Object Oriented Analysis and Design
Project 7

Topic: Fitness Tracking app

Team Members: Prateek Kumar, Rohan Eswara

Final State of the system: The final state of our system is a fully functional fitness tracking application that allows users to create workout plans, track their progress, and monitor their nutrition intake. The implemented features include user authentication, user profile creation and management, nutrition tracking, workout tracking, workout intensity setting, workout history, and a user-friendly interface with modern CSS. The system also utilizes a factory pattern to create class instances and a strategy pattern to implement the workout intensity setting feature. However, due to time constraints and prioritization, some planned features such as social media sharing and workout plan sharing were not implemented. In comparison to Project 5 and 6, the final version of the application has undergone minor changes, including a switch from Flask to react as the back-end server and the addition of nutrition tracking and modern CSS. Overall, the final state of our system is a robust fitness tracking application that fulfills its intended purpose.

Final Class Diagram and Comparison Statement:

(Project7, Project5 Diagram at the end of this doc in that order)

Comparison:

Project 5 and 6 had the Observer-Subscriber pattern to be implemented for the logger class. But we then went ahead and used the default logger package of the AWS lambda function and used the inbuilt Observer-Publisher functionality of the react-app to publish it to the db. We also got rid of the Food List because it felt redundant and since we were not using the food items in any other classes, the decision was easy to make. We added Abstract Factory Class instead of isolated Factory Class to implement Social Media Decorator Class.

Third-Party Code vs. Original Code:

There was no direct code used from other third-party sources however we referenced the templates of the various frameworks and tools that were implemented in the code. Here's the list of URLs that were referenced:

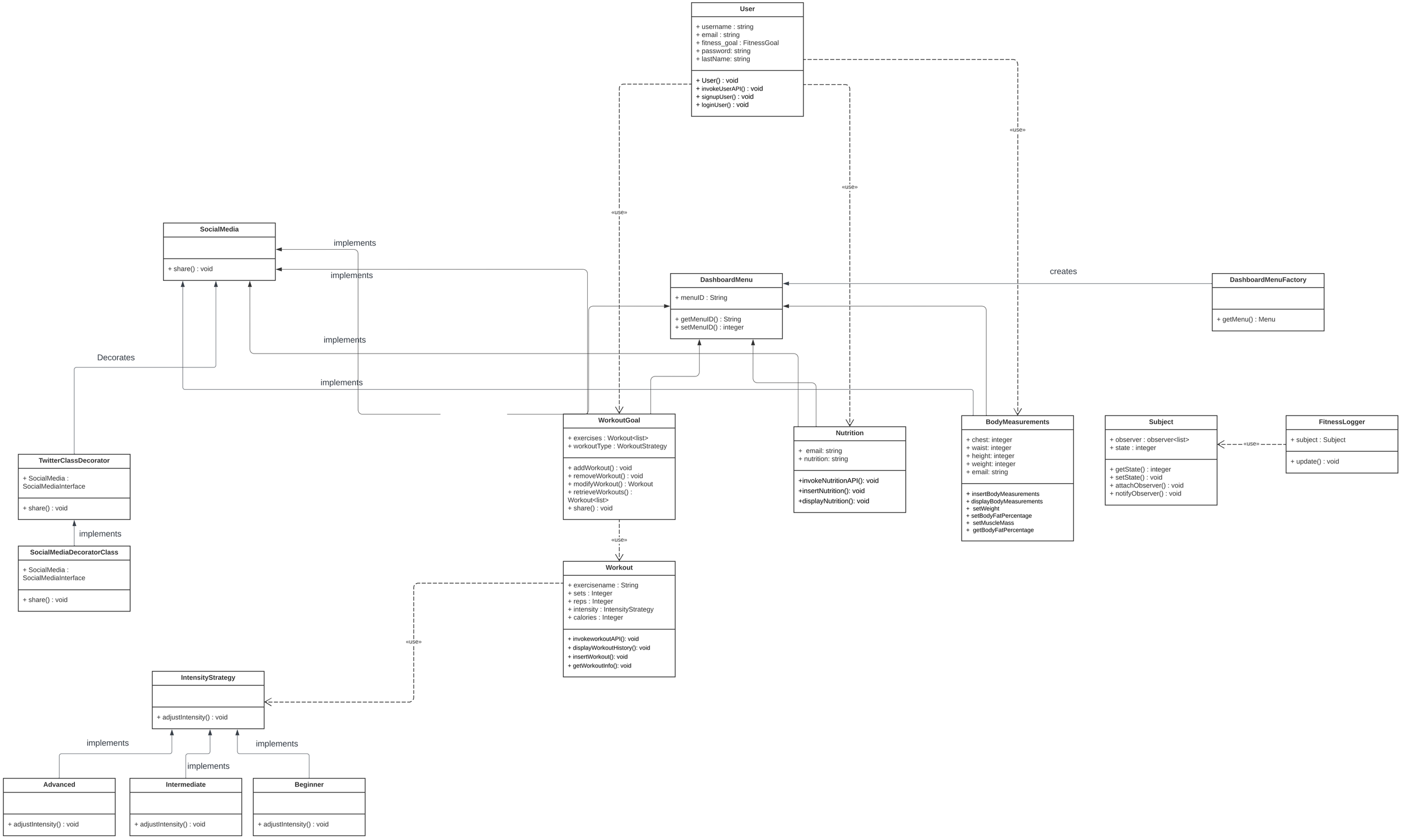
[1] https://docs.aws.amazon.com/lambda/latest/dg/python-handler.html

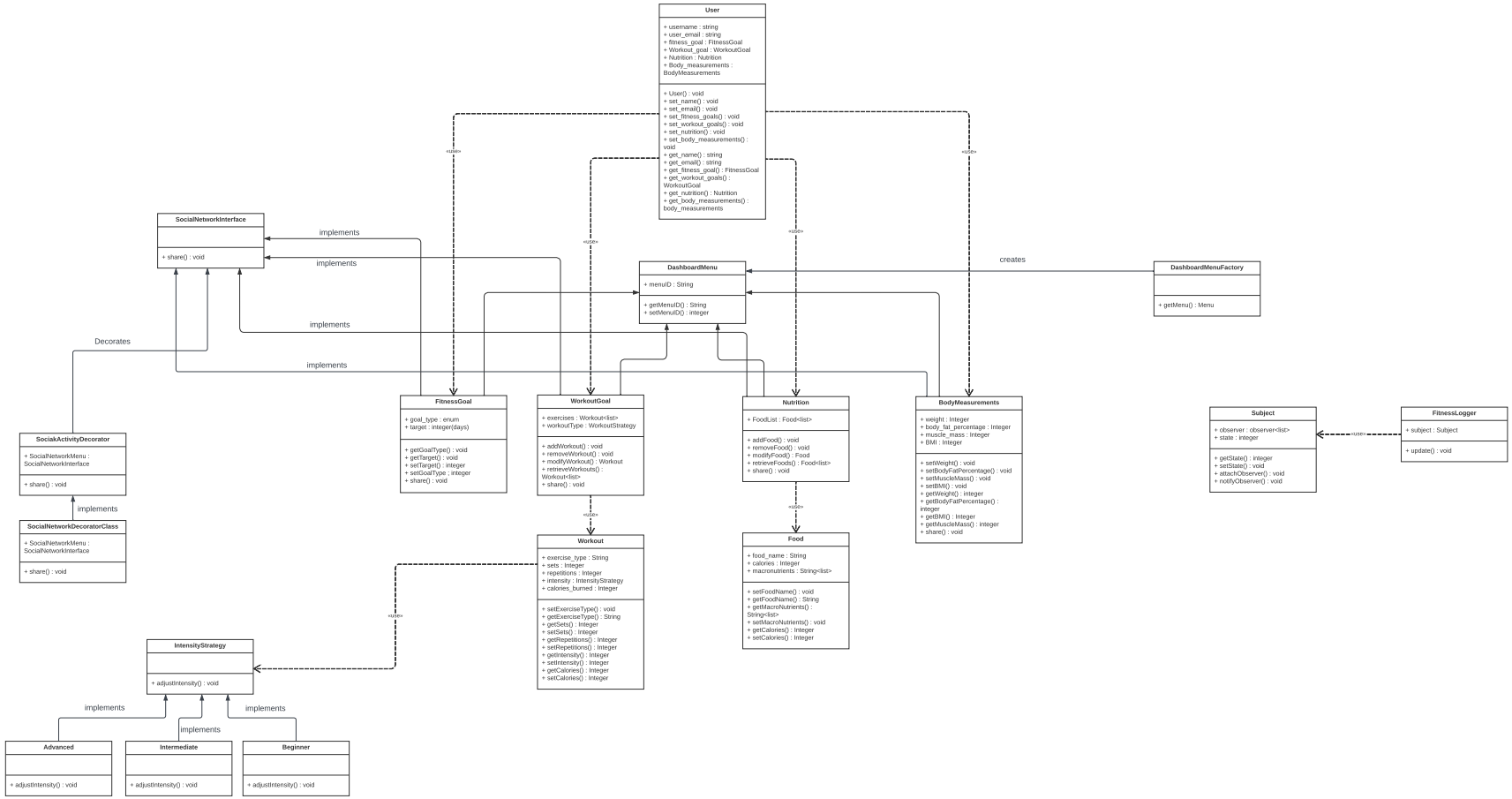
[2] https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

[3] https://react-tutorial.app/app.html
[4] https://aws.amazon.com/getting-started/hands-on/create-nosql-table/
[5] https://stackoverflow.com/questions/15900338/python-request-post-with-param-data

OOAD Process:

1. Use of Design Patterns: One of the key design process elements we used was the implementation of design patterns such as the Singleton pattern, Factory pattern, Observer pattern, and Strategy pattern. The Singleton pattern was used to ensure that only one instance of a class was created, such as for the database connection. The Factory pattern was used to create instances of classes, such as for creating different types of exercises. The Observer pattern was used to notify the UI of changes in the data model, such as when a new workout is added. The Strategy pattern was used to implement different workout intensities based on user preferences.
2. Design of Relationships: Another important design process element we used was the design of relationships between classes and objects. For example, we had a User class that had a relationship with the Workout and Nutrition classes. The User class had a list of Workouts and Nutritions that were associated with it. We also had a relationship between the Exercise and Workout classes. An instance of an Exercise was associated with a Workout instance.
3. Algorithm Design: Lastly, we used algorithm design as a key process element during the design of the fitness application. For example, we had to design algorithms for calculating the number of calories burned during a workout, based on user input such as sets, reps of a workout. We also had to design algorithms for generating a workout plan based on user preferences such as exercise type and intensity. The algorithm design process involved identifying the steps involved in the calculations and determining the most efficient and accurate way to perform the calculations.


User
+ username : string
+ email : string
+ fitness_goal : FitnessGoal
+ password: string
+ lastName: string
+ User() : void
+ invokeUserAPI() : void
+ signupUser() : void
+ loginUser() : void
«use»
«use»
«use»
SocialMedia
+ share() : void
implements
implements
implements
implements
DashboardMenu
+ menuID : String
+ getMenuID() : String
+ setMenuID() : integer
creates
DashboardMenuFactory
+ getMenu() : Menu
Decorates
TwitterClassDecorator
+ SocialMedia : SocialMediaInterface
+ share() : void
implements
SocialMediaDecoratorClass
+ SocialMedia : SocialMediaInterface
+ share() : void
WorkoutGoal
+ exercises : Workout<list>
+ workoutType : WorkoutStrategy
+ addWorkout() : void
+ removeWorkout() : void
+ modifyWorkout() : Workout
+ retrieveWorkouts() : Workout<list>
+ share() : void
«use»
Workout
+ exercisename : String
+ sets : Integer
+ reps : Integer
+ intensity : IntensityStrategy
+ calories : Integer
+ invokeworkoutAPI(): void
+ displayWorkoutHistory(): void
+ insertWorkout(): void
+ getWorkoutInfo(): void
«use»
Nutrition
+ email: string
+ nutrition: string
+invokeNutritionAPI(): void
+insertNutrition(): void
+displayNutrition(): void
BodyMeasurements
+ chest: integer
+ waist: integer
+ height: integer
+ weight: integer
+ email: string
+ insertBodyMeasurements
+ displayBodyMeasurements
+ setWeight
+ setBodyFatPercentage
+ setMuscleMass
+ getBodyFatPercentage
Subject
+ observer : observer<list>
+ state : integer
+ getState() : integer
+ setState() : void
+ attachObserver() : void
+ notifyObserver() : void
«use»
FitnessLogger
+ subject : Subject
+ update() : void
IntensityStrategy
+ adjustIntensity() : void
implements
implements
implements
Advanced
+ adjustIntensity() : void
Intermediate
+ adjustIntensity() : void
Beginner
+ adjustIntensity() : void


User
+ username : string
+ user_email : string
+ fitness_goal : FitnessGoal
+ Workout_goal : WorkoutGoal
+ Nutrition : Nutrition
+ Body_measurements : BodyMeasurements
+ User() : void
+ set_name() : void
+ set_email() : void
+ set_fitness_goals() : void
+ set_workout_goals() : void
+ set_nutrition() : void
+ set_body_measurements() : void
+ get_name() : string
+ get_email() : string
+ get_fitness_goal() : FitnessGoal
+ get_workout_goals() : WorkoutGoal
+ get_nutrition() : Nutrition
+ get_body_measurements() : body_measurements
«USE»
SocialNetworkInterface
+ share() : void
implements
DashboardMenu
+ menuID : String
+ getMenuID() : String
+ setMenuID() : integer
creates
DashboardMenuFactory
+ getMenu() : Menu
Decorates
FitnessGoal
+ goal_type : enum
+ target : integer(days)
+ getGoalType() : void
+ getTarget() : void
+ setTarget() : integer
+ setGoalType : integer
+ share() : void
WorkoutGoal
+ exercises : Workout<list>
+ workoutType : WorkoutStrategy
+ addWorkout() : void
+ removeWorkout() : void
+ modifyWorkout() : Workout
+ retrieveWorkouts() : Workout<list>
+ share() : void
Nutrition
+ FoodList : Food<list>
+ addFood() : void
+ removeFood() : void
+ modifyFood() : Food
+ retreveFoods() : Food<list>
+ share() : void
BodyMeasurements
+ weight : Integer
+ body_fat_percentage : Integer
+ muscle_mass : Integer
+ BMI : Integer
+ setWeight() : void
+ setBodyFatPercentage() : void
+ setMuscleMass() : void
+ setBMI() : void
+ getWeight() : integer
+ getBodyFatPercentage() : integer
+ getBMI() : Integer
+ getMuscleMass() : integer
+ share() : void
SociakActivityDecorator
+ SocialNetworkMenu : SocialNetworkInterface
+ share() : void
SocialNetworkDecoratorClass
+ SocialNetworkMenu : SocialNetworkInterface
+ share() : void
Workout
+ exercise_type : String
+ sets : Integer
+ repetitions : Integer
+ intensity : IntensityStrategy
+ calories_burned : Integer
+ setExerciseType() : void
+ getExerciseType() : String
+ getSets() : Integer
+ setSets() : Integer
+ getRepetitions() : Integer
+ setRepetitions() : Integer
+ getIntensity() : Integer
+ setIntensity() : Integer
+ getCalories() : Integer
+ setCalories() : Integer
Food
+ food_name : String
+ calories : Integer
+ macronutrients : String<list>
+ setFoodName() : void
+ getFoodName() : String
+ getMacroNutrients() : String<list>
+ setMacroNutrients() : void
+ getCalories() : Integer
+ setCalories() : Integer
«USE»
IntensityStrategy
+ adjustIntensity() : void
implements
Advanced
+ adjustIntensity() : void
Intermediate
+ adjustIntensity() : void
Beginner
+ adjustIntensity() : void
Subject
+ observer : observer<list>
+ state : integer
+ getState() : integer
+ setState() : void
+ attachObserver() : void
+ notifyObserver() : void
«USE»
FitnessLogger
+ subject : Subject
+ update() : void