User can see all of their exercise sessions (attribute viewing)
- User can see all their workout data for each exercise session (attribute viewing)
- User can see completion of exercise plan (attribute viewing)

Body Metrics Tracker - Operation Contract Author: Jeremy Huggins

System Operations

- enterDataPoint()
- displayInfo()

Contract: addDataPoint

Operation: addDataPoint(metric: MetricID, value: Double)

Cross References: Use Cases: Body Metrics Tracker.

Pre-conditions: The system is in an instance of openBodyMetricsTracker().

Post-conditions:

- A Metric instance *m* was created. (instance creation)
- *m* was associated with the BodyMetricsTracker. (association formed)
- value was appended to *m.data*. (attribute modification)

Contract: displayInfo

Operation: displayInfo(metric: MetricID)

Cross References: Use Cases: Body Metrics Tracker.

Pre-conditions: The user is in an instance of openExercisePlanner().

Post-conditions:

- User can see the updated metric and all its data points (attribute viewing)

Exercise Planner - Operation Contract Author: Jeremy Huggins

System Operations

- addExercisePlan()
- addWorkout()
- displayInfo()

Contract: addExerciseSession

Operation: addExercisePlan()

Cross References: Use Cases: Exercise Planner.

Pre-conditions: The user is in an instance of openExercisePlanner().
Post-conditions:

- An ExercisePlan instance *exp* was created. (instance creation)
- *exp* was associated with the ExercisePlanner. (association formed)

Contract: addWorkout
Operation: addWorkout(type: WorkoutID, details: String)
Cross References: Use Cases: Exercise Planner.
Pre-conditions: The system is in an instance of addExercisePlan().
Post-conditions:

- A Workout instance *w* was created. (instance creation)
- *w* was associated with the ExercisePlan. (association formed)
- *w.details* was set to details. (attribute modification)

Contract: displayInfo
Operation: displayInfo()
Cross References: Use Cases: Exercise Planner.
Pre-conditions: The user is in an instance of openExercisePlanner().
Post-conditions:

- User can see all of their weekly/monthly workout routine (attribute viewing)
- User can see all their goals related to the workout routine (attribute viewing)

trackSleep(startTime: Time, wakeTime: Time)

Description:

This operation allows a user to log their sleep by entering the time they fell asleep and the time they woke up. The system records the sleep session, calculates the total sleep duration, and updates the user's sleep history.

Preconditions:

- The user has an existing account on the app.
- The user is successfully logged in.
- The sleep tracking feature is operational.

Postconditions:

- A new SleepRecord object is created and stored in the system. (instance creation)
- The system updates the user's sleep history with the new sleep session. (attribute modification)
- The system associates the sleep session with the user's profile. (instance creation)
- If the user has a sleep goal, the system compares the sleep duration to the goal and updates progress. (attribute modification)

Inputs:

- startTime: The time the user fell asleep.
  wakeTime: The time the user woke up.

Outputs:

- totalSleepDuration: The calculated sleep duration in hours and minutes. goalComparison: A status update on whether the sleep goal was met.

Exceptions / Error Handling:

- Invalid Input: If startTime is later than wakeTime or the format is incorrect, the system rejects the entry and prompts the user to re-enter valid values.
- System Unavailability: If the sleep tracker is unresponsive, the system logs the failure and notifies the user to try again later.

Side Effects:

- The sleep record is stored in the user's sleep history.
- If notifications are enabled, the system may send a progress update to the user.

viewLeaderboard(filter: String)

Description:

- This operation allows a user to view the leaderboard, which displays rankings based on various filters such as time period or workout type. The system retrieves and presents the relevant leaderboard data.

Preconditions:

- The user has an existing account on the app.
- The user is successfully logged in.
- The leaderboard system is operational and contains data.

Postconditions:

- The system retrieves the leaderboard rankings from the database. (attribute viewing)
- The leaderboard data is updated based on the selected filter. (attribute modification)
- A leaderboard object is created (if the request requires generating a new leaderboard view). (instance creation)

Inputs:

- filter: A string representing the selected filter (e.g., "weekly", "monthly", "all-time", or workout type). Outputs:
- leaderboardData: A list of ranked users based on the selected filter.

Exceptions / Error Handling:

- Invalid Filter: If the filter is not recognized, the system returns an error message and defaults to the standard leaderboard view.
- System Unavailability: If the leaderboard system is unresponsive, an error is logged, and the user is notified.

Side Effects:

- The leaderboard access time may be logged for analytics.

sendFriendRequest(senderId: UserID, recipientId: UserID)

Description:

- This operation allows a user to send a friend request to another user. If accepted, an association between the two users is created in the system.

Preconditions:

- Both sender and recipient have existing accounts on the app.
- The sender is successfully logged in.
- The friend system is operational.

Postconditions:

- A friend request object is created in the system. (instance creation)
- The recipient is notified of the friend request. (attribute modification)

Inputs:

- senderId: The ID of the user sending the request.
  recipientId: The ID of the user receiving the request.

Outputs:

- requestStatus: A message confirming that the friend request has been sent.

Exceptions / Error Handling:

- Invalid User ID: If either user ID is not found, an error message is returned.
- Friend Request Already Sent: If a request already exists between the two users, an error message is displayed.
- System Unavailability: If the friend system is down, the request is not processed, and an error message is shown.

Side Effects:

- The friend request is stored in the system until it is accepted or declined

Operation Contracts

Use Case: Tracking Calories

System Operations

- openCalorieTracker()
- enterCalories()
- viewCalorieSummary()
- closeCalorieTracker()

Contract 1: openCalorieTracker()

Operation: openCalorieTracker()
Cross References: Use Cases: Tracking Calories **Pre-conditions:**

- The user must have an account.

Post-conditions:

- The system opens up the calorie tracker, ready for input.

Contract 2: enterCalories()

Operation: enterCalories(quantity: double) **Cross References:** Use Cases: Tracking Calories **Pre-conditions:**

- The user is currently in the calorie tracker.

Post-conditions:

- A new calorie entry is logged.
- The entry is linked to the user's daily calorie log.
- The system updates the total calorie count for the day.

Contract 3: viewCalorieSummary()

Operation: viewCalorieSummary() **Pre-conditions:**

- The user has already logged some calorie data.

Post-conditions:

- The system displays the total calories consumed for the day.
- The user sees how many calories they have left to stay within their goal.

Contract 4: closeCalorieTracker()

Operation: closeCalorieTracker()
Cross References: Use Cases: Tracking Calories **Pre-conditions:**

- The user is currently using the calorie tracker.

Post-conditions:

- The system closes the calorie tracker session.

Use Case: Exercise Journal

System Operations

- openExerciseJournal()
- addExerciseSession()
- addWorkout()
- viewExerciseHistory()
- closeExerciseJournal()

Contract 1: openExerciseJournal()

Operation: openExerciseJournal()
Cross References: Use Cases: Exercise Journal **Pre-conditions:**

- The user must have an account.

Post-conditions:

- The system opens up the exercise journal, ready for use.

Contract 2: addExerciseSession()

Operation: addExerciseSession()
Cross References: Use Cases: Exercise Journal **Pre-conditions:**

- The user is inside the exercise journal.

Post-conditions:

- A new workout session is created.
- The session is linked to the user's exercise history.

Contract 3: addWorkout()

Operation: addWorkout(type: WorkoutID, details: String) **Cross References:** Use Cases: Exercise Journal **Pre-conditions:**

- The user has started a workout session.

Post-conditions:

- A new workout entry is created.
- The workout is linked to the current session.
- The system records workout details like exercise type, reps, and weight.

Contract 4: viewExerciseHistory()

Operation: viewExerciseHistory() **Pre-conditions:**

- The user must have logged at least one workout session.

Post-conditions:

- The system displays all previous workout sessions.
- The user can see their progress over time.

Contract 5: closeExerciseJournal()

Operation: closeExerciseJournal()
Cross References: Use Cases: Exercise Journal **Pre-conditions:**

- The user is currently using the exercise journal.

Post-conditions:

- The system closes the journal session.

Use Case: Trainer Managing Workout Class

System Operations

- openClassManager()
- createClass()
- addParticipant()
- removeParticipant()
- updateSchedule()
- closeClassManager()

Contract 1: openClassManager()

Operation: openClassManager()
Cross References: Use Cases: Trainer Managing Workout Class **Pre-conditions:**

- The trainer must have an account.

Post-conditions:

- The system opens up the class management interface.

Contract 2: createClass()

Operation: createClass(className: String)
Cross References: Use Cases: Trainer Managing Workout Class **Pre-conditions:**

- The trainer is using the class manager.

Post-conditions:

- A new class is created.
- The system links the class to the trainer's account.

Contract 3: addParticipant()

Operation: addParticipant(participantID: UserID, className: String) **Cross References:** Use Cases: Trainer Managing Workout Class **Pre-conditions:**

- The class must already exist.

Post-conditions:

- A participant is successfully added to the class.

Contract 4: removeParticipant()

Operation: removeParticipant(participantID: UserID, className: String) **Cross References:** Use Cases: Trainer Managing Workout Class **Pre-conditions:**

- The class must already exist.

Post-conditions:

- The system removes the participant from the class list.

Contract 5: updateSchedule()

Operation: updateSchedule(scheduleDetails: String, className: String) **Cross References:** Use Cases: Trainer Managing Workout Class **Pre-conditions:**

- The class must already exist.

Post-conditions:

- The system updates the class schedule with the new details.

Contract 6: closeClassManager()

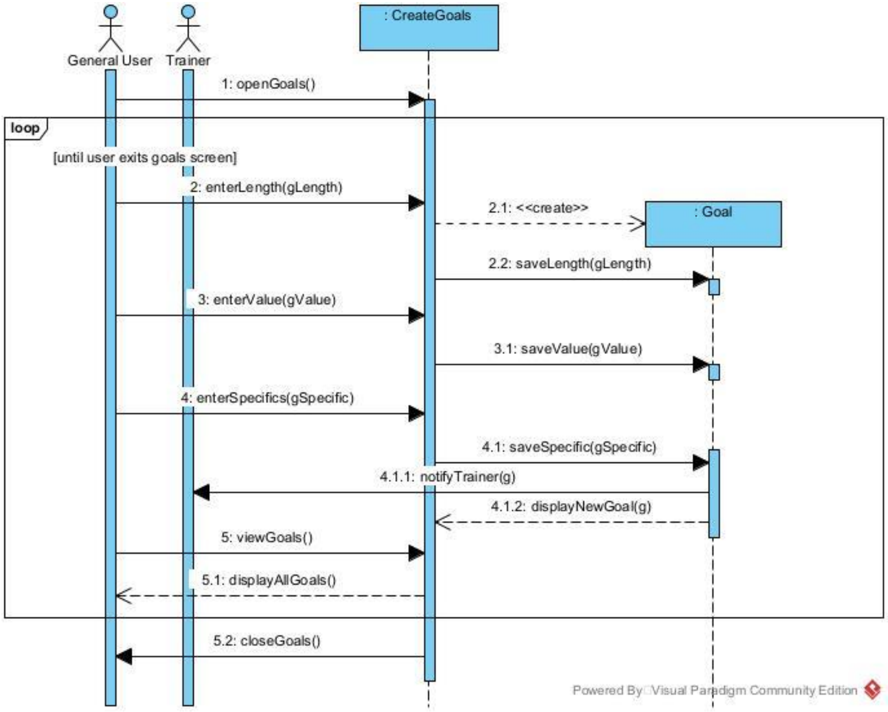
Operation: closeClassManager()
Cross References: Use Cases: Trainer Managing Workout Class **Pre-conditions:**

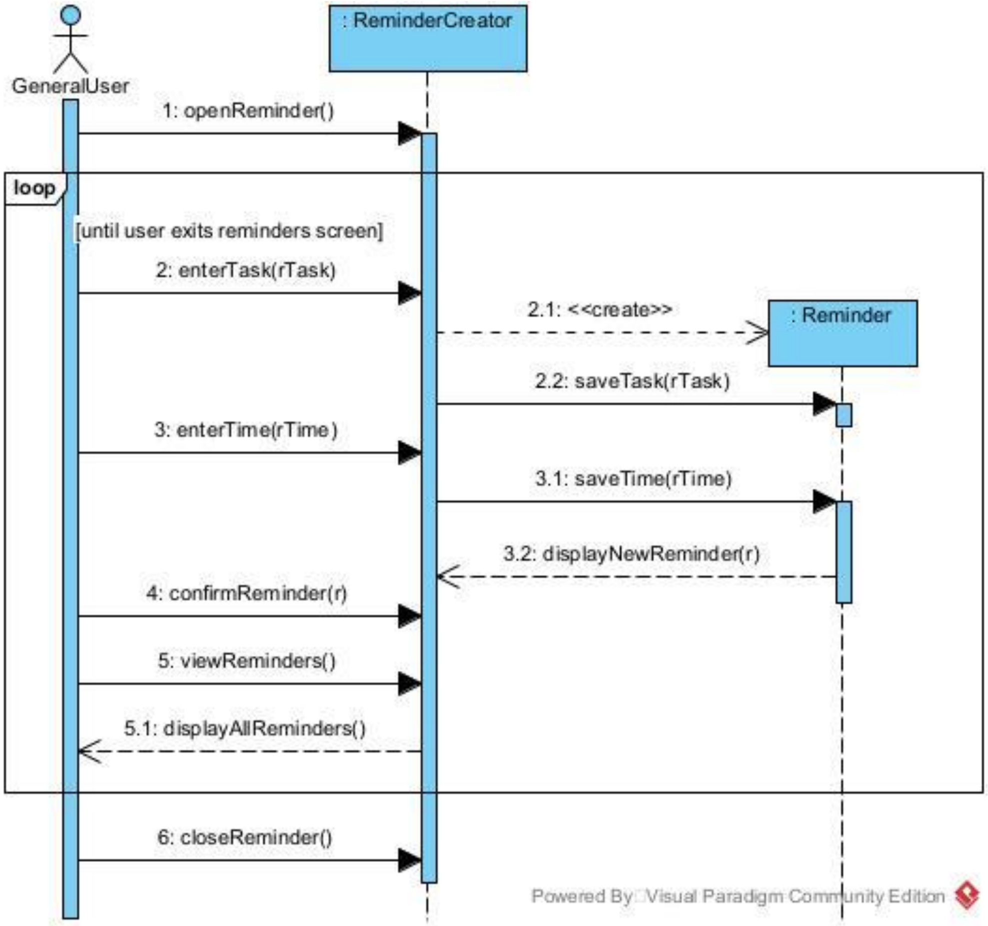
- The trainer is currently managing a class.

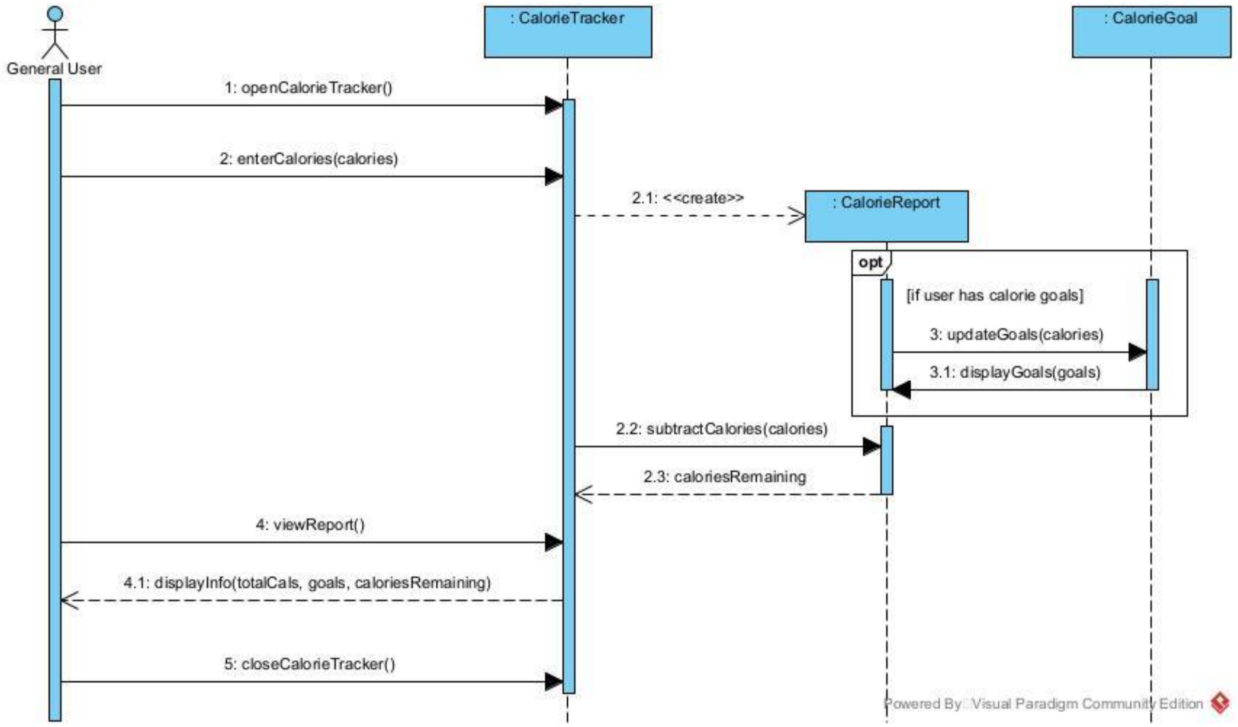
Post-conditions:

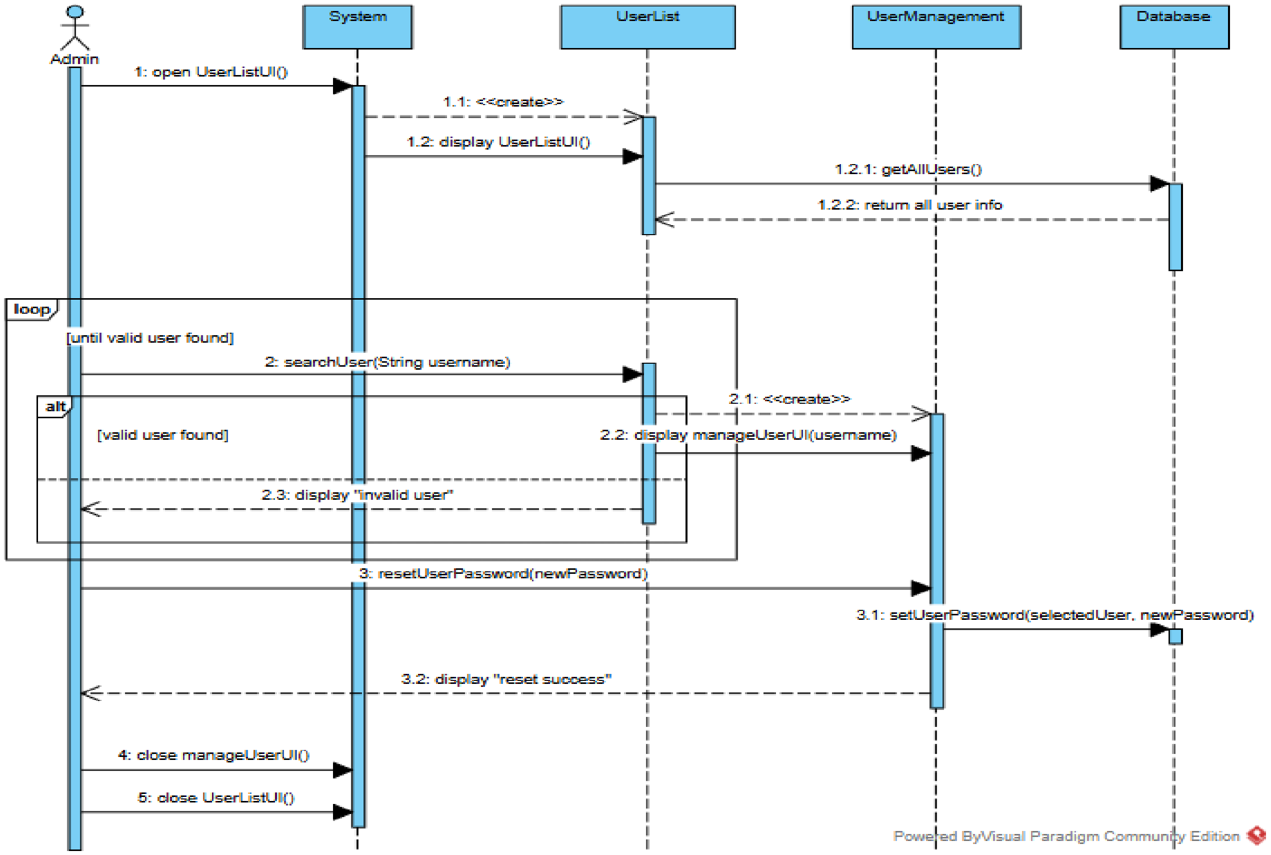
- The class manager session is closed.

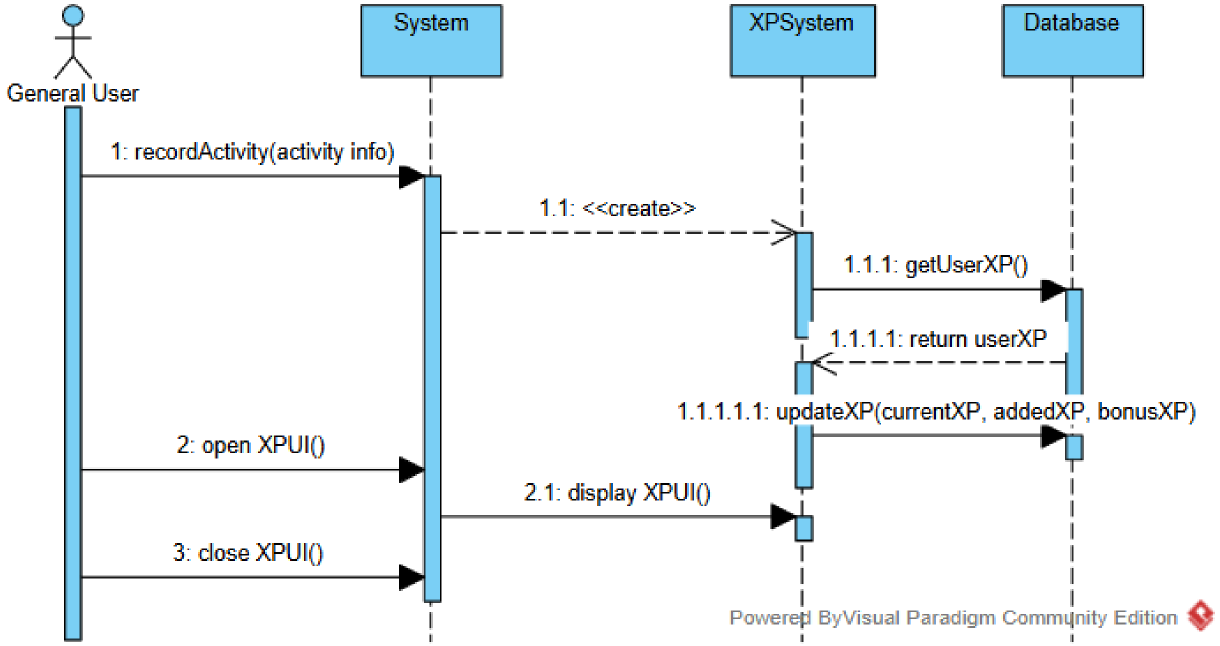
SDs:

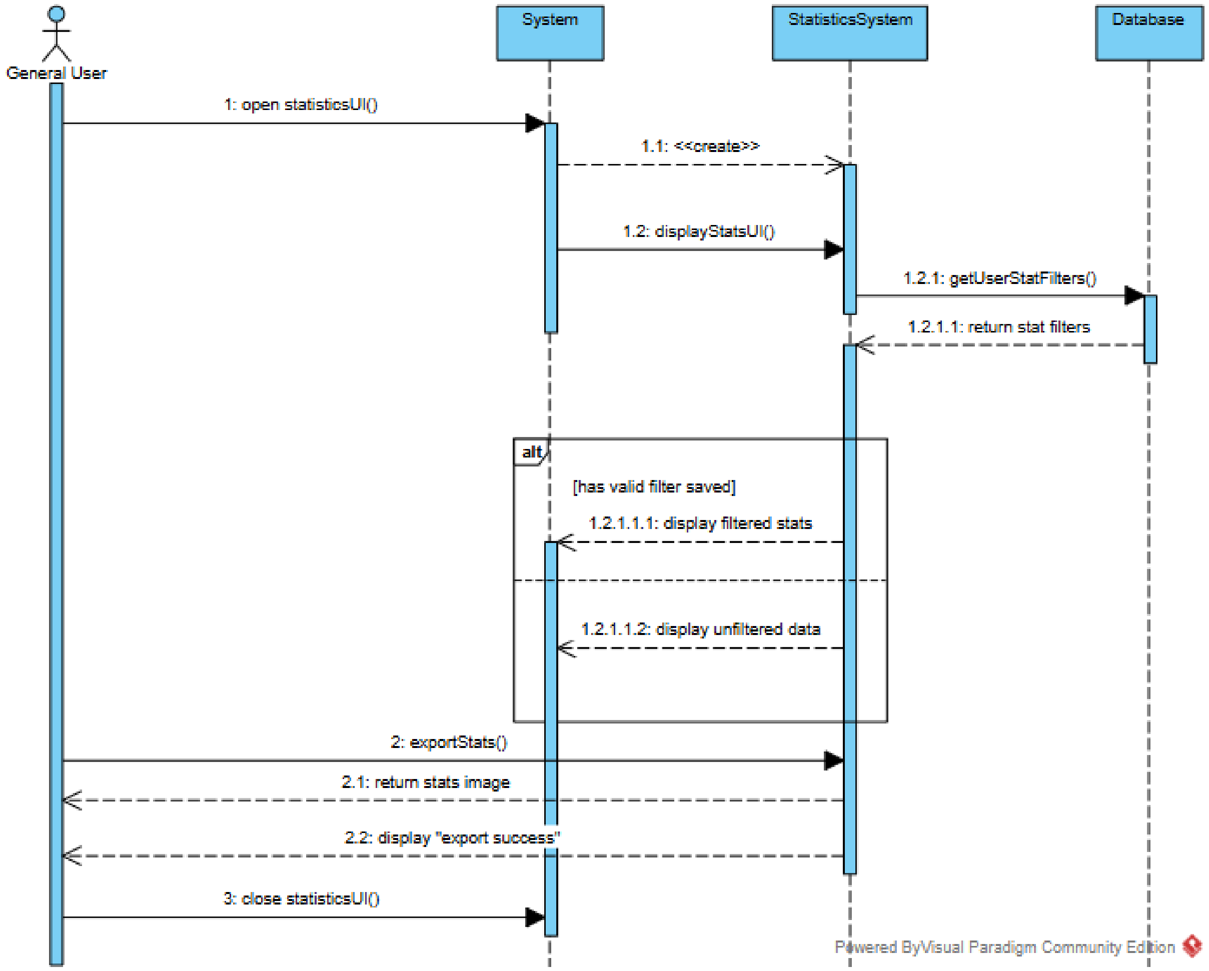

General User
Trainer
: CreateGoals
1: openGoals()
loop
[until user exits goals screen]
2: enterLength(gLength)
2.1: <<create>>
: Goal
2.2: saveLength(gLength)
3: enterValue(gValue)
3.1: saveValue(gValue)
4: enterSpecifics(gSpecific)
4.1: saveSpecific(gSpecific)
4.1.1: notifyTrainer(g)
4.1.2: displayNewGoal(g)
5: viewGoals()
5.1: displayAllGoals()
5.2: closeGoals()
Powered By: Visual Paradigm Community Edition

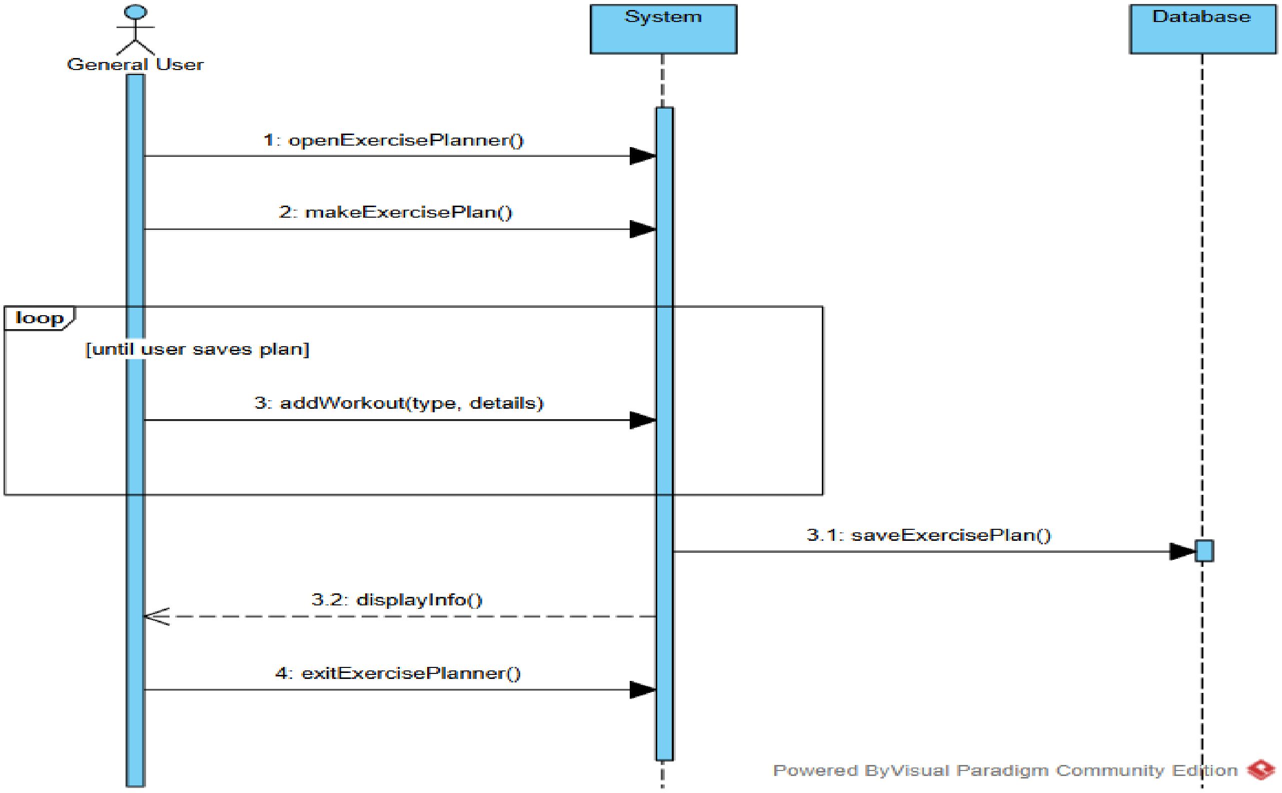

GeneralUser
: ReminderCreator
1: openReminder()
loop
[until user exits reminders screen]
2: enterTask(rTask)
2.1: <<create>>
: Reminder
2.2: saveTask(rTask)
3: enterTime(rTime)
3.1: saveTime(rTime)
3.2: displayNewReminder(r)
4: confirmReminder(r)
5: viewReminders()
5.1: displayAllReminders()
6: closeReminder()
Powered By: Visual Paradigm Community Edition

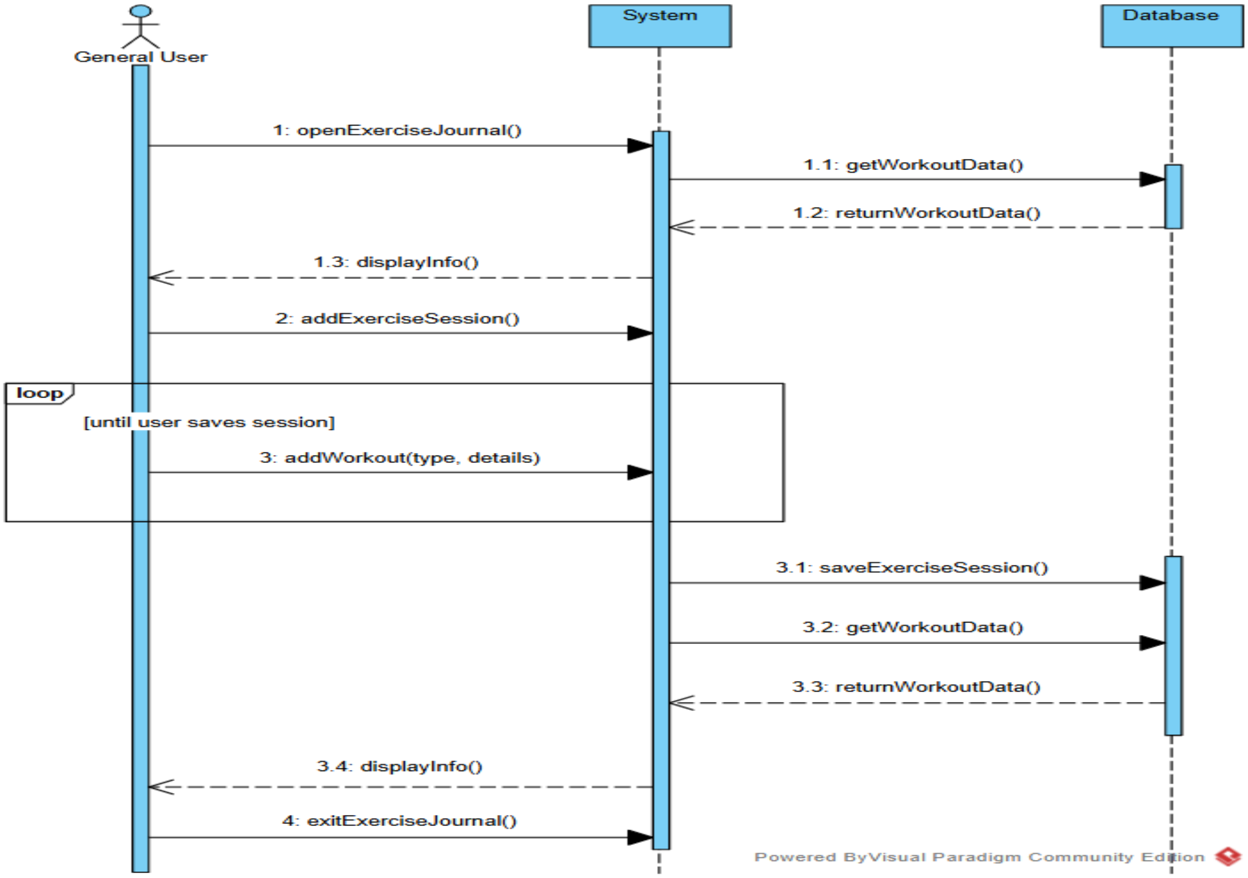

General User
: CalorieTracker
: CalorieGoal
1: openCalorieTracker()
2: enterCalories(calories)
2.1: <<create>>
: CalorieReport
opt
[if user has calorie goals]
3: updateGoals(calories)
3.1: displayGoals(goals)
2.2: subtractCalories(calories)
2.3: caloriesRemaining
4: viewReport()
4.1: displayInfo(totalCals, goals, caloriesRemaining)
5: closeCalorieTracker()
Powered By□Visual Paradigm Community Edition

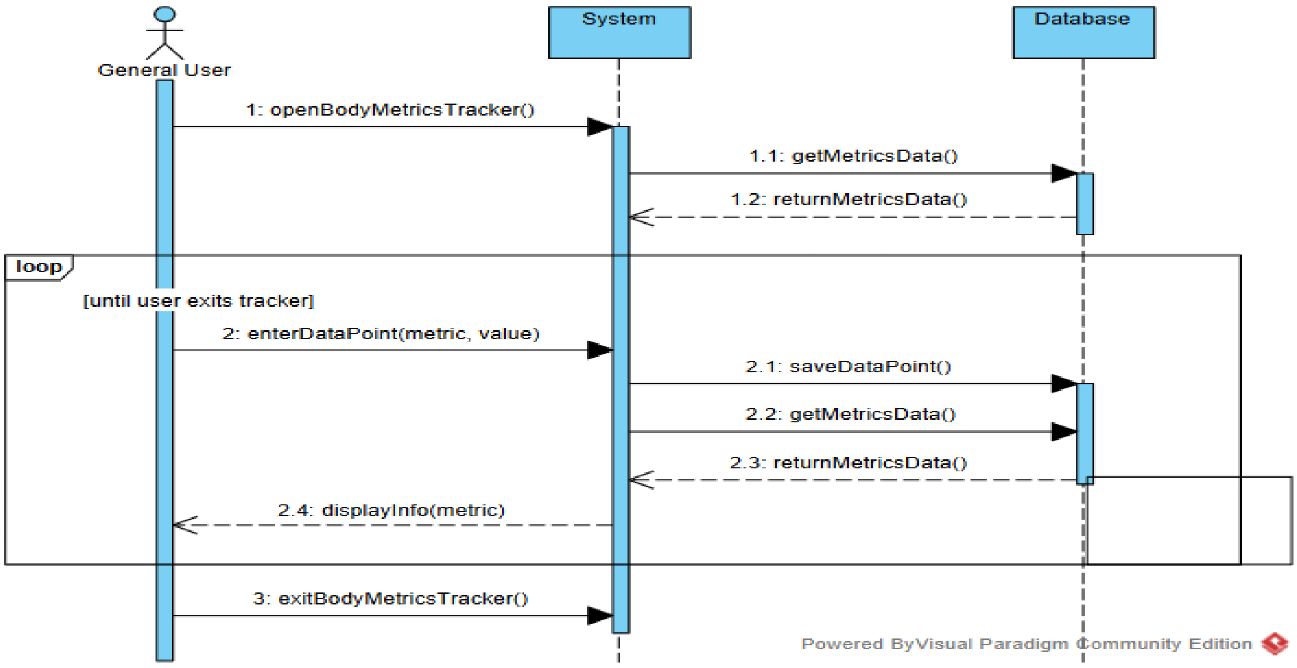

Admin
System
UserList
UserManagement
Database
1: open UserListUI()
1.1: <<create>>
1.2: display UserListUI()
1.2.1: getAllUsers()
1.2.2: return all user info
loop
[until valid user found]
2: searchUser(String username)
alt
2.1: <<create>>
[valid user found]
2.2: display manageUserUI(username)
2.3: display "invalid user"
3: resetUserPassword(newPassword)
3.1: setUserPassword(selectedUser, newPassword)
3.2: display "reset success"
4: close manageUserUI()
5: close UserListUI()
Powered ByVisual Paradigm Community Edition

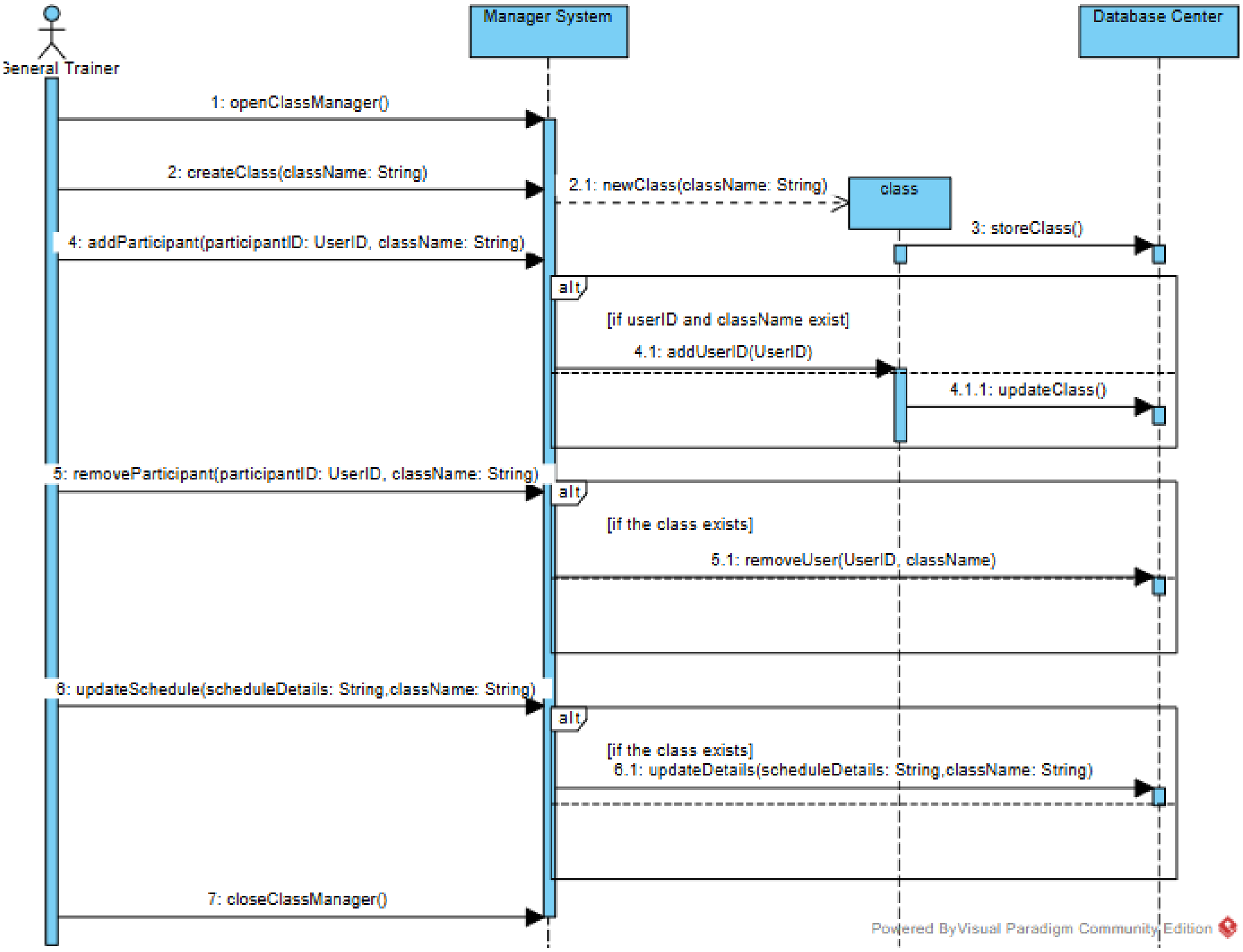

General User
System
XPSystem
Database
1: recordActivity(activity info)
1.1: <<create>>
1.1.1: getUserXP()
1.1.1.1: return userXP
1.1.1.1.1: updateXP(currentXP, addedXP, bonusXP)
2: open XPUI()
2.1: display XPUI()
3: close XPUI()
Powered ByVisual Paradigm Community Edition

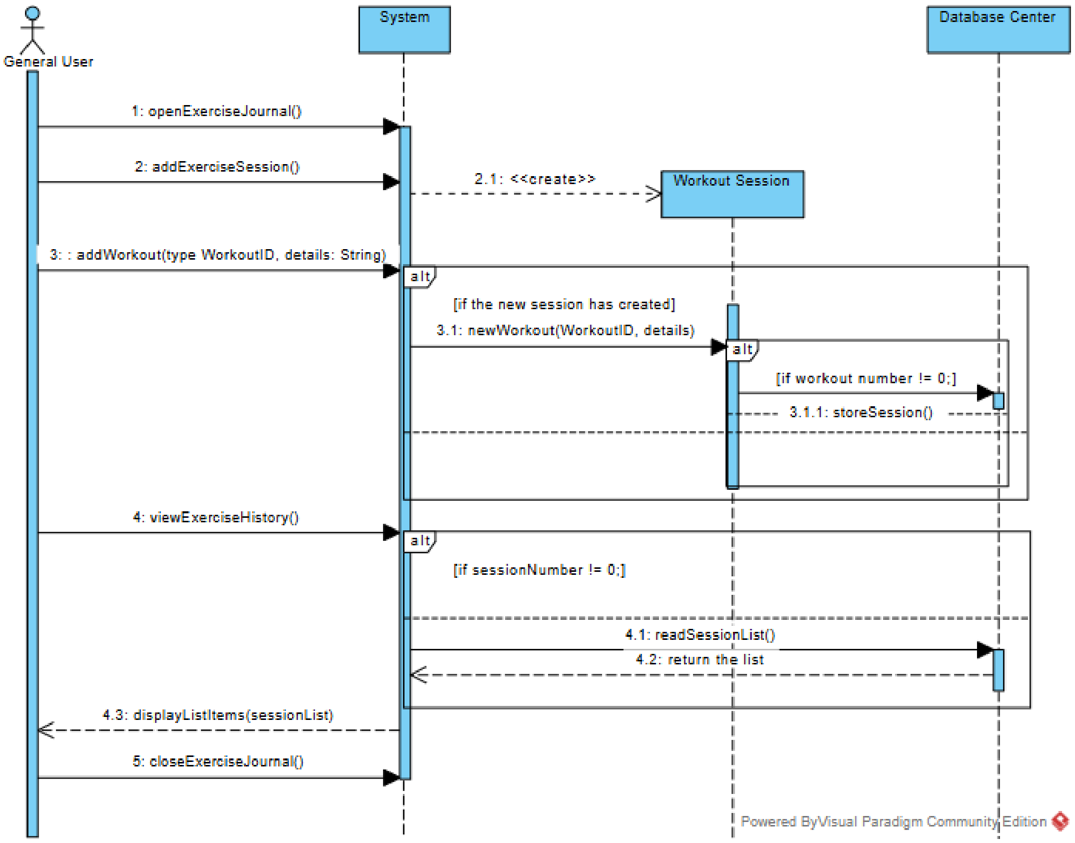

General User
System
StatisticsSystem
Database
1: open statisticsUI()
1.1: <<create>>
1.2: displayStatsUI()
1.2.1: getUserStatFilters()
1.2.1.1: return stat filters
alt
[has valid filter saved]
1.2.1.1.1: display filtered stats
1.2.1.1.2: display unfiltered data
2: exportStats()
2.1: return stats image
2.2: display "export success"
3: close statisticsUI()
Powered ByVisual Paradigm Community Edition

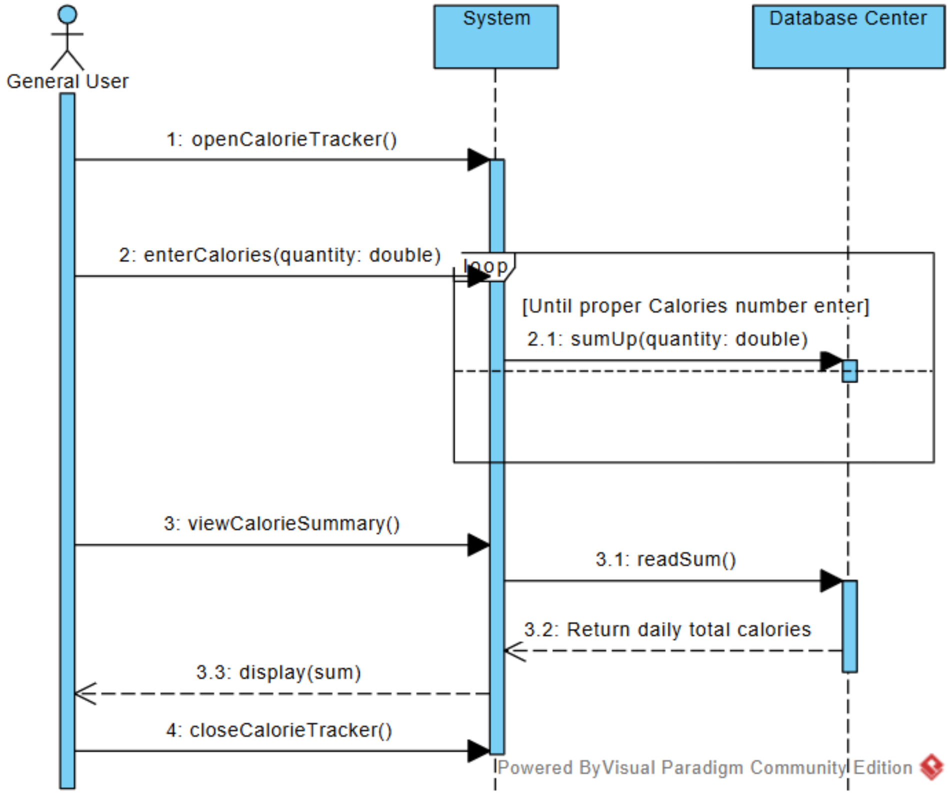

General User
System
Database
1: openExercisePlanner()
2: makeExercisePlan()
loop
[until user saves plan]
3: addWorkout(type, details)
3.1: saveExercisePlan()
3.2: displayInfo()
4: exitExercisePlanner()
Powered ByVisual Paradigm Community Edition

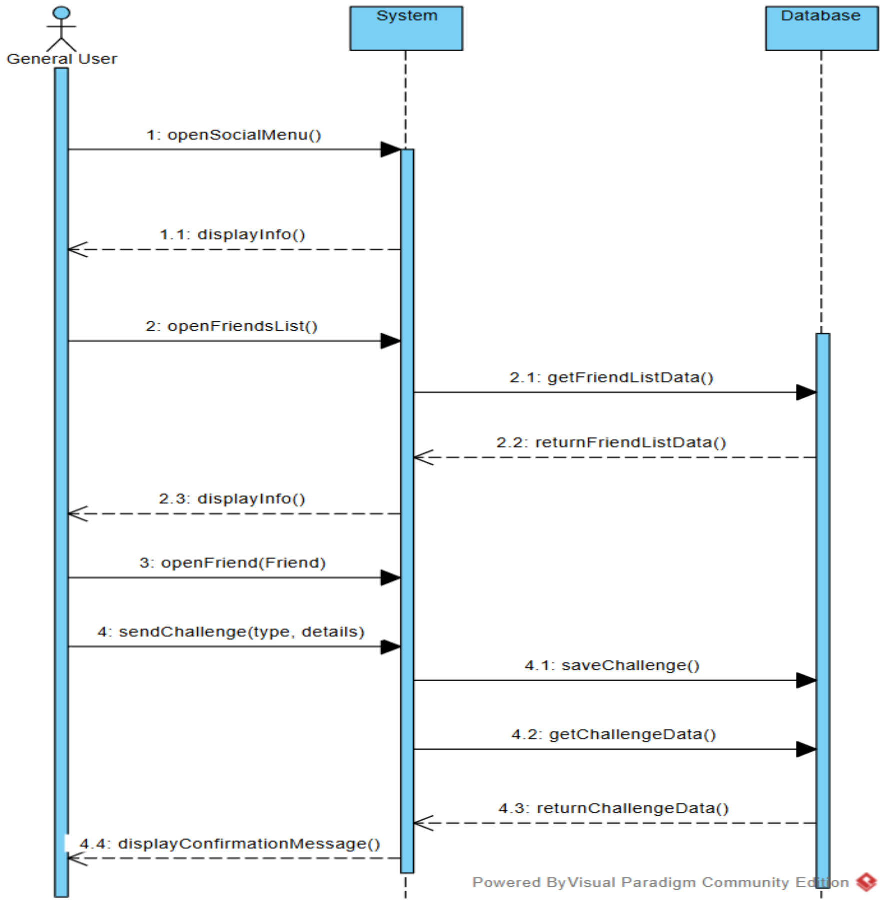

General User
System
Database
1: openExerciseJournal()
1.1: getWorkoutData()
1.2: returnWorkoutData()
1.3: displayInfo()
2: addExerciseSession()
loop
[until user saves session]
3: addWorkout(type, details)
3.1: saveExerciseSession()
3.2: getWorkoutData()
3.3: returnWorkoutData()
3.4: displayInfo()
4: exitExerciseJournal()
Powered ByVisual Paradigm Community Edition

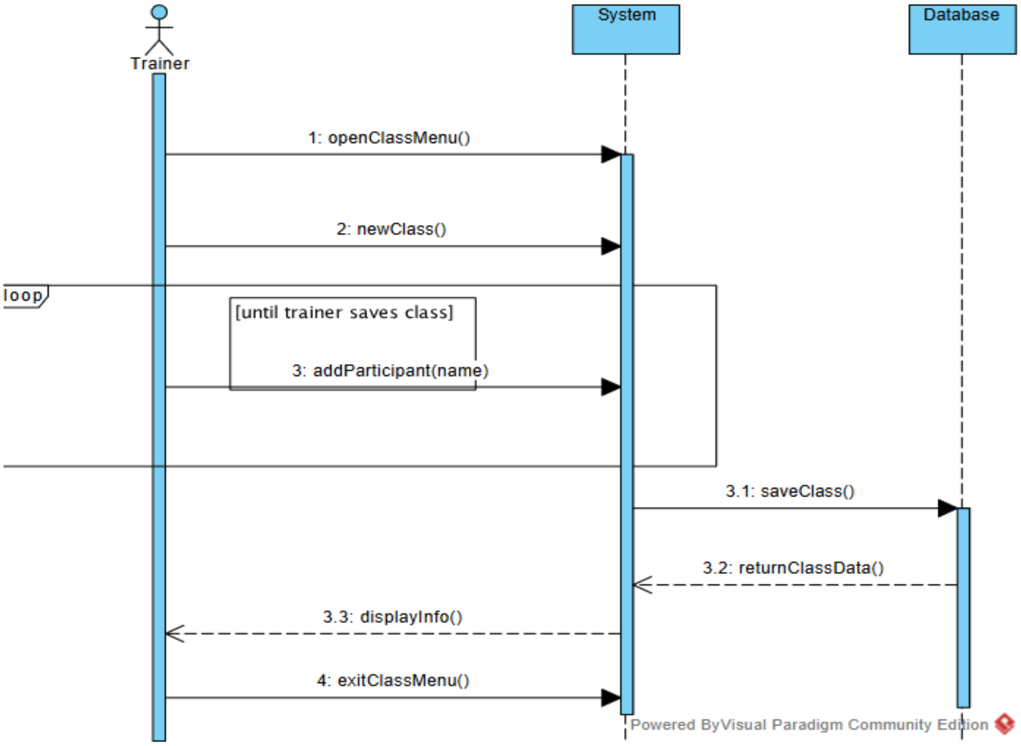

General User
System
Database
1: openBodyMetricsTracker()
1.1: getMetricsData()
1.2: returnMetricsData()
loop
[until user exits tracker]
2: enterDataPoint(metric, value)
2.1: saveDataPoint()
2.2: getMetricsData()
2.3: returnMetricsData()
2.4: displayInfo(metric)
3: exitBodyMetricsTracker()
Powered ByVisual Paradigm Community Edition

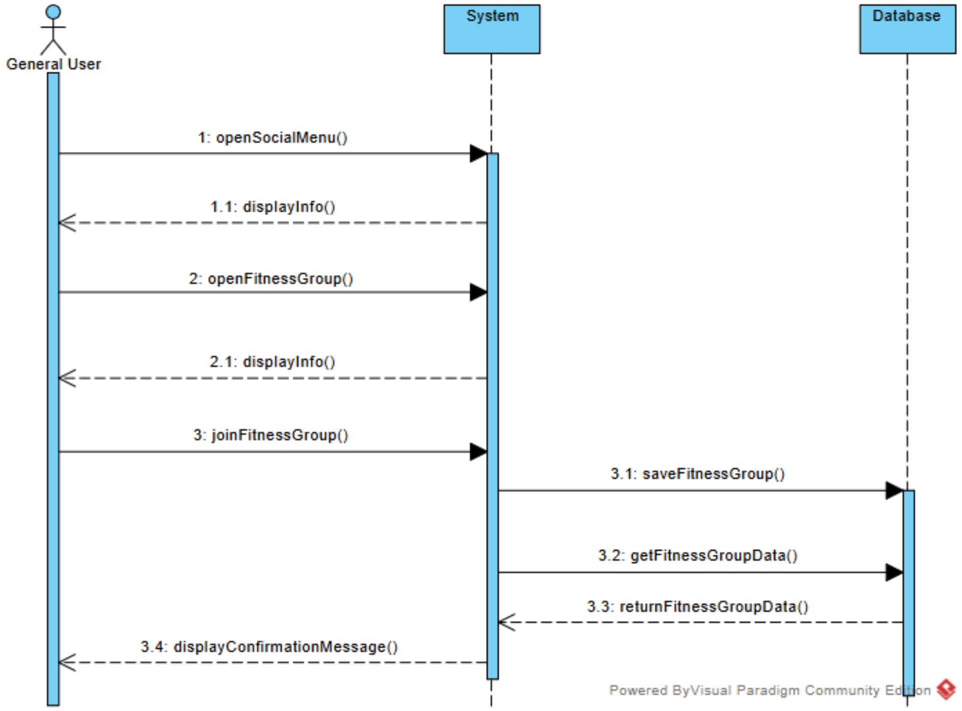

General Trainer
Manager System
Database Center
1: openClassManager()
2: createClass(className: String)
2.1: newClass(className: String)
class
3: storeClass()
4: addParticipant(participantID: UserID, className: String)
alt
[if userID and className exist]
4.1: addUserID(UserID)
4.1.1: updateClass()
5: removeParticipant(participantID: UserID, className: String)
alt
[if the class exists]
5.1: removeUser(UserID, className)
6: updateSchedule(scheduleDetails: String,className: String)
alt
[if the class exists]
6.1: updateDetails(scheduleDetails: String,className: String)
7: closeClassManager()
Powered ByVisual Paradigm Community Edition


General User
System
Database Center
1: openExerciseJournal()
2: addExerciseSession()
2.1: <<create>>
Workout Session
3: : addWorkout(type WorkoutID, details: String)
alt
[if the new session has created]
3.1: newWorkout(WorkoutID, details)
alt
[if workout number != 0;]
3.1.1: storeSession()
4: viewExerciseHistory()
alt
[if sessionNumber != 0;]
4.1: readSessionList()
4.2: return the list
4.3: displayListItems(sessionList)
5: closeExerciseJournal()
Powered ByVisual Paradigm Community Edition

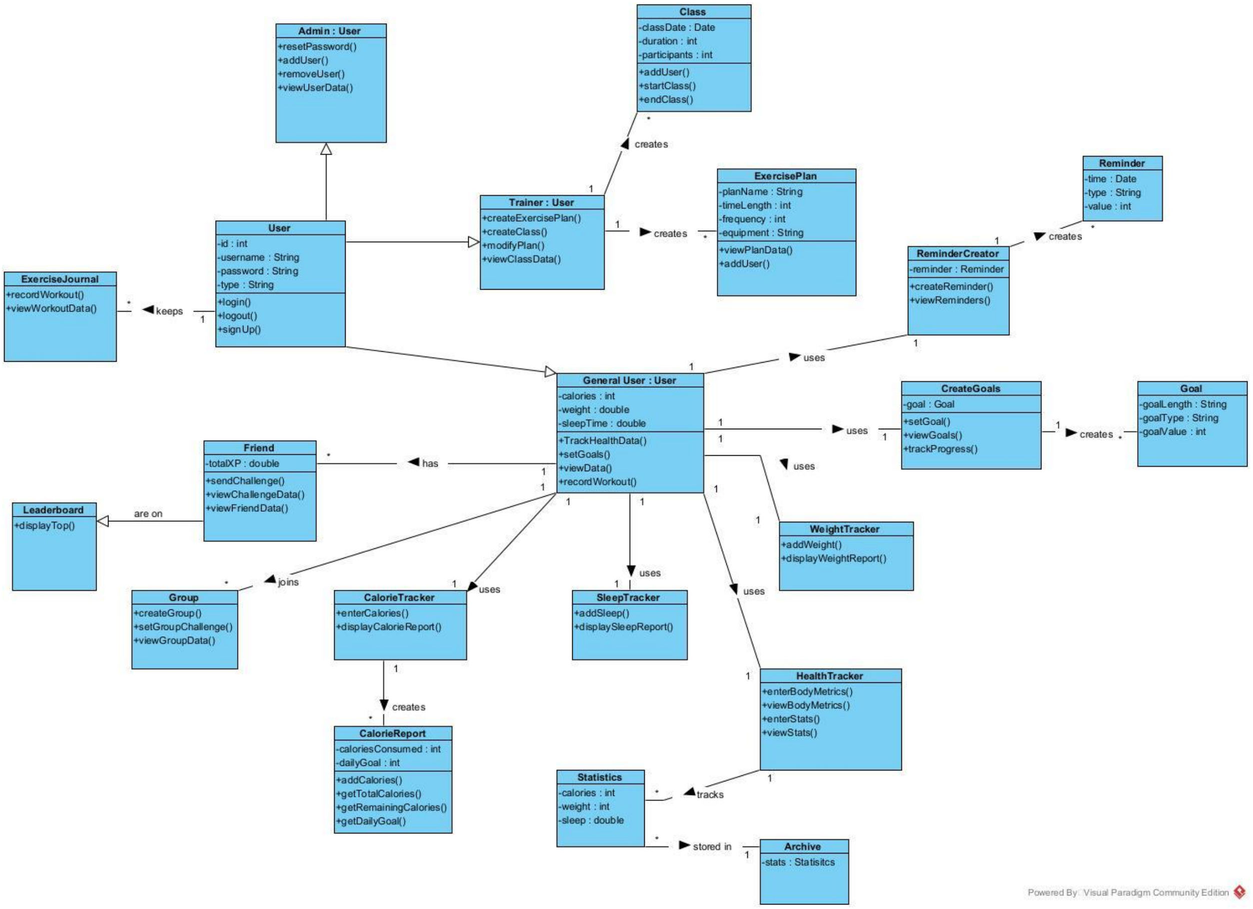

General User
System
Database Center
1: openCalorieTracker()
2: enterCalories(quantity: double)
loop
[Until proper Calories number enter]
2.1: sumUp(quantity: double)
3: viewCalorieSummary()
3.1: readSum()
3.2: Return daily total calories
3.3: display(sum)
4: closeCalorieTracker()
Powered ByVisual Paradigm Community Edition


General User
System
Database
1: openSocialMenu()
1.1: displayInfo()
2: openFriendsList()
2.1: getFriendListData()
2.2: returnFriendListData()
2.3: displayInfo()
3: openFriend(Friend)
4: sendChallenge(type, details)
4.1: saveChallenge()
4.2: getChallengeData()
4.3: returnChallengeData()
4.4: displayConfirmationMessage()
Powered ByVisual Paradigm Community Edition


Trainer
System
Database
1: openClassMenu()
2: newClass()
loop
[until trainer saves class]
3: addParticipant(name)
3.1: saveClass()
3.2: returnClassData()
3.3: displayInfo()
4: exitClassMenu()
Powered ByVisual Paradigm Community Edition


General User
System
Database
1: openSocialMenu()
1.1: displayInfo()
2: openFitnessGroup()
2.1: displayInfo()
3: joinFitnessGroup()
3.1: saveFitnessGroup()
3.2: getFitnessGroupData()
3.3: returnFitnessGroupData()
3.4: displayConfirmationMessage()
Powered ByVisual Paradigm Community Edition

Design Class Diagram:


Admin : User
+resetPassword()
+addUser()
+removeUser()
+viewUserData()
Class
-classDate : Date
-duration : int
-participants : int
+addUser()
+startClass()
+endClass()
*
creates
1
Trainer : User
+createExercisePlan()
+createClass()
+modifyPlan()
+viewClassData()
1
creates
*
ExercisePlan
-planName : String
-timeLength : int
-frequency : int
-equipment : String
+viewPlanData()
+addUser()
Reminder
-time : Date
-type : String
-value : int
*
creates
1
ReminderCreator
-reminder : Reminder
+createReminder()
+viewReminders()
1
uses
1
User
-id : int
-username : String
-password : String
-type : String
+login()
+logout()
+signUp()
ExerciseJournal
+recordWorkout()
+viewWorkoutData()
*
keeps
1
General User : User
-calories : int
-weight : double
-sleepTime : double
+TrackHealthData()
+setGoals()
+viewData()
+recordWorkout()
1
1
uses
1
CreateGoals
-goal : Goal
+setGoal()
+viewGoals()
+trackProgress()
1
creates
*
Goal
-goalLength : String
-goalType : String
-goalValue : int
Friend
-totalXP : double
+sendChallenge()
+viewChallengeData()
+viewFriendData()
*
has
1
1
uses
1
1
WeightTracker
+addWeight()
+displayWeightReport()
Leaderboard
+displayTop()
are on
1
1
*
joins
Group
+createGroup()
+setGroupChallenge()
+viewGroupData()
1
uses
CalorieTracker
+enterCalories()
+displayCalorieReport()
uses
1
SleepTracker
+addSleep()
+displaySleepReport()
uses
1
HealthTracker
+enterBodyMetrics()
+viewBodyMetrics()
+enterStats()
+viewStats()
1
creates
*
CalorieReport
-caloriesConsumed : int
-dailyGoal : int
+addCalories()
+getTotalCalories()
+getRemainingCalories()
+getDailyGoal()
1
tracks
*
Statistics
-calories : int
-weight : int
-sleep : double
*
stored in
1
Archive
-stats : Statisitcs
Powered By: Visual Paradigm Community Edition