Not so Beary Fat Iteration 2

Group Name: Not so Beary Fat

Members:

Lademi Aromolaran: requirements engineer

Jonah Beck: design engineer

Zach Taylor: project manager

Olivia Calusinski: Project librarian

Owen Chipman: Quality Assurance

Github Repo: https://github.com/ocalusinski/NotSoBearyFat

Jira Repo: https://notsobearyfat.atlassian.net/jira/core/projects/NSBF/board?filter=&groupBy=status&isEligibleForUserSurvey=true&isEligibleForUserSurvey=true

Our Website: https://ocalusinski.github.io/NotSoBearyFat/

Current Implementation Status:

the ability to create an account for both a user and a trainer, ability to create a class and add it to the database.

Planned Scope:

implement the ability for users to view their historical data, allow trainers to create recurring meetings, allow the login and password information of users to be changed

Main Roadblocks:

Integrating all the different use cases into a cohesive program, configuring pom.xml file to download the correct dependencies for database and database driver, resolving coloring conflicts between different pages

Current Implementation Status:

Everything is currently separate from each other and not connected to the application layer.

A general user can track calories, sleep, and weight: mostly finished, need to figure out how to have an optional input. Next step is to use the data to create a graph (Olivia)

Admin portal: basic UI right now, still adjusting buttons and figuring out how to create routing logic (Owen)

View dashboard: basic dashboard UI, not integrated with other use cases yet so it cannot display more information yet (Lademi)

Login: not connected to application layer but is mostly done (Jonah)

Trainer creates a class: Skeleton of a trainer creating a class (Zach

Roadblocks:

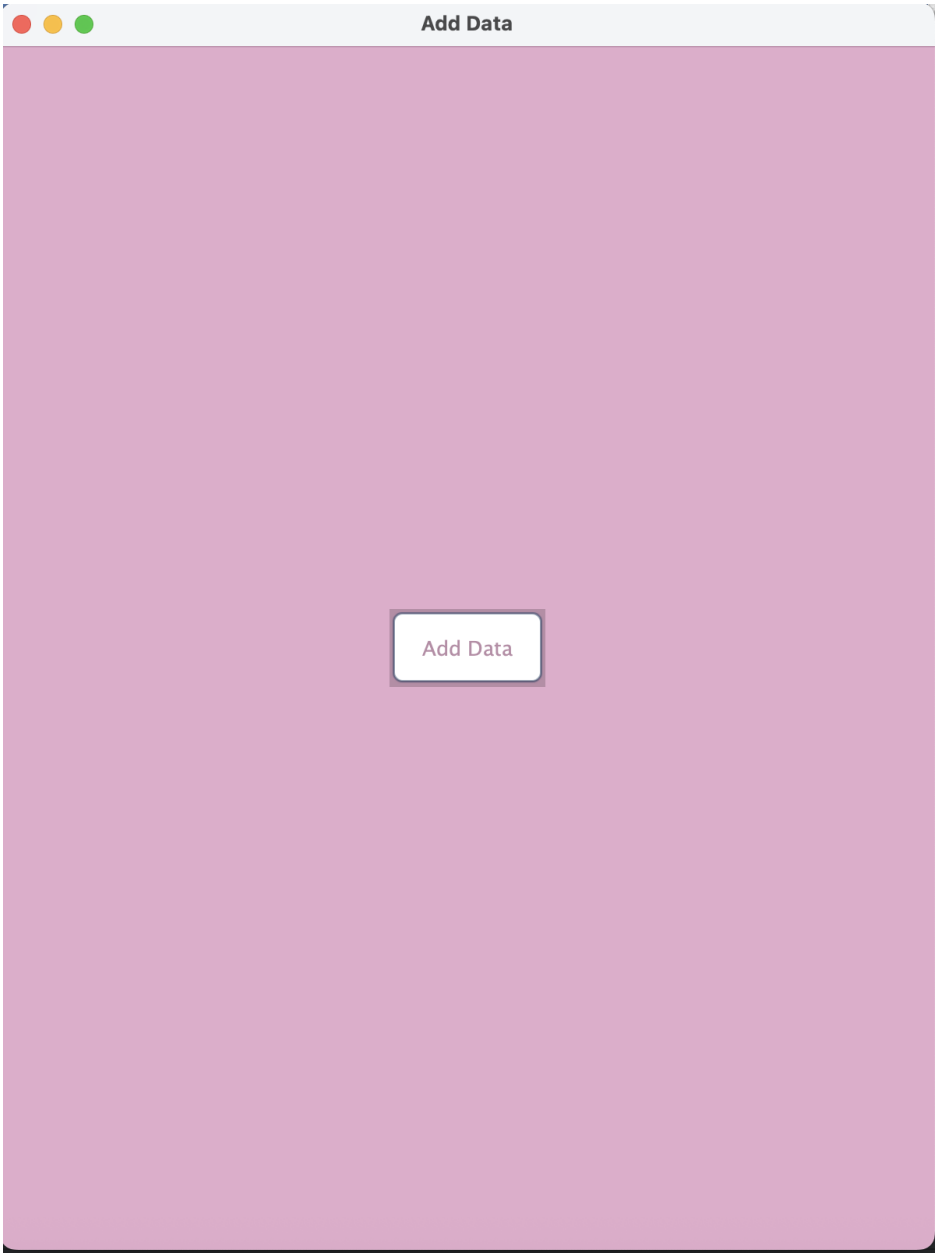
It is taking a long time to code because we are all inexperienced with java Swing

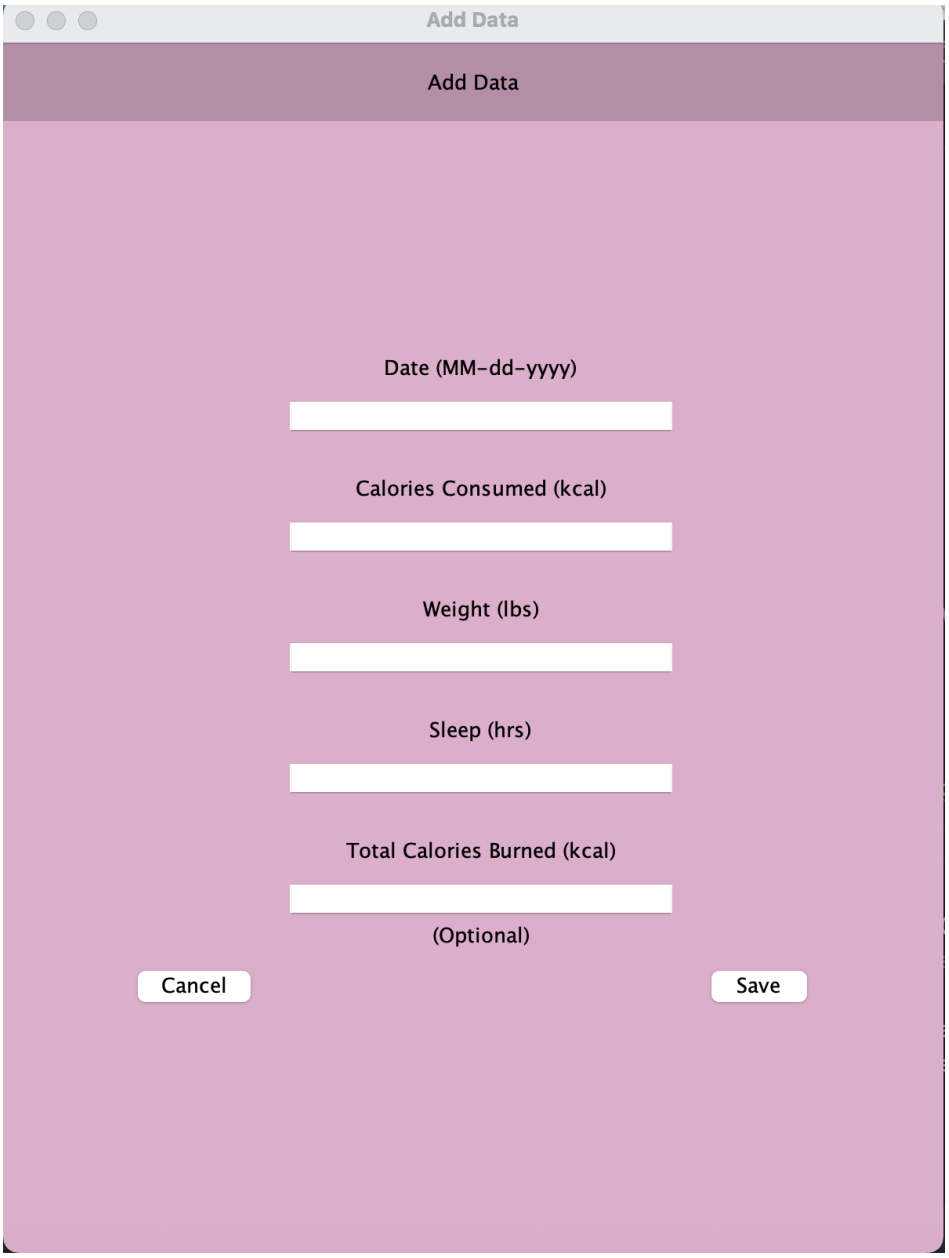
Need to make more time to meet up as a group to integrate everything together.

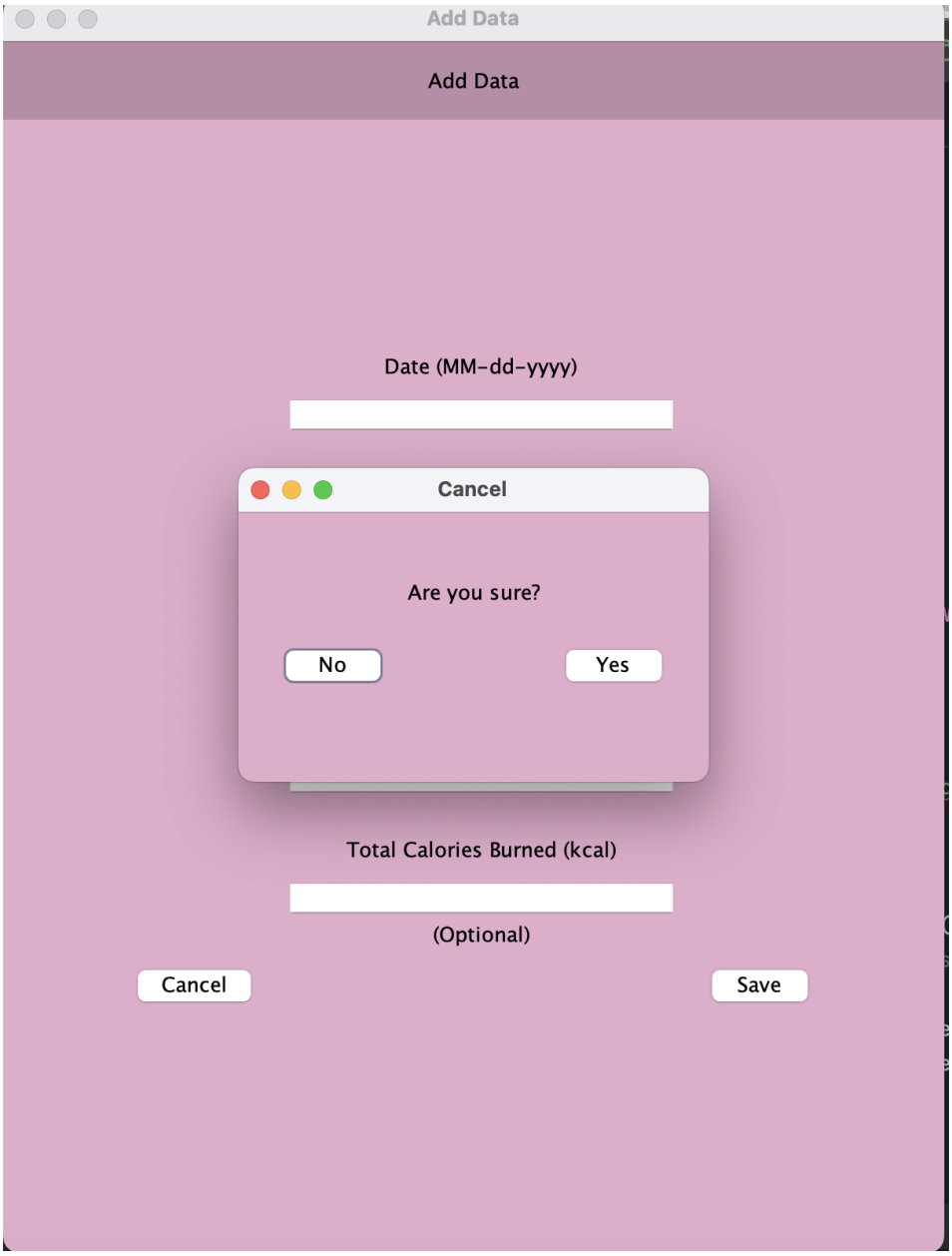
https://notsobearyfat.atlassian.net/jira/core/projects/NSBF/board?filter=&groupBy=status

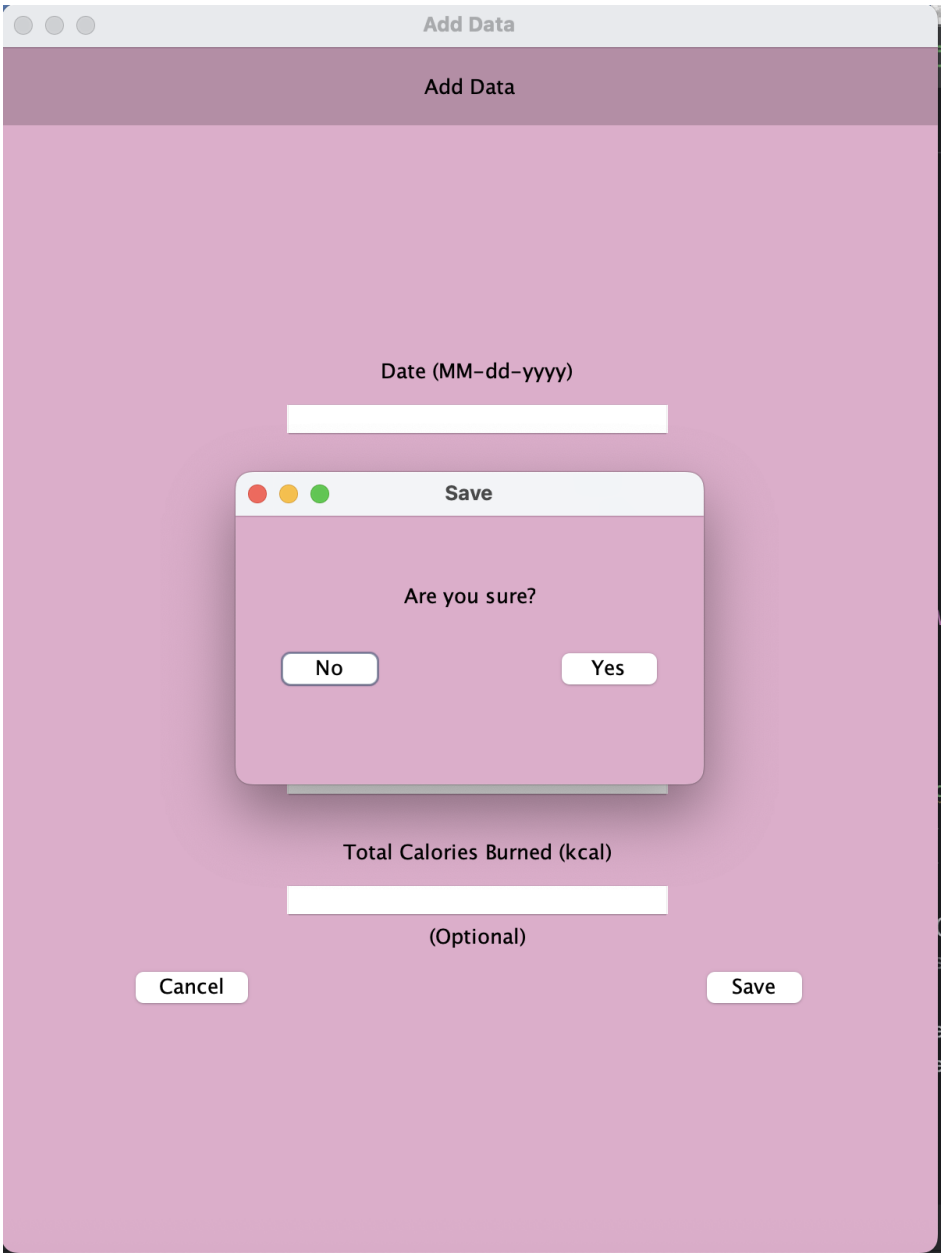
Use Case: A general user can track calorie intake, weight, and sleep (Olivia)

Not currently connected with application layer




Add Data
Add Data
Date (MM-dd-yyyy)
Calories Consumed (kcal)
Weight (lbs)
Sleep (hrs)
Total Calories Burned (kcal)
(Optional)
Cancel
Save

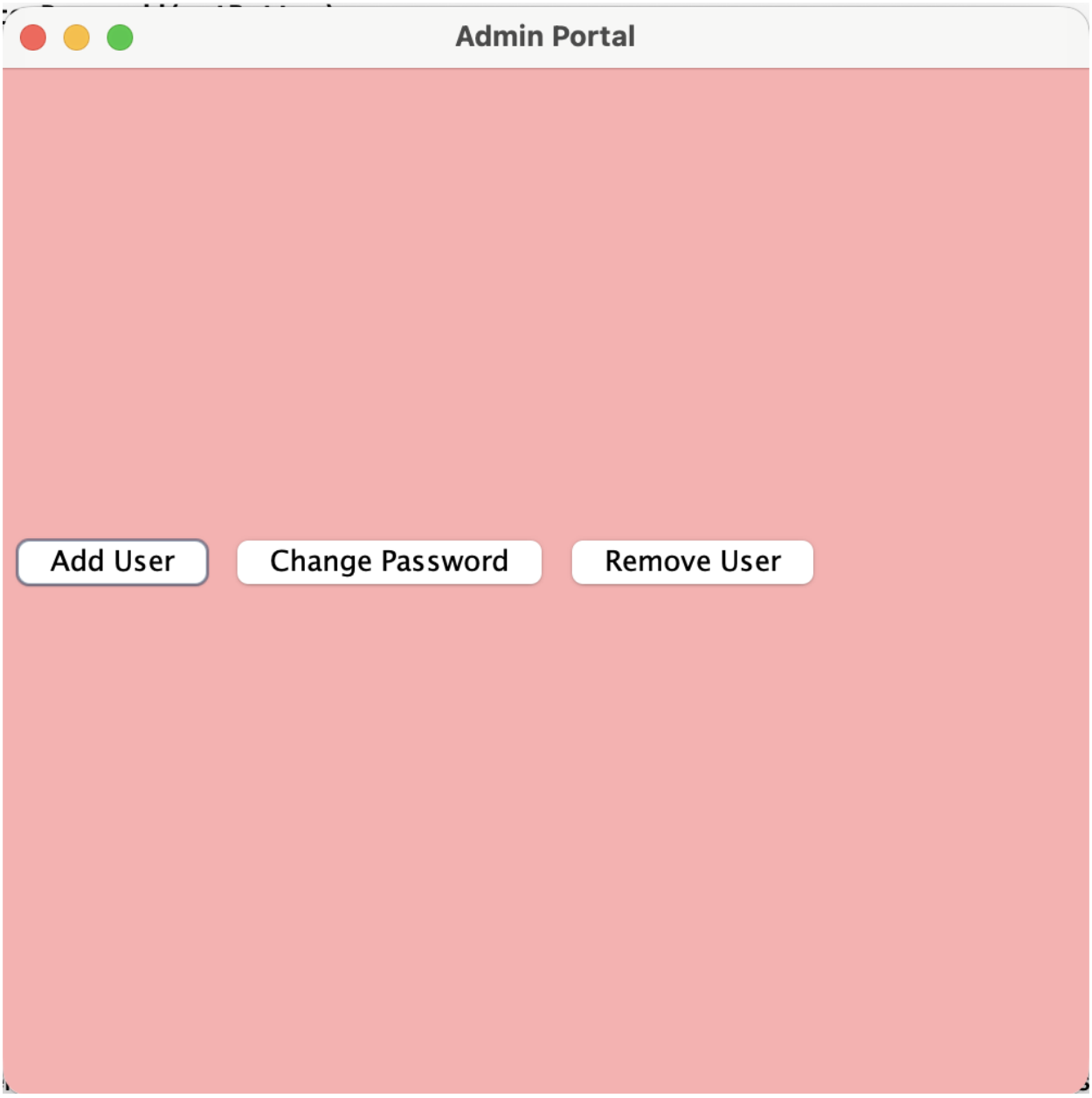

Add Data
Add Data
Date (MM-dd-yyyy)
Cancel
Are you sure?
No
Yes
Total Calories Burned (kcal)
(Optional)
Cancel
Save


Add Data
Add Data
Date (MM-dd-yyyy)
Save
Are you sure?
No
Yes
Total Calories Burned (kcal)
(Optional)
Cancel
Save

Owen Chipman

Use Case: Admin Portal so Admin can Add, Remove, or Update Password of a User

Notes: Not connected to backend logic yet. Still need to fix layout so everything is symmetric and create routing logic



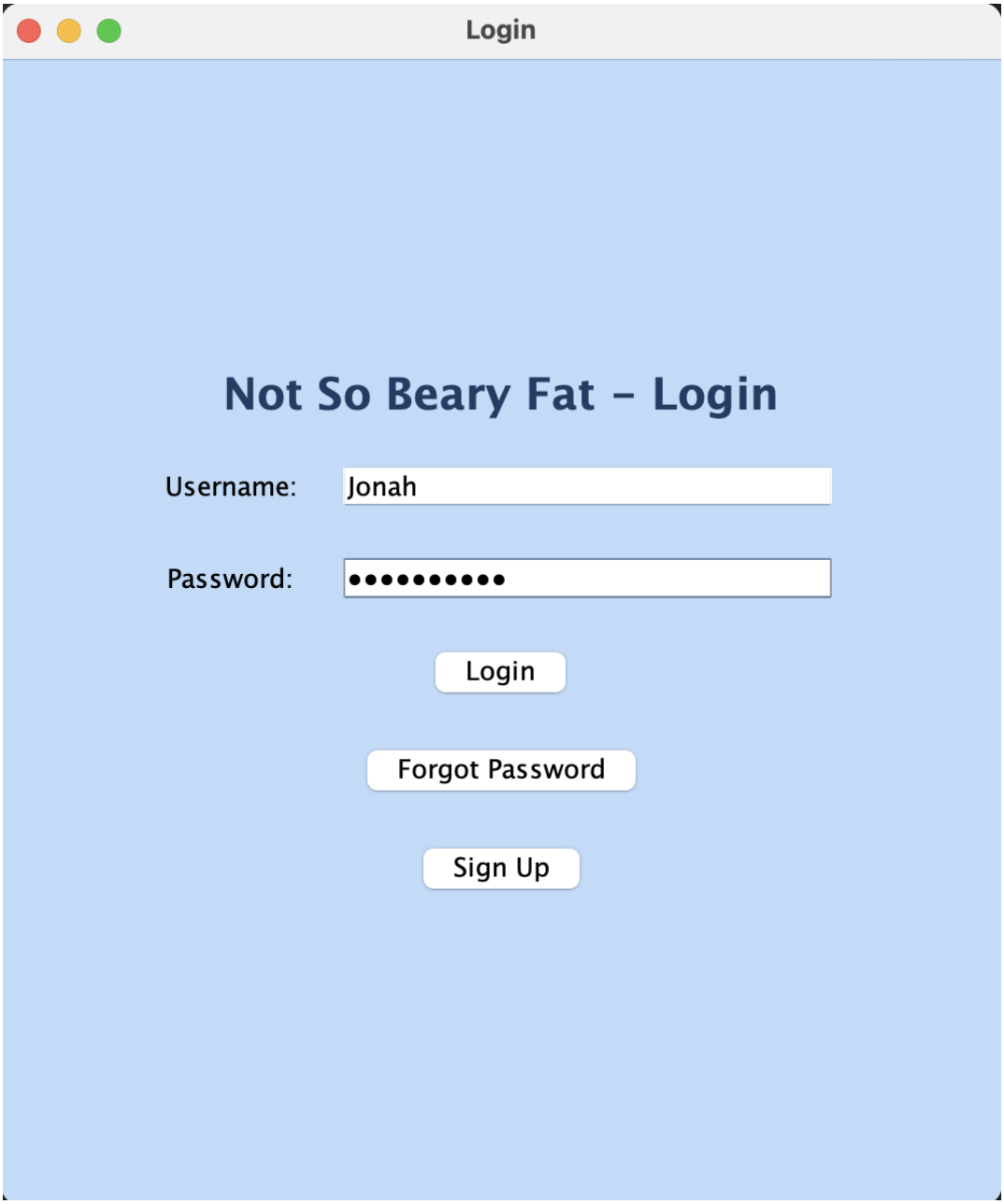
Use Case: View Dashboard (Lademi)

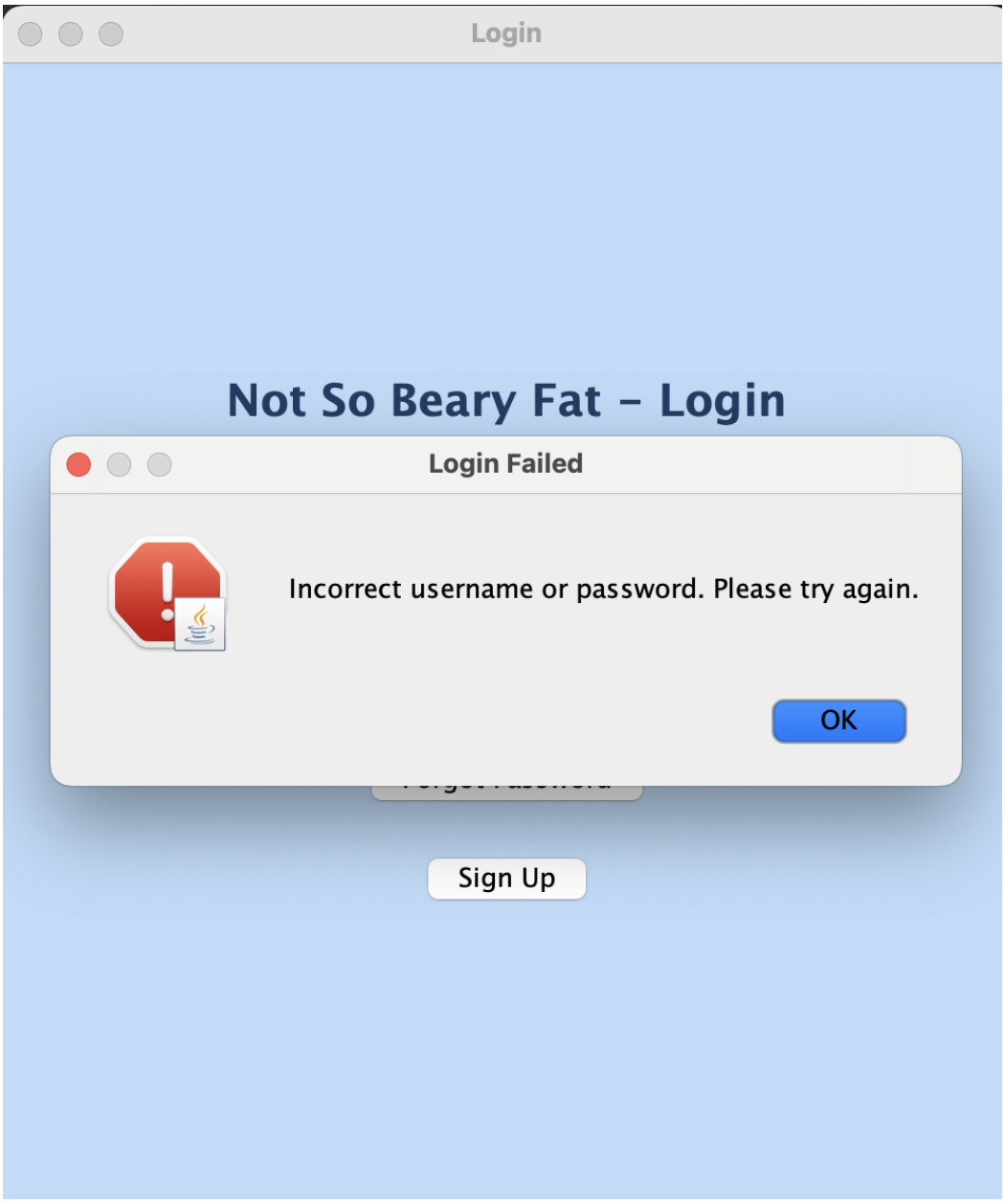
Not currently integrated with application layer

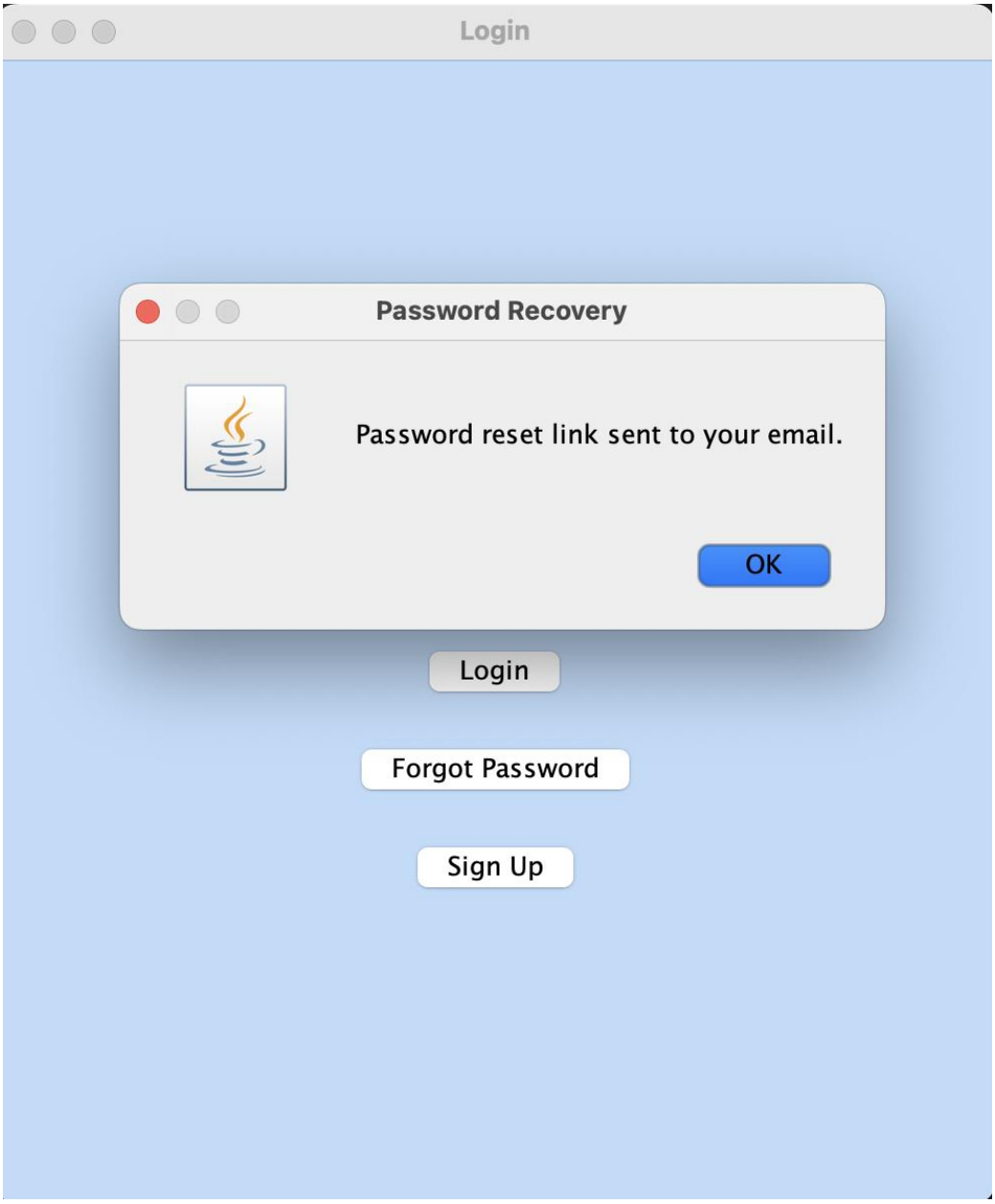
Author: Jonah Beck

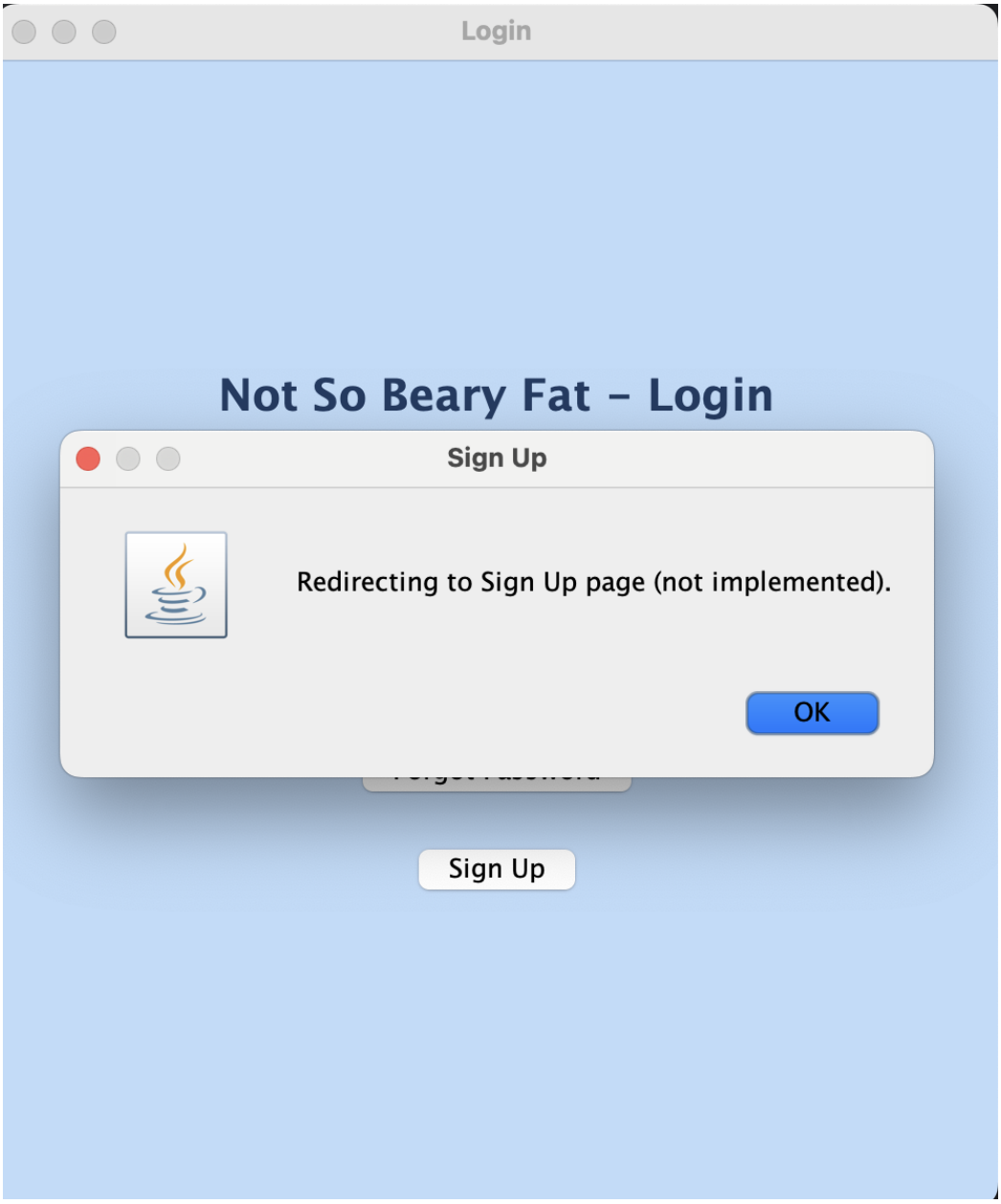
Use Case: User Login

Not currently connected to application layer




Login
Not So Beary Fat - Login
Login Failed
Incorrect username or password. Please try again.
OK
Sign Up

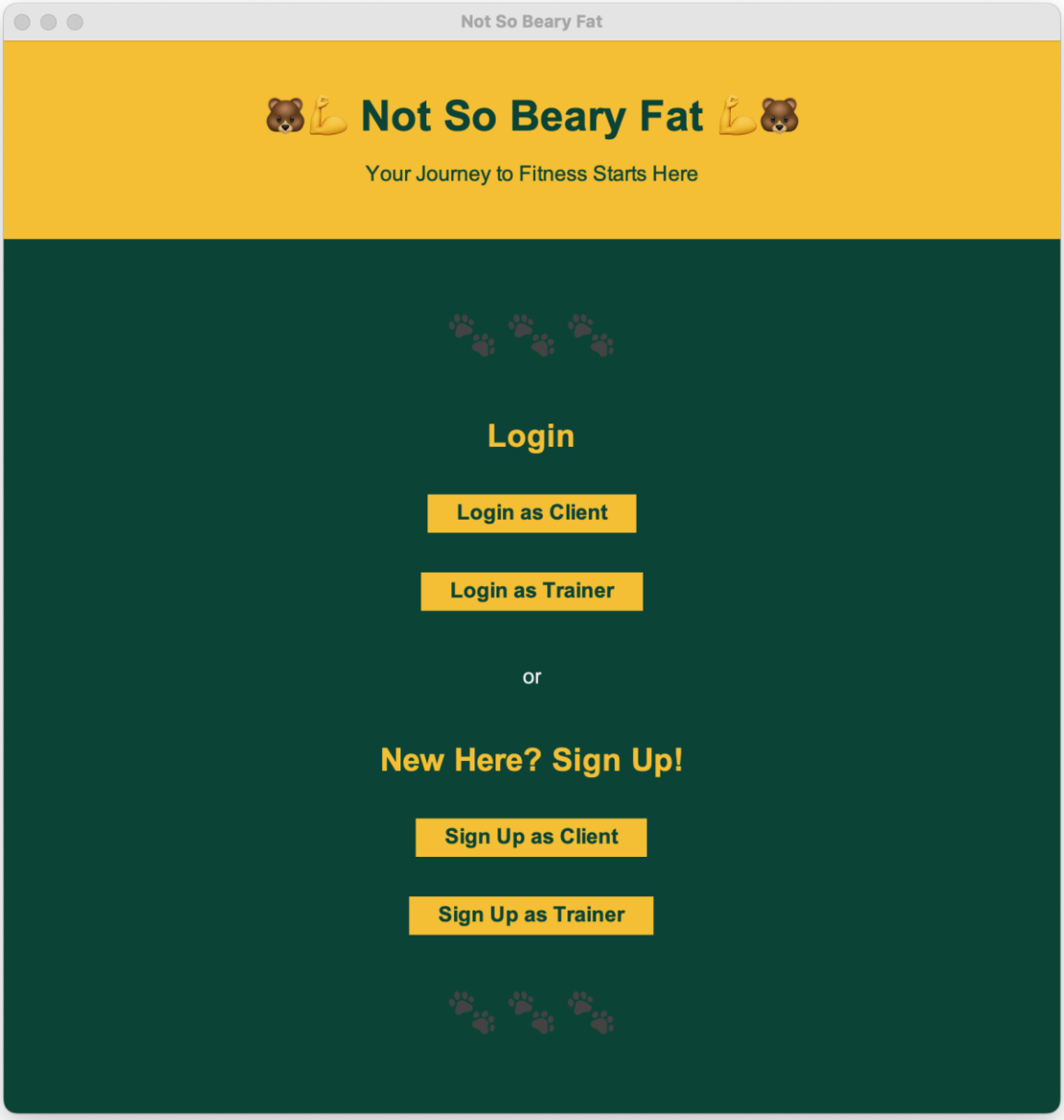

Login
Password Recovery
Password reset link sent to your email.
OK
Login
Forgot Password
Sign Up


Login
Not So Beary Fat - Login
Sign Up
Redirecting to Sign Up page (not implemented).
OK
Sign Up

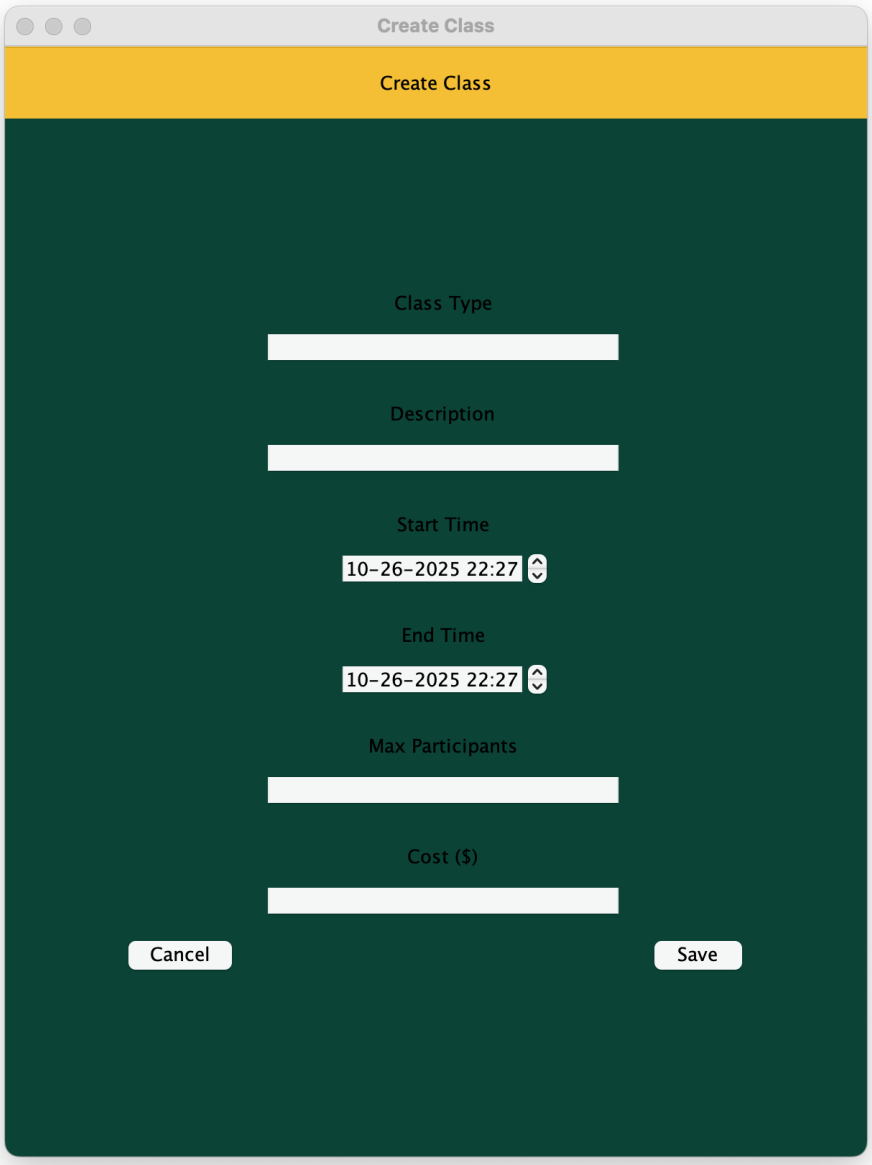
Use Case: Trainer makes a class (Zach)

Not connected to application layer yet

(Another option for home page)




Create Class
Create Class


Create Class
Create Class
Class Type
Description
Start Time
10-26-2025 22:27
End Time
10-26-2025 22:27
Max Participants
Cost ($)
Cancel
Save

Author: Lademi Aromolaran

Use Case: View Dashboard

Not currently connected to application layer

Task Board:

https://notsobearyfat.atlassian.net/jira/core/projects/NSBF/board?filter=&groupBy=status

Website:

https://ocalusinski.github.io/NotSoBearyFat/

Current Use Cases:

- Trainer making classes
- User setting goals
- User making friends
- User workout search
- User log in and out
- Self-paced exercise plans
- Dashboard of users' progress
- General User recording workout
- User tracking data (calories, sleep, etc.)
- User can see historical data
- User can log in
- Admin user management
- Trainer statistics of self-paced exercises
- Trainer can modify an existing class or plan
- User can register for a trainer led class

Use Case: A trainer creates a class (Zach)

ID: Create Class

Scope: User Portal

Level: User goal

Stakeholders and Interests:

- **Trainer (actor):**
  Can create and manage classes for users to sign up for.
- **User:**
  Can view available classes and register for them.

- **System:**
  Checks for scheduling conflicts, saves class details, updates the calendar, and confirms class creation.

Precondition: Trainer is logged into the system

Postcondition: A new class is saved and displayed on the class calendar for users to view and register for

Main Success Scenario:

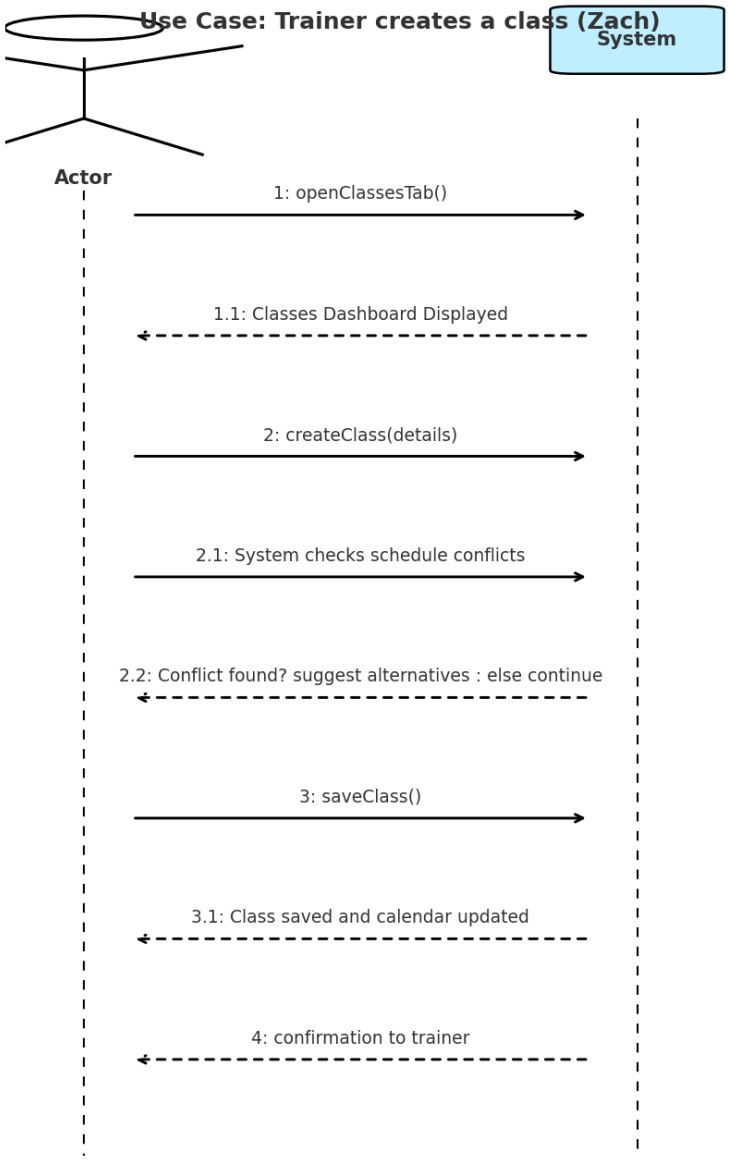
1. Trainer logs into the system
2. Trainer clicks on the “Classes” tab
3. Trainer selects “Create Class”
4. System prompts the trainer for class details (type of workout, description, max participants, time, and cost)
5. System checks for scheduling conflicts with existing classes
6. No conflict is found
7. Trainer clicks “Save”
8. System saves the new class and updates the class calendar
9. Trainer receives a confirmation message that the class has been registered

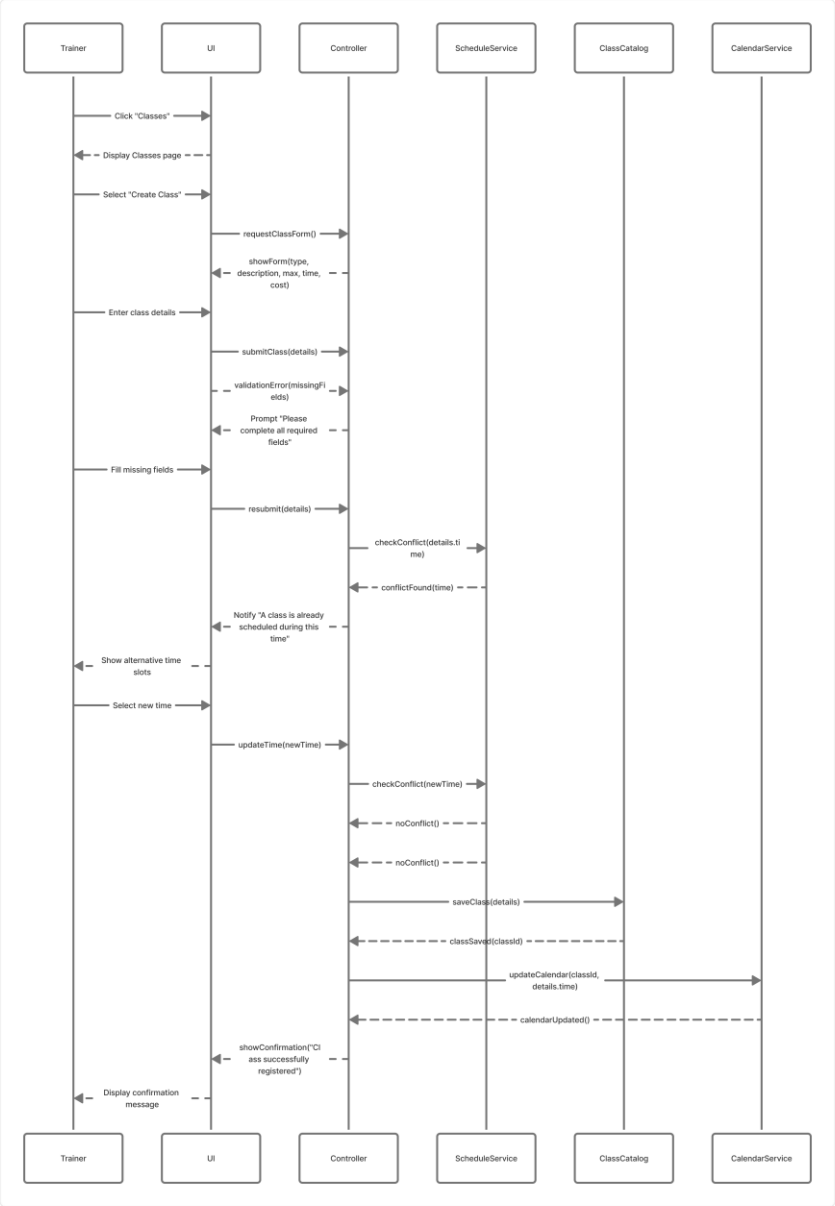
Alternate Scenario 1: Scheduling Conflict

1. Complete steps 1–4 of the main scenario
2. System detects that another class is already scheduled during the selected time
3. System notifies the trainer: “A class is already scheduled during this time”
4. System suggests alternative available time slots
5. Trainer selects a new time
6. Continue at step 7 of the main scenario

Alternate Scenario 2: Missing Required Information

1. Complete steps 1–4 of the main scenario
2. Trainer leaves out a required field (e.g., time, class type, or max participants)
3. System prompts: “Please complete all required fields before saving”
4. Trainer fills in missing information
5. Continue at step 7 of the main scenario


Use Case: Trainer creates a class (Zach)
System
Actor
1: openClassesTab()
1.1: Classes Dashboard Displayed
2: createClass(details)
2.1: System checks schedule conflicts
2.2: Conflict found? suggest alternatives : else continue
3: saveClass()
3.1: Class saved and calendar updated
4: confirmation to trainer



Operation: createClass(type: String, description: String, startTime: DateTime, endTime: DateTime, maxParticipants: int, cost: double)

Cross Reference: Trainer Create Class use case

Pre-conditions:
Trainer is logged in and currently on the “Classes” tab

Post-conditions:

- A new Class instance trainercls is created (instance creation)

- trainercls is added to the system calendar (association formed)
- trainercls is associated with the trainer (association formed)
- Confirmation message sent to trainer (state change)

Use Case: A user earns achievements or badges (Zach)

ID: Earn Achievements

Scope: Workout App

Level: User goal

Stakeholders and Interests:

- **User (actor):**
  Wants recognition for progress and milestones (e.g., completing workouts, meeting goals).
- **System:**
  Tracks user actions and awards badges when criteria are met.
- **Trainer:**
  Can view user badges to monitor progress.
- **Admin:**
  Ensures achievements are awarded accurately and manages badge criteria.

Precondition:
User is logged into the system and performing trackable actions (e.g., workouts, goals, data logs).

Postcondition:
A new badge is awarded and displayed on the user's dashboard under "Achievements."

Main Success Scenario:

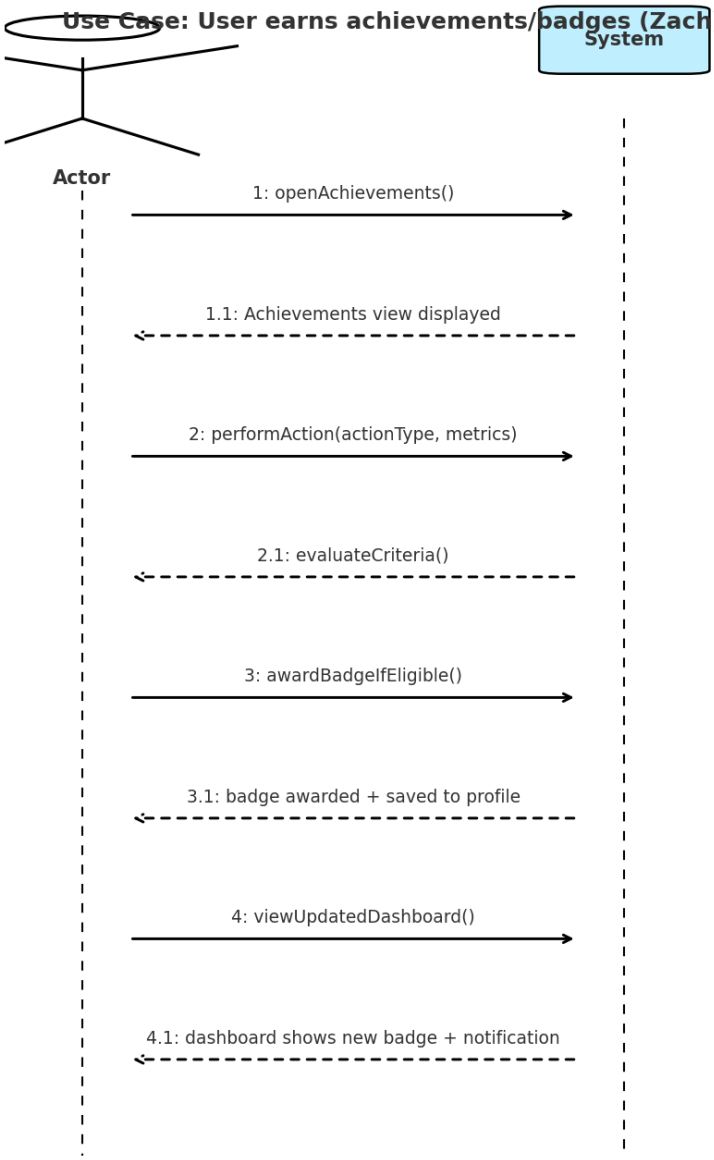
1. User logs into the system
2. User performs an action (e.g., completes a workout, meets a goal)
3. System tracks user's progress
4. System checks if an achievement milestone is reached
5. If milestone is reached, system awards the corresponding badge
6. System saves the badge and associates it with the user's profile
7. System updates the user's dashboard to display the new badge
8. System sends a notification congratulating the user

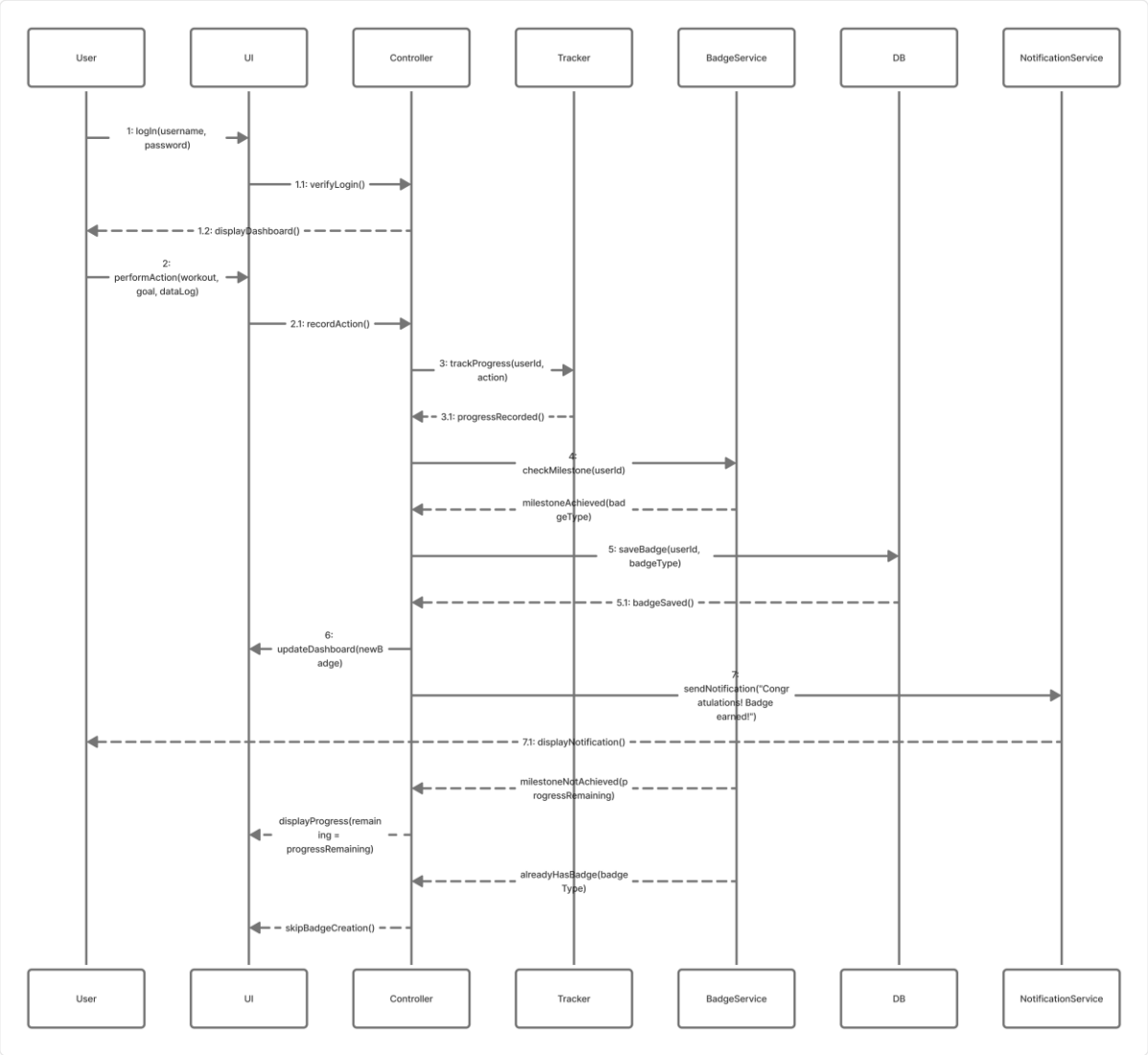
Alternate Scenario 1: Milestone Not Reached

1. Complete steps 1–3
2. System checks milestone criteria and finds that requirements are not met
3. No badge is awarded
4. System optionally displays progress (e.g., "2 more workouts until next badge")

Alternate Scenario 2: Duplicate Badge Prevention

1. Complete steps 1–4
2. System detects the user already has the badge
3. System does not create a duplicate badge
4. No update is made to the dashboard


Use Case: User earns achievements/badges (Zach)
System
Actor
1: openAchievements()
1.1: Achievements view displayed
2: performAction(actionType, metrics)
2.1: evaluateCriteria()
3: awardBadgeIfEligible()
3.1: badge awarded + saved to profile
4: viewUpdatedDashboard()
4.1: dashboard shows new badge + notification



Contract ACH1: awardAchievement

Operation: evaluateAndAwardAchievement(actionType: String, metrics: Map)

Cross Reference: Earn Achievements use case

Pre-conditions:
User is logged in and performing a trackable action (e.g., logging a workout, completing a goal).

Post-conditions:

- A new Achievement instance ach is created (instance creation)

- ach is associated with the user profile and achievements list (association formed)
- User dashboard is updated to display ach (state change)
- A congratulatory notification is queued/sent (state change)

Use Case: A user friends another user (Zach)

ID: Friend User

Scope: User Portal

Level: User goals

Stakeholders and Interests:

- **User (actor):**
  Can search for another user's ID, send a friend request, and view friend data once accepted.
- **Friend (recipient):**
  Can accept or reject friend requests; controls what data is shared.
- **Admin:**
  Can audit friend requests, review disputes

Precondition: User is logged into the system

Postcondition: Friendship link is created; both users can see allowed friend data on their dashboards

Main Success Scenario:

1. User logs into the system

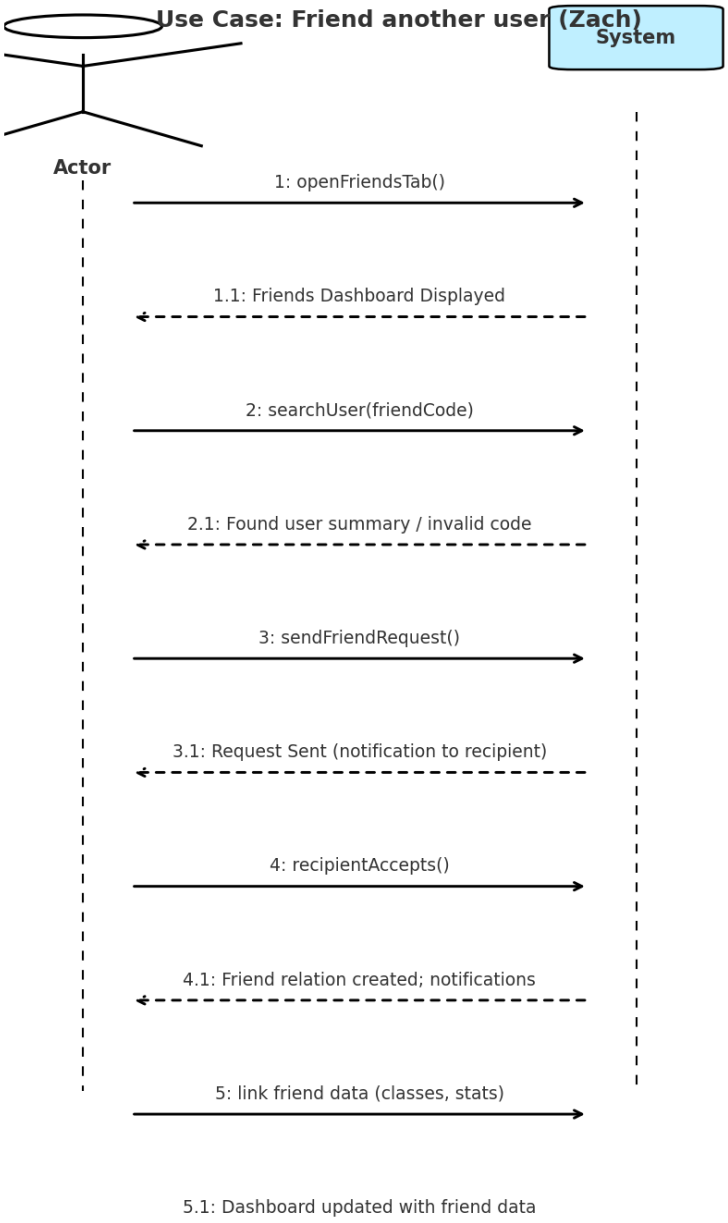
2. User clicks on the “Friends” tab in the dashboard
3. User searches for another user’s ID or friend code
4. System validates the code and finds the target user
5. User clicks “Send Friend Request”
6. Target user receives a friend request notification
7. Target user accepts the request
8. System links both users as friends
9. Friend data appears in both dashboards (e.g., number of classes, shared classes)

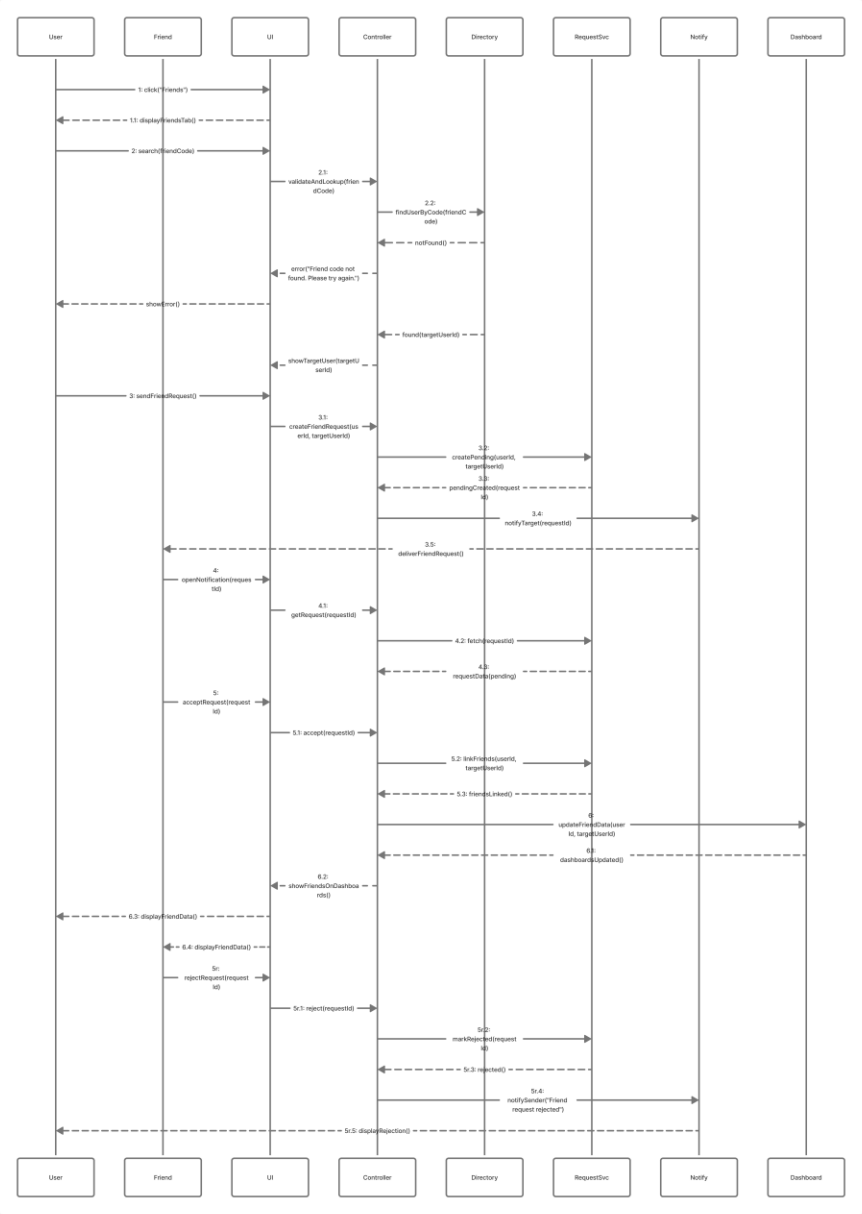
Alternate Scenario 1: Invalid Friend Code

1. Complete steps 1–3 of the main scenario
2. User enters an incorrect or invalid friend code
3. System displays an error message: “Friend code not found. Please try again.”
4. User may retry entering a valid code

Alternate Scenario 2: Rejecting Friend Request

1. Complete steps 1–6 of the main scenario
2. Target user rejects the friend request
3. System notifies the sender: “Friend request rejected.”
4. No friendship link is created, and no data is shared


Use Case: Friend another user (Zach)
System
Actor
1: openFriendsTab()
1.1: Friends Dashboard Displayed
2: searchUser(friendCode)
2.1: Found user summary / invalid code
3: sendFriendRequest()
3.1: Request Sent (notification to recipient)
4: recipientAccepts()
4.1: Friend relation created; notifications
5: link friend data (classes, stats)
5.1: Dashboard updated with friend data



Contract F1: sendFriendRequest

Operation: sendFriendRequest(targetCode: String)

Cross Reference: Friend User use case

Pre-conditions:
User is logged in and currently on the “Friends” tab

Post-conditions:

- A new FriendRequest instance fri is created (instance creation)
- fri is associated with the sender and recipient users (association formed)
- Notification is queued for the recipient (state change)

Use Case: Set Goals and Track Progress (Lademi)

ID: Set Goals and Track Progress

Scope: Not So Beary Fat app

Level: User

Stakeholders and Interests

General User:

- Wants a simple way to set personal fitness goals (e.g., weight loss, muscle gain).
- Wants flexibility to choose preferred workouts, frequency, and difficulty level.
- Expects progress to be tracked automatically with clear charts and updates.
- Wants the ability to update or change goals if circumstances change.
- Wants reminders and encouragement to stay consistent.

System:

- Provides an intuitive interface for creating and updating goals.
- Stores and tracks user progress in a reliable manner.
- Displays visual feedback, milestones, and motivational messages.
- Handles missing entries by sending reminders without disrupting the experience.

Precondition

- User is logged into the app with valid credentials.

Postcondition

- Goals are saved, and the tracking system updates based on the user's input.
- Progress is visible in charts or summaries, with reminders if entries are missing.

Main Success Scenario

1. User logs in to the app.
2. System displays the goal-setting interface.
3. User selects an overall fitness objective (e.g., lose weight, build strength).
4. User defines specific targets such as number of weekly workouts, exercise type, or calorie intake.
5. User adjusts frequency and intensity to match personal preference.
6. User saves the new goals.
7. System confirms goals are stored and links them to the progress tracker.
8. User begins logging data (workouts, calories, sleep, etc.).
9. System updates charts and shows how close the user is to reaching targets.
10. Milestones or achievements are highlighted when progress is made.

Alternate Success Scenario A (Adjusting Goals)

1. User reviews progress and decides to change an existing goal.
2. System displays current goals.
3. User edits the routine or targets and saves the changes.
4. System updates and continues tracking with new parameters.

Alternate Success Scenario B (No Recent Activity)

1. User has not logged data for at least 7 days.

2. System shows reminder: “No entries in the past week. Add a workout or meal to continue tracking.”
3. Quick-log controls are displayed so the user can add data quickly.

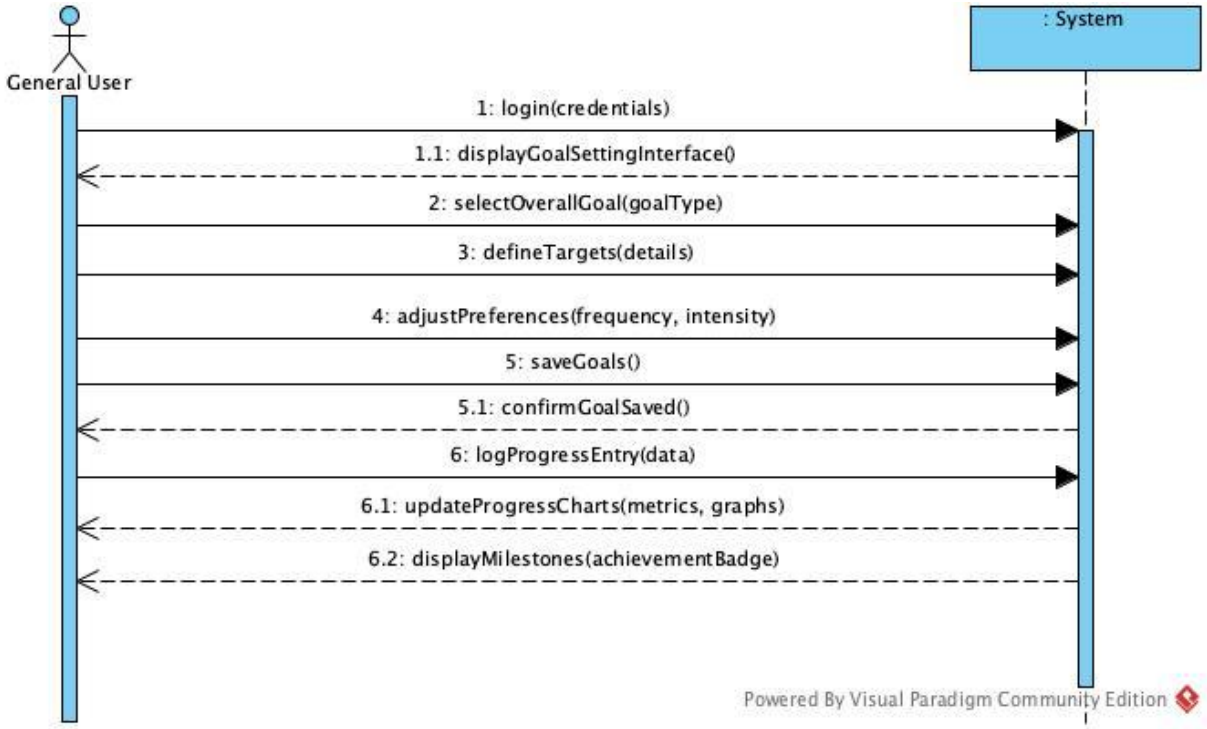
Alternate Success Scenario C (Milestone Celebration)

1. User achieves a milestone (e.g., five consecutive workouts, reaching a calorie goal).
2. System displays a celebratory message or badge.
3. User is encouraged to maintain progress.

Alternate Success Scenario D (System Error)

1. User attempts to save a goal but system cannot process the request.
2. Error message appears: “Unable to save goals right now. Please try again later.”
3. User may retry later or continue using other parts of the app.

SSD:



Contract SG1: saveGoals

Operation: saveGoals(userID: integer, goalDetails: object)

Cross References: Use Case – Set Goals and Track Progress, SSD – Set Goals and Track Progress

Preconditions:

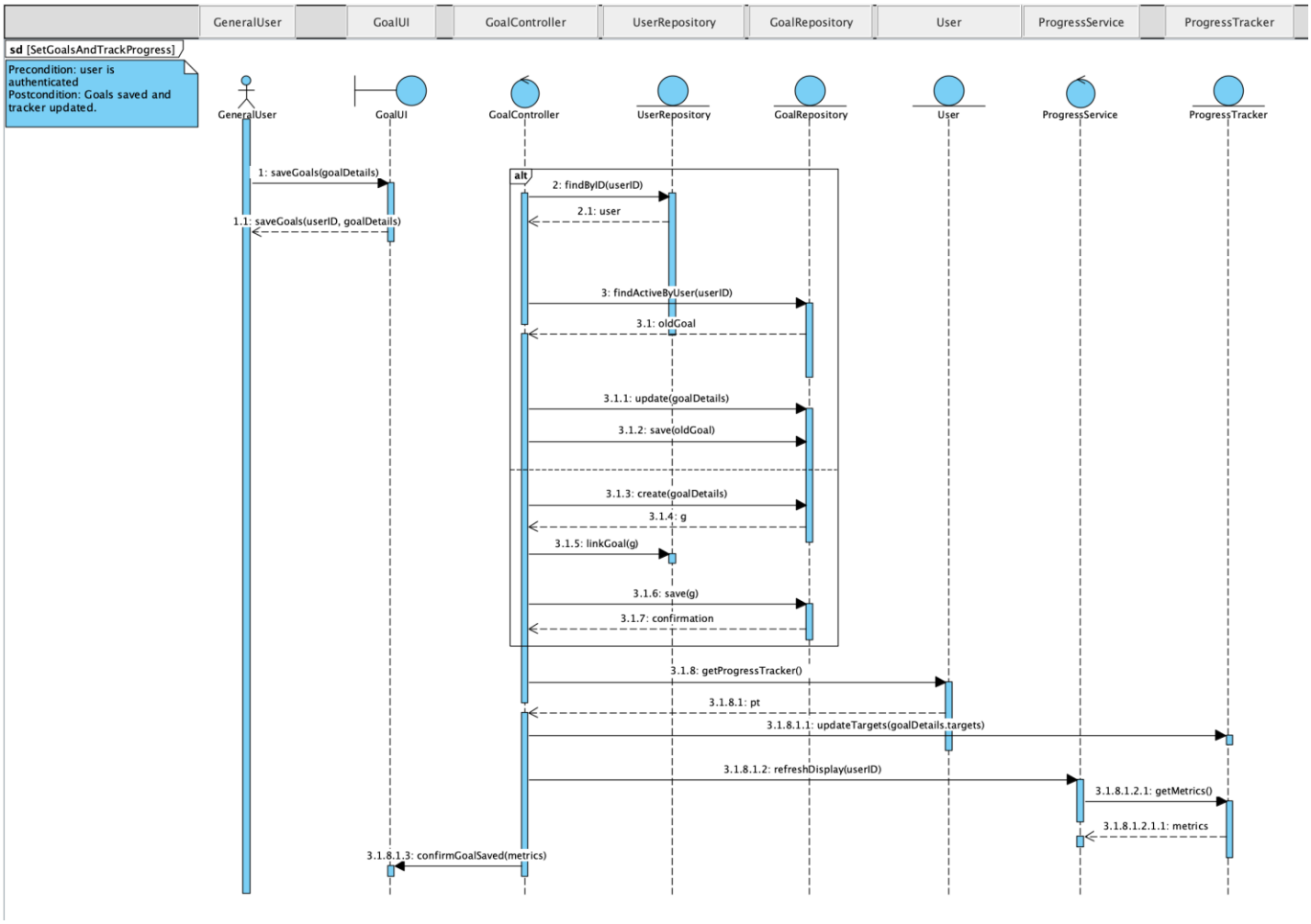
- The user is already logged in with valid credentials.

Postconditions:

- A new Goal object *g* is created. (instance creation)
- *g* is linked to the User account. (association formed)
- Goal attributes such as overall objective, preferred workouts, frequency, intensity, and calorie targets are recorded in *g*. (attribute modification)
- *g* is connected to the ProgressTracker *pt*. (association formed)
- The ProgressTracker is updated to reflect the user's new goals. (attribute modification)
- If the user updated an existing goal, the old values are overwritten with the new ones. (attribute modification)

- If no progress entries exist for the past 7 days, a Reminder object *rm* is created. (instance creation)
- *rm* is associated with the User. (association formed)
- When a milestone is achieved, a Milestone object *m* is created and tied to the User. (instance creation, association formed)
- If saving fails, an ErrorMessage object *em* is generated. (instance creation)
- The ProgressTracker display is refreshed so charts and metrics reflect the latest state. (attribute modification)

SD:



Use Case: Create Self-Paced Exercise Plans (Lademi)

ID: Create Self-Paced Exercise Plans

Scope: Not So Beary Fat app

Level: Trainers

Stakeholders and Interests

Trainer:

- Wants to design structured plans that users can follow independently.
- Needs to provide all the necessary details so the plan is useful.
- Expects the system to confirm when a plan has been created successfully.
- Doesn't want to lose work if something is left incomplete.

General User:

- Wants access to reliable trainer-created plans in the workout library.
- Relies on accurate details to decide whether a plan fits their goals and available equipment.

System:

- Supplies a form for trainers to enter required details.
- Ensures important fields are completed before allowing the plan to be saved.
- Stores the plan securely and publishes it to the workout library.
- Confirms successful creation or displays errors if something is missing.

Preconditions

- The trainer is logged into the app with valid credentials.

Postconditions

- A new self-paced plan is stored in the system and becomes available in the workout library for general users.

Main Success Scenario

1. Trainer logs into the app.
2. System displays the trainer dashboard.
3. Trainer chooses “Create Self-Paced Plan.”
4. System presents the plan creation form.
5. Trainer enters the title, description, fitness level, equipment needs, session length, and frequency.
6. Trainer saves the plan.
7. System validates inputs, stores the plan, and shows a confirmation message.
8. The plan now appears in the workout library for users to browse.

Alternate Scenario A (Missing Required Fields)

1. Trainer leaves one or more required fields blank (e.g., session length or title).
2. System prevents the save and highlights the missing sections.
3. Trainer fills in the missing details.
4. System accepts the submission and saves the plan.

Alternate Scenario B (System Error During Save)

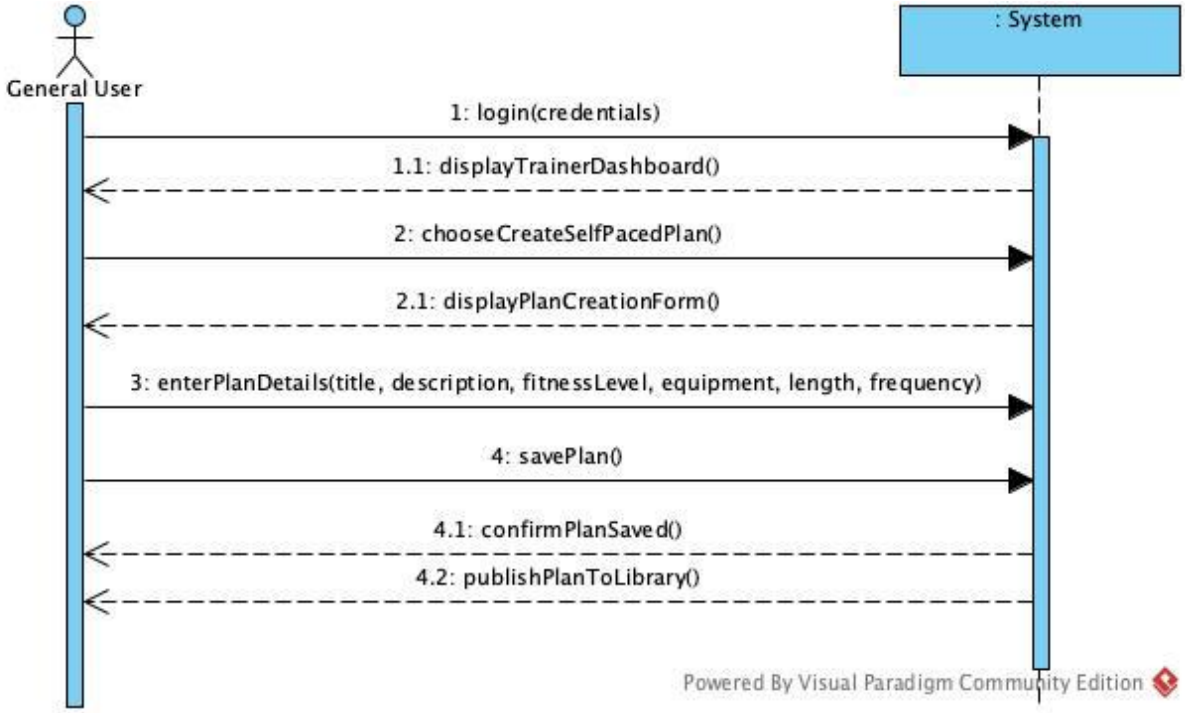
1. Trainer fills out the plan and selects “Save.”
2. System encounters an error and cannot complete the action.
3. A message appears: “Unable to save your plan right now. Please try again later.”
4. Trainer can retry once the system is functioning.

Alternate Scenario C (Editing After Creation)

1. Trainer decides to update an existing plan.

2. System shows the list of trainer-created plans.
3. Trainer selects one, makes changes, and saves.
4. System updates the stored version, and the changes appear in the workout library.

SSD:



Contract CP1: savePlan

Operation: savePlan(trainerID: integer, planDetails: object)

Cross References: Use Case – Create Self-Paced Exercise Plans, SSD – Create Self-Paced Exercise Plans

Preconditions:

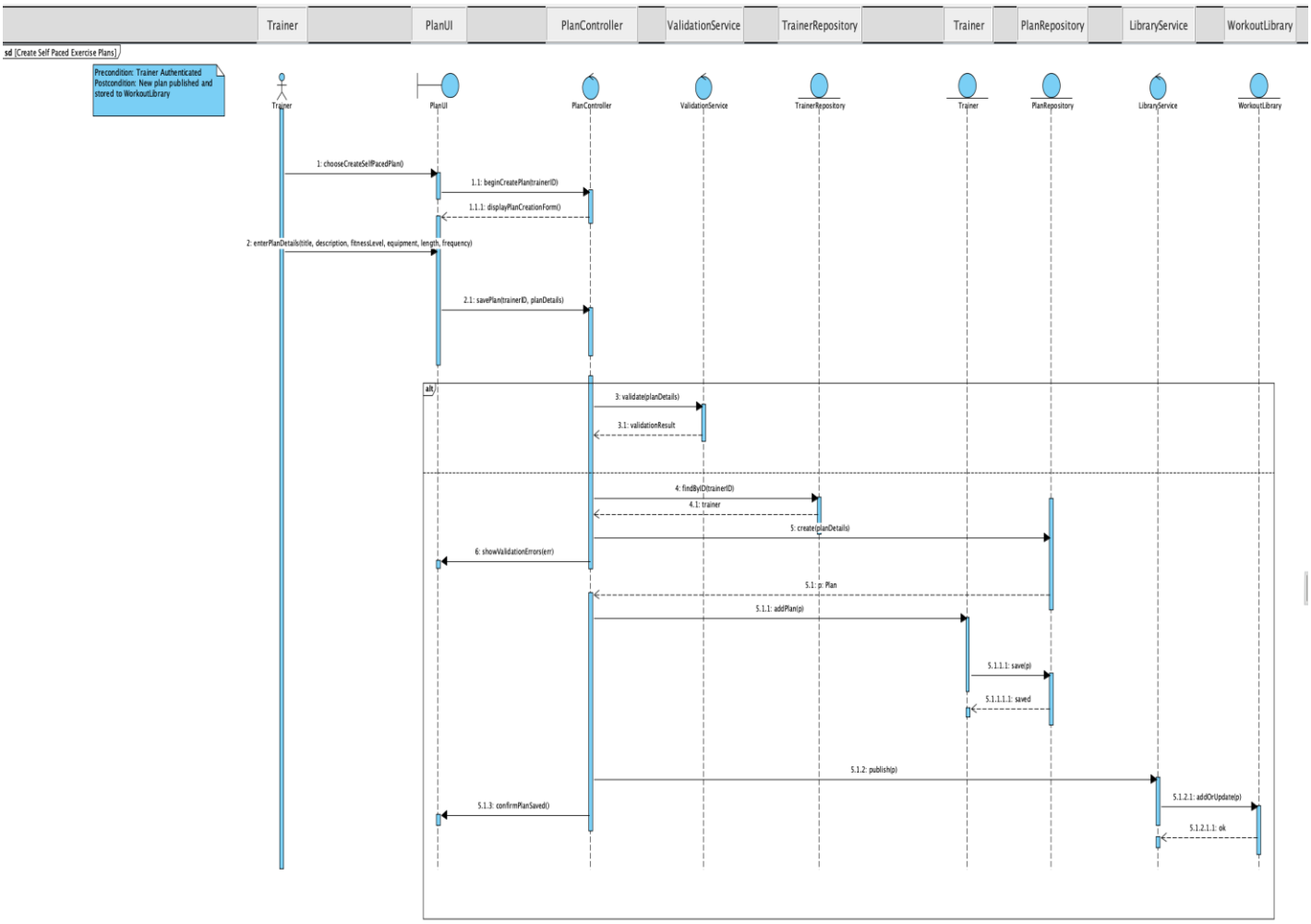
- The trainer has logged in successfully with valid credentials.

Postconditions:

- A new Plan object *p* was created. (instance creation)
- *p* was associated with the Trainer. (association formed)
- Plan attributes (title, description, fitness level, equipment needs, session length, frequency) were saved in *p*. (attribute modification)

- *p* was validated by the system to ensure required fields were complete. (attribute modification)
- *p* was published to the WorkoutLibrary *wl*. (association formed)
- If fields were missing, an Error object *err* was created. (instance creation)
- *err* highlighted the missing attributes, and *p* remained unsaved until corrections were made. (attribute modification)
- If the save failed due to a system error, an ErrorMessage object *em* was created. (instance creation)
- If an existing plan was edited, the old attributes were replaced with updated values in *p*. (attribute modification)
- WorkoutLibrary *wl* was updated to include the new or updated plan. (attribute modification)

SD:



Use Case: View Dashboard (Lademi)

ID: View Dashboard

Scope: Not so Beary Fat app

Level: User

Stakeholders and Interests

General User:

1. Wants quick access to their current progress.
2. Wants accurate charts and summaries that reflect their logged data.
3. Wants reminders if no recent entries are available.

System:

1. Automatically displays the dashboard after login.
2. Provides up-to-date data whenever possible.
3. Handles missing or incorrect entries without breaking the user's experience.

Precondition:

1. The user has logged in successfully with valid credentials.

Postcondition:

1. The user sees an accurate snapshot of their progress or is given reminders/errors if data cannot be displayed correctly.

Main Success Scenario:

4. User enters valid login credentials.
5. System redirects them to the dashboard page.
6. Dashboard displays calorie intake, sleep data, and current weight trends.
7. Dashboard shows weekly workout graphs including days active and calories burned.
8. User reviews progress within a few seconds of logging in.

Alternate Success Scenario A (No Recent Data):

4. User logs in after not entering data for a week or more.
5. Dashboard shows a message: *"No entries for the last 7 days. Add today's calories or workout to get back on track."*
6. Dashboard provides quick-log buttons so the user can add data immediately.

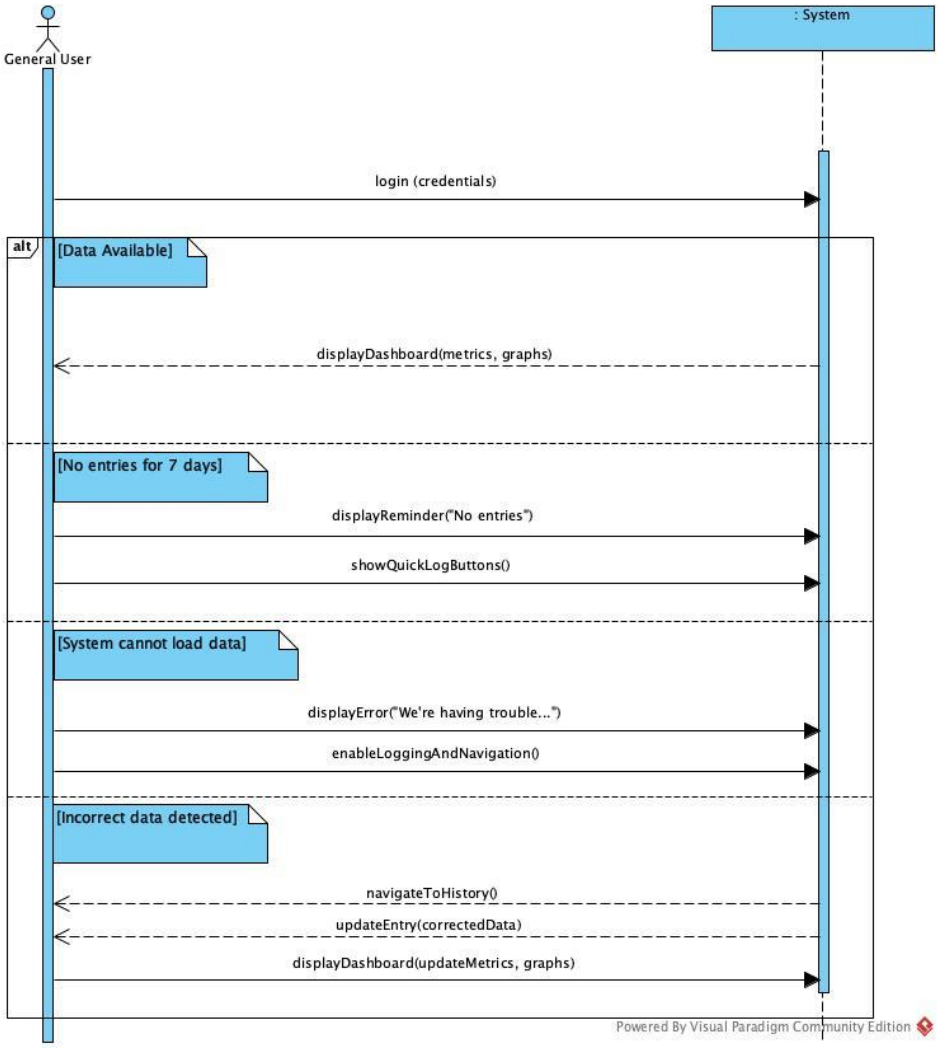
Alternate Success Scenario B (Loading Error):

2. User logs in but the system cannot load dashboard data.
3. Dashboard shows an error: *“We’re having trouble displaying your progress right now.”*
4. User is still able to log new entries and explore other features while waiting.

Alternate Success Scenario C (Incorrect Data):

2. User logs in and sees incorrect information (e.g., calories showing as zero).
3. User navigates to their data history and corrects the entry.
4. Dashboard refreshes and displays the updated values.

SSD:



Contract VD1: displayDashboard

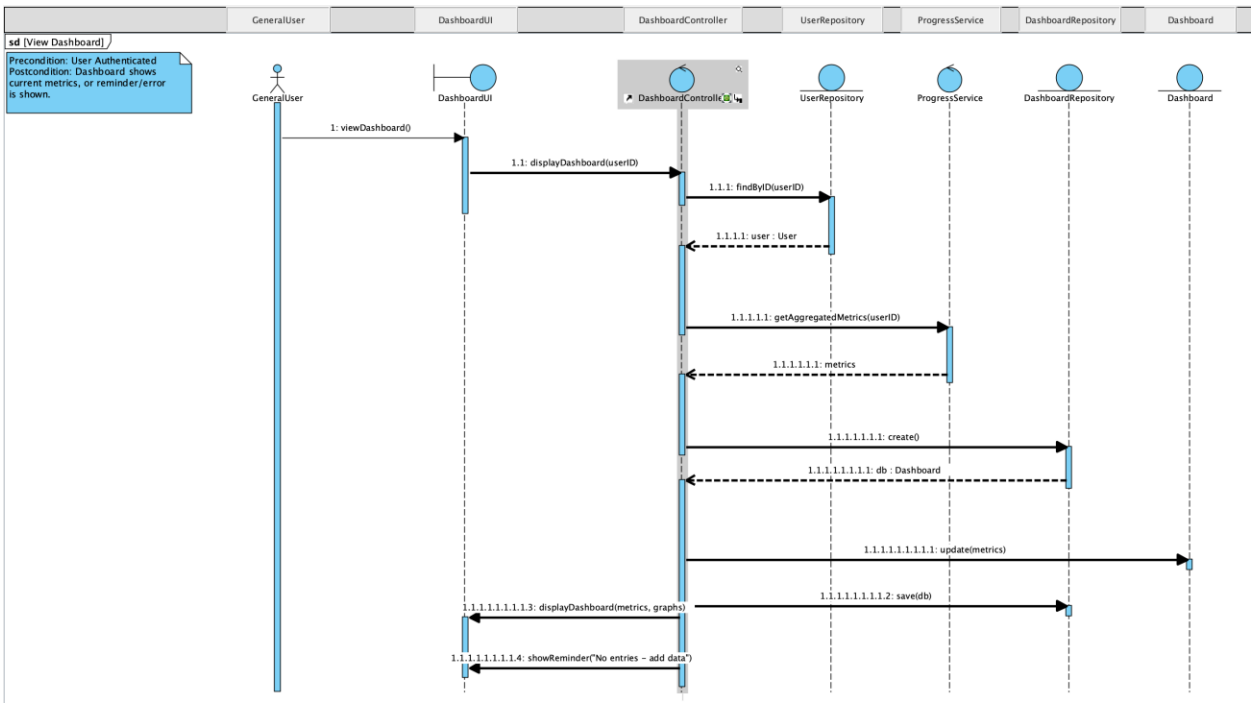
Operation: displayDashboard (userID: integer)
Cross References: Use Case – View Dashboard, SSD – View Dashboard

Pre-conditions: The user has logged in successfully with valid credentials.

Post-conditions:

- A Dashboard *db* instance was created. (instance creation)
- *db* was associated with the User. (association formed)
- User progress data (calories, weight, sleep, workouts) attributes were updated in *db*. (attribute modification)
- If no entries exist for the last 7 days, a Reminder object *rm* was created. (instance creation)
- If no entries exist for the last 7 days, Quick-log controls were associated with *db*. (association formed)
- If system loading fails, an Error message object *em* was created. (instance creation)
- If incorrect data was detected, the User corrected entries, and the updated values replaced the old ones in *db*. (attribute modification)
- Old/invalid *db* display state was replaced with refreshed metrics. (attribute modification)f

SD:



Use Case: A general user records a workout (Olivia)

ID: Record workout

Scope: Not So Beary Fat app

Level: User goal

Stakeholders and Interests:

User:

- Can record a workout and view it on the dashboard

Trainer:

- Can record a workout and view it on the dashboard

- Can see workouts completed by users

Admin:

- Can record a workout and view it on the dashboard
- Can see workouts completed by users and trainers

Precondition: User logged into the system

Postcondition: Workout is saved to workout dashboard

Main Success Scenario:

1. User logs into system
2. User clicks on the “Workouts” tab
3. User clicks “Add Workout”
4. “Add Workout” screen appears
5. User fills in type of workout, duration of the workout (in minutes), and date it was completed
6. User clicks “Save”
7. The workout and subsequent information will appear on the workout dashboard

Alternate Scenario 1: Track Calories Burned:

1. Complete steps 1-5
2. User fills in an optional section labeled “Calories burned”
3. User clicks “Save”
4. The workout will appear on the dashboard and include information about calories burned

Alternate Scenario 2: Update Workout:

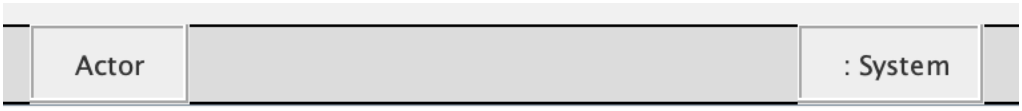
1. Complete steps 1-3
2. User navigates to previous workout that needs updating
3. User clicks the “edit” button
4. The workout appears with all the boxes filled in with the previously inserted information
5. User updates information
6. User can click “Save” or “cancel”
7. Clicking “Save” will update the information while clicking “Cancel” will revert the information back to its original state
8. The updated information will appear on the workout dashboard

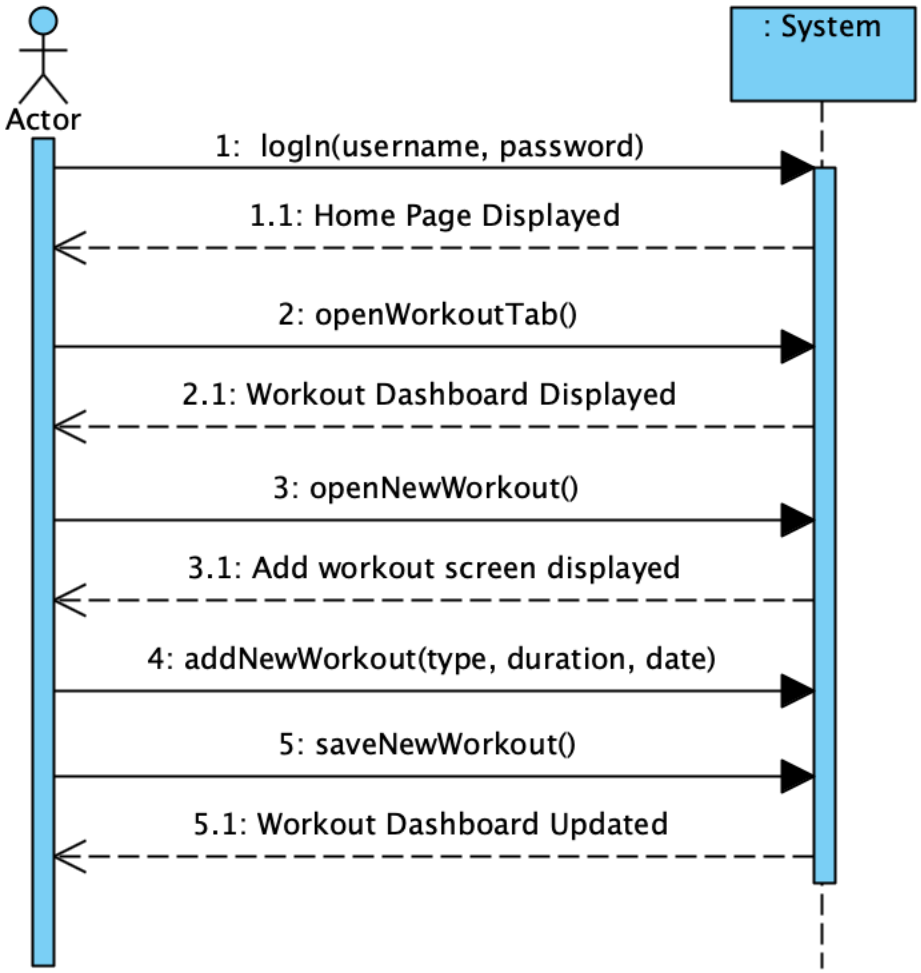
Alternate Scenario 3: Delete Workout:

1. Complete steps 1-3

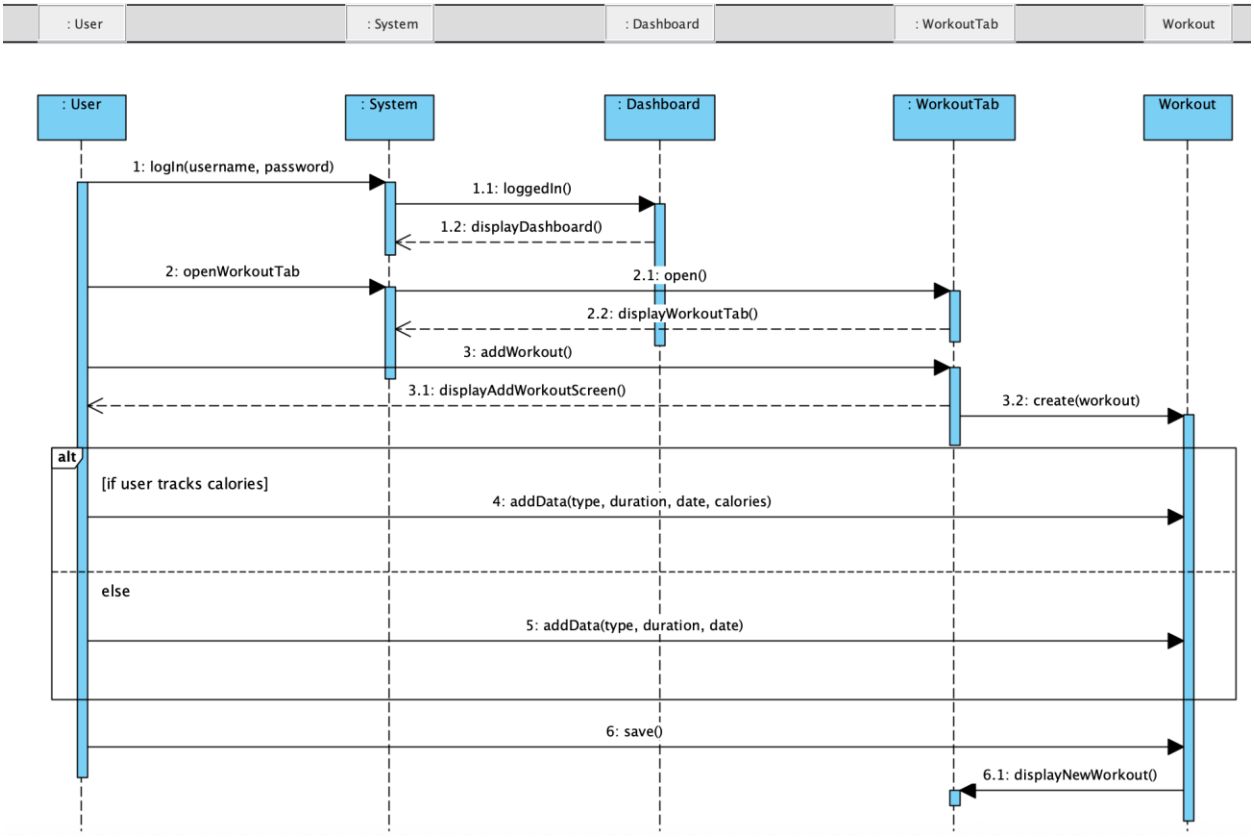
2. User navigates to previous workout that needs updating
3. User clicks the “edit” button
4. The workout appears
5. User clicks “Delete Workout”
6. The system will prompt a message that asks if the user is sure they want to delete the workout
7. The user will click “Yes”
8. The workout will be removed from the workout dashboard

SSD (Olivia):





SD:



Contract CO2: addNewWorkout

Operation: addNewWorkout(type: Workout, duration: int, date: Date)

Cross Reference: Use Case: Historical Trends

Pre-conditions: User has logged in and is currently on the “Add Workout” screen

Post-conditions:

- A newWorkout instance *sli* was created (instance creation)
- *sli* is associated with workout dashboard (association formed)

Use Case: A general user can track calorie intake, weight, and sleep (Olivia)

ID: Track data

Scope: Not So Beary Fat app

Level: User goal

Stakeholders and interests:

User:

- Can add data
- Historical trends will be updated

Trainer:

- Can add data
- Historical trends will be updated
- Can view user data

Admin:

- Can add data
- Historical trends will be updated
- Can view user and trainer data

Precondition: User has logged into the system

Postcondition: Data is added to historical trends

Main Success Scenario:

1. User logs into system
2. User navigates to “Historical Trends” page
3. User clicks “Record Data”

4. Screen prompts user to select the date of the data
5. User inputs calorie intake, weight, and sleep (in hours)
6. User selects “Save”
7. Updated data will be reflected in the graphs

Alternate Scenario 1: Update Data:

1. Complete steps 1-4
2. Previous data will be displayed on the screen
3. User inputs new data in desired boxes
4. User selects “Save”
5. Updated data will be reflected in the graphs

Alternate Scenario 2: Delete Data:

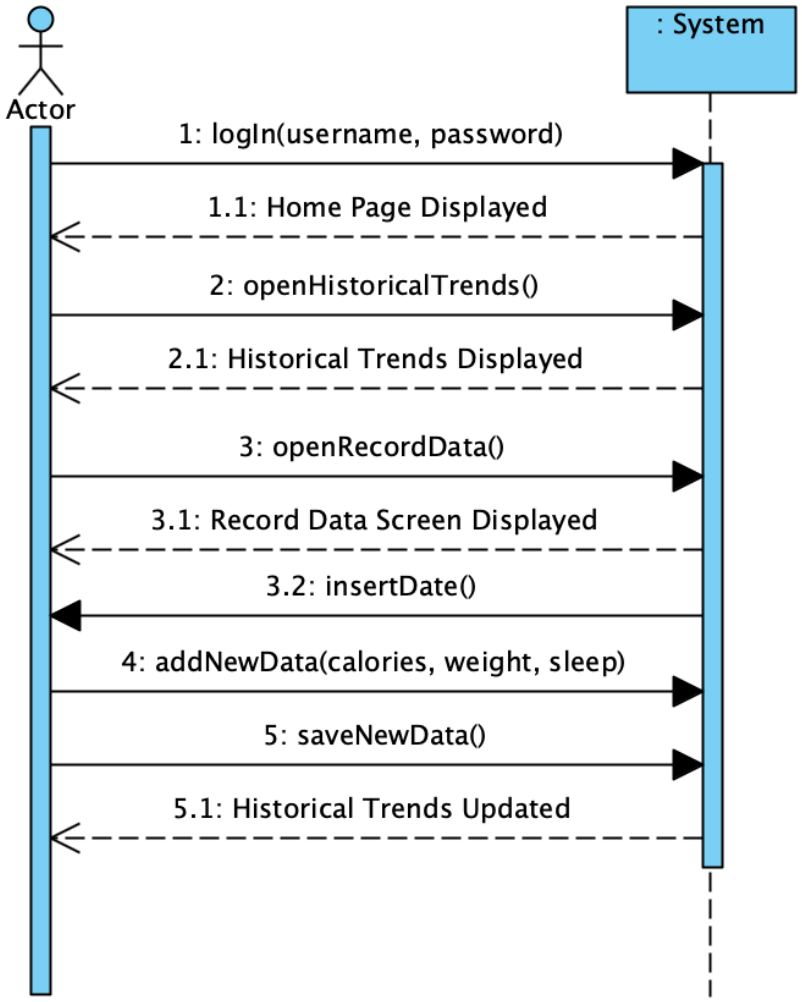
1. Complete steps 1-4
2. Previous data will be displayed on the screen
3. User selects “Delete Data”
4. The system asks the user to confirm they want to delete the data
5. User confirms
6. Updated data will be reflected in the graphs

Alternate Scenario 3: User adds calories burned:

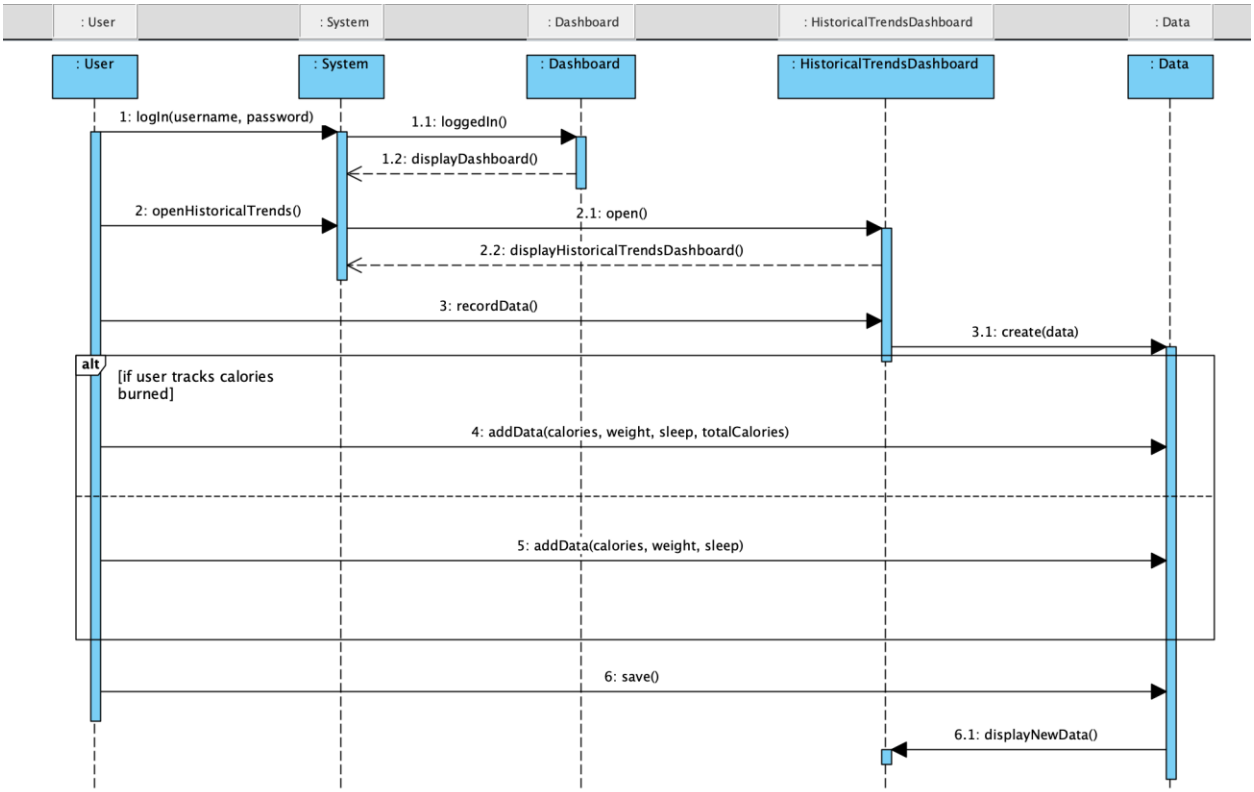
1. Complete steps 1-5
2. User clicks “Add Calories Burned (optional)”
3. User uses data from Apple Watch or similar device to add total calories burned for the day
4. User clicks “Save”
5. Updated data will be reflected in the graphs

SSD (Olivia):


Actor
: System


Actor
: System
1: logIn(username, password)
1.1: Home Page Displayed
2: openHistoricalTrends()
2.1: Historical Trends Displayed
3: openRecordData()
3.1: Record Data Screen Displayed
3.2: insertDate()
4: addNewData(calories, weight, sleep)
5: saveNewData()
5.1: Historical Trends Updated

SD:



Contract CO2: addNewData

Operation: (calories : int, weight : double, sleep : int)

Cross Reference: Use Case: Historical Trends

Pre-conditions: User has logged in, User is on “Record Data” screen

Post-conditions:

- An addNewData *sli* was created (instance creation)
- *sli* is associated with historical trends (association formed)
- Historical trends is updated (attribution modification)

_____

_____

_____

_____

Use Case: A general user can see their historical trends (Olivia)

ID: Historical Trends

Scope: Not So Beary Fat app

Level: User goal

Stakeholders and interests:

User:

- Can navigate to historical trends page and view graphs of inputted data

Trainer:

- Can navigate to historical trends page and view graphs of inputted data
- Can view graphs of user's inputted data

Admin:

- Can navigate to historical trends page and view graphs of inputted data
- Can view graphs of user's inputted data
- Can view graphs of trainer's inputted data

Precondition: User logged into system, user has entered data, including: Calories consumed, weight, and sleep, user has entered workout data

Postcondition: Historical trend graphs are updated

Main Success Scenario:

1. User logs into the system
2. User selects "Historical Trends"
3. A page appears with 3 line-graphs displaying historical data, each one representing one topic: calories consumed, weight, or sleep
4. User clicks the arrow on the bottom right of the screen

5. A new page appears with 2 graphs: one line-graph displaying workout duration and a bar graph displaying types of workouts completed
6. User clicks on the arrow on the bottom left of the screen to navigate back to original page
7. User clicks on a graph
8. The graph is enlarged to show the details more clearly

Alternate Scenario 1: User input total calories burned (Data):

1. Complete steps 1-2
2. A page appears with 4 line-graphs displaying historical data, each one representing one topic: calories consumed, total calories burned, weight, or sleep
3. Complete steps 4-8

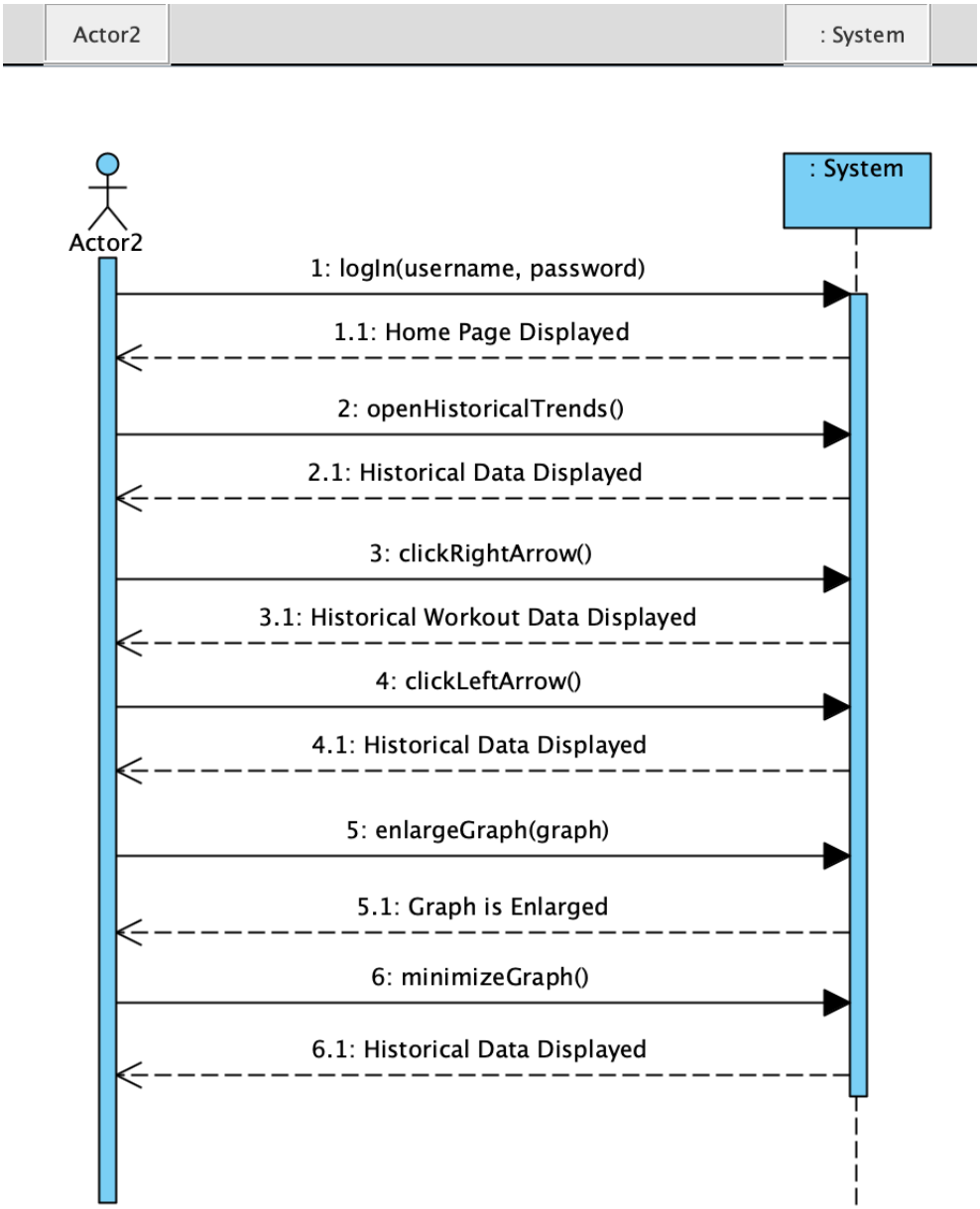
Alternate Scenario 2: User input calories burned (during workout):

1. Complete steps 1-4
2. A new page appears with 3 graphs: two line-graphs, one displaying workout duration and the other displaying calories burned during the workout, and a bar graph displaying types of workouts completed
3. Complete steps 6-8

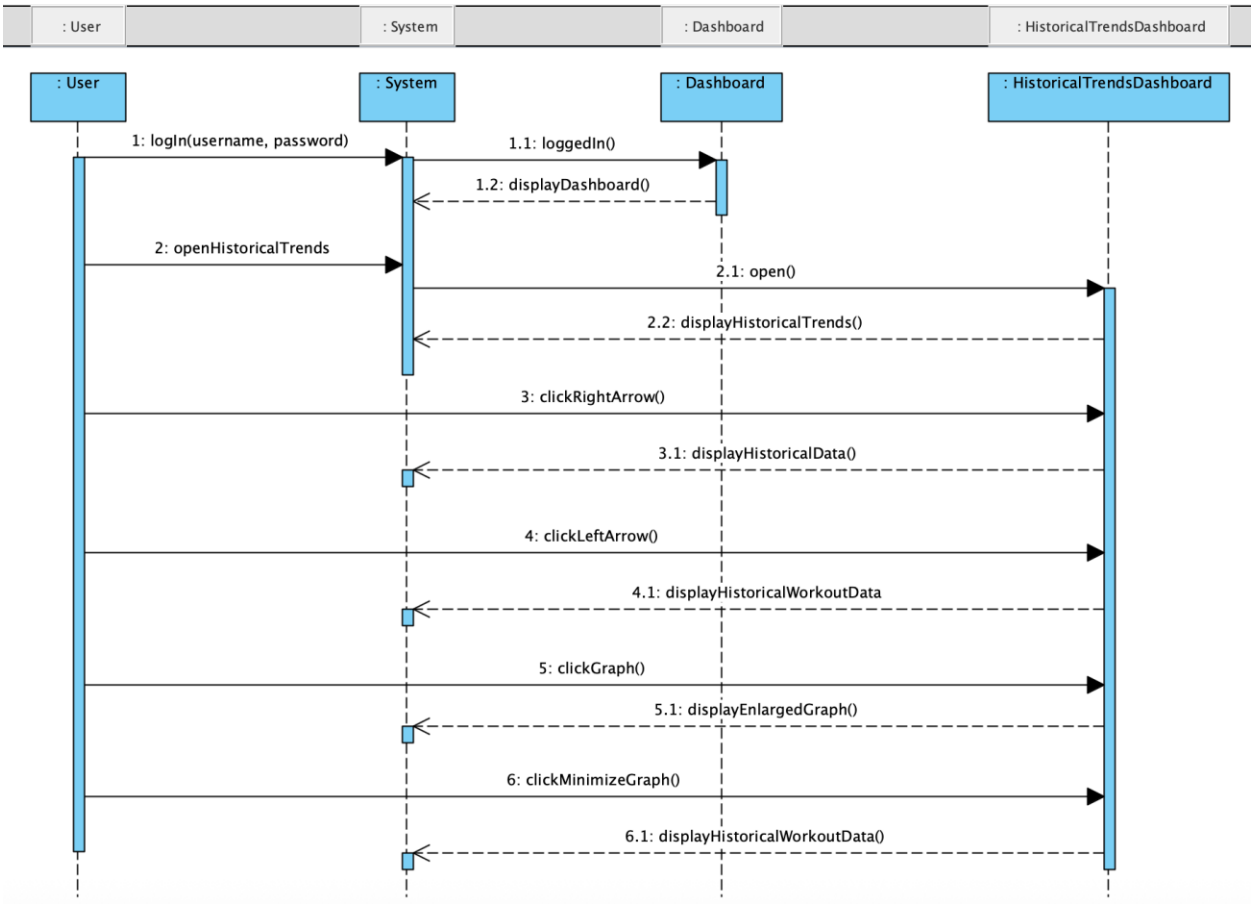
Alternate Scenario 3: User has not entered any data or workouts:

1. Complete steps 1-2
2. A blank page is displayed with a message stating “No data to display” and two buttons, one marked “Add data” and the other marked “Add workout”
3. User selects desired button
4. User adds information (Use Case ID: Record Workout or Use Case ID: Track Data)
5. After saving data/workouts, graphs will be displayed with their respective information
6. Complete steps 4-8

SSD (Olivia):


Actor2
: System
Actor2
: System
1: logIn(username, password)
1.1: Home Page Displayed
2: openHistoricalTrends()
2.1: Historical Data Displayed
3: clickRightArrow()
3.1: Historical Workout Data Displayed
4: clickLeftArrow()
4.1: Historical Data Displayed
5: enlargeGraph(graph)
5.1: Graph is Enlarged
6: minimizeGraph()
6.1: Historical Data Displayed

SD:



Contract CO2: openHistoricalTrends

Operation: openHistoricalTrends()

Cross Reference: Use Case: Track Data, Use Case: Record Workout

Pre-conditions: User logs in, user has tracked data, user has recorded a workout

Post-conditions:

- Data graph is created upon first instance of tracked data (instance creation)
- Workout graph is created upon first instance of workout data (instance creation)
- Data graphs are updated based on data inputted (attribute modification)
- Workout graphs are updated based on data inputted (attribute modification)

Use Case: User can login (Jonah)

ID: Login

Scope: Not so Beary Fat app

Level: User Goal

Stakeholders and Interests:

General User:

- Secure, quick access to their fitness account to track health and workouts

Trainer:

- Needs access to create, modify, and view classes/plans

Admin

- Needs access to manage users and system accounts

System

- Must authenticate credentials securely, prevent unauthorized access and provide account recovery.

Precondition:

- The user (General User, Trainer, or Admin) has already created an account with a valid username and password.
- The app is accessible, and network connectivity is available

Postcondition:

- **Success:** User is authenticated and redirected to their respective home dashboard (General User, Trainer, or Admin).

- **Failure:** User is denied access and offered password reset/retry options.

Main Success Scenario

1. The user (General User, Trainer, or Admin) opens the application.
2. The system displays the login/sign-up screen.
3. The user selects “Login.”
4. The system prompts for username and password.
5. The user enters valid credentials.
6. The system verifies credentials against the database.
7. The system authenticates the user.
8. The user is redirected to their correct dashboard:
    8.1. General User: Fitness Dashboard
    8.2. Trainer: Trainer management page
    8.3. Admin: Admin control portal

Alternate Scenario A: Invalid Credentials

1. Steps 1-4 as above.
2. The user enters an invalid username or password.
3. The system rejects the login attempt.
4. The system displays: “Incorrect username or password. Please try again.”
5. The user can retry credentials.

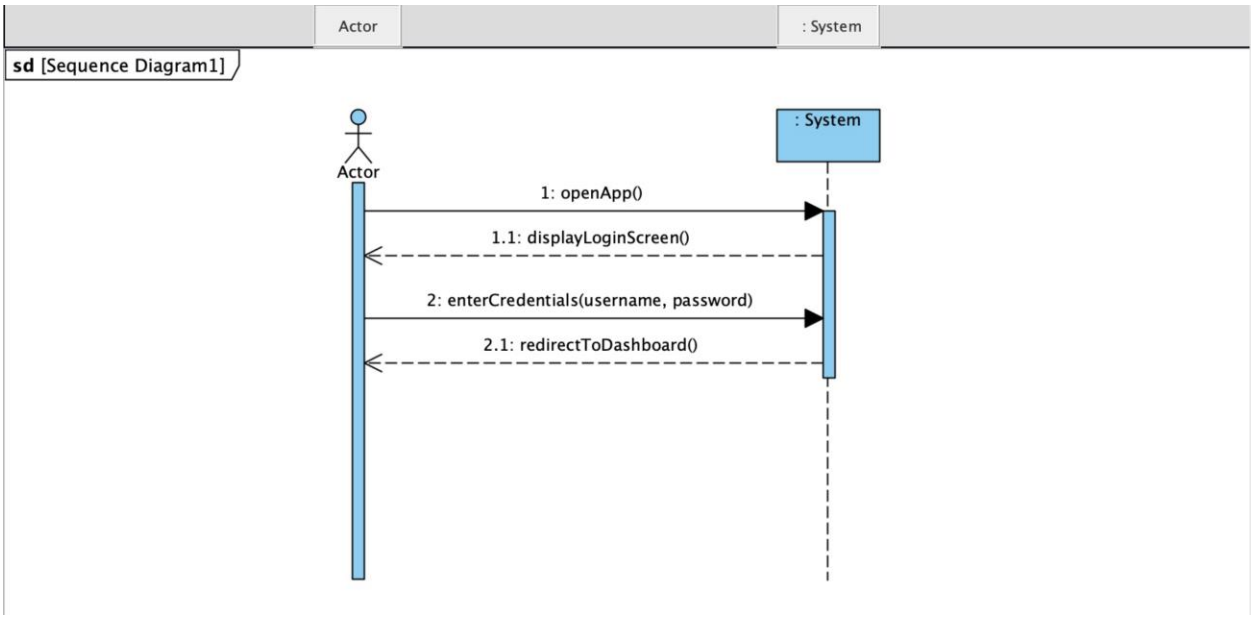
Alternate Scenario B: Forgot Password

1. Steps 1-4 as above.
2. The user selects “Forgot Password.”
3. The system prompts for a registered email.
4. The user enters their email.
5. The system sends a password reset link.
6. The user resets their password successfully.
7. The user re-attempts login and is authenticated.

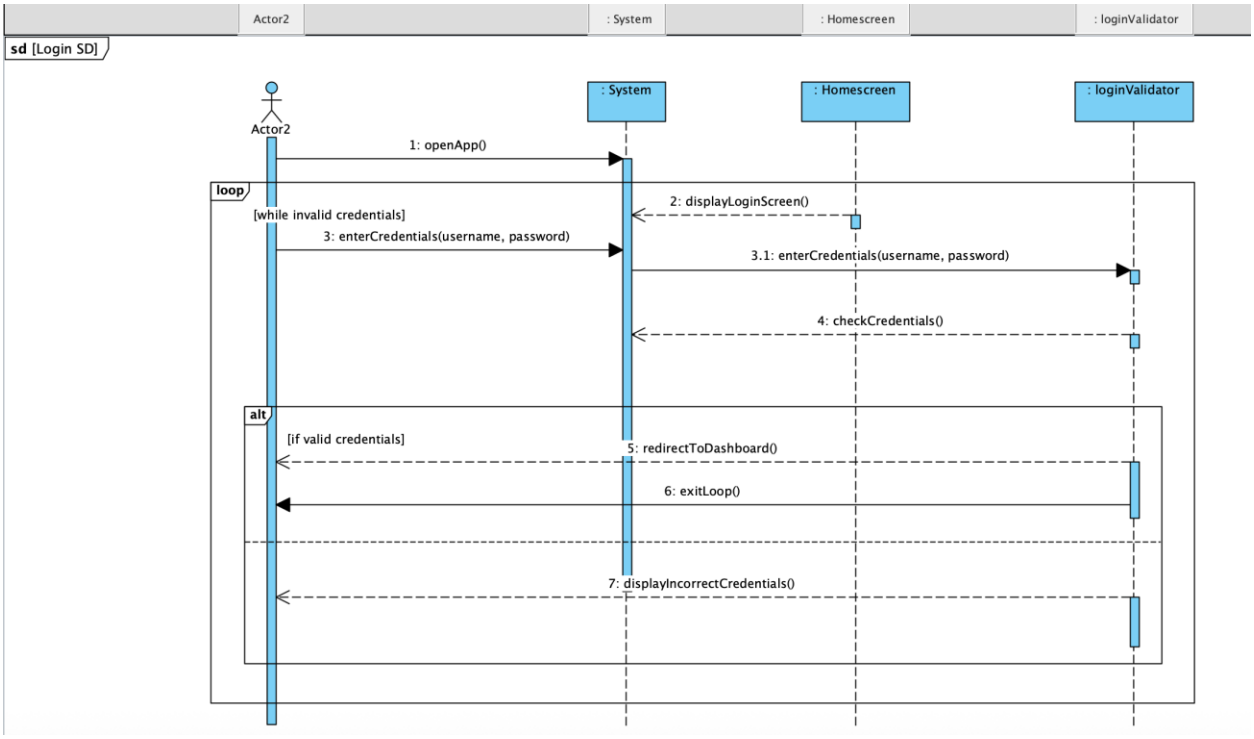
Alternate Scenario C: System Unavailable

1. Steps 1-3 as above.
2. The system fails to connect to the authentication service.
3. The system displays: “Login temporarily unavailable. Please try again later.”
4. The user cannot log in until the issue is resolved.

SSD:



SD:



Contract: loginUser

Author: Jonah

Operation:
loginUser(username: string, password: string)

Cross References:
Use Case: User can login

Pre-conditions:

- User has an existing account with valid credentials.
- System is accessible.

Post-conditions:

- A Session **s** is created (instance creation)
- **s** is linked to the User (association formed)
- user is redirected to the correct dashboard (General, Trainer, Admin). (association formed).
- If credential are invalid, no session created, ErrorMessage object **em** is displayed. (instance creation).
- If "forgot password" is selected, a PasswordResetRequest **pr** is created. (instance creation).
- If the system was unavailable, an ErrorMessage **em** is displayed (instance creation).

Use Case: A trainer can modify an existing class or plan (Jonah)

ID: Modify Class

Scope: Not so Beary Fat app

Level: Trainer/Admin

Stakeholders and Interests:

General User:

- A participant is notified of changes such as time, equipment descriptions or if they were removed from the class.

Trainer:

- Needs to be able to modify class details such as time, description, participant limits, difficulty level, or equipment restrictions.

Admin:

- Has the same options as the trainer.
- Can also delete/add classes or plans.

System:

- Blocks invalid change attempts.

Precondition:

- Trainer has access to the trainer dashboard and can login.
- Trainer has a class/plan that they may modify.
- Trainer has access to schedule of classes.

Postcondition:

- **Success:** Trainer makes changes to the class/plan, and it is successfully updated. User is notified of time/equipment change if they are in the class.
- **Failure:** The change is blocked, and the system notifies the trainer of what was invalid. Returns trainer to page with classes.

Main Success Scenario:

1. Trainer is logged in and opens class menu.
2. They select one of their classes and select modify.
3. The trainer modifies information such as time, description, participant limits, difficulty level, or equipment restrictions.
4. The system does not detect any invalid inputs and the change is approved.
5. Users in the class receive an email regarding the change if it affects them. (Time/equipment change).

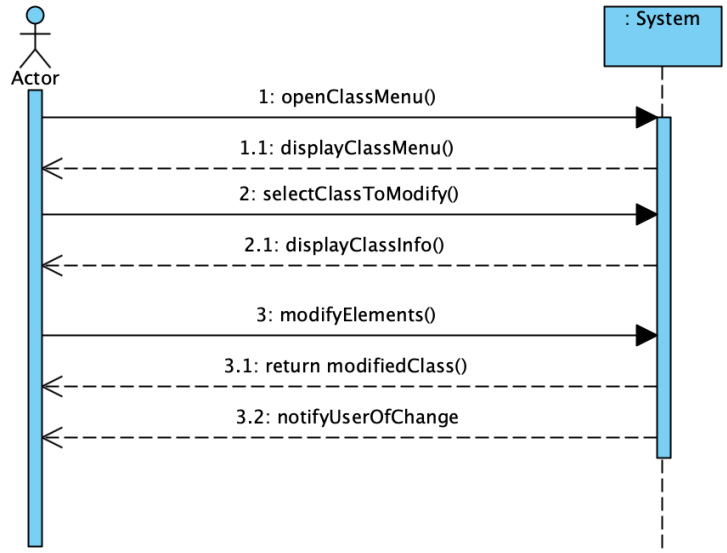
Alternate Failure Scenario:

1. Steps 1-3 as above.
2. The system detects an invalid input and blocks the change. The trainer is notified of what made the change invalid and they are returned to their modify class screen.

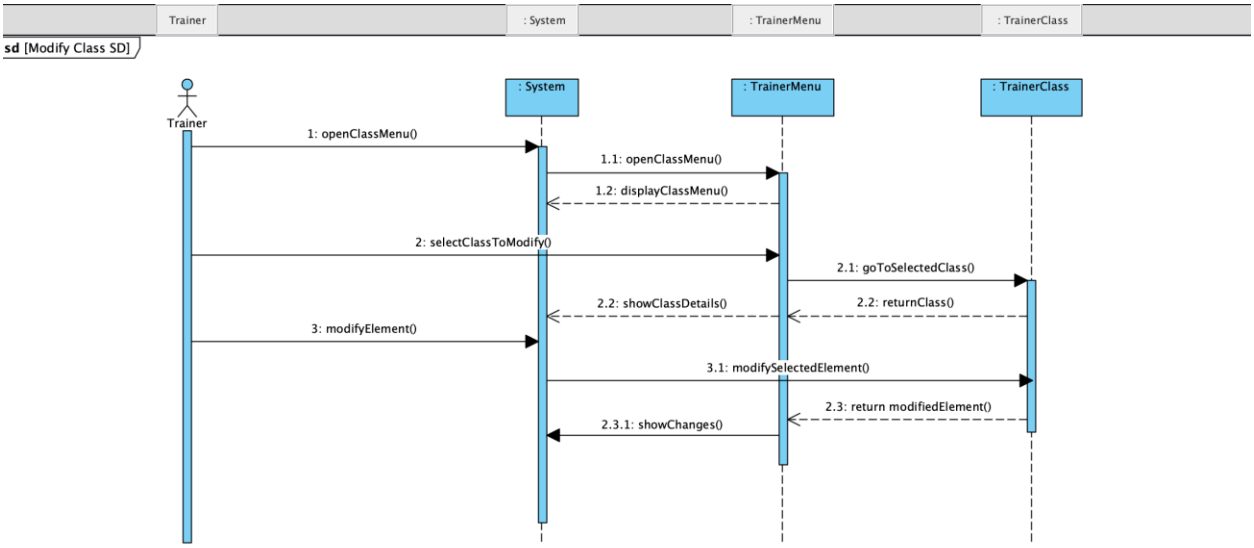
Alternate Failure Scenario:

1. System fails at any point above.

SSD:



SD:



Contract: Modify Class/Plan

Author: Jonah

Operation:

modifyClass()

Cross References:

- Use Case - User can login
- Use Case – Trainer can create a class

Precondition:

- Trainer has a class
- System is accessible
- Trainer makes valid changes

Postcondition:

- Class is successfully modified (attribution modification)
- User is notified of changes if applicable through object Message **m** (instance creation)

Use Case: A general user can register for a trainer led class (Jonah)

Prereq. Use Case: User can log in and view available classes

Main Success Scenario:

A general user logs in and navigates to “Classes.” Thye browse the available trainer led sessions, filter by time, type, or difficulty, and select a class to register for. The system verifies the prerequisites are met and the class is not full. If successful, the user is enrolled, the class appears on their dashboard/calendar, and the trainer receives an updated roster.

Alternate Scenario (class full):
If the class if already at maximum capacity, the system notifies the user and provides options to join a waitlist or browse similar classes.

Use Case: Register for a Trainer-Led Class (Jonah)

ID: classRegister

Scope: Not so Beary Fat app

Level: User Goal

Primary Actor: General User

Stakeholders and Interests:

- General User: reliable registration for trainer led classes that fit their schedule.
- Trainer: Needs an accurate roster of registered users.
- Admin: Needs reliable data of users in classes.
- System: Must enforce class limits and maintain accurate schedules.

Preconditions:

- User is logged in.
- User can view available classes.
- Class catalog and trainer schedules are accessible.

Postconditions:

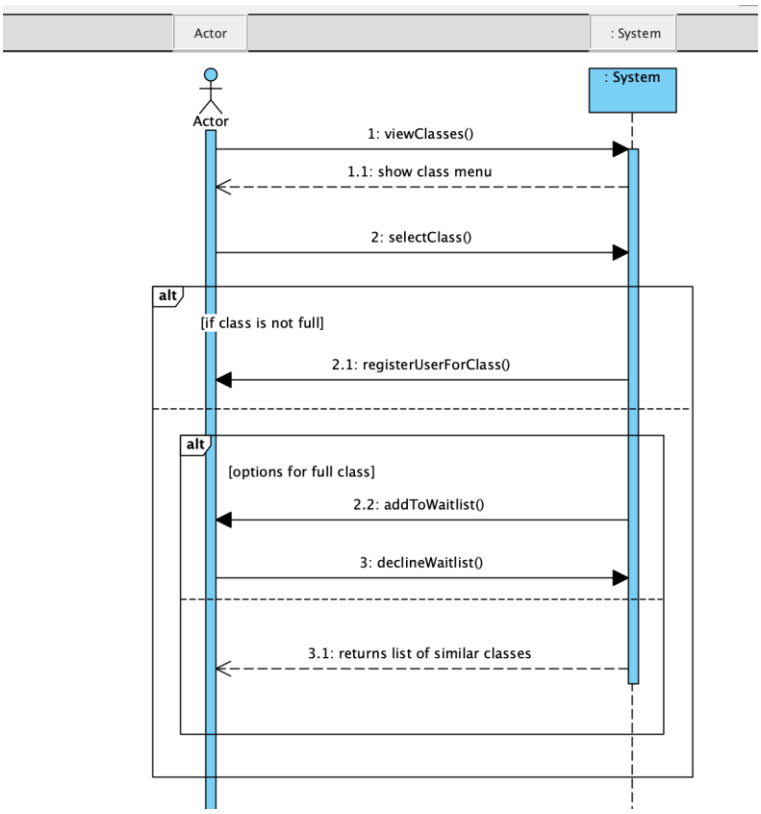
- **Success:** User is registered, the class is added to their dashboard/calendar, and the trainer's roster updates.
- **Failure (class full):** User is not enrolled, offered waitlist option or alternative classes.

Main Success Scenario:

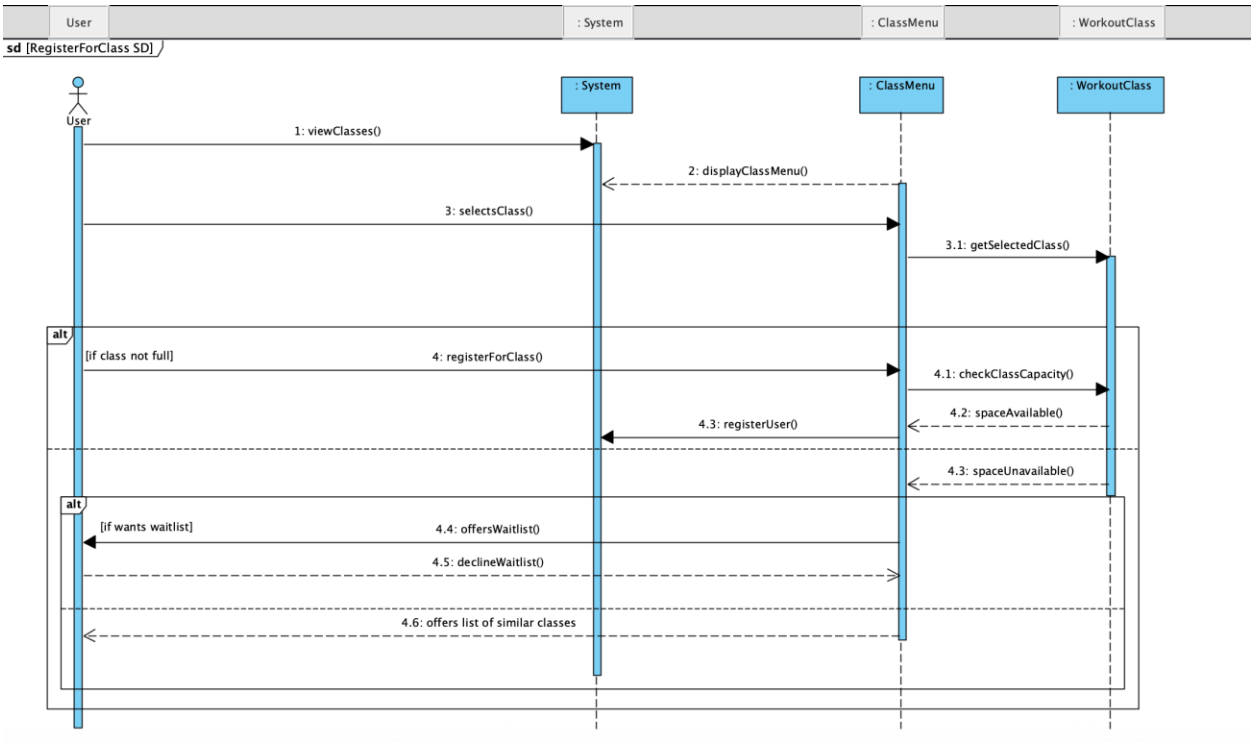
1. The user logs in and navigates to "Classes."
2. The system displays a list of available trainer-led sessions.
3. The user browses and applies filters (time, type, difficulty).
4. The user selects a desired class.
5. The system checks class capacity.
6. If space is available, the system registers the user.
7. The class appears on the user's dashboard/calendar.
8. The trainer's roster is updated to include the new user.
9. The system confirms successful enrollment to the user.

Alternate Scenario A (Class Full):

- Step 6: The system detects the class has reached maximum capacity.
- The system notifies the user.
- The system provides options:
    - Join the waitlist.
    - Browse similar available classes.
- User selects an option.



SD:



Contract: Register For Class

Author: Jonah Beck

Operation: classRegister()

Cross References:

- Use Case - Trainer can create a class

Precondition:

- Classes are available
- System is accessible
- User is logged in

Postcondition:

- User successfully registers for class (attribution modification)
- If class is full ,user is added to Waitlist object **w** or looks for another class. (attribution modification)/(instance creation).

_____

_____

_____

_____

Owen Chipman, fully dressed use case, User management

ID: User Management

Scope: Not so Beary Fat app

Level: Admin

Stakeholders and Interests:

Basic User:

- Able to be removed/added to the application
- Able to have their password reset

Trainer:

- Able to be removed/added to the application
- Able to have their password reset

Admin

- Able to be removed/added to the application
- Able to have their password reset

Precondition: Admin is logged in

Postcondition: The proper user was able to have their account status changed

Main Success Scenario:

1. User wishes to change their password
2. User provides admin with their current username and new password they'd like their account to be associated with
3. Admin logs in to admin portal
4. Admin locates user via username and changes their password

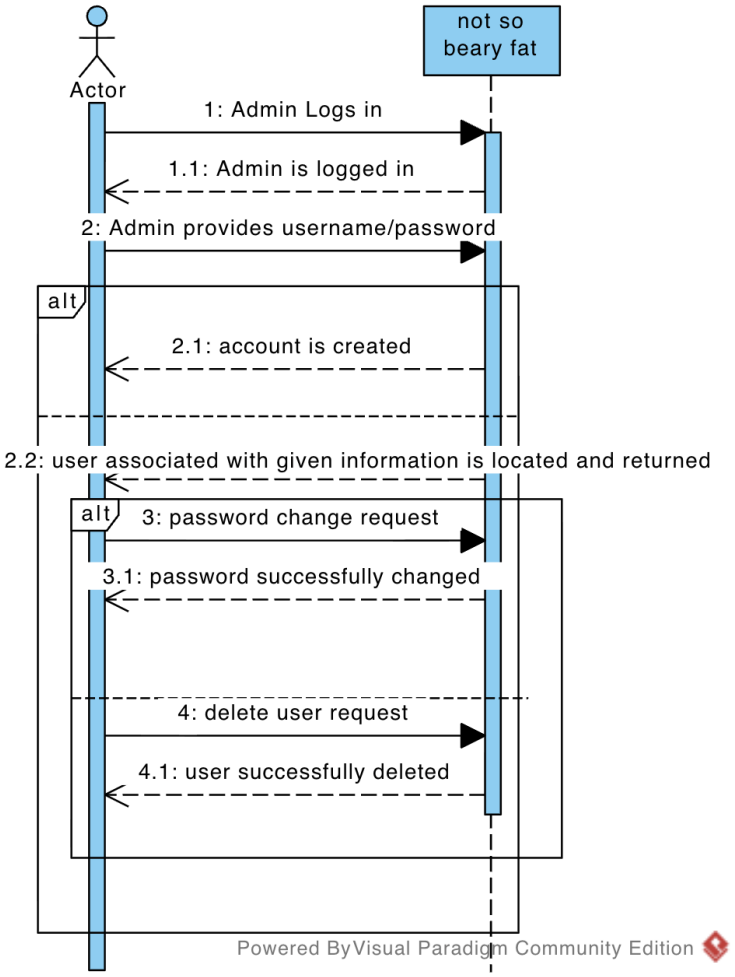
Alternate Success Scenario A:

1. New user wishes to join *Not so Beary Fat*
2. Admin receives preferred username password for said user
3. Admin logs in to admin portal
4. Admin creates new account with stated username and password
5. New user is now able to login

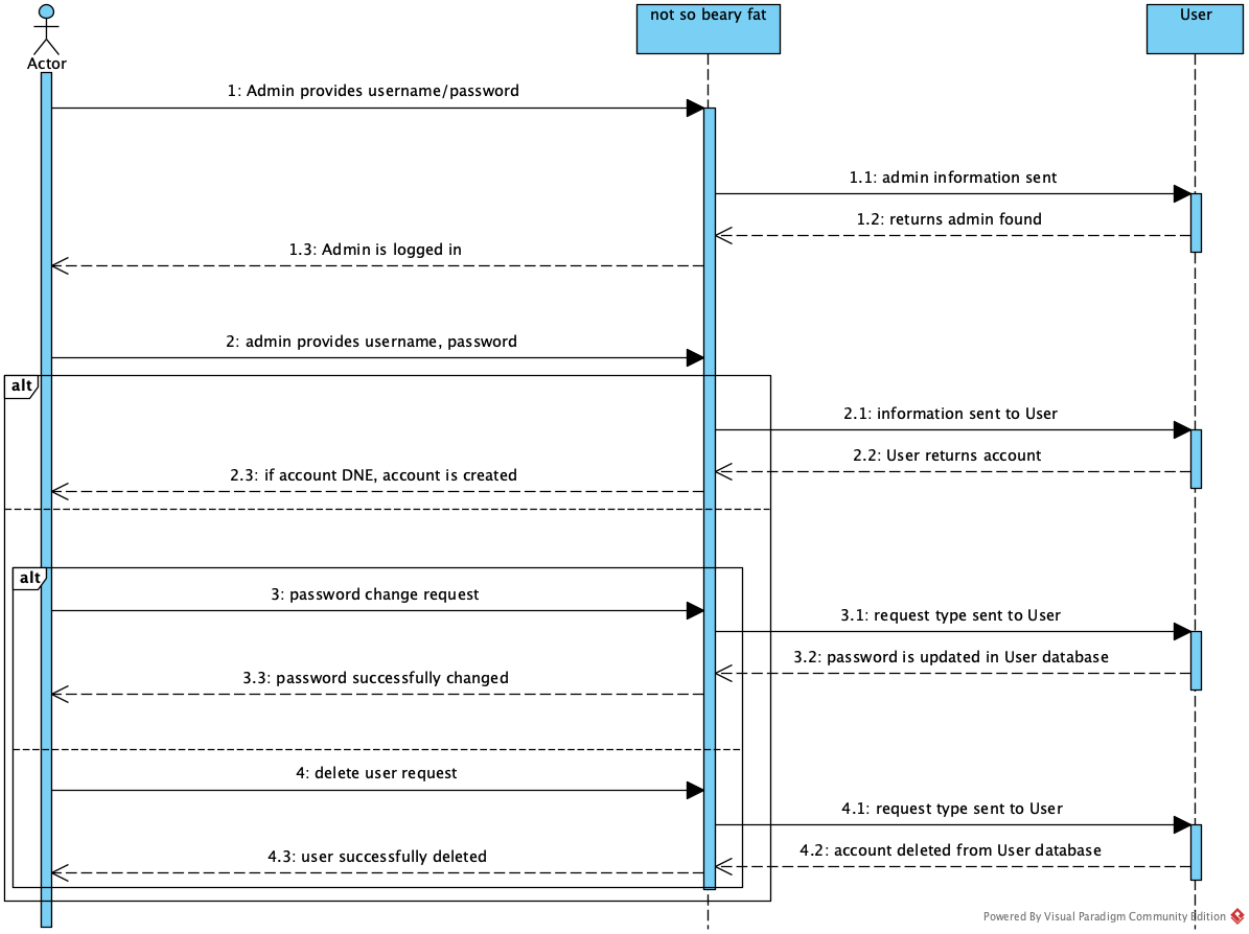
Alternate Success Scenario B:

1. New user wishes to be removed from *Not so Beary Fat*
2. User provides admin with username
3. Admin locates user by username
4. Admin removes user from application

SSD:



SD:



Operation Contract: User Management

Author: Owen Chipman

Contract:

- Operation name: User Management
- Responsibility: Allows for user login information to be added, deleted, or modified
- Precondition: System Admin is logged in
- Postcondition: the proper user has had their information altered in a satisfactory manner. (instance created or deleted)

_____

_____

_____

_____

Workout Search, Fully Dressed Use case: Owen, Chipman

ID: Workout Search

Scope: Not so Beary Fat app

Level: Basic User, Trainer

Stakeholders:

Basic User:

- Able to search by a given category:
    - Workout type
    - Targeted muscle group
    - Difficulty Level
    - Time of day
- Able to sign up for a workout
- Able to subscribe to a specific class

Trainer:

- Able to search by a given category:
    - Workout name
    - Workout type
    - Targeted muscle group
    - Difficulty Level
    - Date/Time of day
- Not able to sign up for classes as a trainer. Classes should already be associated with trainers after they create them

Precondition: Basic user wishes to sign up for a class

Postcondition: Basic user is registered

Main Success Scenario A:

1. User wishes to sign up for a class

2. User searches for workout by name or filters for relevant classes
3. User finds class they wish to take
4. User signs up for specific class

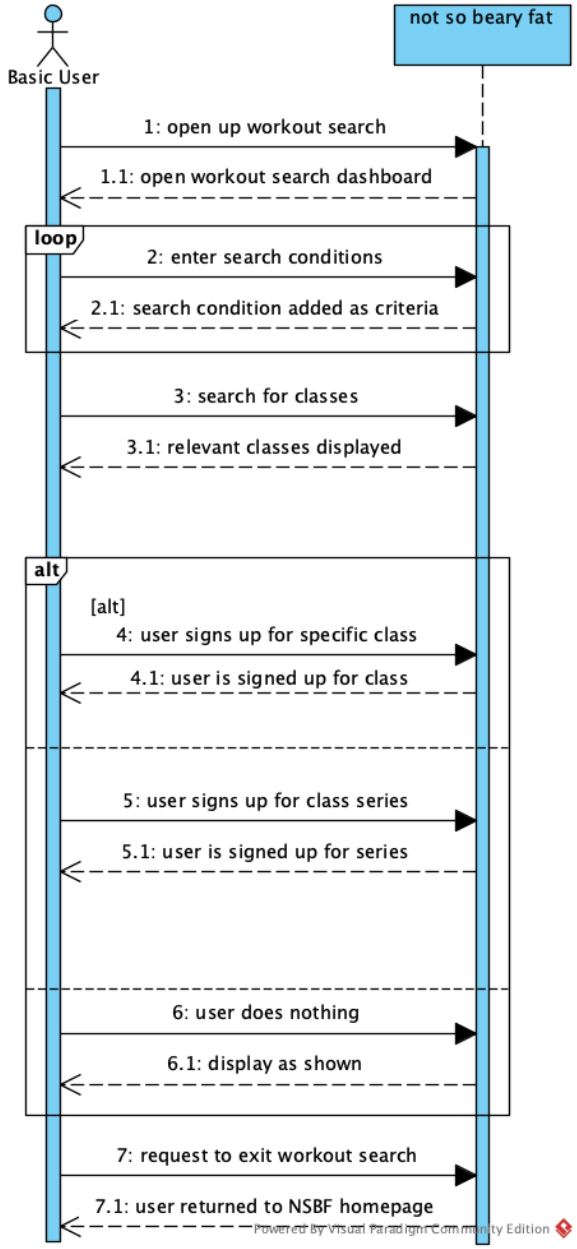
Alternate Success Scenario A:

1. User wishes to subscribe to a class series
2. User searches for workout by name or filters for relevant classes
3. User finds class they wish to take
4. User subscribes to series and is automatically registered for all classes in the series

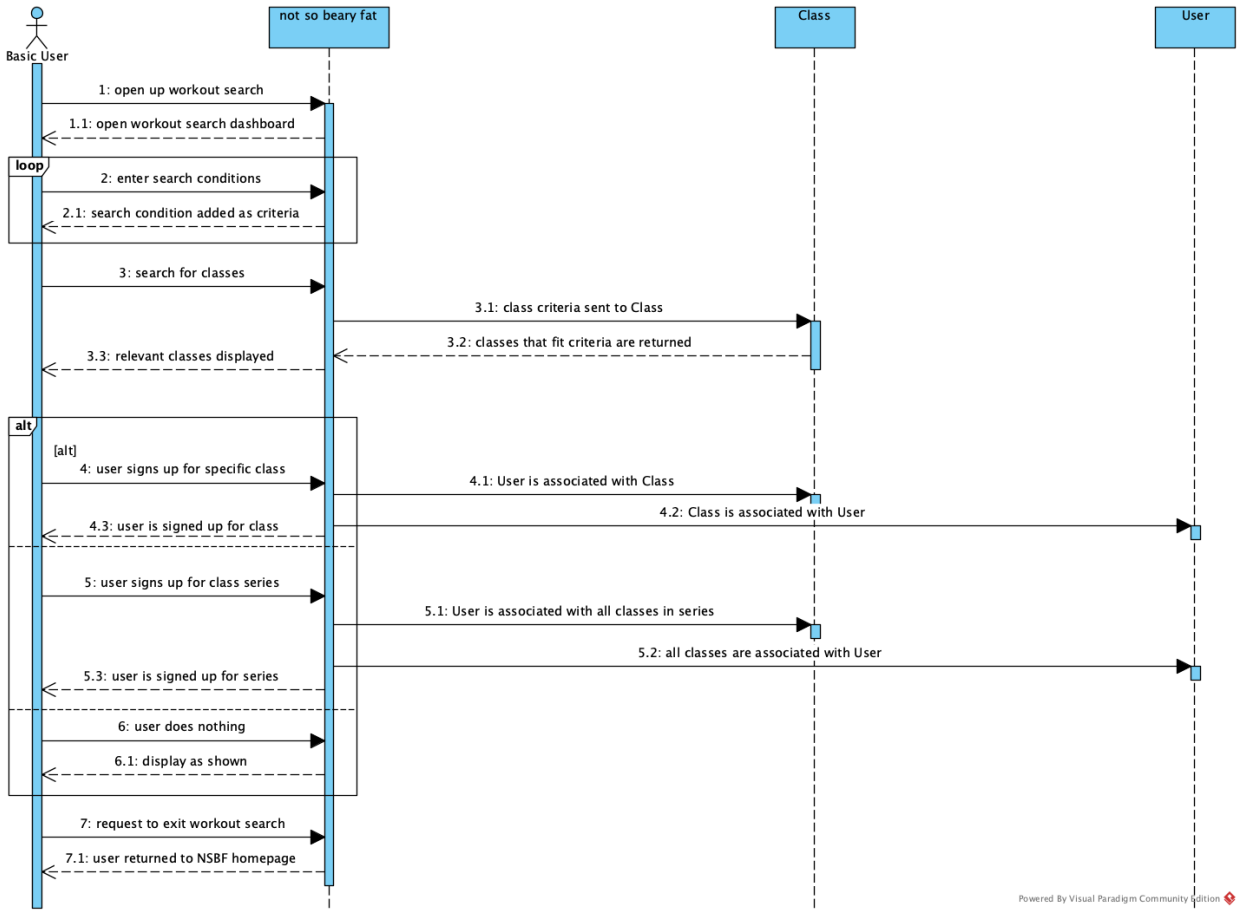
Alternate Success Scenario B:

1. User wishes view available classes without signing up for any
2. User searches for workout by name or filters for relevant classes
3. User exits workout search

SSD:


Basic User
not so beary fat
1: open up workout search
1.1: open workout search dashboard
loop
2: enter search conditions
2.1: search condition added as criteria
3: search for classes
3.1: relevant classes displayed
alt
[alt]
4: user signs up for specific class
4.1: user is signed up for class
5: user signs up for class series
5.1: user is signed up for series
6: user does nothing
6.1: display as shown
7: request to exit workout search
7.1: user returned to NSBF homepage
Powered by Visual Paradigm Community Edition

SD:



Operation Contract: Workout Search

Author: Owen Chipman

Contract:

- Operation name: Workout Search
- Responsibility: Allows for Basic Users to search for and sign up for workouts
- Precondition: Basic user is logged in and wants to search for a workout
- Postcondition: Basic user has reviewed the Not So Beary Fat workout offerings and selected a class of their choosing (Association created)

Trainer Statistics, Fully Dressed Use case, Owen Chipman

ID: Trainer Statistics

Scope: Not so Beary Fat app

Level: Trainer

Stakeholders and Interests:

Trainer:

- Should be able to log in to the trainer statistics portal
- Should be able to set time intervals by a day, week, and month basis
- Should be able to see progress of one individual
- Should be able to view the progress of demographics of clients based off of gender, age, and BMI
- Should be able to message a patient to send congratulations on great progress, or notify them that they need to lock in

Main Success Scenario:

1. Trainer wishes to see the progress of an individual user
2. Trainer logs in statistics portal
3. Trainer searches for that client by name
4. Trainer sets her desired timeframe
5. Weight, sleep, and caloric information are displayed to the screen

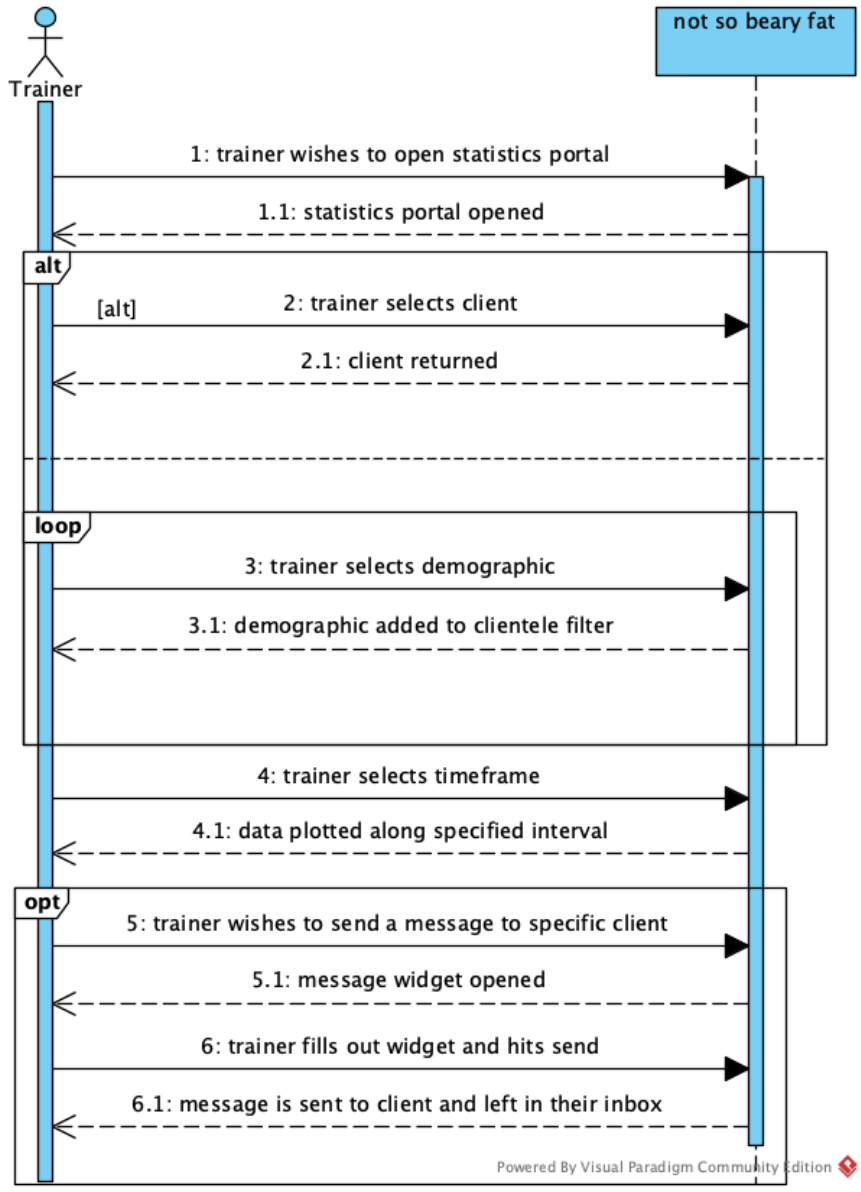
Alternate Success Scenario A:

1. Trainer wishes to the progress of her clients within a specific demographic
2. Trainer chooses the desired demographic
3. Trainer sets desired timeframe
4. Weight, sleep, and caloric information are displayed to the screen

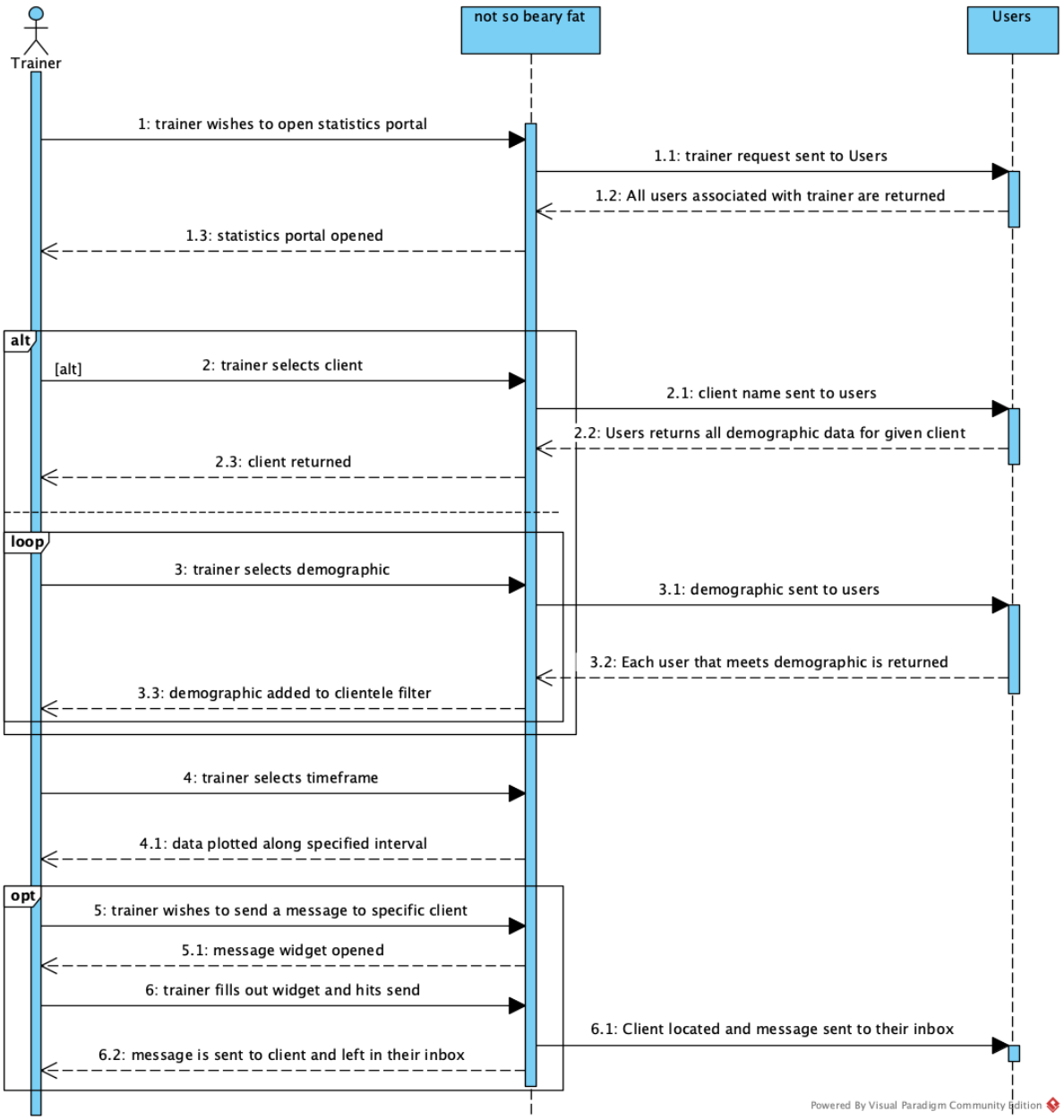
Alternate Success Scenario B:

1. Same as main success scenario A
2. Trainer is able to send a message of congratulations / encouragement to their client

SSD:


Trainer
not so beary fat
1: trainer wishes to open statistics portal
1.1: statistics portal opened
alt
[alt]
2: trainer selects client
2.1: client returned
loop
3: trainer selects demographic
3.1: demographic added to clientele filter
4: trainer selects timeframe
4.1: data plotted along specified interval
opt
5: trainer wishes to send a message to specific client
5.1: message widget opened
6: trainer fills out widget and hits send
6.1: message is sent to client and left in their inbox
Powered By Visual Paradigm Community Edition

SD:



Operation Contract: Trainer Statistics

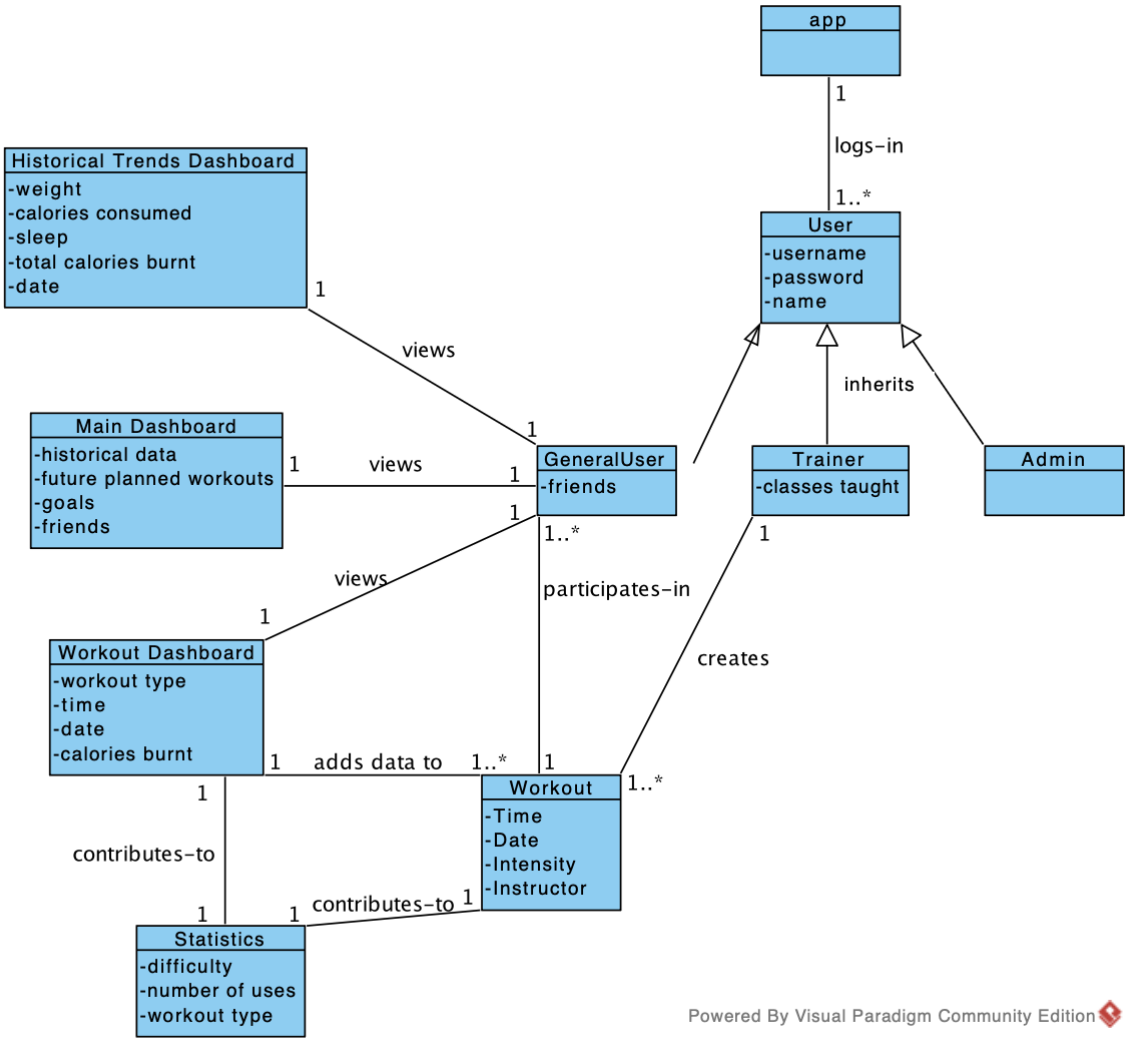
Author: Owen Chipman

Contract:

- Operation name: Trainer Statistics
- Responsibility: Allows for a trainer to view the progress over time of their clientele on an individualized or generalized basis
- Precondition: Trainer is logged in and wishes to view the progress of her clients

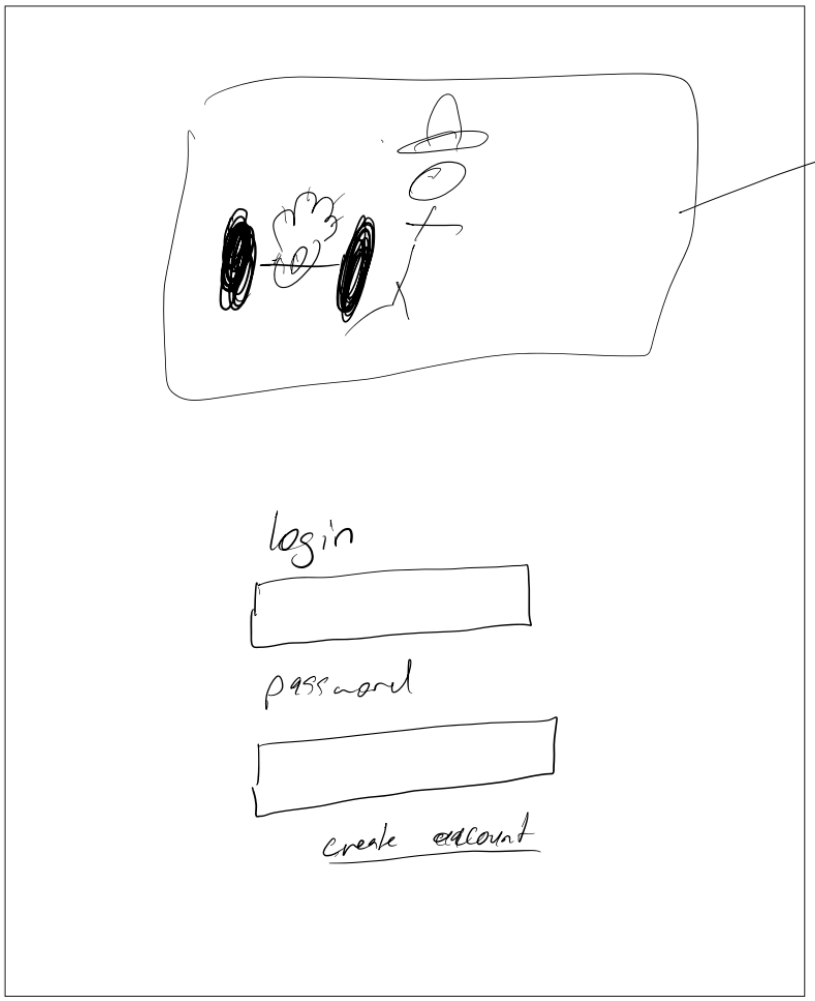
- Postcondition: Trainer is able to see the workout metrics of her clients and has sent out a word of encouragement / congratulations if deemed necessary (Association created)

Domain Model:

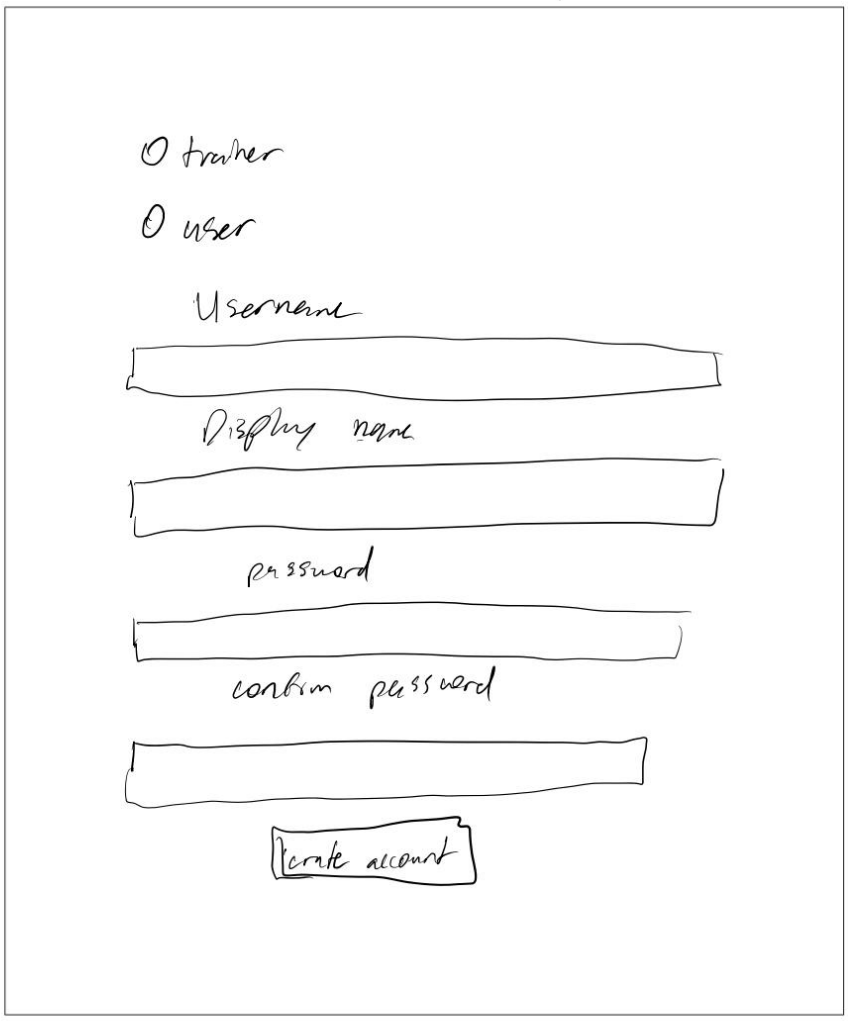

app
1
logs-in
1..*
User
-username
-password
-name
Historical Trends Dashboard
-weight
-calories consumed
-sleep
-total calories burnt
-date
1
views
1
inherits
Main Dashboard
-historical data
-future planned workouts
-goals
-friends
1
views
1
GeneralUser
-friends
Trainer
-classes taught
Admin
1
1..*
views
1
participates-in
1
creates
Workout Dashboard
-workout type
-time
-date
-calories burnt
1
adds data to
1..*
1
1..*
Workout
-Time
-Date
-Intensity
-Instructor
1
contributes-to
1
1
contributes-to
1
Statistics
-difficulty
-number of uses
-workout type
Powered By Visual Paradigm Community Edition

Wireframe:

Home Screen

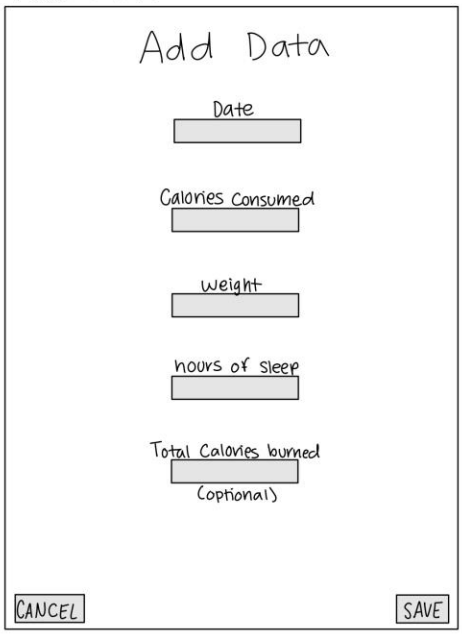


Create account page

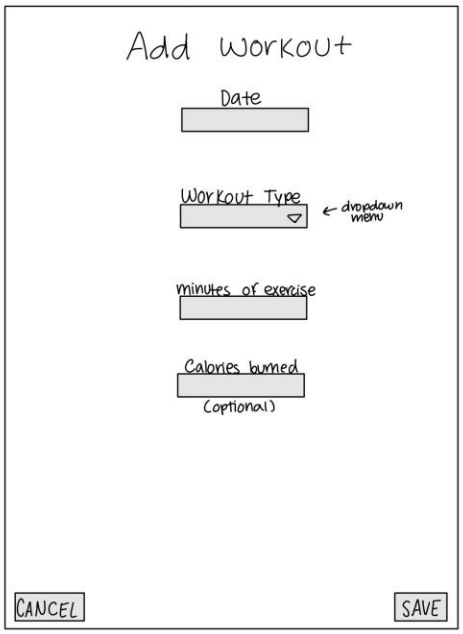


User Dashboard

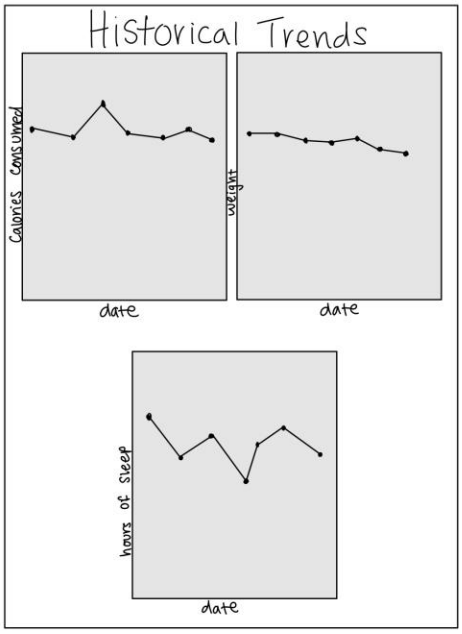
Add Data



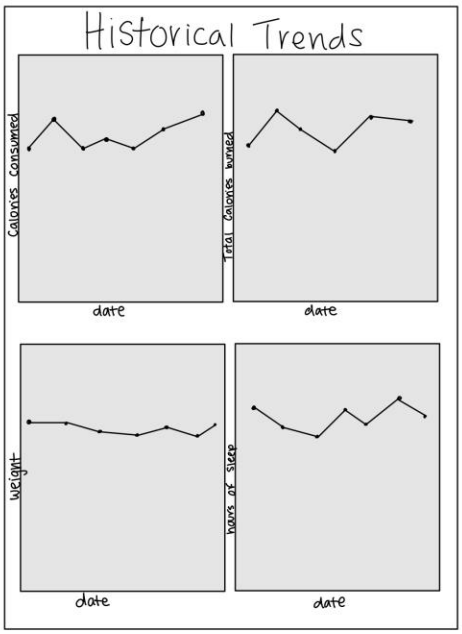
Add Workout



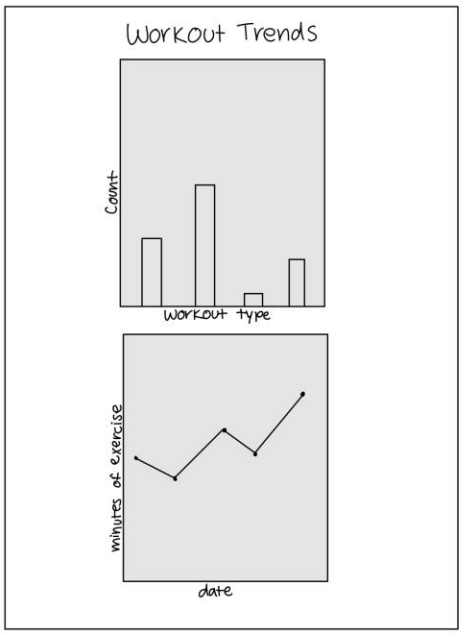
normal:



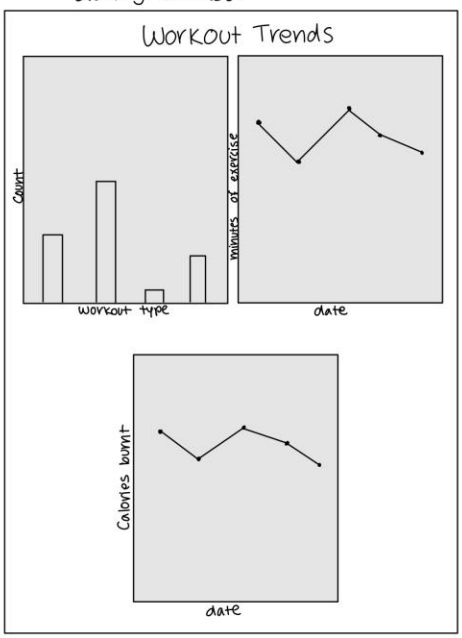
Alt. User tracks total calories burned:



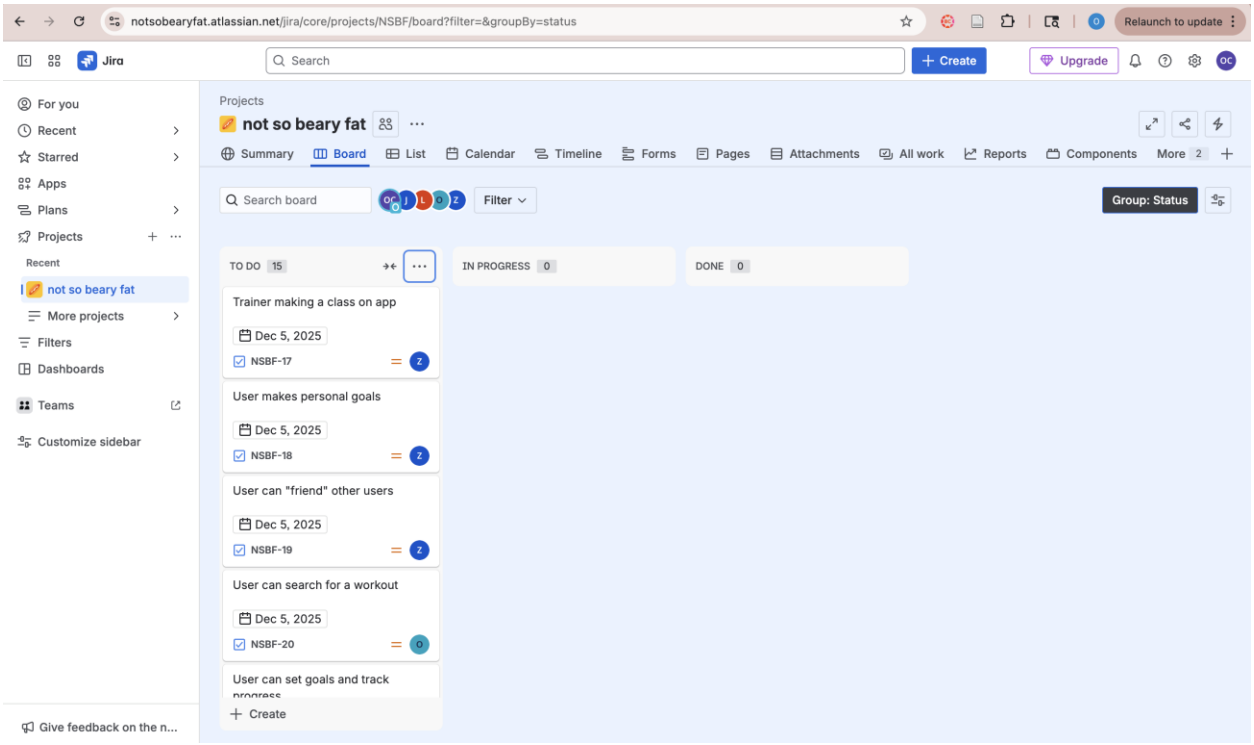
normal:

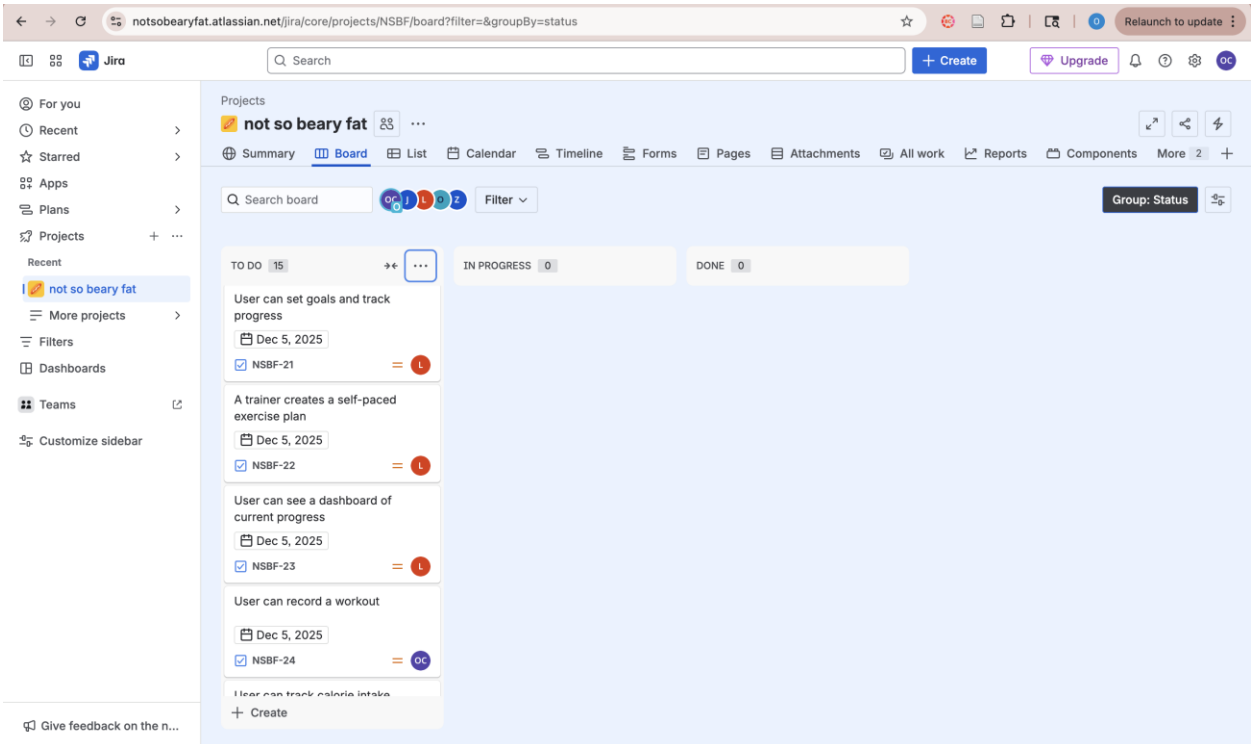


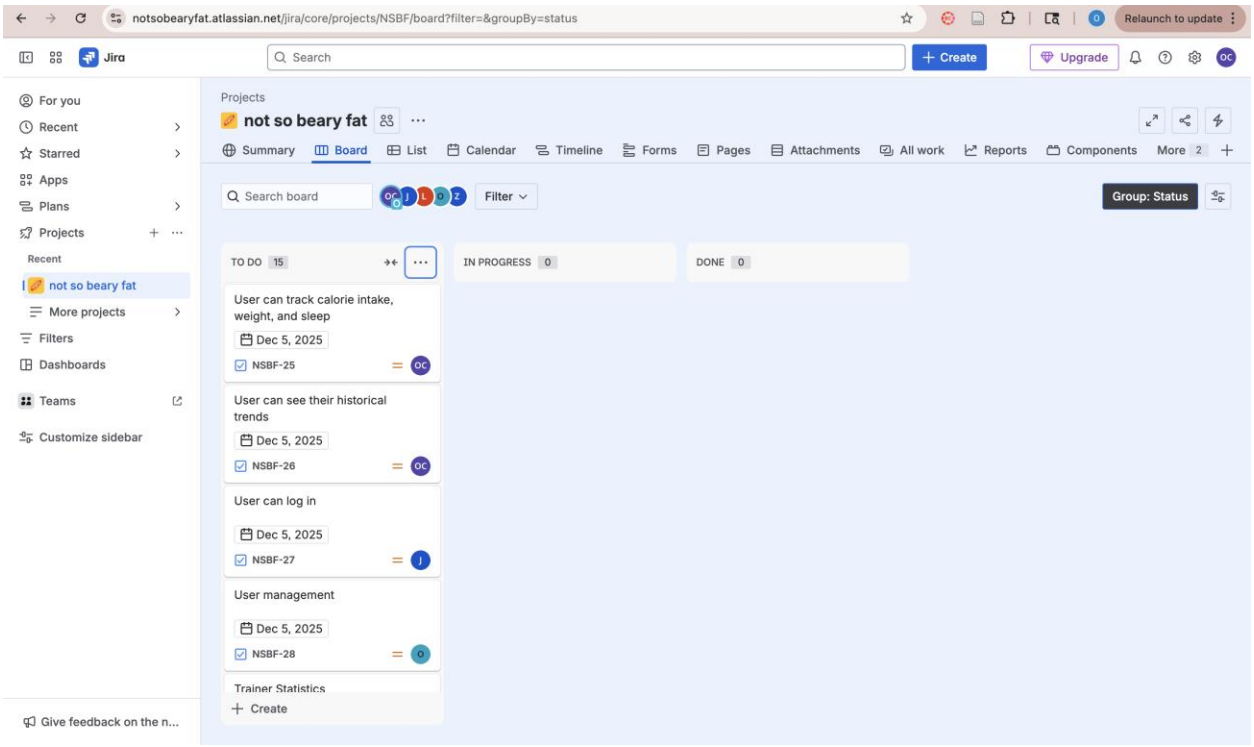
Alt. User tracks calories burnt during workout

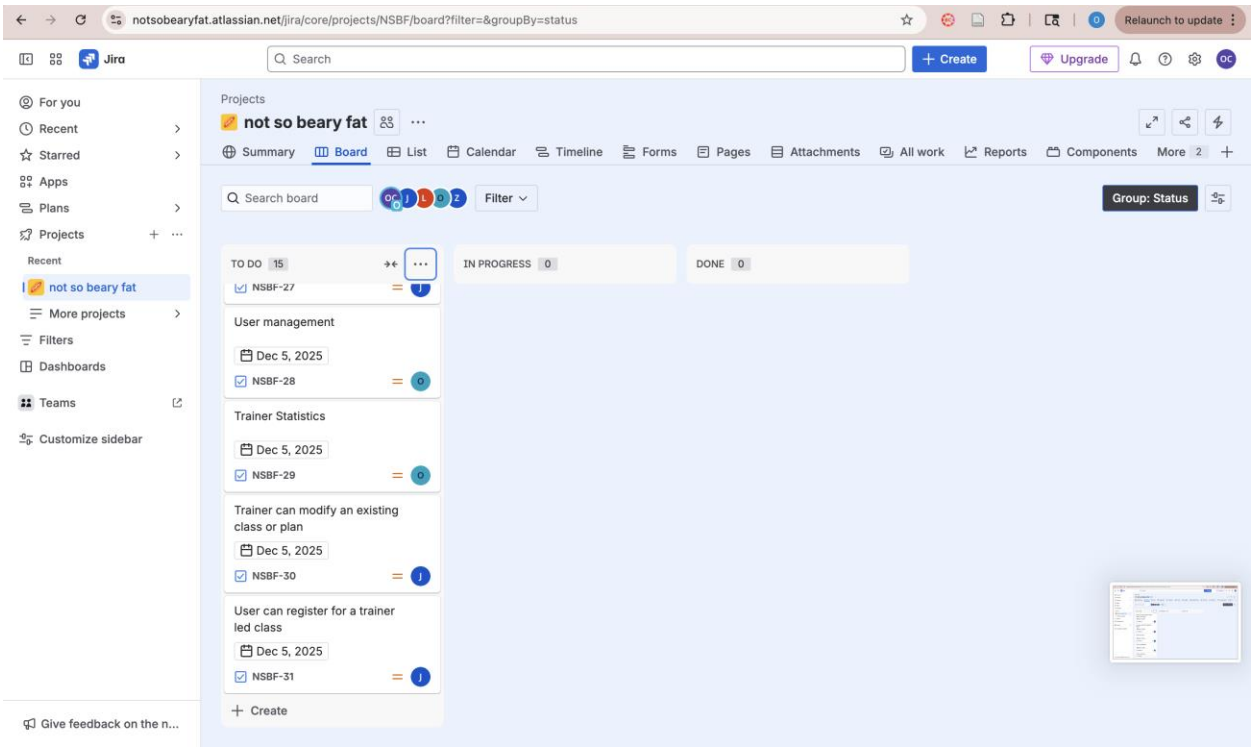


Trello:

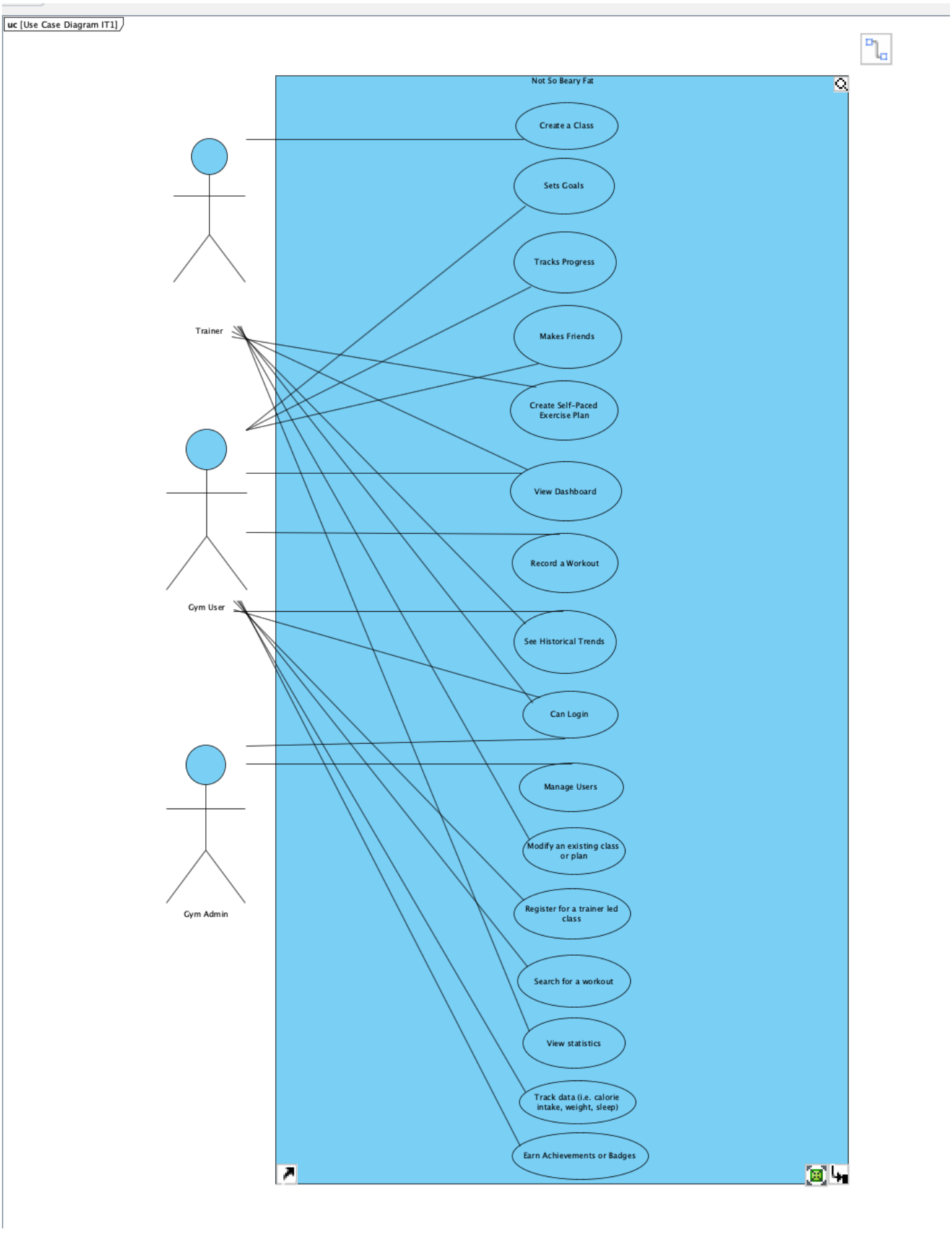

notsobearyfat.atlassian.net/jira/core/projects/NSBF/board?filter=&groupBy=status
Relaunch to update
Jira
Search
Create
Upgrade
For you
Recent
Starred
Apps
Plans
Projects
Recent
not so beary fat
More projects
Filters
Dashboards
Teams
Customize sidebar
Give feedback on the n...
Projects
not so beary fat
Summary
Board
List
Calendar
Timeline
Forms
Pages
Attachments
All work
Reports
Components
More 2
Search board
Filter
Group: Status
TO DO 15
IN PROGRESS 0
DONE 0
Trainer making a class on app
Dec 5, 2025
NSBF-17
User makes personal goals
Dec 5, 2025
NSBF-18
User can "friend" other users
Dec 5, 2025
NSBF-19
User can search for a workout
Dec 5, 2025
NSBF-20
User can set goals and track
Create


notsobearyfat.atlassian.net/jira/core/projects/NSBF/board?filter=&groupBy=status
Relaunch to update
Jira
Search
Create
Upgrade
For you
Recent
Starred
Apps
Plans
Projects
Recent
not so beary fat
More projects
Filters
Dashboards
Teams
Customize sidebar
Give feedback on the n...
Projects
not so beary fat
Summary
Board
List
Calendar
Timeline
Forms
Pages
Attachments
All work
Reports
Components
More 2
Search board
Filter
Group: Status
TO DO 15
IN PROGRESS 0
DONE 0
User can set goals and track progress
Dec 5, 2025
NSBF-21
A trainer creates a self-paced exercise plan
Dec 5, 2025
NSBF-22
User can see a dashboard of current progress
Dec 5, 2025
NSBF-23
User can record a workout
Dec 5, 2025
NSBF-24
Create


notsobearyfat.atlassian.net/jira/core/projects/NSBF/board?filter=&groupBy=status
Relaunch to update
Jira
Search
Create
Upgrade
For you
Recent
Starred
Apps
Plans
Projects
Recent
not so beary fat
More projects
Filters
Dashboards
Teams
Customize sidebar
Give feedback on the n...
Projects
not so beary fat
Summary
Board
List
Calendar
Timeline
Forms
Pages
Attachments
All work
Reports
Components
More 2
Search board
Filter
Group: Status
TO DO 15
IN PROGRESS 0
DONE 0
User can track calorie intake, weight, and sleep
Dec 5, 2025
NSBF-25
User can see their historical trends
Dec 5, 2025
NSBF-26
User can log in
Dec 5, 2025
NSBF-27
User management
Dec 5, 2025
NSBF-28
Trainer Statistics
Create



Use Case Diagram:


uc [Use Case Diagram IT1]
Not So Beary Fat
Create a Class
Sets Goals
Tracks Progress
Makes Friends
Create Self-Paced Exercise Plan
View Dashboard
Record a Workout
See Historical Trends
Can Login
Manage Users
Modify an existing class or plan
Register for a trainer led class
Search for a workout
View statistics
Track data (i.e. calorie intake, weight, sleep)
Earn Achievements or Badges
Trainer
Gym User
Gym Admin