Software Title:

MyFit App

Team Members:

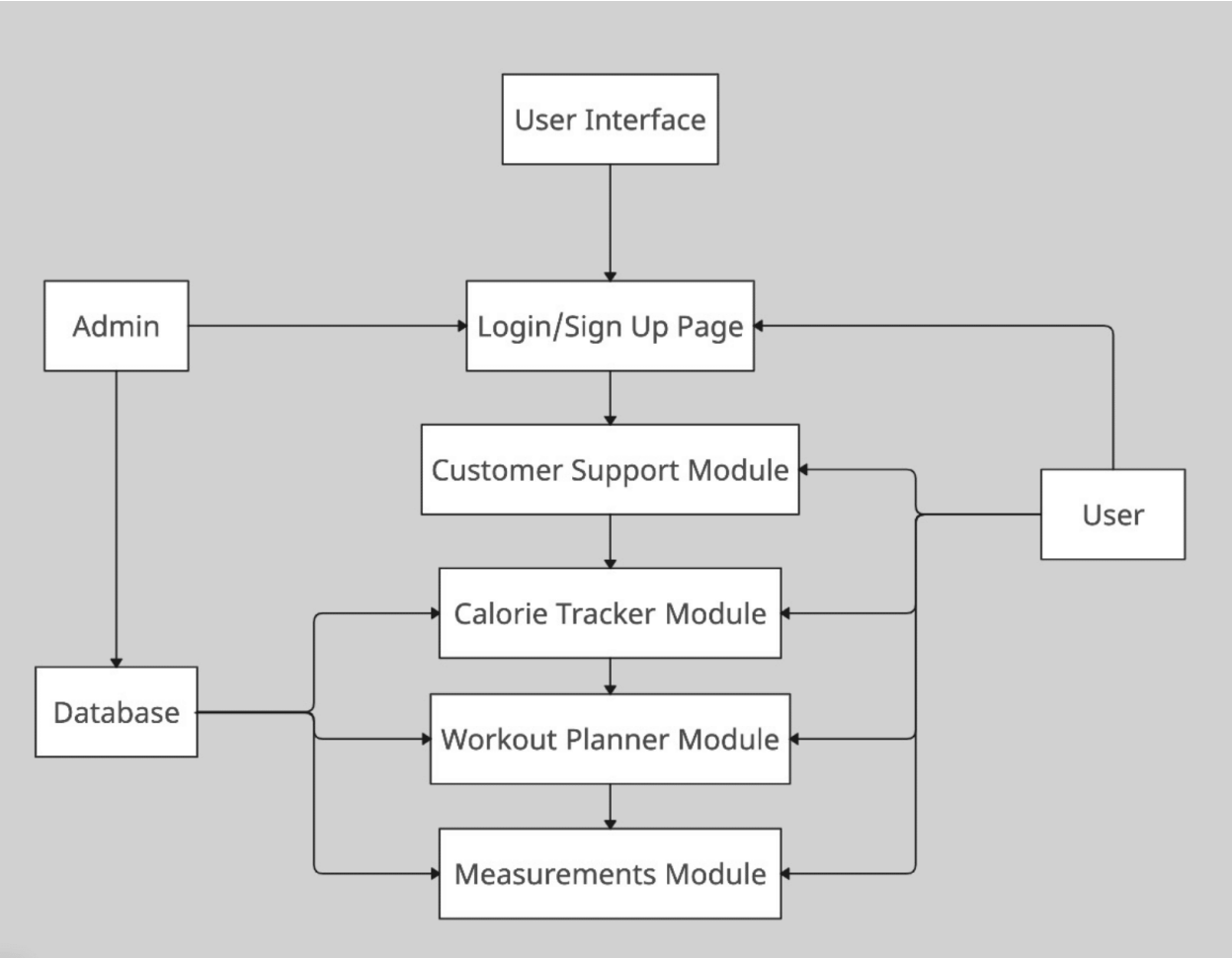
Teddy Barker
Sama Seif
Wissam Almasri
Umar Umar

System Overview:

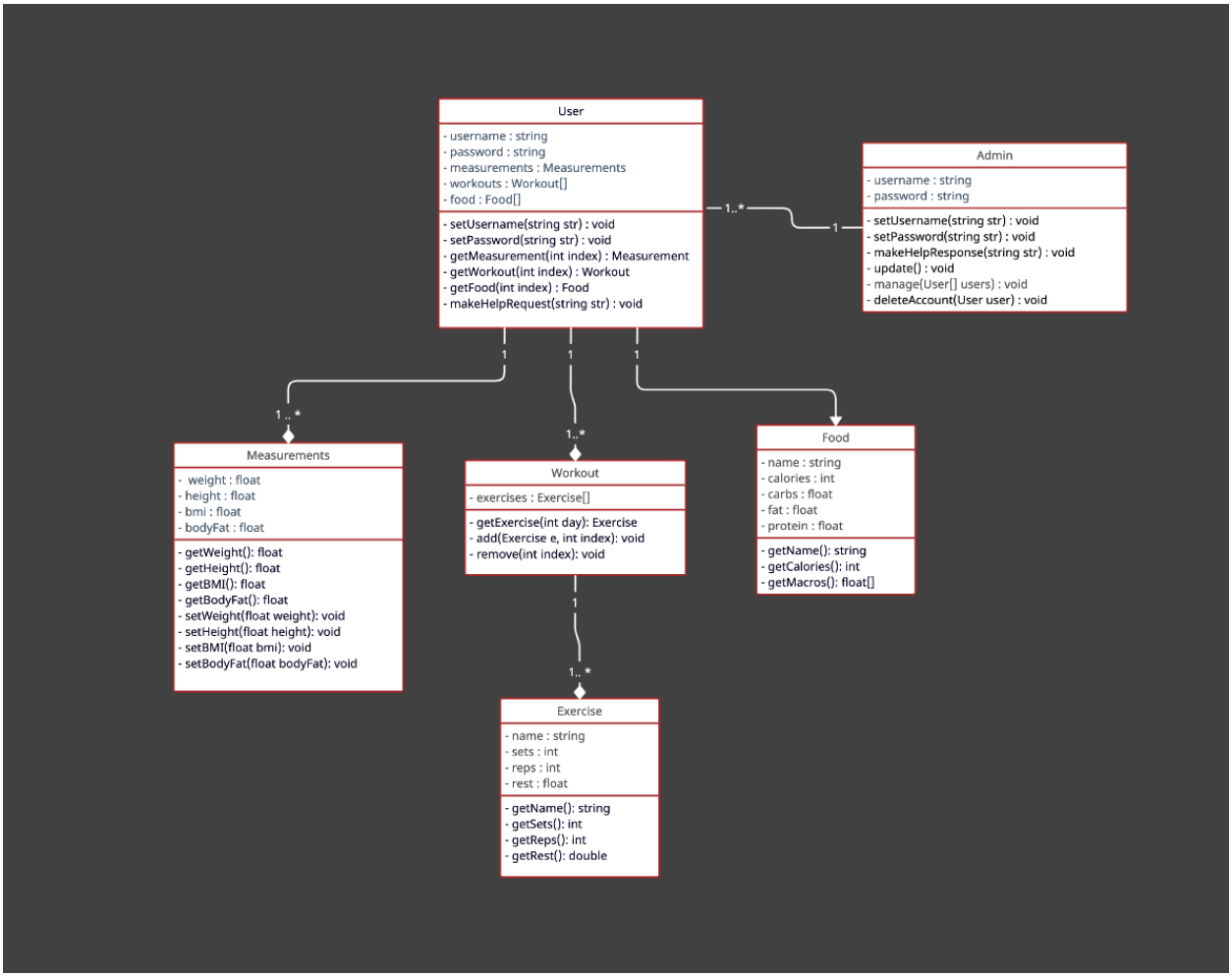
MyFit App serves the purpose of giving users the ability to track, monitor, and program their fitness goals by providing a dynamic user experience that solves multiple fitness goals including meal tracking, calorie counting, workout programming, weight monitoring, and goal setting. By utilizing this app and its features, users with the general goal of increasing their physical health and fitness can achieve their goal no matter what the specificity of their individualized requirements are. Overall, MyFit App aims to see the world become a healthier, happier place.

Software Architecture Overview

Architectural Diagram of all Major Components



UML Class Diagram



Class Descriptions:

- ❖ User
  - The user class controls a client that is using the app. It maintains the attributes of username, password, measurements, workouts, and food. User name is composed of 3 other defined classes and has an association with the Admin class.
  - Attributes
    - username: string that controls unique user identification and login details
    - password: encrypted string that controls user login details
    - measurements: instance of class measurements that controls the users personal information like height, weight, bmi, and body fat percentage.

        - workouts: list of workouts that compose the user's desired workout program
        - food: list of food items that the user has eaten throughout the days
    - Operations:
        - setUsername(): void function that allows user to change their username if they want to
        - setPassword(): void function that allows user to change their password if they want to
        - getMeasurement(): returns the measurements associated with the user
        - getWorkout(int index): returns the specified workout associated with the user
        - getFood(int index): returns the specified food item logged by the user
        - makeHelpRequest(string str): void function that controls a user customer service request

- Admin
    - The admin class controls an administrator of the company that has more selective control over aspects of the application such as maintenance and responding to customer service requests.
    - Attributes
        - username: string that controls unique admin identification and login details
        - password: encrypted string that controls admin login details
    - Operations:
        - setUsername(): void function that allows admin to change their username if they want to
        - setPassword(): void function that allows admin to change their password if they want to
        - makeHelpResponse(string str): void function that allows admin to respond to a request
        - update(): void function that allows the admin to make any necessary updates to the app
        - manageUser(User[] users): void function that allows the admin to manage a list of users

        - deleteAccount(User user): void function that allows the admin to delete a user's account if they want

- Measurements
    - This class resembles a collection of user measurements and manages attributes weight, height, body mass index (bmi), and body fat percentage (bodyFat). This allows users to see their progress change overtime so they can achieve their goals.
    - Attributes
        - weight: float that represents the user's weight in pounds
        - height: float that represents the user's height in inches
        - bmi: float that represents the user's body mass index
        - bodyFat: float that represents the user's body fat percentage
    - Operations
        - getWeight(): returns weight
        - getHeight(): returns height
        - getBMI(): returns bmi
        - getBodyFat(): returns bodyFat
        - setWeight(float weight): void function that changes weight
        - setHeight(float height): void function that changes height
        - setBMI(float bmi): void function that changes bmi
        - setBodyFat(float bodyFat): void function that changes bodyFat
- Workout
    - This class represents a collection of exercises that formulate a specific workout routine. It manages a list of exercises as an attribute.
    - Attributes
        - exercises: list of exercises that comprise the workout routine
    - Operations
        - getExercise(int day): returns the Exercise of the specified day in the workout
        - add(Exercise e, int index): void function that adds the specified exercise to the workout
        - remove(int index): void function that deletes an exercise from the program
- Exercise

- This class represents all the information for one exercise and manages a string for the name, two integers: sets and reps, and a float for the time in seconds of rest required to recover from this movement.
- Attributes:
  - name: string that describes the name of the exercise
  - sets: integer value for number of sets for desired exercise
  - reps: integer value for the number of reps per set
  - rest: float value to represent the rest time in seconds after exercise
- Operations
  - getName(): returns the string for the name of the exercise
  - getSets(): returns sets
  - getReps(): returns reps
  - getRest(): returns rest time

❖ Food

- This class represents one food item. It manages a name, calories, carbs, fat, and protein attributes.
- Attributes
  - name: string for the food item's name
  - calories: integer value of caloric specifications
  - carbs: float value of number of grams of carbs
  - fat: float value of the number of grams of fats
  - protein: float value of the number of grams of protein
- Operations
  - getName(): returns the name
  - getCalories: returns calories
  - getMacros: returns a float array of 3 elements of the calories, fats, and proteins

Development Plan and Timeline

Task	Team Member Responsible
planning and conceptualization (1-2 months)	Ted Leader
Market Research (1 week)	Umar
define Goals and objectives (1 week)	Umar
Conceptualize Features (2 weeks)	Wissam
Design Wireframes and protoypes (2 weeks)	Sama
Development (3 -6 months)	Ted Leader
Backend Development (2-3 months)	Ted
Fronttend Development (2-3 months)	Umar
nutrition tracking module (2-3 months)	Wissam
workout planning Module (2-3 months)	Sama
Integration with wearable Devices (1-2 weeks)	Umar
Testing and Quality Assurance (1-2 months)	Ted Leader
Unit Testing (2-3 weeks)	Wissam
Integration Testing (2-3 weeks)	Sama
User Acceptance Testing (3-4 weeks)	Ted
Deployment (1-2) Months	Ted Leader
App Store Submissio (2-3 weeks)	Umar and Wissam
Launch Marketing Campaign (1-2 weeks)	Sama
App Launch (1 day)	Team Work
Post-Launch (OnGoing)	Ted Leader
User Support and feedback (ongoing)	Wissam
Regular updates and feature Enhancements (ongoing)	Umar
Marketing and Growth (ongoing)	Sama
Bug fixes and maintenace (ongoing)	Ted

Verification Test Plan

Unit Tests:

- User Class Test Set:
  - Test 1: setUsername() and getUsername(): Test to ensure that the setUsername() and getUsername() functions in the User class work correctly.
    - Pseudo Code Examples of Test 1:

```
// Initialize a User object
user = User()

// Test 1: Check if setUsername() and getUsername() work correctly
user.setUsername("NewUsername")
if (user.getUsername() == "NewUsername")
      print("setUsername() and getUsername() functions work
      correctly.")
else
      print("Error: setUsername() or getUsername() functions are
      not working as expected.")
```

  - Test 2: getWorkout() with invalid index: Test to verify that the getWorkout() function properly handles invalid index inputs and returns an appropriate error message.
    - Pseudo Code Examples of Test 2:

```
// Initialize a User object
user = User()

// Workouts is a list of workout objects
workouts = {workout1, workout2, workout3} // Sample workout list

// Test 2: Check if getWorkout() function handles invalid index
properly
try:
      invalid_index = 5  // Index 5 is invalid
      user.getWorkout(invalid_index)
      print("Error: getWorkout() did not raise an exception for an
      invalid index.")
except IndexError:
      print("getWorkout() handles invalid index correctly.")
```

- Exercise Class Test Set:
  - Test 1: getSets() and getReps(): Test to confirm that the getSets() and getReps() functions in the Exercise class return the expected values.
    - Pseudo Code Examples of Test 1:

```
// Initialize an Exercise object
exercise = Exercise()

// Sets and reps are attributes of the Exercise class
exercise.setSets(3) // Setting the number of sets to 3
exercise.setReps(12) // Setting the number of reps to 12

// Test 1: Check if getSets() and getReps() functions work
correctly
if (exercise.getSets() == 3 && exercise.getReps() == 12)
      print("getSets() and getReps() functions work correctly.")
else
      print("Error: getSets() or getReps() functions are not
      working as expected.")
```

- Test 2: getRest() with negative rest time: Test to ensure that the getRest() function properly handles negative rest time inputs and returns an appropriate error message.
  - Pseudo Code Examples of Test 2:

```
// Initialize an Exercise object
exercise = Exercise()

// Rest is an attribute of the Exercise class
exercise.setRest(-5)  // Setting a negative rest time

// Test 2: Check if getRest() function handles negative rest time
properly
if (exercise.getRest() >= 0)
      print("Error: getRest() did not handle negative rest time
      properly.")
else
      print("getRest() handles negative rest time correctly.")
```

- Measurements Class Test Set:
  - Test 1: getHeight() and getWeight(): Test ensures that the getHeight() and getWeight() functions in the Measurements class are functioning correctly.
    - Pseudo Code Examples of Test 1:

```
// Initialize a Measurements object
measurements = Measurements()

// Height and weight are attributes of the Measurements class
measurements.setHeight(68.5)  // Setting the height to 68.5 inches
measurements.setWeight(150.2)  // Setting the weight to 150.2
pounds
```

```
// Test 1: Check if getHeight() and getWeight() functions work
correctly
if (measurements.getHeight() == 68.5 && measurements.getWeight()
== 150.2)
      print("getHeight() and getWeight() functions work
      correctly.")
else:
      print("Error: getHeight() or getWeight() functions are not
      working as expected.")
```

- Test 2: setBMI() with valid BMI value: Test validates that the setBMI() function in the Measurements class works correctly with a valid BMI value.
  - Pseudo Code Examples of Test 2:

```
// Initialize a Measurements object
measurements = Measurements()

//Bmi is an attribute of the Measurements class
valid_bmi = 24.5  // 24.5 is a valid BMI value
measurements.setBMI(valid_bmi)

// Test 2: Check if setBMI() function works correctly with a valid
BMI value
if (measurements.getBMI() == valid_bmi)
      print("setBMI() function works correctly with a valid BMI
      value.")
else
      print("Error: setBMI() function is not working as
      expected.")
```

- Food Class Test Set:
  - Test 1: getName() and getCalories(): Test verifies that the getName() and getCalories() functions in the Food class work correctly.
    - Pseudo Code Examples of Test 1:

```
// Initialize a Food object
food = Food()

// Name and calories are attributes of the Food class
food.setName("Chicken Salad")  // Setting the food name
food.setCalories(350)  // Setting the calorie value

// Test 1: Check if getName() and getCalories() functions work
correctly
if (food.getName() == "Chicken Salad" && food.getCalories() ==
350)
```

```
        print("getName() and getCalories() functions work
        correctly.")
else
        print("Error: getName() or getCalories() functions are not
        working as expected.")
```

- Test 2: getMacros() with valid macro values: Test ensures that the getMacros() function in the Food class works correctly with valid macro values.
  - Pseudo Code Examples of Test 2:

```
// Initialize a Food object
food = Food()

// Fats, proteins, and calories are attributes of the Food
class
valid_fats = 12.5  // 12.5 grams is a valid fat value
valid_proteins = 25.2  // 25.2 grams is a valid protein value
valid_calories = 350  // 350 is a valid calorie value
food.setFats(valid_fats)
food.setProteins(valid_proteins)
food.setCalories(valid_calories)

// Test 2: Check if getMacros() function works correctly with
valid macro values
macros = food.getMacros()
if (macros[0] == valid_calories and macros[1] == valid_fats and
macros[2] == valid_proteins)
      print("getMacros() function works correctly with valid macro
      values.")
else
      print("Error: getMacros() function is not working as
      expected.")
```

Functional Tests:

- Test 1: Workout and Exercise Functionality Test
  - Description: This test ensures the proper functioning of the workout and exercise components within the application. It involves adding and removing exercises in a workout routine and checking the accuracy of exercise information retrieval.
  - Expected Outcome: The test should demonstrate that exercises can be effectively managed within the workout routine, and exercise information is readily accessible and accurate.
- Test 2: User and Measurements Functionality Test

- Description: This test aims to validate the seamless interaction between the user and measurements components of the application. It focuses on setting and retrieving user-specific information, including username, measurements, workouts, and food items.
- Expected Outcome: The test should verify that user-related data can be properly set and retrieved, ensuring the application's accurate handling of user-specific information.

❖ Test 3: Meal Tracking Test

- Description: This test concentrates on the meal tracking functionality within the application. It involves logging food items and ensuring accurate calorie counting and nutritional tracking.
- Expected Outcome: The test should confirm that the application effectively logs food items, accurately calculates nutritional intake, and provides users with a comprehensive meal tracking feature.

System Tests:

❖ Test 1: Authentication and Access Control Test

- Objective: Verify the authentication system's integrity and the effectiveness of access control mechanisms.
- Steps:
  - Attempt to log in with valid credentials and ensure successful access.
  - Try logging in with invalid credentials and confirm appropriate denial of access.
- Expected Outcome: The authentication system should accurately authenticate valid users and prevent unauthorized access attempts effectively.

❖ Test 2: Account Management and Data Security Test

- Objective: Ensure the secure management of user accounts and the robustness of data security measures.
- Steps:
  - Test the account deletion process and ensure all user data is appropriately removed.
  - Check for any potential data remnants or security vulnerabilities after account deletion.

- Expected Outcome: The test should confirm that the account deletion process results in the complete removal of user data, with no residual data or security loopholes left in the system.