DreamTeam, Inc.

DreamTeam Athletics

Project Test Plan Iteration 2 v2

Prepared by:
Ben Hancock-Song
Henry Howe
Sebastian Olson
Conor Wellman

Release Date
December 14, 2023

Table of Contents

Project Summary

It can be challenging to track a group's progress remotely. For example, coaches for college athletics teams lose the ability to train with athletes when they travel home for breaks. This can lead to people skipping workouts, overtraining, and greater risk of injury. Access to a community that shares your interests can increase motivation. Currently, there is a void of applications that encourage working out via an online community. This product can help to fill that void, benefitting both users and the virtual fitness market. In particular, focusing on interconnectedness will strengthen the likelihood that our user base increases at a faster rate, as users will recommend DreamTeam Athletics to friends, teammates, and workout partners.

Our goal is to create a simple interface in which a user can input workouts they complete, track their progress over time, and share their profile with other users. Additionally, we aim to have a workout tracking function that runs during your workout instead of needing to manually input afterwards. Our first integration of this aim will be a timer function that, upon stopping, adds the workout to the database. We would also like to add a GPS function that can track running or biking workouts. We would show users' locations on the map and track them through time and space. We would also like users to have the ability to attach comments and media to workouts that they can use to reflect later, including a pre-workout "readiness score" and images.

Another overarching goal for this project, most likely for a secondary iteration, is to implement an option to send your profile information to a designated account. This will allow users to compare progress and recreate some sense of community when working out alone. Additionally, it will enable sports teams and workout partners to track progress even from long distances. Finally, to build the sense of connectedness within the app, we hope to implement workout and challenge assignments. This will allow coaches to assign workouts and friends to challenge each other to create a sense of community on the app.

Document Versioning

Date	Owner	Comment
12/4/23	Ben Hancock-Song	Initial text, test case descriptions
12/4/23	Henry Howe	Test cases written
12/14/23	Conor Wellman	Test cases written
12/14/23	Ben Hancock-Song	Test cases written

Boundary Test Cases

Create_Profile

```
Private user passwordHash(password,user) {
        //modifies: hashmap userList/profile CSV file
        //returns: newly created user
        //throws: if name associated with user is already in hashmap

}
```

Obvious Tests:

- Username already exists:
  - Eg. username “bob” -> existing usernames “bob”

Load_Profile

```
private user readfromProfileCSV(password,user) {
        //returns: user found from running input against CSV of existing user profiles
        //throws: if name associated with user is not found in CSV file, if password associated
        //with inputted username doesn’t match

}
```

Obvious Tests:

Existing usernames contain username:

- username “bob” -> existing usernames “bobert, bobbeta”

Password does not match

- Password “passsword”-> existing password “passhword”

User_Model

```
private void writetoProfileCSV(password,user) {
        //modifies: profile list CSV file
        //throws: if profile is not found or user does not have permission to edit
}
```

Obvious Tests:

Existing usernames contain usernames:

- username “bob” -> existing usernames “bobert, bobbeta”

```
private array getUsers() {
        //returns: array of existing users
        //throws: if user list is empty
}
```
Obvious Tests:

Return correct user
- Return user “user” - User array{}

```
private void addUser(password,user) {
        //modifies: user list array
        //throws: if added user name matches a username that already exists, if password field is
        //left blank
}
```
Obvious Tests:

Add correct user:
- User array: {bob, bobbo} -> added user: “bob”

```
private void editUser(password,user) {
        //modifies: information on specific user
        //throws: if updated username matches a username that already exists, if password field is
        //left blank
}
```
Obvious Tests:
Edit username correctly, Check that edited user does not exist
- Edited username: “Bobby” -> existing usernames{ “bob”, “bobbert”, “bobby”}

Check that new password is not empty
- Entered Password: “”

```
public workouts_model getInstance(){
        //gets singleton instance of userModel class
}
```
Obvious Tests:
- Check only one instance exists
- Create instance, return instance

Edit_Profile

public String changeUsername(password,user) {
    //returns: new username as string
    //modifies: name associated with user
    //throws: if updated username matches a username that already exists
}
Obvious Tests:

Check that new username does not exist
- Edited username: “Bobby” -> existing usernames{ “bob”, “bobbert”, “bobby”}

public string changePassword(password,user) {
    //returns: new password as string
    //modifies: password associated with user
    //throws:if password field is left blank, if same password
}

Obvious Tests:

Check new password is not empty
- Edited password: “”

Check new password does not equal old password
- New password “New” -> Old password “old”

View_Friends
public void addFriend(user) {
    //modifies: friend list with new friend
    //throws: if entered username doesn't exist
    //throws: if already friend
}
Obvious Tests:
Entered username is equal to existing username
- Searched username: “Bobby” -> existing usernames{ “bob”, “bobbert”}
- Existing friends: {“Bob”, “Bobby”}

public string searchFriend(user_input) {
    //returns: searched friend
    //throws: if friend doesn't exist

```
}
```
Obvious Tests:

Method returns nothing when friend not added
- Searched friend: “NotBob” -> Existing friends: {“Bob”, “Bobby”}

Add_Workout
```
public exercise addExercise(type, duration) {
        //modifies: list of exercises associated with selected workout
        //returns: exercise object with associated data
        //throws: if exercise type does not exist, if type field is left blank,
        //if duration field is left blank
}
```
Obvious Tests:
Ensure exercise has all fields completed correctly
- Entered Exercise: “This is not what exercise inputs are supposed to look like”

```
public String addComment(input) {
        //modifies: comment list of a selected workout
        //returns: inputted comment as string
        //throws: if comment is left blank, if comment contains offensive language
}
```
Obvious Tests:
Handle empty comments
- Entered Comment: “”
- Entered Comment “*****”

```
public void submitWorkout(exercises[], string comment) {

        //modifies: list of completed workouts
        //throws: if workout doesn’t contain any exercises
}
```
Obvious Tests:
Ensure workout enters correctly and outputs in correct format
- Entered Workout: Exercises{}

Workouts_Model
```
public workouts[] getWorkouts() {
        //returns: workouts
        //throws: if invalid request
}
```

Obvious Tests:

Returns requested workout

- Requested Workout: existing workouts{}

```
public void addWorkouts() {

        //modifies: adds workout to model
        //throws: if invalid workout format
}
```

Obvious Tests:

Workout must have proper format to enter

- Entered Workout: “Incorrect Workout format”

```
public workouts_model getInstance(){
        //gets singleton instance of workouts_model class
}
```

Obvious Tests:

- Check only one instance exists
- Create instance, return instance

Edit_Workout

```
public exercise editExercise() {

        //modifies: edits previous entered exercise
        //throws: if new exercise format is identical or is entered incorrectly

}
```

Obvious Tests:

Ensure format is correct on edited workouts

- Edited exercise: “Incorrect format”

```
public string editComment() {

        //modifies: edits comment from previous workout
        //throws: if comment was selected incorrectly
        //throws if foul language
}
```

Obvious Tests:
Handle empty comments

- Entered Comment: “****”
- Entered Comment: “”

```
public void submitWorkout() {
        //modifies: submits edited workout
        //throws: if nothing was changed
}
```

Obvious Tests:
Handle duplicate workouts

- Submitted workout “Workout” -> existing workout “Workout”

Get_Workouts

```
public exercise[] getExercises() {

        //returns exercises
        //throws if exercise cannot be returned
}
```

Obvious Tests:

- Requested workout “Workout” -> existing workout “Something went wrong”

Workout

```
Public Workout() {
        //constructor for new workout object
}
```

Obvious tests:

- Create new workout, check all attributes not null

User

```
Public User() {
        //contructor for new user object
}
```

Obvious Tests:

- Create new user, check all attributes not null

WorkoutBuilder

```
Public workout workoutBuilder(){
        //TODO Replace with actual algorithm
        Return workout;
}'
```

Tests:

- Build workout, check completed workout

UserNormal

```
void update(){
        //TODO Replace with actual algorithm
}
```

Tests:

- Update user, check user updates

NetworkModel

```
Connection connect(){
        //TODO Replace with actual algorithm
}
```

Tests:

- Check network connection can be established
- Check network includes relevant databases
- Check for database editing permissions