Project Overview

Garth Terlizzi, Parakh Jaggi

Estimated time worked on project: 65 hours each

April 26, 2018

The Nutrition Trakr is a personal calorie and exercise tracker with Java FX and Apache Derby implementation. After a quick registration and survey, consisting of the user's height, weight, and dimensions the user is free to start tracking calories with our database of categories and foods. If the user eats a food that currently isn't in the database, then he or she will have the option to add the food and the number of calories that food has to the database. The user can also do the same with exercises. The user can select an exercise that he or she has completed, or they can add a new exercise for all users to see. The users can also see their monthly progress with a colorful bar chart.

Although the Fitness Trakr seems like a single person app, the users can actually compete with each other. Using our leaderboard, the users can compare their Trakr scores and try to one up each other to get the coveted first place. If you do not want to compete with other users, you can still get a sense of accomplishment with our ribbon system. A great Trakr score will give the user a Gold ribbon, an average score a Silver ribbon, and a bad score a Bronze ribbon.

The user will be able to register and the user account remains after the termination of the program. The user will be able to input what foods he has eaten for the day, and the calories associated with the food are added to their tracker. The user will be able to input what foods he has eaten for the day, and the exercise associated with the food are added to their tracker. The user will be able to view his Fitness Score, a calculation based off the information in his tracker. The user should be able to view his BMI/ BFA. The user should be able to see the graphs based on information from the trackers, regardless if there is a lack of information from the tracker. The user will be able to view his Fitness Score compared to everyone else in the database, leaderboard style. The user should be able to view his goals and edit his goals.

Usability:

The program must have a working login/sign up page and a dashboard with buttons leading to working pages.

The dashboard should be simple, with buttons leading to the actions.

The graphics should be monochromatic, not flashy.

The program should be well documented

Reliability:

The program should always open on start-up.

Input Errors should be validated.

Errors should result in a message box, then close the program.

Performance:

The program should have a relatively fast response time for database retrieval.

The program should not use any threads or event that causes multiple processes to run.

Supportability:

The program should be compatible on Windows and IOS devices.

Business Rules:

The user needs to have an account in order to access the dashboard.

The user should not create an account if their email is already tied to another account.

The user should be able to use outside utilities (scale, tape measurer, height-market) to know their BFA attributes.

The user should know how many calories they ate/burned from outside sources if the information is not contained within the database.

Vision Statement

To provide an opportunity for health-conscious people to set their goals and then track their lifestyle habits quantitatively and view their progress over time.

Junit Testing Analysis

The Junit tests covers mostly database alteration, as our application is a database-driven project. There are test cases resolving the inserts of food, users, and tracker entries. We also test bad loads in SQL statements (such as searching for a user that does not exist). Other notable tests are a test that verifies that categories display all values in a list, in the proper order, and a test that verifies a leaderboard is initialized. However, with testing databases, we run the risk of adding unnecessary data to the tables, eventually reducing the access time of our SQL queries. To counteract that, we have created delete functions that remove our test-case variables from our schema at the end of each test. We also test the user class to verify that a male user and female user output the correct body fat measurements and body mass index. Because most of our functions are infused with the graphics, it is hard to test anything other than database calls and functions.

Use Cases:

Use Case ID:	UC Access Dashboard
Scope:	Sign Up Form
Level:	User goal
Stake Holders and Interests:	Unregistered User- Person trying to register for Fitness Trakr. Registered User- The end result of the use case.
Description:	The user signs up to for a user account within the program Fitness Trakr.
Preconditions:	The user has opened the application.
Post conditions:	Registration occurs and is saved.
Main Success Scenario:	1. The user clicks the sign-up button. 2. The user enters their first/last name. 3. The user enters their email address. 4. The user enters their password. 5. The user enters their gender. 6. The user enters their height/weight. 7. The user enters their waist and neck measurement. 8. If all the fields above are valid, the form is submitted and the user becomes registered.

Extensions:	1.* The user already has an account, and logs in 3.* The email is taken, and the form prevents a submission until the field is changed to a valid input.

Use Case ID:	UC InputData/View Graphs
Scope:	Data Input and Collection
Level:	User goal
Stake Holders and Interests:	User- Person entering data for Fitness Trakr
Description:	The user logs into their account, adds their entries for the day, and then views their data compared to the past 30 days.
Preconditions:	The user has an account registered within the database.
Post conditions:	Data entry is saved, and the graph screen is showing.
Main Success Scenario:	1. The user enters his username and password. 2. The user is directed to the dashboard. 3. The user selects “Track my Calories” 4. The user inputs a food category, and selects from a drop down list. 5. The user exits to the dashboard. 6. The user selects “View graphs” to see his data for the past 30 days.
Extensions:	1.* The input is entered wrong, and the program re-allows for proper entry. 4.* The food is not in the database, and the user inputs a new food item with its calories, category name, and name

Use Case ID:	UC View BFA/BMI/FitnessScore
Scope:	Data Input and Collection
Level:	User goal
Stake Holders and Interests:	User- Person editing goals for Fitness Trakr. MaleUser- Uses a different set of data to calculate BFA/BMI. FemaleUser- Uses a different set of data to calculate BFA/BMI.
Description:	The user logs into their account, edits their profile to contain current information, and checks their BMI/BFA.
Preconditions:	The user has an account registered within the database.
Post conditions:	Data entry is saved, and the view BMI/BFA is saved.
Main Success Scenario:	1. The user enters his username and password. 2. The user is directed to the dashboard. 3. The user selects “Views Profile” 4. The user edits the information to ensure everything is current and up-to-date. 5. The user returns to the dashboard. 6. The user selects FitnessScore. 7. The user views BMI/BFA.

Extensions:	1.* The input is entered wrong, and the program re-allows for proper entry. 4.1 The information is up-to-date, in which case no editing is necessary. 4.2a If the user is Male, the information is only waist measurement and neck. 4.2b If the user is Female, a hip measurement is required.

Use Case ID:	UC Add Goal
Scope:	Data Input and Collection
Level:	User goal
Stake Holders and Interests:	User- Person entering data for Fitness Trakr
Description:	The user logs on, views their goals, and adds a new goal.
Preconditions:	The user has an account registered within the database.
Post conditions:	Data entry is saved, and the goal is stored in the database
Main Success Scenario:	The user enters his username and password. The user is directed to the dashboard. The user selects “View Goals” The user types a goal in the text field The user submits the goal The goal appears on the goal List
Extensions:	1.* The input is entered wrong, and the program re-allows for proper entry. 6.1 The user has more than 5 goals, in which case not all goals are shown. 6.2 The user deletes a goal from the drop down list of goals. 6.2.1. The user accidentally deletes a goal, and which case he uses the undo button.

System Operations

1. Login()
2. Register()
3. LoadUser(Username)
4. AddCaloriesToFoodTracker(CategoryFood)
5. InputFitnessData(Height,Weight)
6. SetGoals()
7. calculateFitnessScore()
8. ViewData()

Operation Contracts

Name	**Login()**
Responsibilities	Verifies that the user can access personal data across sessions, even after the application is closed.
Cross-references	All Use Cases
Output	None
Pre-condition	The user has an account with the system, and the username and password are entered correctly
Post-condition	The user’s data is loaded and can interact with the rest of the functions

Name	Register()
Responsibilities	Verifies that the user can create personal data maintained across sessions, even after the application is
Cross-references	All Use Cases
Output	None
Pre-condition	The user does not have an account with the system
Post-condition	The user has a permanent account associated with the system

Name	LoadUser(Username)
Responsibilities	Successes a database and gets all the data from the table, filling the user account with values as neces
Cross-references	All Use Cases
Output	None
Pre-condition	A login or registration has been accepted
Post-condition	Valid data has been initialized

Name	AddCaloriesToFoodTracker(Category,Food)
Responsibilities	Adds calories to a tracker, which will be used to formulate a fitness score.
Cross-references	UC Input Data/View Graphs
Output	None
Pre-condition	The loadUser operation has occurred
Post-condition	Data is initialized

Name	InputFitnessData(Height,Weight)
Responsibilities	Allows the user to input their measurements to calculate their progress over time.
Cross-references	UC View BFA,BMI, FitnessScore
Output	None
Pre-condition	The loadUser operation has occurred
Post-condition	Data is initialized

Name	calculateFitnessScore()
Responsibilities	Takes user information to calculate a FitnessScore, BMI, and BFA
Cross-references	UC View BFA,BMI, FitnessScore
Output	A FitnessScore page with BMI and BFA info
Pre-condition	The loadUser() operation has occurred
Post-condition	All 3 numbers are calculated

Name	SetGoals
Responsibilities	Shows the goals of the user to help track progress
Cross-references	UC Add Goal
Output	None
Pre-condition	The user has an account with the system, and the username and password are entered correctly
Post-condition	The user's goal is loaded and can be retrieved later

Name	ViewData()
Responsibilities	Shows the data in terms of graphs stemming from the calories tracker
Cross-references	UC Input Data/View Graphs
Output	A calorie bar graph
Pre-condition	A user has valid information in the calorie data form
Post-condition	A bar chart is created

Justification of Grasps

Controller: We used Controller to carryout user interactions with the home screen of the app. For example, if the user choses to open the add calories screen, then the dashboard controller will handle the user interaction (mouse click) and give a call to the add calorie controller which will handle all user input and add it to the database.

Creator: Throughout the project we had to use many temporary arrays to retrieve data from the database. We made these arrays using the creator grasp and the Abstract Factory design pattern. This really refined our code and made the creation of new Arrays easier to implement.

High Cohesion: Each screen that shows up in our app has a unique controller class which only deals with the methods and on-screen objects for that screen. This implementation uses high cohesion which makes each function specific to its rule and makes it not have more responsibility than it needs.

Indirection : To separate the database statements and the Java FX code we created a class to indirectly connect the two. This allows for cleaner code and reusability. The DatabaseGateway class inserts, selects, removes objects from the database that is inputted by the user. This means that the controller classes and the database code could be reused if needed by another project.

Information Expert: To ensure that each system action is delegated to a class we used Information Expert. This allowed us to create a class for each controller, which delegated each screens responsibility to a controller, so no class is doing two classes worth of work.

Low Coupling: We integrated Low coupling by ensuring that each User has no awareness to a Goal or Food. All the User has is a calorie count which is linked to food and goals using the database as a middleman.

Polymorphism: Polymorphism is used in the Female and Male users. The equations for calculating Body Mass Index is different for males and females. So, the Male User and FemaleUser is inherited from the User super class. We also use it to delegate different database functions base on functionality

Pure Fabrication: To ensure Low Coupling and High Cohesion we had to create a controller and a Database gateway. These two made it so each class is specific to its role and so classes will not know about each other if they don't need to. We also used the DatbaseGateway as fabricated classes to ensure access to the database.

Do Not Talk To Strangers: Users have no interaction with other users, and Goals and Calories have no interaction with each other. This ensures that no class is interacting with an unrelated class. The only classes that have a relation with each other are the ones that need to.

Design Patterns

Singleton: To interact with the Database in our app, an instance of the DatabaseGateway class must be created. Since DatabaseGateway has the potential to modify the data in the database, we are only allowing one instance of it at any time, using singleton. This lowers the risk of unnecessary interaction between the Database and the rest of the application. If a method requires an instance of the DatabaseGateway the class will return the same instance over and over so that there is never more than one instance.

Abstract Factory: When interacting with the database the creation of temporary ArrayLists was necessary. When selecting Food, Calories, or Goals, an ArrayList would have to be created to hold the large amounts of data. In order to support reusability and code arbitration we used a ListFactory to produce a List whenever a controller needs one.

Command: To promote high cohesion we made a controller class to handle each screen that is viewable by the user. Since each of these controllers have to execute a command to display, the command pattern was used. This allows us to use the controller.execute method on each controller and assume that it will successfully open the page.

Observer: When updating a user's information many different fields must be filled. If a field becomes changed, then the database must be notified in order for the data to be saved. This is implemented as an observer pattern. The observer pattern allows us to notify all observers anytime a field is changed, which in return will update the database.

State: We wanted to reward the user with a ribbon if their FitnessScore is good. A user can get a Gold ribbon for a great score, a Silver ribbon for a okay score, and a Bronze ribbon for a bad score. In order to recognize the level at which the user is currently in we used the State pattern. This allows the system to know how which ribbon the user deserves at all times.

Prototype : The user has the ability to add, remove, and undo goals. When undoing a goal, a copy of the goal must be created so that the goal can be added back into the database. This happens through the prototype pattern. The goal gets cloned using the cloneable interface and gets sent to the database using an instance of the DatabaseGateway.

Memento: When the user deletes a goal, the goal must be saved somewhere so that an undo will add the goal back to the goal list. These deleted goals are saved using the Memento pattern. A deleted goal will be saved into an arrayList created by the Factory, and then held until the user requests the goal to be undoed.

> An estimated 65 hours each were spent on the project(130 hours total). In total we had 2663 lines of logical code (not including fxml), which if we had to guess based on author, Garth had 1400 lines and Parakh had 1200 lines. Our git repository is https://terlizzit@bitbucket.org/terlizzit/fitnesstrackerproject.git. There was a total of 114 commits.

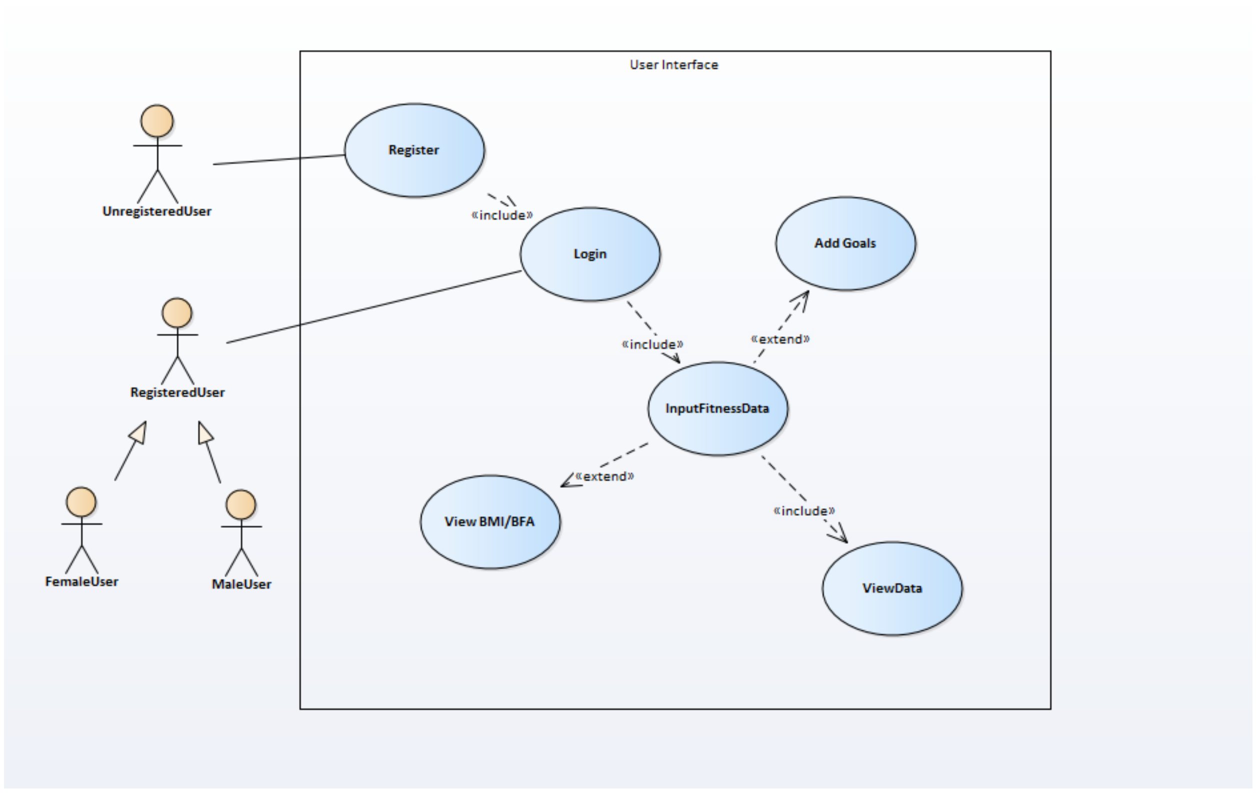
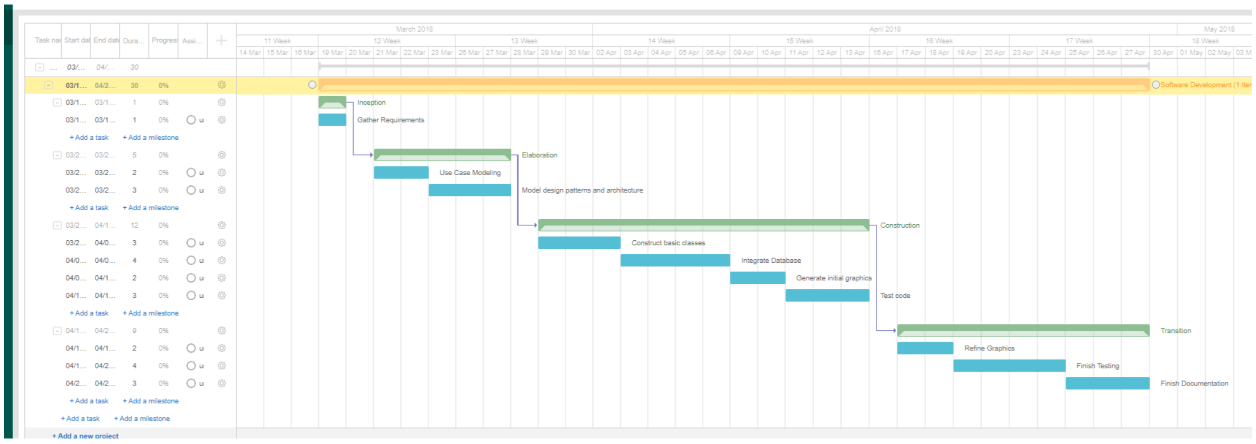
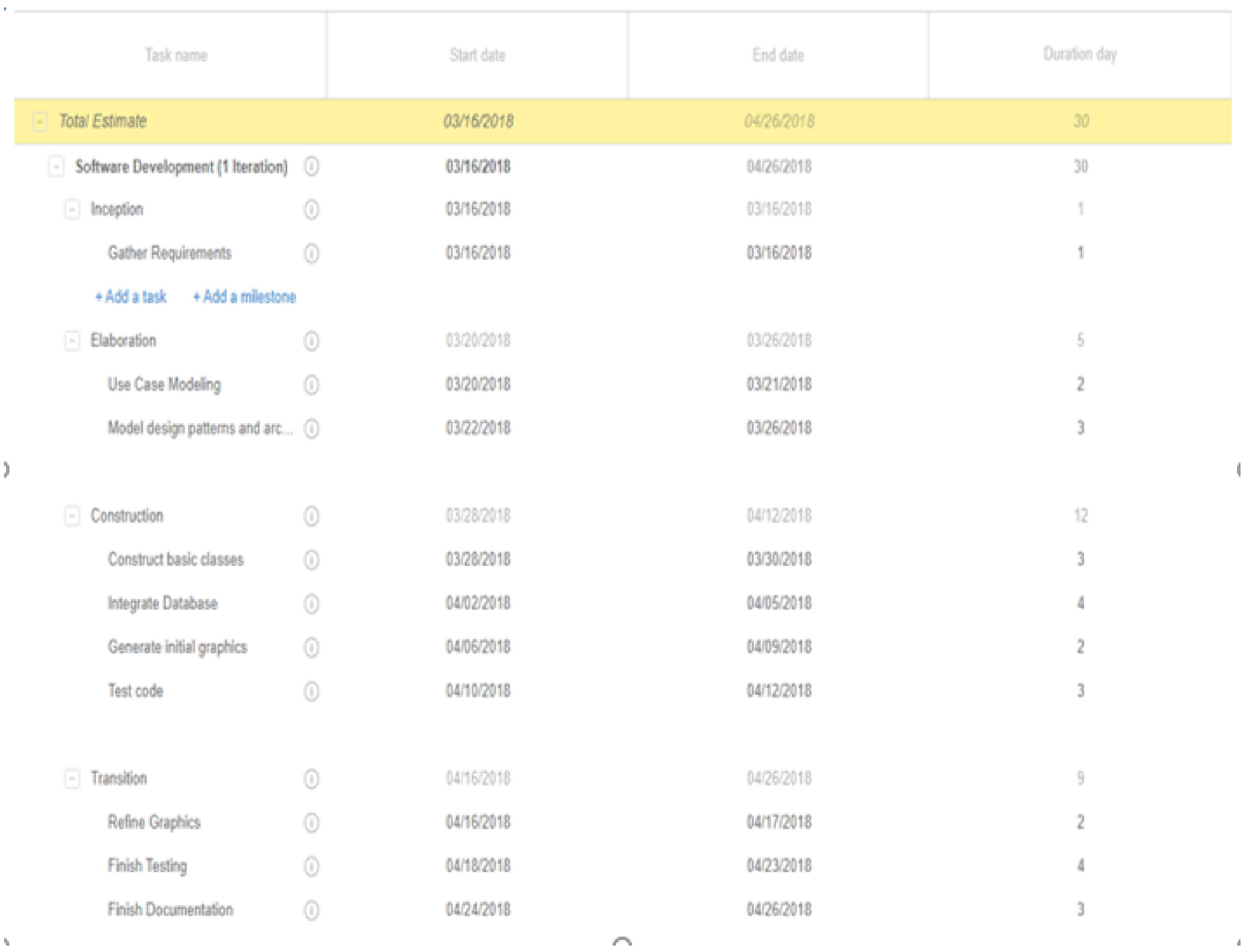


Figure 1: USE CASE MODEL



Task nar	Start dat	End date	Dura...	Progres	Assi...
...	03/...	04/...	30		
	03/1...	04/2...	30	0%	
	03/1...	03/1...	1	0%	
	03/1...	03/1...	1	0%	ui
+ Add a task	+ Add a milestone				
	03/2...	03/2...	5	0%	
	03/2...	03/2...	2	0%	ui
	03/2...	03/2...	3	0%	ui
+ Add a task	+ Add a milestone				
	03/2...	04/1...	12	0%	
	03/2...	04/0...	3	0%	ui
	04/0...	04/0...	4	0%	ui
	04/0...	04/1...	2	0%	ui
	04/1...	04/1...	3	0%	ui
+ Add a task	+ Add a milestone				
	04/1...	04/2...	9	0%	
	04/1...	04/1...	2	0%	ui
	04/1...	04/2...	4	0%	ui
	04/2...	04/2...	3	0%	ui
+ Add a task	+ Add a milestone				
+ Add a task	+ Add a milestone				
+ Add a new project					

Figure 2: GANNT DIAGRAM



Task name	Start date	End date	Duration day
Total Estimate	*03/16/2018*	*04/26/2018*	*30*
Software Development (1 Iteration)	03/16/2018	04/26/2018	30
Inception	03/16/2018	03/16/2018	1
Gather Requirements	03/16/2018	03/16/2018	1
+ Add a task + Add a milestone			
Elaboration	03/20/2018	03/26/2018	5
Use Case Modeling	03/20/2018	03/21/2018	2
Model design patterns and arc...	03/22/2018	03/26/2018	3
Construction	03/28/2018	04/12/2018	12
Construct basic classes	03/28/2018	03/30/2018	3
Integrate Database	04/02/2018	04/05/2018	4
Generate initial graphics	04/06/2018	04/09/2018	2
Test code	04/10/2018	04/12/2018	3
Transition	04/16/2018	04/26/2018	9
Refine Graphics	04/16/2018	04/17/2018	2
Finish Testing	04/18/2018	04/23/2018	4
Finish Documentation	04/24/2018	04/26/2018	3

Figure 3: GANNT CHART

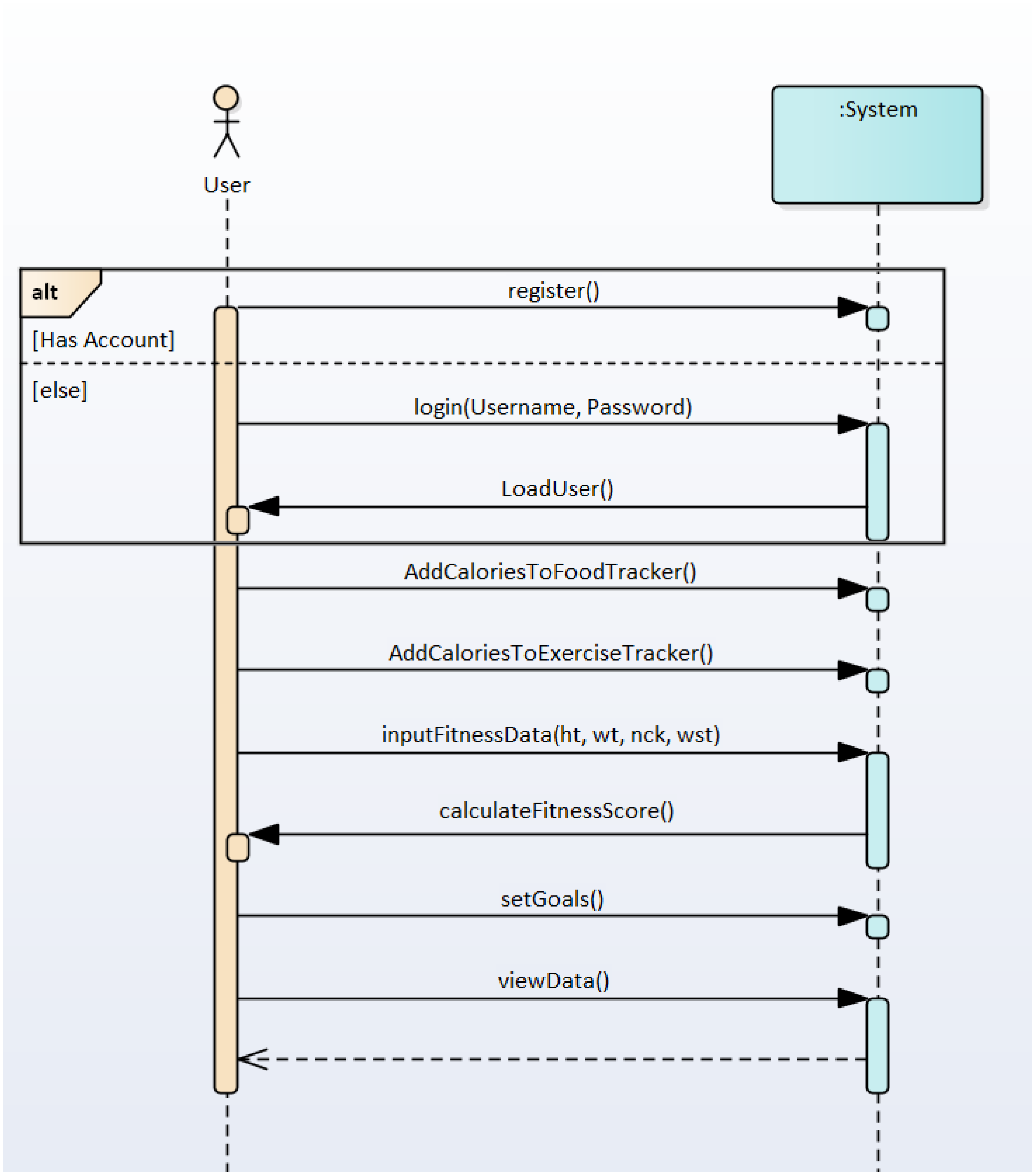

User
:System
alt
[Has Account]
register()
[else]
login(Username, Password)
LoadUser()
AddCaloriesToFoodTracker()
AddCaloriesToExerciseTracker()
inputFitnessData(ht, wt, nck, wst)
calculateFitnessScore()
setGoals()
viewData()

Figure 4: SYSTEM SEQUENCE DIAGRAM

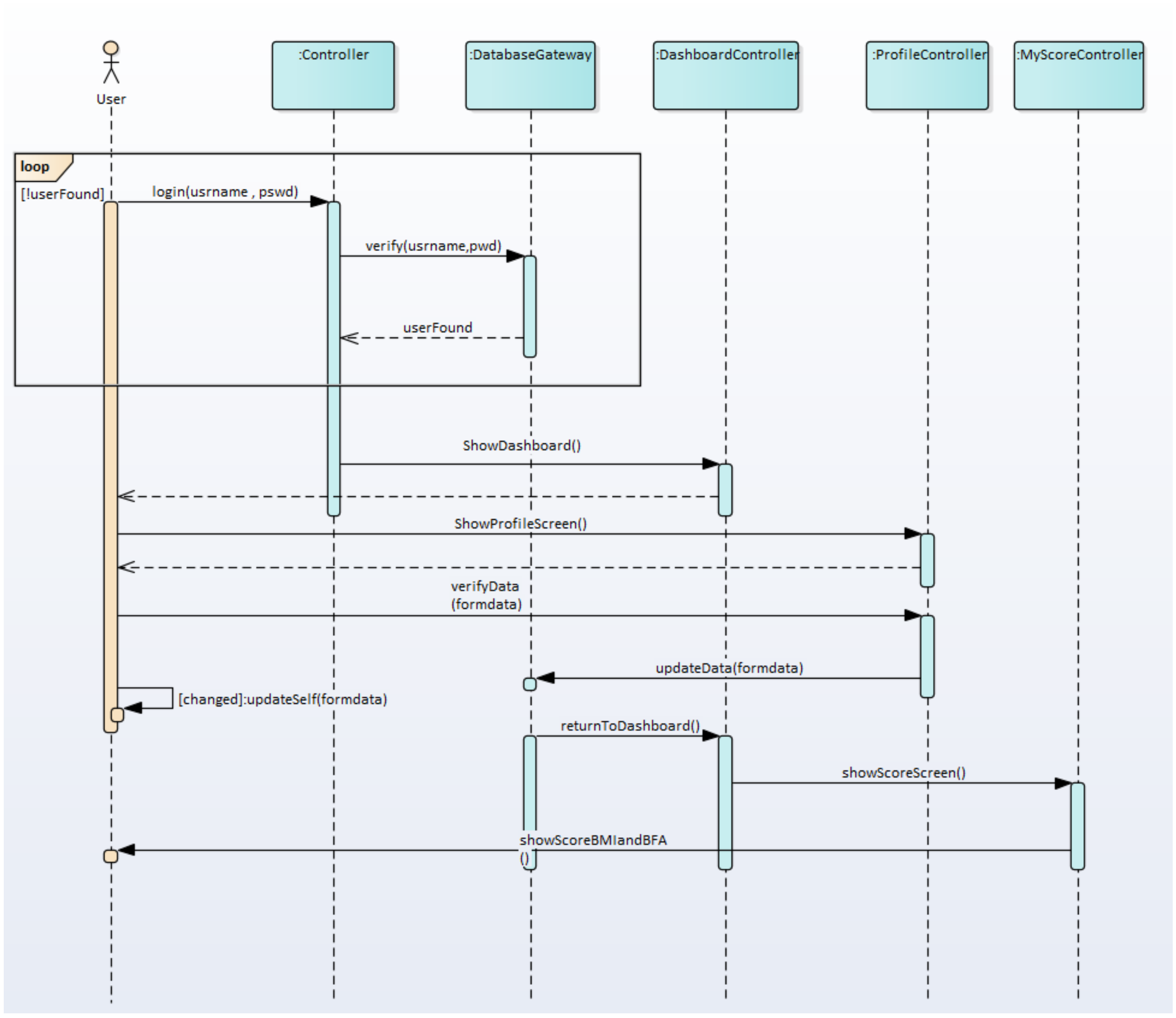


Figure 5: Sequence Diagram (UC View BFA,BMI, FitnessScore)

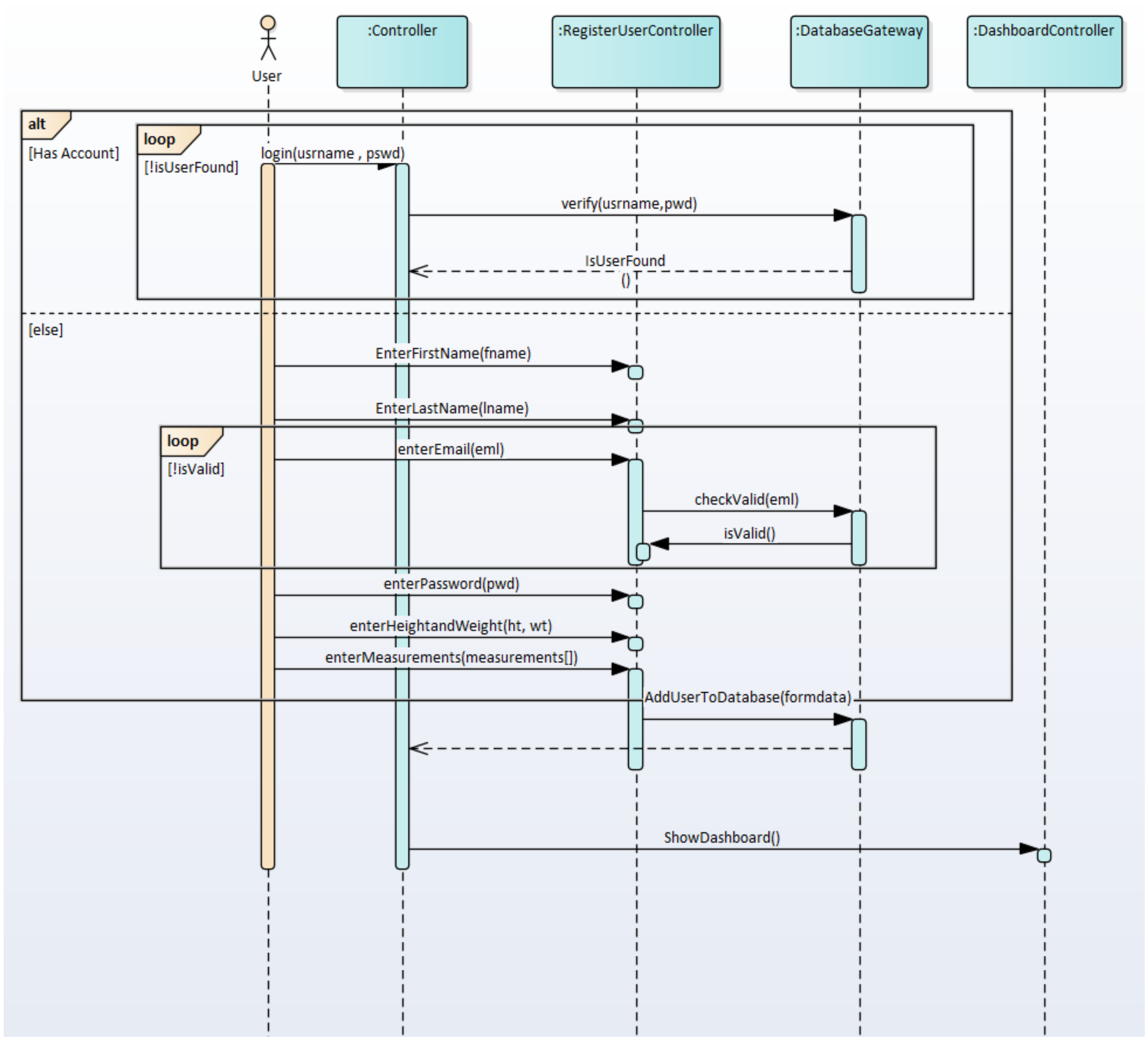


Figure 6: Sequence Diagram (UC View Dashboard)

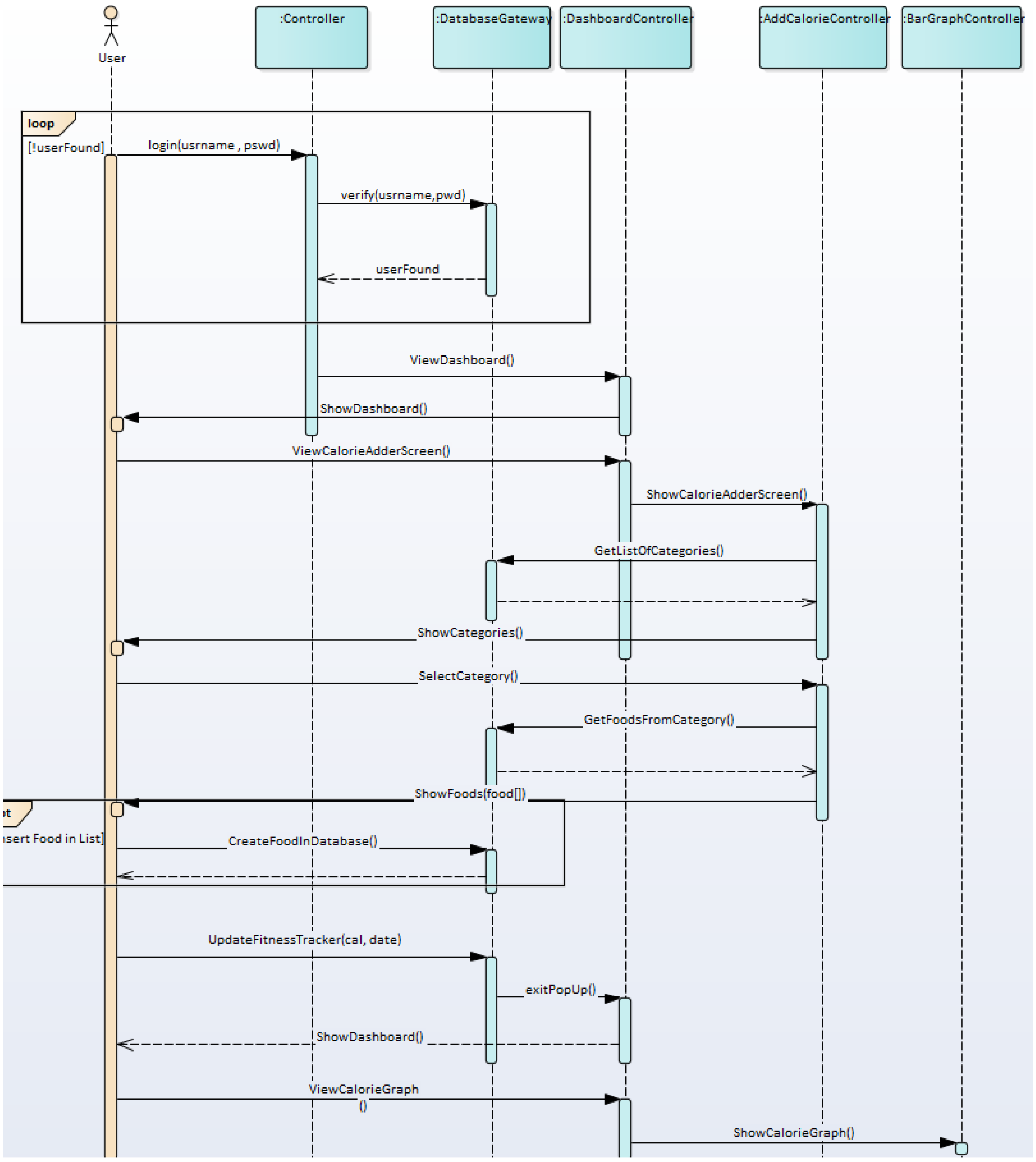


Figure 7: Sequence Diagram (UC Input Data/View Graphs)

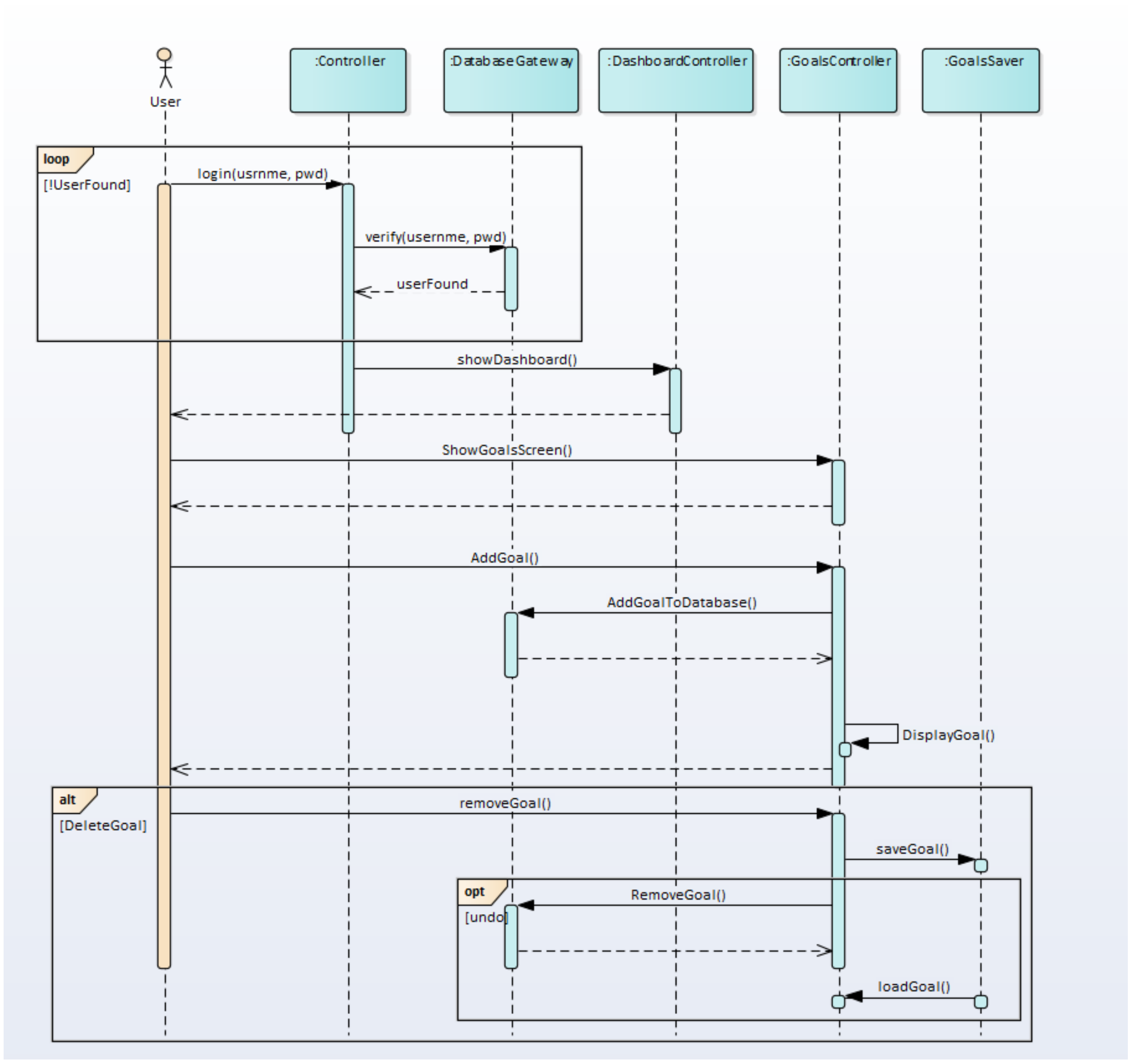


Figure 8: Sequence Diagram (Add Goal)

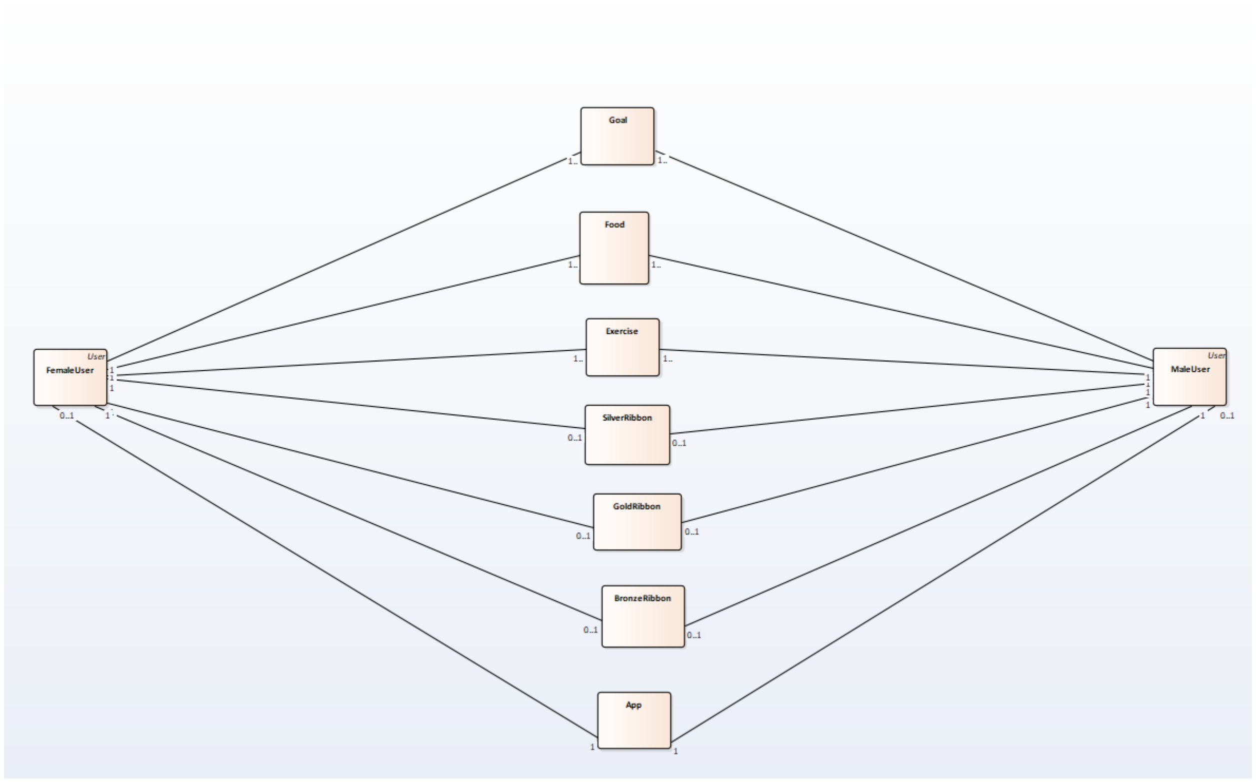


Figure 9: Domain Model

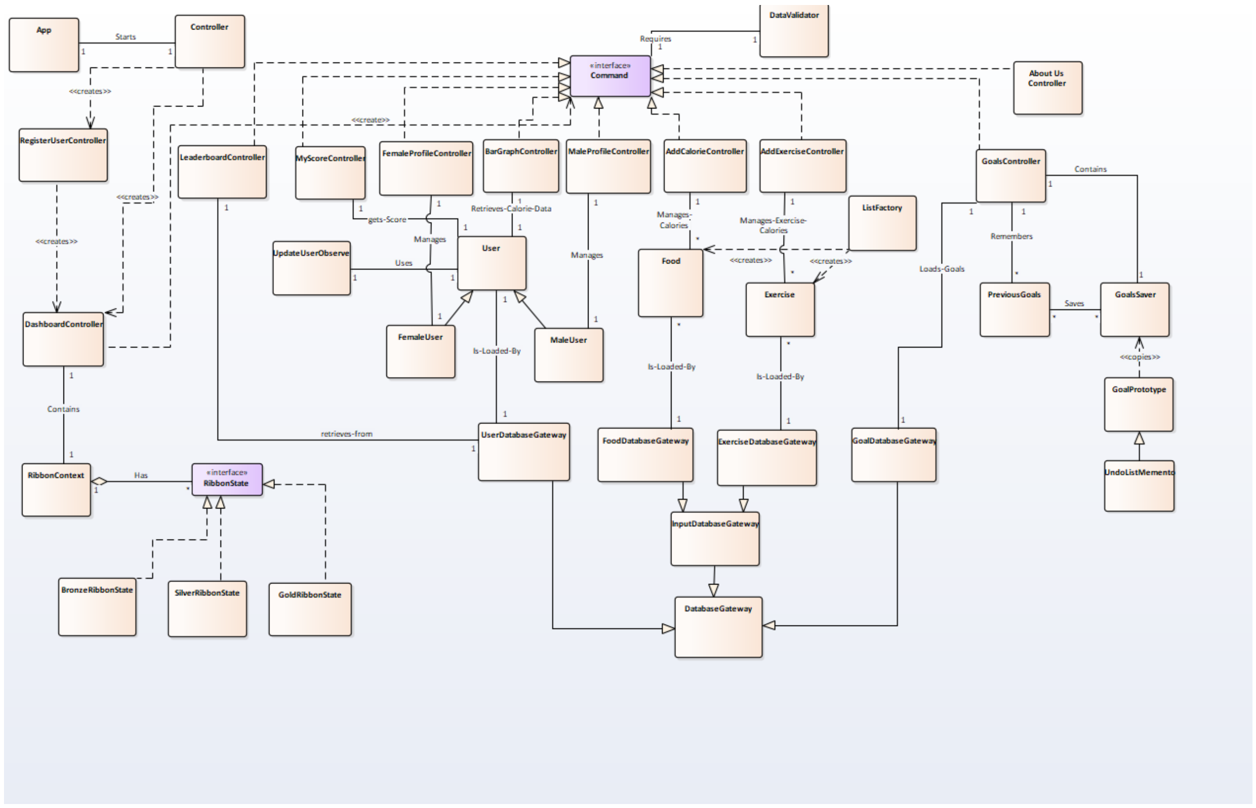


Figure 10: Design Model (Basic)

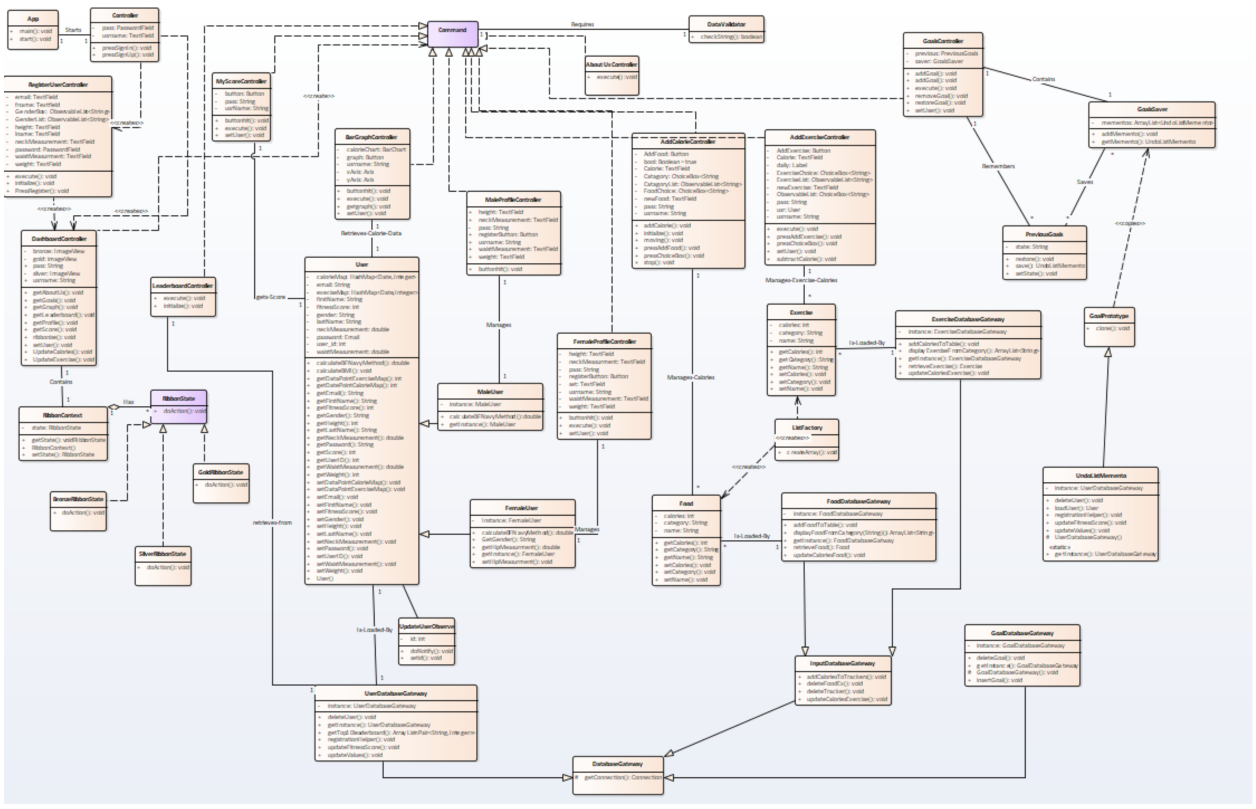


Figure 11: Design Model (FullyDressed)