Aegle

E+1

Sebastian Zawadzki - 201632011
sebastianzawadzki@gmx.de

Kay Gillmann - 201632012
kaygillmann@aol.de

Kevin Oh (오주환) – 201520936
kevinoh94@gmail.com

Lucy Liu – 201632020
lucy.liu@myy.haaga-helia.fi

Tam Tran – 201632027
h8265@student.jamk.fi

Final Report (Elaboration 2)
7 June 2016

Domain Analysis & Design

1 Table of Contents

No.	Date	Changed chapters	Description of change	Authors
1	2016-03-24	all	Inception Phase Draft 1	Team E+1
2	2016-05-02	2 and 3	2 became 3	Team E+1
3	2016-05-02	3.1.1 Use Case Diagram	Deletion of Use Case 1 and 7, changed name of new Use Case 1, 2 and 4	Team E+1
4	2016-05-02	3.1.2 Use Case List	Content of Use Case 3	Team E+1
5	2016-05-02	3.1.3 Use Case Text	Content of Use Case 3, added Use Case 1, 2, 4 and 5	Team E+1
6	2016-05-08	8 Glossary	Updated/added some de-scriptions	Team E+1
7	2016-06-03	4.1 Domain Model Diagram	Changes in diagram and class description	Team E+1

8	2016-06-03	5.1 UC1 – Do exercise program	Changes in all diagrams, addition of explanation	Team E+1
9	2016-06-03	5.2 UC1 – Do diet program	Changes in all diagrams, addition of explanation	Team E+1
10	2016-06-03	5.3 UC1 – Interact with social media	Changes in all diagrams, addition of explanation	Team E+1
11	2016-06-03	5.4 UC1 – Interact with online content	Changes in all diagrams, addition of explanation	Team E+1
12	2016-06-06	Glossary	Changed and added definitions	Team E+1

No.	Date	New chapters	Authors
1	2016-03-24	1 Table of Content, 2 Vision and 3 Requirements	Team E+1
2	2016-05-02	4 Domain Model	Team E+1
3	2016-05-02	5 System Sequence Diagram	Team E+1
4	2016-05-06	6 Design Model	Team E+1
5	2016-06-03	5.5 UC5 – Manage Users	Team E+1
6	2016-06-03	6.1.12 Combined Design Class Diagram for Use Case 1	Team E+1
7	2016-06-03	6.1.13 Combined Design Sequence Diagram for Use Case 1 Main 1	Team E+1
8	2016-06-03	6.1.14 Combined Design Sequence Diagram for Use Case 1 Main 2	Team E+1
9	2016-06-05	6.2.3 Combined Design Class Diagram for Use Case 2	Team E+1
10	2016-06-05	6.2.4 Combined Design Sequence Diagram for Use Case 2	Team E+1
11	2016-06-05	6.3.4 Combined Design Class Diagram for Use Case 3	Team E+1
12	2016-06-05	6.3.5 Combined Design Sequence Diagram for Use Case 3	Team E+1
13	2016-06-05	6.4.5 Combined Design Class Diagram for Use Case 4	Team E+1
14	2016-06-05	6.4.6 Combined Design Sequence Diagram for Use Case 4	Team E+1
15	2016-06-05	6.5.4 Combined Design Class Diagram for Use Case 5	Team E+1
16	2016-06-05	6.5.5 Combined Design Sequence Diagram for Use Case 5	Team E+1
17	2016-06-05	6.6 Design Class Diagram of the System	Team E+1
18	2016-06-05	7 Architecture	Team E+1

AJOU UNIVERSITY

19	2016-06-05	8 Conclusion	Team E-1
20	2016-06-06	9 Appendix	Team E-1

2 Vision

2.1 Introduction

We envision a next generation, social, all-in-one smart fitness companion application, Aegle, designed to have the flexibility to be useful for anyone interested in fitness, containing multiple user interface mechanisms supported by a variety of devices intuitively. The idea behind Aegle arose from the inconvenience and general incompleteness of existing fitness related applications. This system will provide a convenient way for users to start and maintain a regular exercise program, along with a proper diet.

Aegle is also integrated with its own social media platform. Users will be able to share their personal achievements and milestones, along with the ability to fit in with others in your own niche of the health community. The greatest advantage of the social media platform is the ability for users to share their workout programs and recipes. Other users can try them out and leave reviews. The most popular workouts and recipes will then be showcased and recommended to users. Doing this will ensure that Aegle will always have the best information available to our users.

2.2 Positioning

2.2.1 Business Opportunity

A recently published study found that only 2.7 % of U.S. adults live a healthy lifestyle. A significant part of living a healthy lifestyle comes from fitness and diet. Almost anyone could benefit from eating a better diet and being more active. Yet the number of healthy people is so low because of the barriers to getting fit, such as the time, money, and know-how required. Existing fitness related applications are numerous but are lacking in important features. Current applications fail to be essential. They are not satisfying to use. Aegle will be different because of how its community driven. Since users will be able to get information and advice directly from experts it will be more exciting and useful for users to use.

2.2.2 Problem Statement

Fitness is a science and can be intimidating for new users because of the overwhelming amounts of information and conflicting regimes. Since everyone is different there is no correct way to get fit. Our system is designed to ease the burden of information by providing users with accurate information verified by other users.

2.2.3 Product Position Statement

Aegle is designed to be the ultimate fitness application, specifically geared towards beginners, but useful for anyone. Aegle combines the exercise and diet aspects of fitness and provides users with relevant information and research so that anyone can just dive in and start becoming more fit.

The most unique part of Aegle comes from its community. A specially designed social media seamlessly integrated with the system will allow users to share their workout programs and favorite recipes.

Aegle will give users the ability to create highly detailed meal plans with nutrition and calorie content of individual meals to help them map out the road to a healthier lifestyle.

The goal behind Aegle is to give a complete beginner the ability to step right in and begin their journey to a healthier lifestyle the moment they download the application.

2.2.4 Alternatives and Competition

There are numerous fitness and diet related applications already available on the market. What differentiates Aegle from similar apps is Aegle's more intuitive and useful user interface, and the sharing system for workout programs and diet ideas. Aegle offers a much more detailed dietary plan, letting users map out each individual meal to reach their calorie goals. The biggest selling point of Aegle is the fact that it brings everything together so that anyone can begin to live a healthier lifestyle with just one application and minimum research.

2.3 Stakeholder Description

2.3.1 Stakeholder Summary

System Administrator - Designed the software and pushes out updates in order to gain a userbase. The application will be free to download but offer greater features in the form of in-app purchases in order to make a profit.

Social Media Administrator - Moderates social media and the sharing system. The driving force behind the success of the application is its community. Creating and establishing a large community will ensure that good, relevant information will always be available to users. Thus further growing the user base, and profit.

2.3.2 User Summary

General User - Can share content that worked for them, leave reviews for others' content. Can see and download popular content from the social media. Can join and participate in communities.

Certified User - Certified health experts are verified and given special privileges. Their posts and reviews hold greater weight in the community.

2.3.3 Key High-level Goals and Problems of the Stakeholders

Table 2-1 Key High-level Goals

High-Level Goal	Priority	Problems and Concerns	Current Solutions
Fast and easy to get started exercise routines	high	- Exercise routines need to meet certain standards - Everyone is different so a routine that works for one person may not be right for someone else	Leave it to the user to do the research and figure out a routine that will work for them
Healthy and nutritious dieting	high	- Recipes must be cheap and easy to make - Recipes need to be delicious - Recipes need to be flexible for varying calorie targets	There are plenty of services available for figuring out proper calorie and nutrition balance but not as easy to design a meal plan to match
Sharing of accurate and relevant information in the health world	medium	- Fitness is not a completely established science -There is a lot of misinformation and conflicting information	User can join existing health communities to ask questions and gain knowledge

2.3.4 User Goals

General User - Wants to start and maintain an effective workout program along with a healthy diet. Don't want to be burdened by complicated information and overly expensive or complicated foods.

Certified User - Having trained health professionals recognized improves the quality of the content available and allows the experts to help others while promoting themselves and therefore profiting from using the application.

2.3.5 User Environment

- General user can log in, connect to the integrated social media with an existing social media account (such as google or Facebook)
- Users can see and download exercise routines from the integrated social media
- Users can find their recommended calorie intake based on their goal, height, weight, age, sex, and activity-level
- Users can see and download recipes to design a meal plan to match their recommended calorie intake
- Users can see and get advice from Certified Users who are verified health experts

- Users can share personal milestones and information through the integrated social media
- Social media administrators moderate the content that is uploaded and interactions between the community
- System administrators ensure safety of the user's personal information and manages verification of Certified Users

2.4 Product Overview

2.4.1 Product Perspective

Other existing apps only offer the part of an exercising or a diet program. With Aegle the user is able to generate his own exercising program, combined with a balanced food plan to get the most efficacy. Also the user able to share his achievements at social media platforms or compare it with other users from the app, to check his goals. Aegle is an all-in-one app to structure the daily routine as much as possible.

2.4.2 Summary of Benefits

Table 2-2 Summary of Benefits

Features	Benefits
Display exercises for the user	Helps user begin and maintain a regular exercise routine
Display food plan	Helps user begin and maintain a healthy diet
Get information about the substance in the food	Makes it easier to create a meal plan with the proper calorie and nutritious content
Share workout goals to the social media	Give users motivation and be a part of a community

2.4.3 Assumptions and Dependencies

- The social media has a large enough user base that people are actively participating in sharing and reviewing content
- Aegle needs a sufficiently sized database of foods that users can pick from to get nutrition and calorie information in order to create meal plans
- Certified health experts join the community along with generally fitness experienced people

2.4.4 Other Requirements and Constraints

- The design of an interface (GUI) that shows enough information for the user but still clearly arranged
- The idea behind Aegle is the promise of a great wealth of useful information, but we are relying on having a user base to provide that information. So we will need to provide a large amount of information to begin with in order to drive the creation of a user base
- We are relying on establishing a user base large enough that health professionals would want to join and participate in order to promote themselves to a large community. So we need users to draw in professionals in order to draw in users

3 Requirements

3.1 Functional Requirements

3.1.1 Use Case Diagram

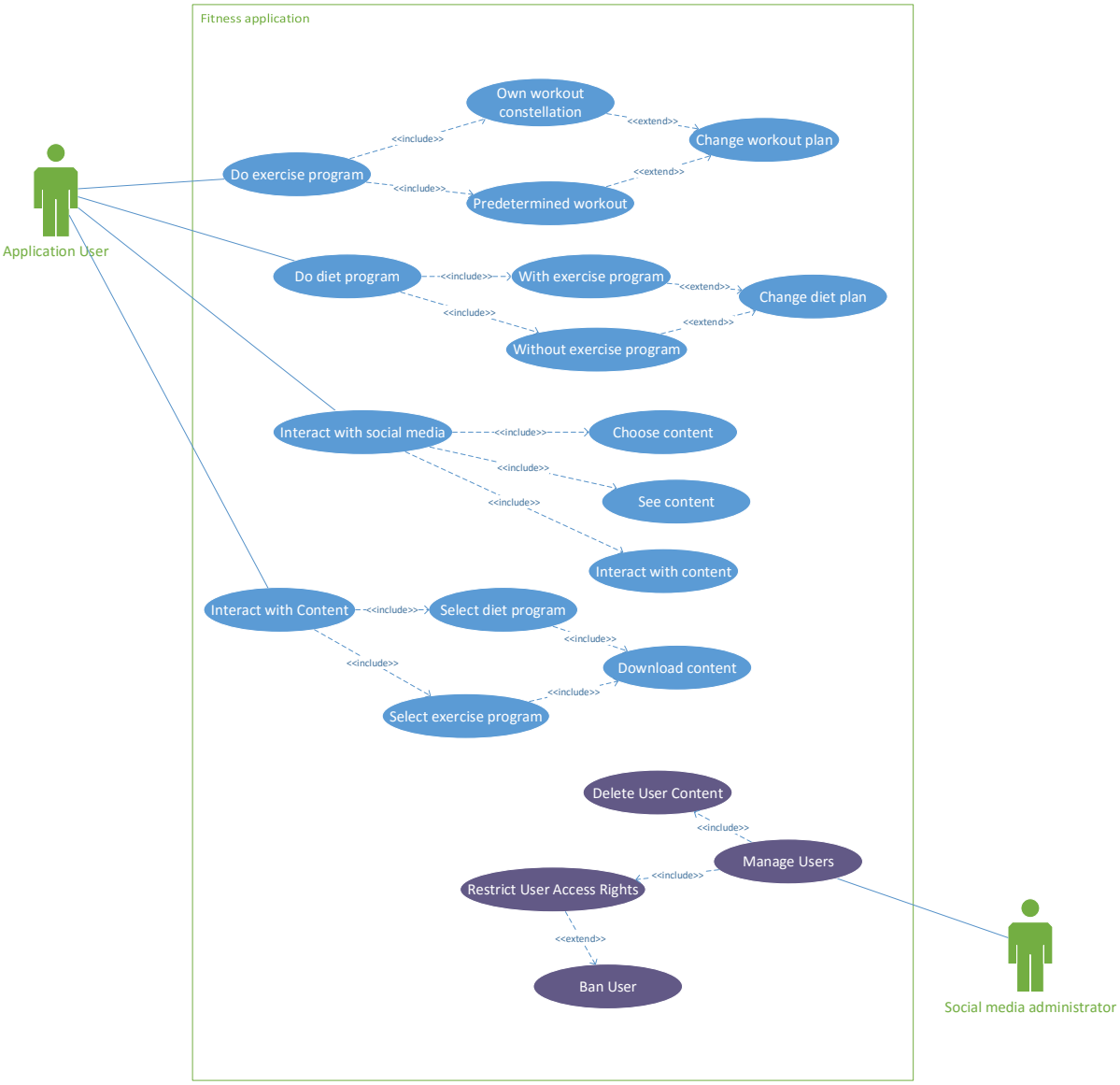


Figure 3-1 Use case diagram

3.1.2 **Use Case List**

UC1. Do exercise program (User)

- User chooses a kind of exercise program and the intensity of exercise
- User can choose to allow the system to use his information to suggest alternative exercise programs
- User starts exercise
- System measures health status during the exercise
- User finishes exercise and gives info like time spent
- System provides feedback on health and fitness status and saves it (if allowed)

UC2. Do diet program (User)

- User inputs body information for age, weight, height, sex, activity level, goal weight, and time frame. (optional)
- System saves the information about the user (optional)
- System provides a daily calorie amount in order to reach the goal weight in the time frame
- User creates or selects an appropriate diet program
- User starts the diet program
- System creates a daily planner for the user, as specific as the user wants. Planning individual meals is recommended.

UC3. Interact with the social media (User)

- User select the ”Feed/Media” button
- User chooses from the list what content one would like to see
- User see the feed from chosen category
- User interact with the content from the feed
- User select the diet, exercise or both to share
- User push ”Share” to finish

UC4. Interact with content (User)

- User accesses database with all content. Separate sections for exercise programs and diet programs.
- User chooses type of content what he wants to see

- System will highlight popular content / up voted shared content
- User chooses specific file to see
- User sees the content of the file
- User can download chosen content
- User exits the database

UC5. Manage users (Social media administrator)

- Social media administrator access social media
- Social media administrator gets report about inappropriate user
- Social media administrator checks the reported user
- Social media administrator manages the user

3.1.3 Use Case Text

Use Case Text – 1

Use case name: Do exercise program (User)

Level: User-goal

Primary Actor: User, Database

Stakeholders and Interests:

- User: wants to choose an exercise program to achieve his attempted goal without any problem.
- Social media application: wants to share or offer exercises for all users who use the application.
- Data base: wants to store/offer all the information from the social media application and the user without missing data and any problem.

Preconditions:

- The user has to create an account for the application
- The user has to choose if he wants to do his exercise program with or without a diet
- If the user chose to do his exercise with a diet, a diet program must be configured
- The user has to download an exercise program before he starts a predetermined workout

Postconditions:

- Confirmation from user that the workout is finished.
- System has stored the information between the user and the Social Media platform frequently and at any time.

Main Success Scenario 1 (own workout constellation):

1 User chooses that he wants to do an exercise program
2 User chooses that he wants to create his own workout
3 System gives user information about the selectable exercises
4 User selects all exercise he wants to do in the workout
5 User chooses intensity and duration of workout
6 System asks the user if he wants to save his workout
7 User starts exercise
8 System measures health status during the exercise
9 User confirms that he finished a set of his exercise
10 User confirms that he finished his complete workout

Main Success Scenario 2 (predetermined workout):

1 User chooses that he wants to do an exercise program
2 User chooses that he wants to do a predetermined workout
3 User chooses a workout that he downloads before
4 User chooses intensity and duration of workout
5 System gives user information about his chosen workout
6 User starts exercise
7 System measures health status during the exercise
8 User confirm that he finished a set of his exercise
9 User confirm that he finished his complete workout

Extensions (Alternative Flows):

*a. In case of system failure in any time
    1 User reboots the system and returns to its former state.
    2 The system is restored to its original state.

*b. In case of wanting cancellation of using this system
    1 User terminates the service on the app.

2 System deletes all data related to the user.

3 Pop up a message to notify passenger that the system end

*c. In case that the app has no connection to the database

1 The user checks the internet connection of his device

1a. If the connection of the device is bad

1 The user reconnects to his device

2 The user reboots the application

1b. If the database cannot be reached

1 The user reboots the application

Main 1 (own workout constellation)

4 If the system doesn't offer the exercise that the user wants

1 User chooses similar exercise to his request

6 If the system doesn't save the information

a. System gives the user an error message

b. User checks the connection to his Wi-Fi

7 In case the user wants to change his workout.

1 The user breaks off his workout

2 The app asks the user if he really wants to break off the current workout

3 The database erases the information that was stored for the current workout

4 The user chooses a new workout

9 In case the system doesn't save the confirmation

a. System gives the user an error message

b. The user tries to save his data again

i. In case that it is not able to confirm a finished set

1. The user skips the confirmation

2. The user starts with the next exercise

10 In case the system doesn't save the confirmation

c. System gives the user an error message

d. The user tries to confirm again

i. In case that it is not able to confirm a finished set
    1. The user skips the confirmation
    2. The user starts with the next exercise

Main 2 (predetermined workout)

3a. In case that the system cannot load information for a predetermined workout.
    1 User restarts the app
    2 User chooses his workout again

3b. In case that the system cannot load information for a predetermined workout.
    1 User downloads the workout again
    2 User starts the workout

5. In case that the system doesn't provide the information about the workout
    1 System gives the user an error message
    2 User downloads the chosen workout again

6. In case the user wants to change his workout.
    1 User breaks off his workout
    2 System asks the user if he really wants to break off the current workout
    3 Database erases the information that was stored for the current workout
    4 User chooses a new workout

8. In case the system doesn't save the confirmation
    a. The user confirms his finished exercise again
        i. In case that it is not able to confirm a finished set
            1. The user skips the confirmation
            2. The user starts with the next exercise

9. In case the system doesn't save the confirmation
    a. The user confirms his finished workout again
        i. In case that it is not able to confirm a finished set
            1. The user skips the confirmation
            2. The user starts with the next exercise

Special Requirements:

- Aegle hinges on there being an active user base to drive content
- In the early stages of Aegle, we will provide some workouts ideas for a balanced exercise set
- User should be able to easily adjust own workouts in order to create own exercises

Technology and Data variations list:

The System is optimized for Tizen environment.

Frequency of Occurrence:

Always.

Open issues:

None.

Use Case Text - 2

Use case name: Select a diet program (User)

Level: User goal

Primary Actor: User, social media application, Database

Stakeholders and Interests:

- User: wants to get a balanced program with exercise and/or diet program
- Social media application: wants to share or offer exercise and/or diet plans.
- Data base: wants to store/offer all the information from the social media application and the user

Preconditions:

- The user has to create an account for the app
- The user has to choose if he wants to do his diet with or without an exercise program.

Postconditions:

- The database has to store the information frequently.
- The meal plans display the total nutritional information for the day.
- The system keeps a record of all list of meals and user's stats.

Main Success Scenario:

1 User starts a diet.
2 User inputs his information and receives a daily calorie target.
3 User starts a meal list for the day.
4 User can select individual meals to add food items to the list from a database.
5 User can share their meal plan to the social media to help others in a similar situation.

Extensions (Alternative Flows):

*a. In case the app doesn't work correctly

1 The user stops the app and reboot the system.
2 The user starts again and return to his current plan.

1. In case the user wants to change his diet plan

1. The user deletes the current plan.
2. The database erases the information that is stored.
3. The user creates a new plan.

4a. In case the food the user wants to load isn't in the database.

1. The user adds his own recipe
2. The user loads his own recipe

4b. In case the recipe that user wants to load isn't in the database.

1. The user downloads the plan from the social media platform.
2. The user loads the downloaded recipe.

4c. In case the user wants to use a complete meal plan from the database.

1. Rather than adding individual food items the user can use a complete meal plan from the database.
2. The user can find and download whole meal plans from the social media platform.

5.. In case the user wants to leave reviews.

1. The user writes a review of the meal plan.
2. The review is saved in online content.

Special Requirements:

- Aegle hinges on there being an active user base to drive content.
- In the early stages of Aegle, we will provide some meal plans and recipe ideas for various calorie targets.
- User should be able to easily adjust meal plans in order to create variation.

Frequency of Occurrence:

- Very frequent. The diet program is the biggest part of Aegle. Users are expected frequently check their meal plans, make changes, and to try new meal plans recommended by others.

Open Issues:

None

Use Case Text – 3

Use case name: Interact with the social media (User)

Level: User goal

Primary Actor: User, social media application, Database

Stakeholders and Interests:

- User: wants to browse and interact with ”Feed/Media”
- Social media application: wants to share and/or offer content
- Data base: wants to store/offer all the information from the social media application and the user

Preconditions:

- The user has an account or create an account for the app
- - The user has to choose from the list what content one would like to see

Postconditions:

- The database had stored the information about user's interactions

Main Success Scenario:

1 User chooses "Feed/Media" button
2 User chooses content
3 User sees content
4 User interacts with content

Extensions (Alternative Flows):

*a. In case the app doesn't work correctly
    1 The user stops the app and reboots the system
    2 The user starts the app again and tries to see the content

*b. In case the user can't access the database
    1 The user stops the app and reboot the system
    2 The user checks his connection
    3 The user restarts his router
    4 The user reconnects to the internet
    5 The user starts the app again and tries to see the content

1 In case the system doesn't react
    1 The user stops the app and reboots the system
    2 The user starts again and return to his current plan

2 In case the content isn't available
    1 The user refreshes the "Feed/Media"
    2 The user tries to load the content again

3 In case the user doesn't see the content
    1 The user reboots app
    2 he user tries to load the content again

4 User cannot interact with content
    1 The user refreshes the content

2 The user tries again

Special Requirements:

- Touch screen UI on the mobile device. Text size and colors can be adjustable for user preference (e.g. bad sight, color blindness, etc.…)
- Language internationalization on the text displayed

Frequency of Occurrence:

- Up to user. User can decide when he wants to browser the ”Feed/Media”

Open issues:

Is the uploaded content appropriate?

Use Case Text – 4

Use case name: Interact With Content (User)

Level: User goal

Primary Actor: User, Database

Stakeholders and Interests:

- User: wants to compare and get a matching exercise program and diet program according to his personal goals
- Data base: wants to store all the uploaded information by the user and recommend the most popular programs to the user

Preconditions:

- The user has created an account for the app.
- The user has to be connected with the internet.
- The user chose the content he wants to see.
- The user has to have free memory space on his device for the downloaded content.

Postconditions:

- None

Main Success Scenario:

1. User accesses the database.
2. User selects between exercise and diet program.
3. Database highlights popular content.
4. User chooses a specific file to see.
5. User downloads the chosen content.
6. User sees the content of the file.
7. User decides to search for another file. Repeat from step 2 until user decides not to see anymore files.
8. User exits the database.

Extensions (Alternative Flows):

*a. In case the app doesn't work correctly

1. The user stops the app and reboot the system
2. The user starts again and return to his current plan

*b. In case the user doesn't have connection to the database

1. The user stops the app and reboots the system.
2. The user checks his connection.
3. The user restarts his router.
4. The user reconnects to the internet.
5. The user starts the app again and try to see the content.

1a. In case the user doesn't have connection to the database

1. The user checks the internet connection of his device.

1b. If the connection of the device is bad.

1. The user stops the app and reboot the system.
2. The user reconnects to the internet.
3. The user starts the app again and tries to see the content.

1c. If the connection didn't get better.

1. The user stops the app and reboot the system.
2. The user restarts his router.
3. The user reconnects to the internet.

4 The user starts the app again and tries to see the content.

5a In case the user can't download the file.

1 The user checks the free memory space on his device.

5b If the memory on the device is full.

1 The user deletes data to get enough free memory space for the file.

6a. In case the user can't see the content of the file.

1 The user checks if he has an appropriate app to open the type of the file.

6b. If the user doesn't have an appropriate app to open the type of the file.

1 The user installs an appropriate app.

6c. If the user still can't see the content of the file.

1 The user checks the size of the file.
2 The user reports the issue.
3 The system/admin checks the reported file.

Special Requirements:

- User should be able to easily navigate through the folders in the database.

Frequency of Occurrence:

- Frequent. The database enables the user to compare their programs. Users are expected to frequently compare their exercise and diet program and update it often in order to have the newest and best available information when starting an exercise or diet program.

Open Issues:

- None

Use Case Text – 5

Use Case Name: Manage Users

Level: Administrator goal

Primary Actor: Social Media Administrator

Stakeholders and Interests:

- Social Media Administrator: Wants the content in social media to be politically correct and ensures that there's no cybercrime happening
- User: Wants to have safe experience in using application's social media.

Preconditions:

- There are users
- Administrator has access to social media content

Postconditions:

- None

Main Success Scenario:

1 Social media administrator access social media
2 Social media administrator gets report about inappropriate user
3 Social media administrator checks the reported user
4 Social media administrator manages the user

Extensions (Alternative Flows):

*a. In case the app doesn't work correctly
    1 The social media administrator stops the app and reboot the system
    2 The social media administrator starts again and return to his current plan

*b. In case the user doesn't have connection to the database
    1 The social media administrator stops the app and reboots the system.
    2 The social media administrator checks his connection.
    3 The social media administrator restarts his router.
    4 The social media administrator reconnects to the internet.

5 The social media administrator starts the app again and continues his current plan.

1. Social media administrator notices inappropriate user

1 Social media administrator keep track on the user's behavior
2 Social media administrator sends warning message to user

3a. Social media administrator moderates user

1 Social media administrator restricts user's rights

3b. Social media administrator moderates user

1 Social media administrator bans user for certain time

3c. Social media administrator moderates user

1 Social media administrator cancel user's account

4a. Social media administrator manage user

1 Social media administrator restore user's deleted account

4b. Social media administrator manage user

1 Social media administrator reset user's password

4c. Social media administrator manage user

1 Social media administrator restore user's forgotten account

Special Requirements:

- Social media administrator should be able to access all the reported content easily

Frequency of Occurrence:

- Very frequent. Since the application is for all ages and users are expected frequently check the content regularly, the content must be kept appropriate at all times.

Open Issues:

- How quickly can social media administrator manage the reported user

3.2 Non-functional Requirements

3.2.1 Introduction

This section explains all of the requirements not expressed in the use cases

3.2.2 Functionality

Security

- If the user starts a new training program and didn't agree the security policy there will be a popup-window to ask the user about his security settings

Safety

- The user should keep his devices on which he uses the app updated in order to avoid the loss of his local stored personal data

3.2.3 Usability

Connection error

- If the user isn't connected to the internet there will be a sign that signifies that some services are blocked until the user connect again to the internet

Simplicity

- The use of this app on the watch shall be intuitive. The buttons are clearly labeled and the user can easily navigate through the different features of the app. Users aged from 12 to 80 should be able to use the app without any major problems

Regional requirements

- The app is designated for the Korean market. Although the displayed text is provided in English and payments shall be done in USD to enable foreigners living in Korea to use the app too

3.2.4 Reliability

Availability

- The app should work properly in 99.99% of used times. A restart of the app should be the main action which can fix any problems during the use of the app.

Health safety

- The app should provide high quality exercise programs and diet recipes. The feed-back feature and certified users shall provide that the chance of damaging the user's health is minimized.

Changeability

- Changes to the app are done through updates from the system administrator.

3.2.5 Purchased components

In-app purchases possible

- The user can unlock certified diet recipes and exercise programs with money.

3.2.6 Implementation Constraints

- Need a significant amount of content in the beginning to attract early users.

4 Domain Model

4.1 Domain Model Diagram

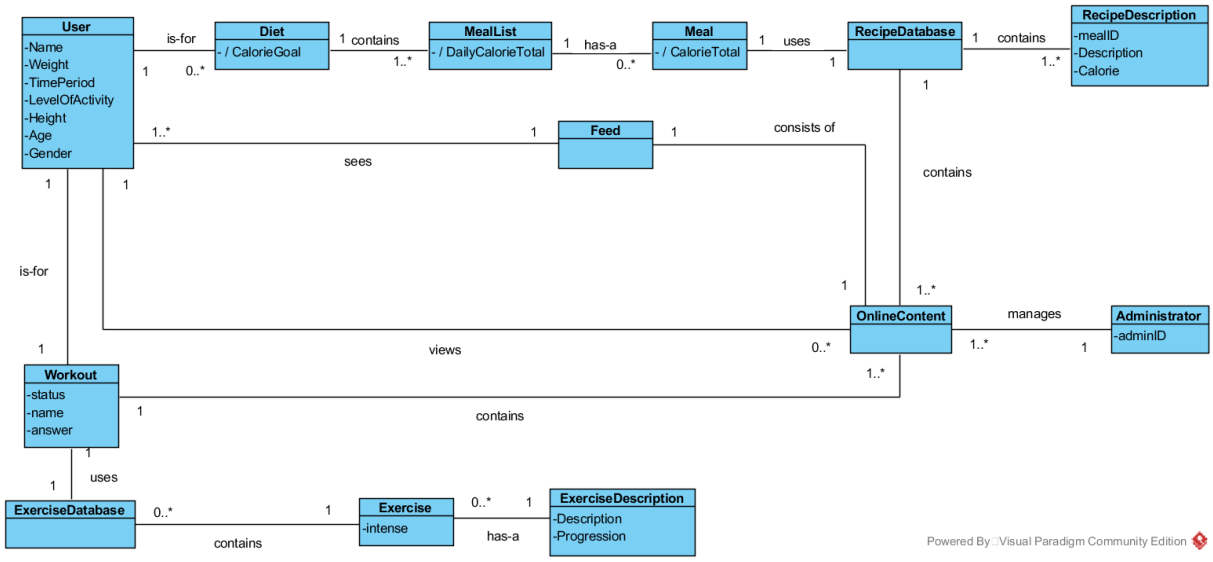


Figure 4-1 Domain Class Diagram

Figure 4-1 shows the conceptual model of the howl system, using associations between the classes what represents the real world problem.

4.1.1 Domain classes

Table 4-1 description for domain class User

User	
Attributes	Name Weight TimePeriod LevelOfActivty Height Age Gender
Association	- One user makes one diet - One user makes one workout - One or more user uses one feed

Table 4-2 description for domain class Diet

Diet	
Attributes	CalorieGoal
Association	- One diet is made by one user - One diet contains a meal list - One diet uses user information (optional)

Table 4-3 description for domain class MealList

MealList	
Attributes	DailyCalorieTotal
Association	- One or more meal lists are contained by one diet - One meal list contains one or more meals

Table 4-4 description for domain class Meal

Meal	
Attributes	CalorieTotal
Association	- One or more meals are contained by one meal list - One meal uses a recipe catalog

Table 4-5 description for domain class RecipeCatalog

RecipeDatabase	
Attributes	-
Association	One recipe catalog contains one or more recipe descriptions

Table 4-6 description for domain class RecipeDescription

RecipeDescription	
Attributes	mealID Description Calories
Association	One or more recipe descriptions are contained by one recipe catalog

Table 4-7 description for domain class Workout

Workout	
Attributes	status name
Association	- One workout is created by one user - One workout contains one exercise list

Table 4-8 description for domain class Workout

ExerciseDatabase	
Attributes	-
Association	- One exercises catalog contains one or more exercises

Table 4-9 description for domain class Workout

Exercise	
Attributes	intense
Association	- One exercise contains one exercise description - One or more exercises are contained by one exercise list

Table 4-10 description for domain class ExerciseDescription

ExerciseDescription	
Attributes	Description Progression
Association	- One exercise description are contained by one exercise

Table 4-11 description for domain class OnlineContent

OnlineContent	
Attributes	-
Association	- One online content contains one workout - One online content contains one recipe catalog

Table 4-12 description for domain class Feed

Feed	
Attributes	-
Association	- One feed is used by one or more users - One feed uses one online content

5 System Sequence Diagram

5.1 UC1 – Do exercise program

5.1.1 UC1 Main 1 – SSD

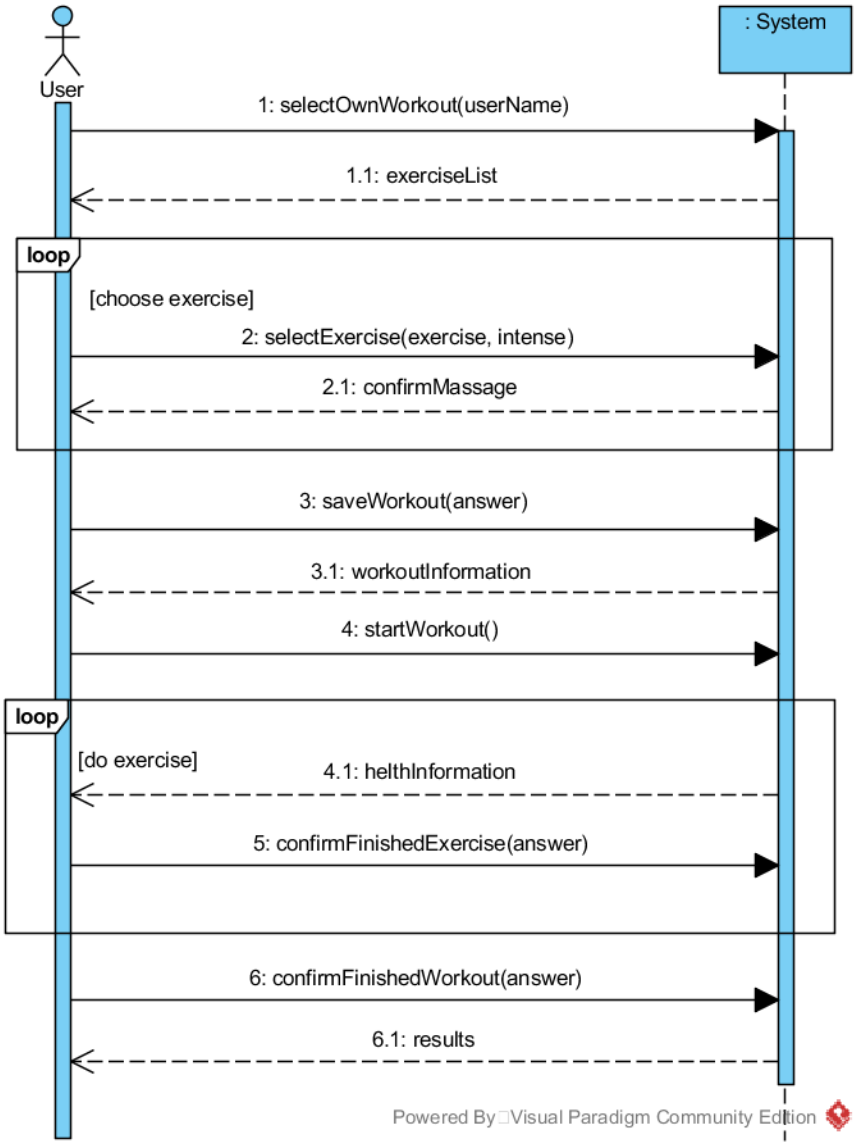


Figure 5-1 Use Case 1 SSD, Main1

Figure 5-1 represents the first main success scenario which the user can reach in use case 1. It shows that the user selects that he wants to do his own workout (operation 1:) by adding exercises. After the user decided to do his own workout, the system displays a list of exer-

cises that the user can choose (operation 1:1). In the first loop, the user adds this exercises (operation 2:) and becomes a confirm massage from the system (operation 2:1). After that, the user can choose if he wants to save his workout (operation 3:) to use it afterwards. The system displays all added exercises again (operation 3:1). After the user starts his workout (operation 4:), the system will give him information about his health status during the workout (operation 4:1). The uses need to confirm his finished exercise (operation 5:) and at the end, his finished workout (operation 6:). The system shows the result from the done workout (operation 6:1).

5.1.2 UC1 Main 1– Operation Contracts

Table 5-1 UC1 Main 1 - selectOwnWorkout(userName : String)

Operation:	selectOwnWorkout(username : String)
Cross references:	Do exercise program
Preconditions:	- none
Postconditions:	- A OwnWorkout instance ow was created - ow was associated with a User - Attribute ow.name was initialized with userName

This system operation is to show different exercises to the user that he can add from the system.

Table 5-2 UC1 Main 1 - selectExercise(exercise : Exercise, intense : Intense)

Operation:	selectExercise(exercise : Exercise, intense : Intense)
Cross references:	Do exercise program
Preconditions:	- There is a own workout is underway
Postconditions:	- Attribute exercise.intense was initialized with intense - Workout was associated with ExerciseCatalog

This system operation is to add exercises to the user workout.

Table 5-3 UC1 Main 1 - saveWorkout(answer : boolean)

Operation:	saveWorkout(answer : boolean)
Cross references:	Do exercise program
Preconditions:	- There is a own workout is underway
Postconditions:	- Attribute userWorkout was initialized with current workout - User was associated with Workout

Table 5-4 UC1 Main 1 - startWorkout()

Operation:	startWorkout()
Cross references:	Do exercise program
Preconditions:	- There is a own workout is underway
Postconditions:	- Attribute workout.status became true

This system operation starts the workout.

Table 5-5 UC1 Main 1 - confirmFinishedExercise(answer : boolean)

Operation:	confirmFinishedExercise(answer : boolean)
Cross references:	Do exercise program
Preconditions:	- There is a own workout is underway
Postconditions:	- Attribute workout.answer became true

This system operation confirms a finished exercise.

Table 5-6 UC1 Main 1 - confirmFinishedWorkout(answer : boolean)

Operation:	confirmFinishedWorkout(answer : boolean)
Cross references:	Do exercise program
Preconditions:	- There is a own workout is underway
Postconditions:	- Attribute workout.status became false

This system operation confirms the finished workout.

5.1.3 UC1 Main 2 – SSD

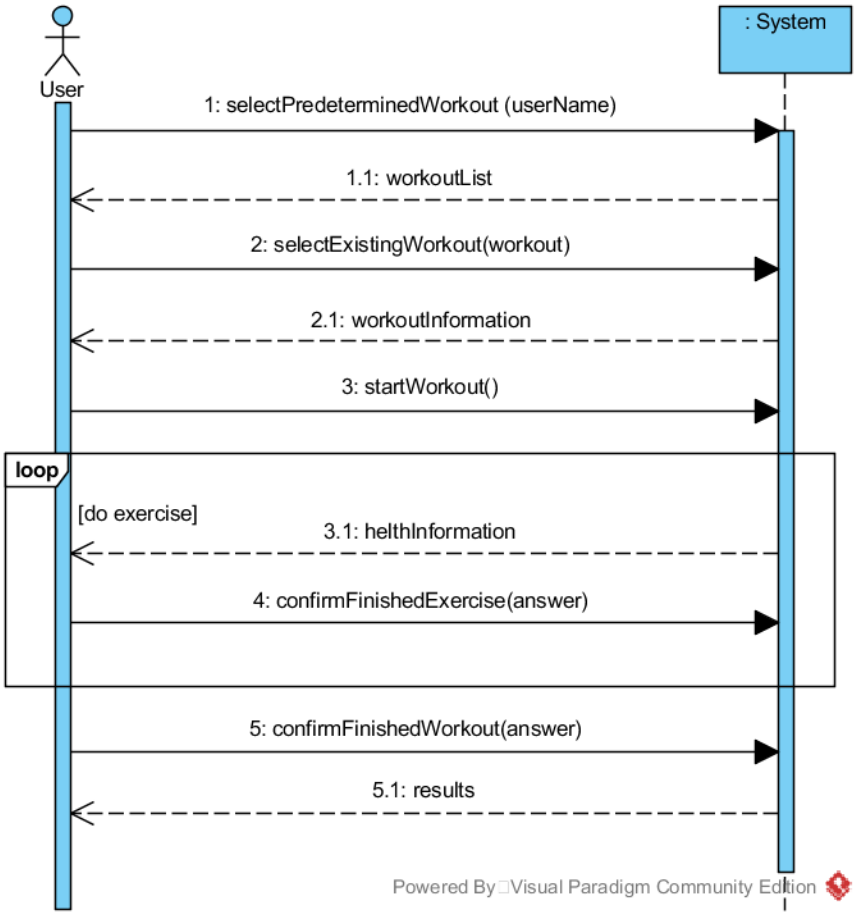


Figure 5-2 Use Case 1 SSD, Main2

Figure 5-2 represents the second main success scenario which the user can reach in use case 1. It shows that the user selects that he wants to do a workout which was saved in the online content (operation 1:). After the user decided to do a predetermined workout, the system displays a list possible workouts that the user can choose (operation 1:1). The user adds this workout (operation 2). The system displays the chosen workout (operation 2:1). After the user starts his workout (operation 3:), the system will give him information about his health status during the workout (operation 3:1). The user needs to confirm his finished exercise (operation 4:) and at the end, his finished workout (operation 5:). The system shows the result from the done workout (operation 5:1).

5.1.4 UC1 Main 2 – Operation Contracts

Table 5-7 UC1 Main 2 - selectPrederterminedWorkout(username : String)

Operation:	selectPrederterminedWorkout(username : String)
Cross references:	Do exercise program
Preconditions:	none
Postconditions:	- A PrederterminedWorkout instance pw was created - pw was associated with User - Attribute pw.name was initialized with userName

This system operation displays the different workouts that the user can choose.

Table 5-8 UC1 Main 2 - selectExistingWorkout(exerciseList : List<Exercise>)

Operation:	selectExistingWorkout(exerciseList : List<Exercise>)
Cross references:	Do exercise program
Preconditions:	- There is a predetermined workout underway
Postconditions:	- Attribute exerciseList was initialized with exerciseList - Workout was associated with OnlineContent

This system operation selects the chosen workout.

Table 5-9 UC1 Main 2 - startWorkout()

Operation:	startWorkout()
Cross references:	Do exercise program
Preconditions:	- There is a predetermined workout underway
Postconditions:	- Attribute workout.status became true

This system operation starts the workout.

Table 5-10 UC1 Main 2 - confirmFinishedExercise(answer : boolean)

Operation:	confirmFinishedExercise(answer : boolean)
Cross references:	Do exercise program
Preconditions:	- There is a predetermined workout underway
Postconditions:	- Attribute workout.answer became true

This system operation confirms a finished exercise.

Table 5-11 UC1 Main 2 - confirmFinishedWorkout(answer : boolean)

Operation:	confirmFinishedWorkout(answer : boolean)
Cross references:	Do exercise program
Preconditions:	- There is a predetermined workout underway
Postconditions:	- Attribute workout.status became false

This system operation confirms the finished workout.

5.2 UC2 – Do diet program

5.2.1 UC2 – SSD

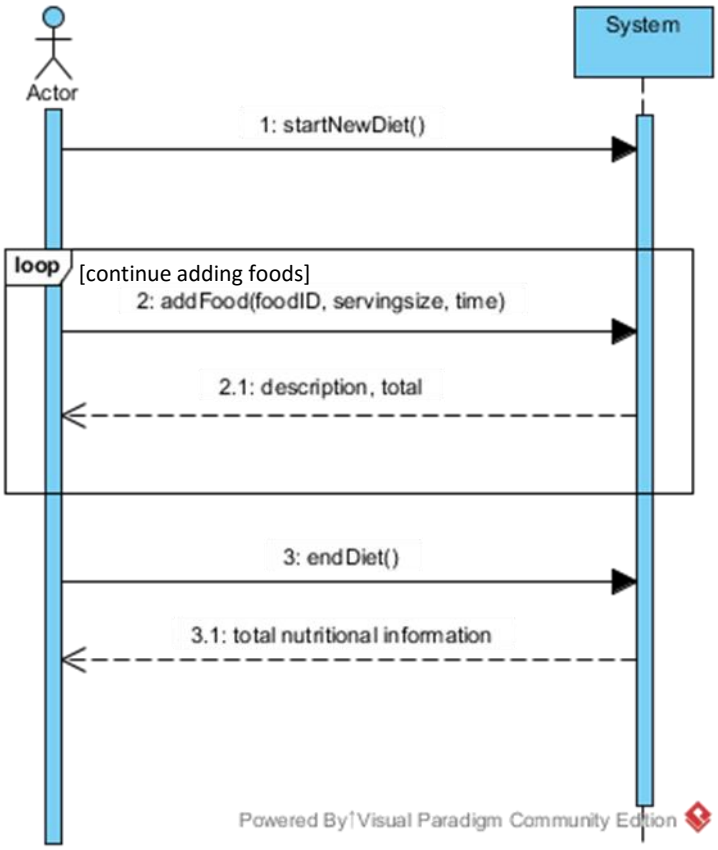


Figure 5-3 Use Case 2 SSD

User begins a Diet for the day. The user can add a food item specifying the serving size and time of day it is eaten. Once user is finished adding food items changes are saved and the total values are displayed.

5.2.2 UC2 – Operation Contracts

Table 5-12 UC2 - startNewDiet()

Operation:	startNewDiet()
Cross references:	Do Diet Program
Preconditions:	- None
Postconditions:	- Do Diet Program - d was associated to user - Attributes of MealList were initialized

Table 5-13 UC2 - addFood(food, servingsize, Time)

Operation:	addFood(food, servingsize, Time)
Cross references:	Do Diet Program
Preconditions:	- MealList has been initialized
Postconditions:	- A Meal instance m was created - m was associated with current MealList. - An instance of class Time was created (for which meal i.e. lunch). - Attributes of Meal were initialized

5.3 UC3 – Interact with the social media

5.3.1 UC3 – SSD

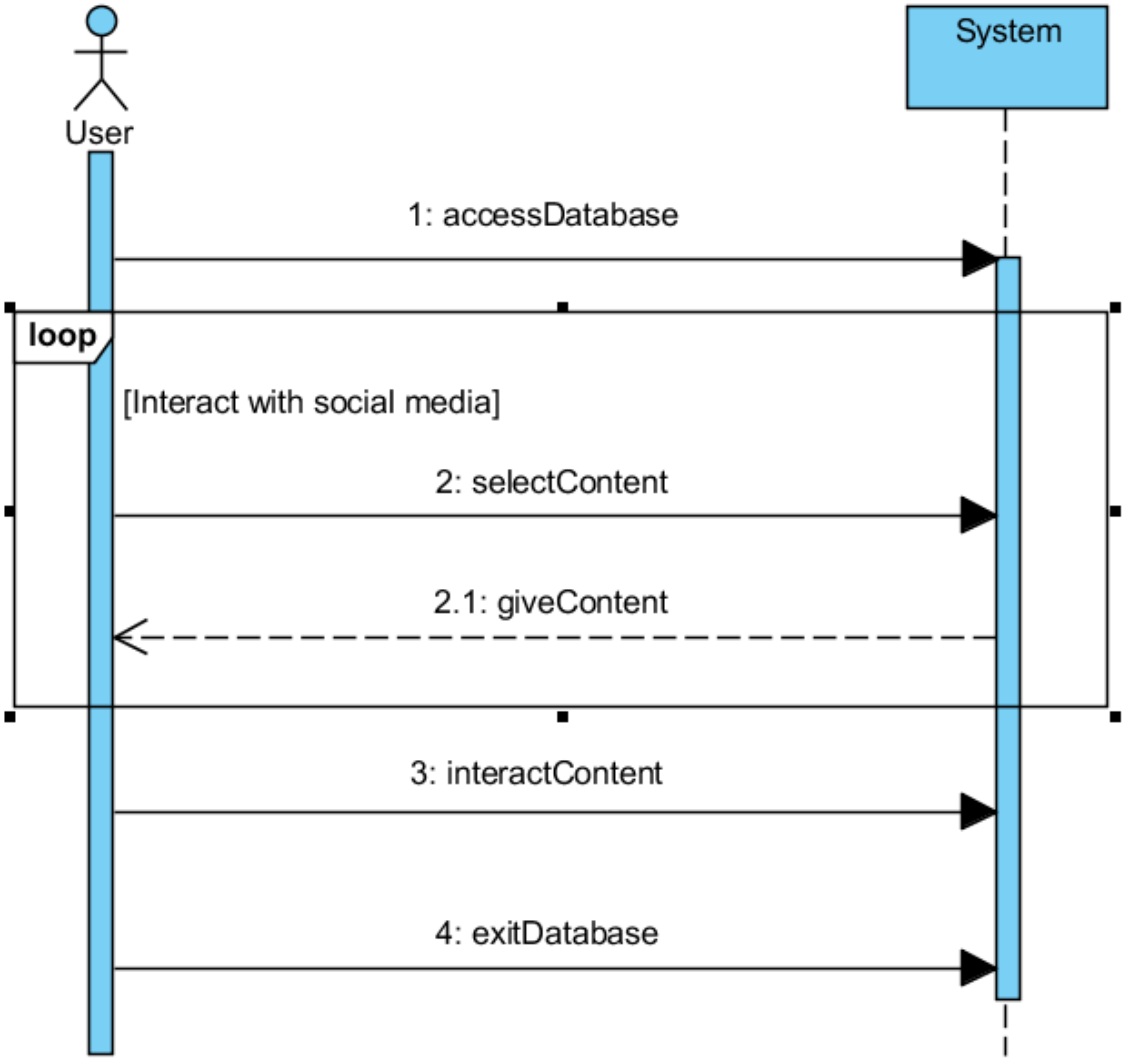


Figure 5-4 Use Case 3 SSD

User logs in to the system and get access to Social Media database. Then User selects what content he wants to see. Information about that is sent to database and database gives back the content what the User choose. User interacts with content and this interaction will be stored into database for others to see. User logs out after he is done.

5.3.2 UC3 – Operation Contracts

Table 5-14 UC3 - accessDatabase ()

Operation:	accessDatabase ()
Cross references:	Interact with social media
Preconditions:	- User has already an account or user creates an account
Postconditions:	- User has access to Database

When User logs in and wants to interact with “Feed/Media”, User will have input his account or create a new one, if he doesn’t have already.

Table 5-15 UC3 - selectContent()

Operation:	selectContent()
Cross references:	Interact with social media
Preconditions:	- User has already log in and have a access to database
Postconditions:	- Information about User’s requested content has been sent

When User logs in and wants to interact with “Feed/Media”, User will have to select what kind of content one would like to see. After User has selected the type of content he or she wants to see, information about the choice will be sent to System.

Table 5-16 UC3 - giveContent()

Operation:	giveContent()
Cross references:	Interact with social media
Preconditions:	- System has information about what content User wants to receive
Postconditions:	- User’s requested content has been delivered to User

System receives an information about User’s request to see chosen content and will deliver requested content to User.

Table 5-17 UC3 - interactContent()

Operation:	interactContent()
Cross references:	Interact with social media
Preconditions:	- User has received requested content

Postconditions:	- System stored information about the User's interaction with the contents - Comment, rate or shared content has been submitted

When User receives the content from the System User can start to interact with the content. Interactions will be stored into database. For example, if User wrote a comment on video, it will be stored to database to others to see.

5.4 UC4 – Interact with online content

5.4.1 UC4 – SSD

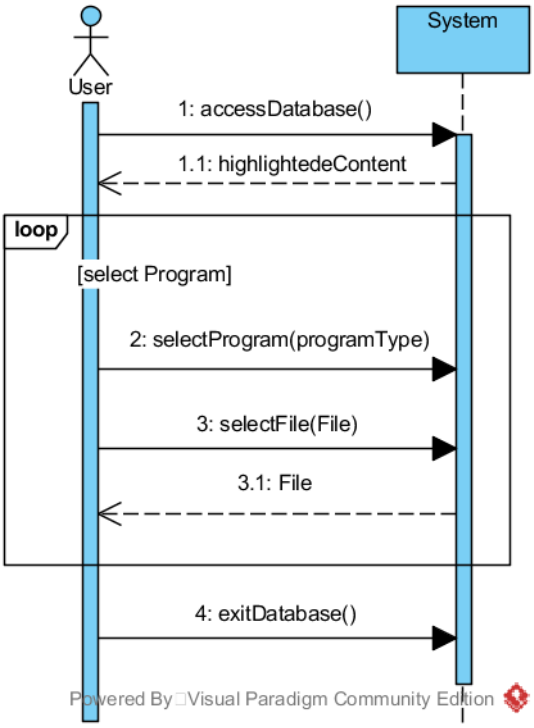


Figure 5-5 Use Case 4 SSD

Figure 5-5 shows the way how the user is interacting with the online content to show the newest feeds and download a diet or workout form the online content. The user access the online content with operation 1:. The system will show the newest contents to the user after that. The loop represents the progress to selection type (diet or workout, operation 2:) and the chosen file (operation 3:). The system will send the chosen file to the user (operation 3:1). If the user finish the download, he leaves (operation 4:).

5.4.2 UC4 – Operation Contracts

Table 5-18 UC4 - accessDatabase()

Operation:	accessDatabase()
Cross references:	InteractWithContent
Preconditions:	- None
Postconditions:	- user was associated with the Feed

	- Feed was associated with the OnlineContent

This operation is to access to online content.

Table 5-19 UC4 - selectProgram(programType)

Operation:	selectProgram(programType)
Cross references:	InteractWithContent
Preconditions:	- None
Postconditions:	- OnlineContent.Catalog[] (variable) was initialized - depending on programType either an ExerciseCatalog or a RecipeCatalog was saved into with the OnlineContent.Catalog[]

This system operation is to select a program type (diet or workout).

Table 5-20 UC4 - selectFile(File)

Operation:	selectFile(File)
Cross references:	InteractWithContent
Preconditions:	- There are files to select
Postconditions:	- user.programm (variable) was initialized - file was associated with the user

This system operation is to select, and finally to become the program that the user wants.

Table 5-21 UC4 - exitDatabase()

Operation:	exitDatabase()
Cross references:	InteractWithContent
Preconditions:	- None
Postconditions:	- feed association with the database was deleted - user association with the feed was deleted

This system operation is to leave the online conten

5.5 UC5 – Manage users

5.5.1 UC5 – SSD

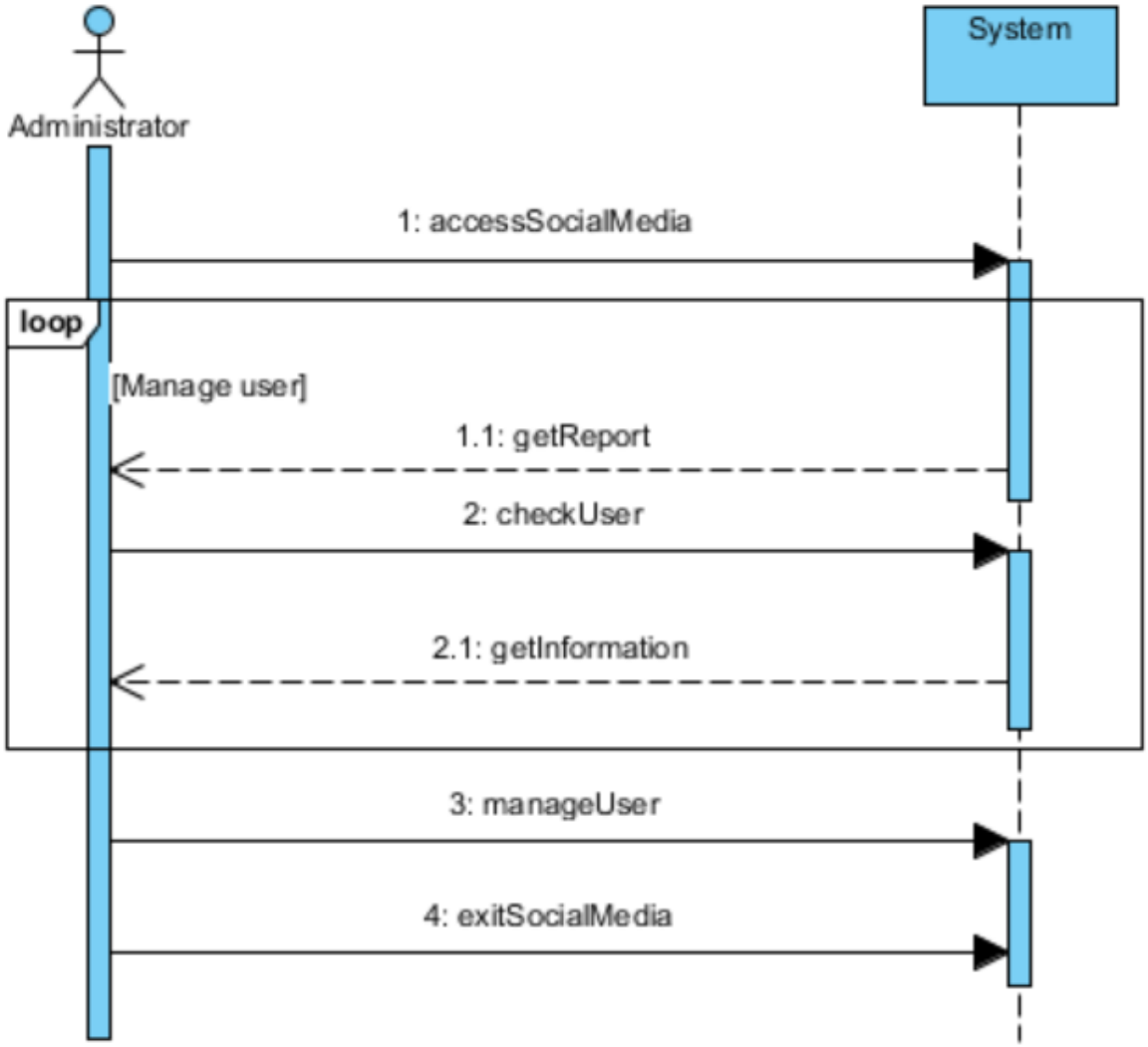


Figure 5-6 Use Case 5 SSD

Social media administrator logs into the system and get access to social media database. Social media administrator gets notification about reported users. Then social media administrator manages the users the way he thinks is best. Information about managing users will be stored into database. Social media administrator logs out after he is done.

5.5.2 UC5 – Operation Contracts

Table 5-22 UC5 - accessSocialMedia()

Operation:	accessSociaMedia()
Cross references:	Manage Users
Preconditions:	- There is connection to Social media
Postconditions:	- The social media administrator is associated with social media

Table 5-23 UC5 - checkUser()

Operation:	checkUser()
Cross references:	Manage users
Preconditions:	- Access to user database
Postconditions:	- Request of information about user is sent

Table 5-24 UC5 – manageUser()

Operation:	manageUser()
Cross references:	Manage users
Preconditions:	- There are users to manage
Postconditions:	- None

Table 5-25 UC5 – exitSocialMedia()

Operation:	exitSocialMedia()
Cross references:	Manage users
Preconditions:	- None
Postconditions:	- Connection with the social media is disconnected

6 Design Model

6.1 Use Case 1 - Do Exercise Program Realization

6.1.1 UC1 Main1 - Operation 1

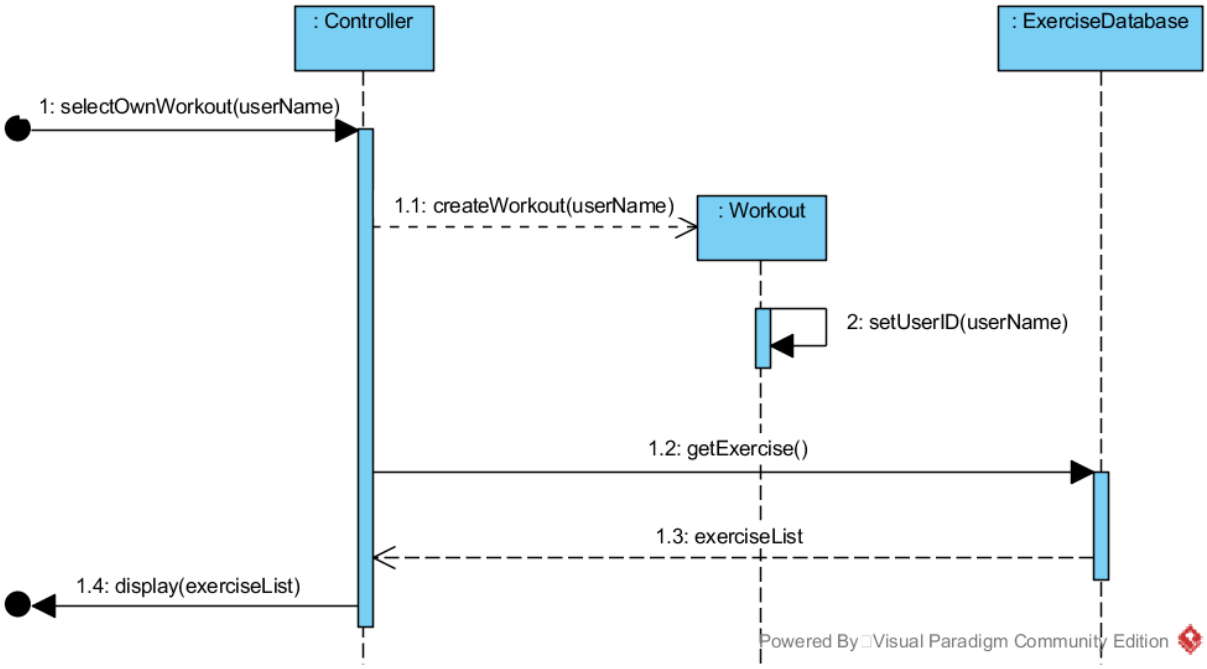


Figure 6-1 UC1 Main1 Sequence Diagram for Operation 1

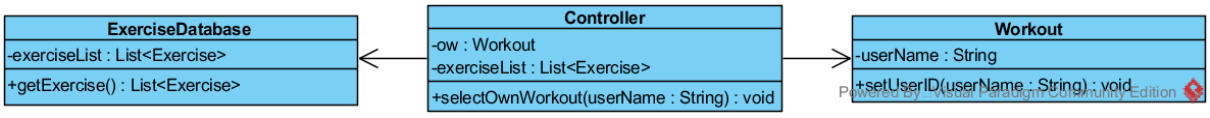


Figure 6-2 UC1 Main1 Class Diagram Operation 1

Table 6-1 UC1 GRASP Main1 Operation 1

Creator	- Controller creates a Workout
Information Expert	- Controller knows username - ExerciseDatabase knows a list of exercises (exerciseList)
Low Coupling	- Controller does not need to know how to bring exerciseList
High Cohesion	- Workout has only one responsibility. Set name of user.
Controller	- Controller represents a handler of all system events, used by user.

6.1.2 UC1 Main1 - Operation 2

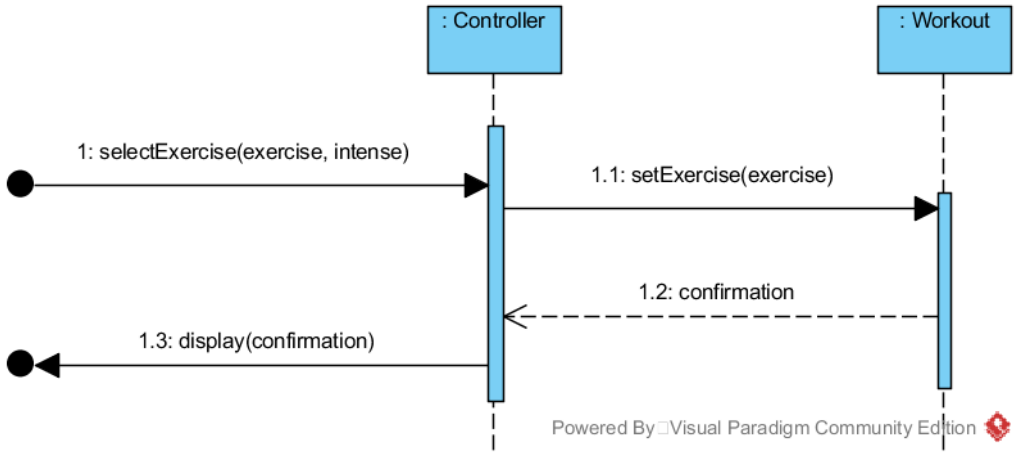


Figure 6-3 UC1 Main1 Sequence Diagram for Operation 2

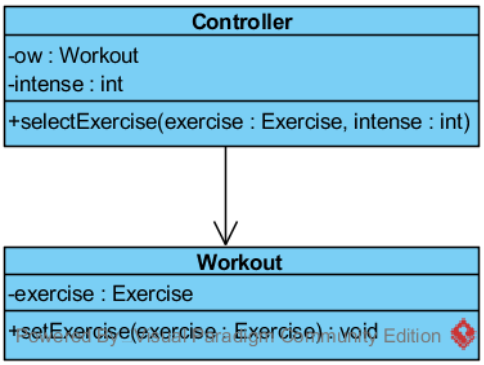


Figure 6-4 UC1 Main1 Class Diagram Operation 2

Table 6-2 UC1 GRASP Main1 Operation 2

Creator	- None
Information Expert	- User knows exercise and intense - Workout knows confirmation
Low Coupling	- Controller doesn't need to know how to set exercise in workout
High Cohesion	- Controller has only one responsibility. Set exercise.
Controller	- Controller represents a handler of all system events, used by user.

6.1.3 UC1 Main1 - Operation 3

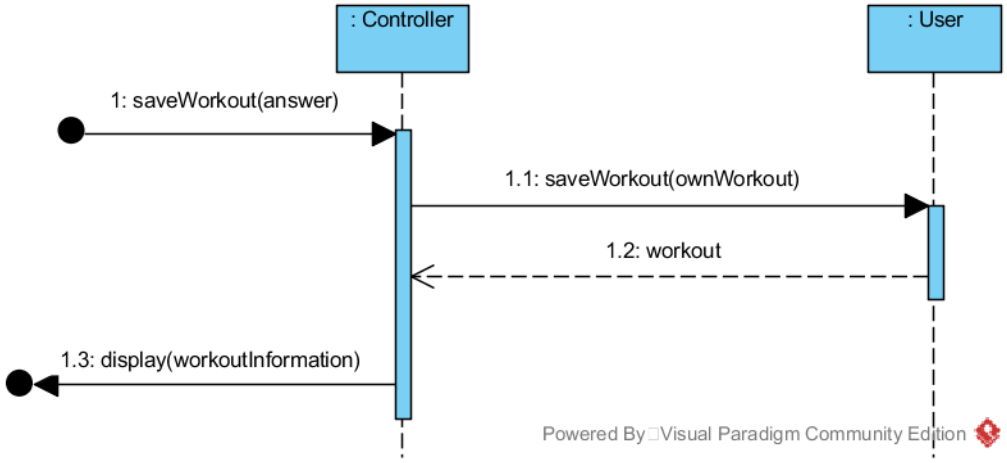


Figure 6-5 UC1 Main1 Sequence Diagram for Operation 3

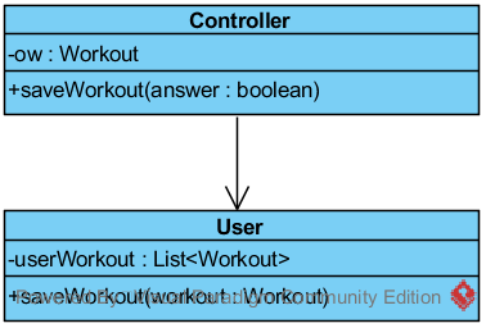


Figure 6-6 UC1 Main1 Class Diagram Operation 3

Table 6-3 UC1 GRASP Main1 Operation 3

Creator	- None
Information Expert	- User knows answer
Low Coupling	- Controller doesn't need to know how to save workout
High Cohesion	- Controller has only one responsibility. Save workout.
Controller	- Controller represents a handler of all system events, used by user.

6.1.4 UC1 Main1 - Operation 4

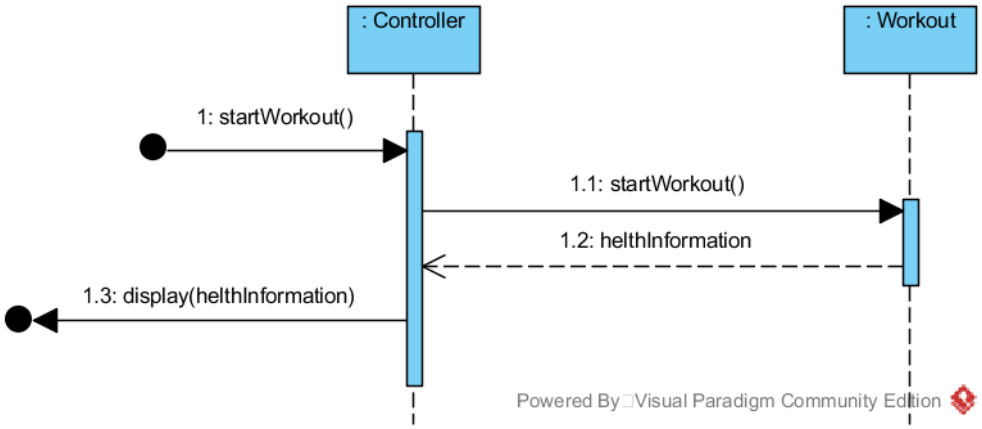


Figure 6-7 UC1 Main1 Sequence Diagram for Operation 4

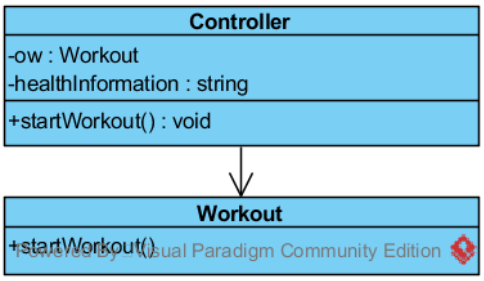


Figure 6-8 UC1 Main1 Class Diagram Operation 4

Table 6-4 UC1 GRASP Main1 Operation 4

Creator	- None
Information Expert	- Workout knows healthInformation
Low Coupling	- Controller doesn't need to know how to start workout
High Cohesion	- Controller has only one responsibility. Start workout.
Controller	- Controller represents a handler of all system events, used by user.

6.1.5 UC1 Main1 - Operation 5

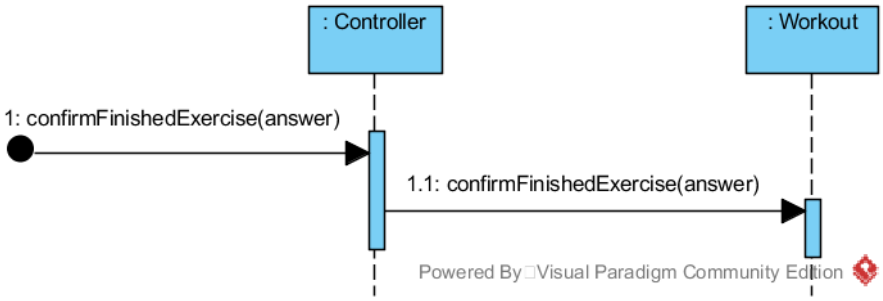


Figure 6-9 UC1 Main1 Sequence Diagram for Operation 5

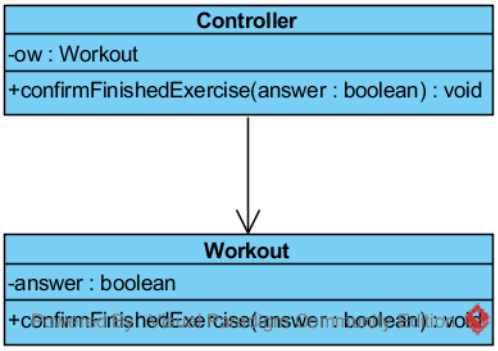


Figure 6-10 UC1 Main1 Class Diagram Operation 5

Table 6-5 UC1 GRASP Main1 Operation 5

Creator	- None
Information Expert	- Controller knows answer
Low Coupling	- Controller doesn't need to know how to confirm a finished exercise
High Cohesion	- Controller hast only one responsibility. Confirm a finished exercise.
Controller	- Controller represents a handler of all system events, used by user.

6.1.6 UC1 Main1 - Operation 6

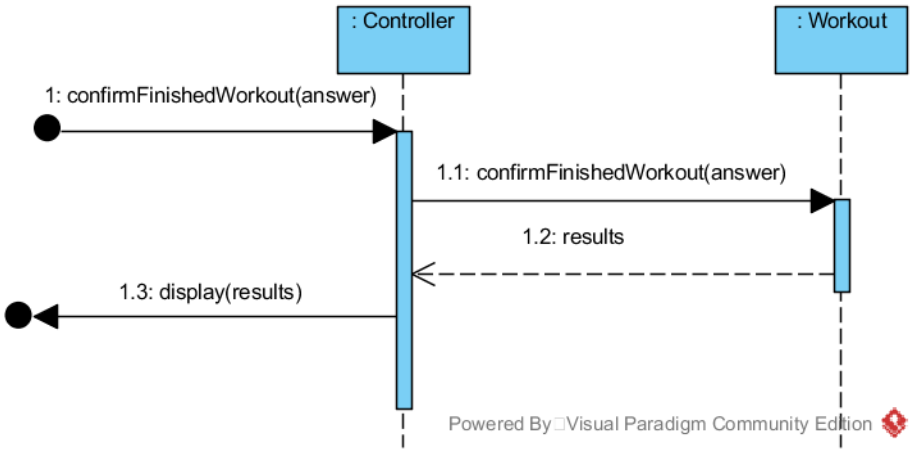


Figure 6-11 UC1 Main1 Sequence Diagram for Operation 6

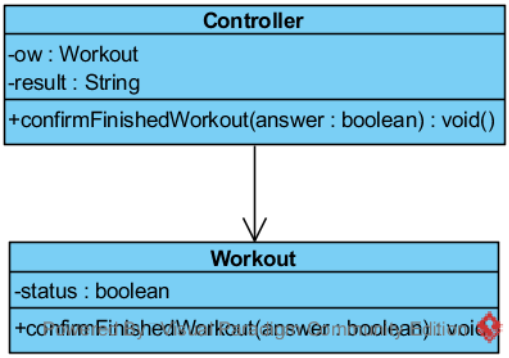


Figure 6-12 UC1 Main1 Class Diagram Operation 6

Table 6-6 UC1 GRASP Main1 Operation 6

Creator	- None
Information Expert	- Controller knows answer - Workout knows result
Low Coupling	- Controller doesn't need to know how to confirm a finished workout
High Cohesion	- Controller has only one responsibility. Confirm a finished workout.
Controller	- Controller represents a handler of all system events, used by user.

6.1.7 UC1 Main2 - Operation 1

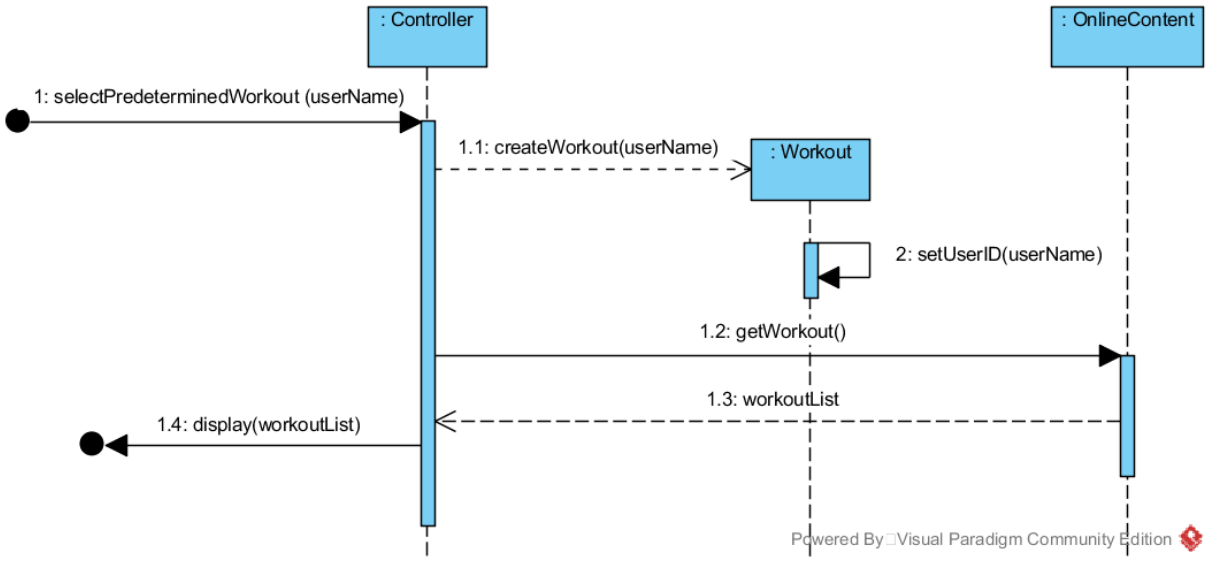


Figure 6-13 UC1 Main2 Sequence Diagram for Operation 1

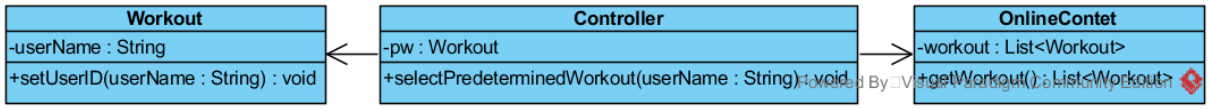


Figure 6-14 UC1 Main2 Class Diagram Operation 1

Table 6-7 UC1 GRASP Main2 Operation 1

Creator	- Controller creates a Workout
Information Expert	- Controller knows user name - OnlineContent knows workoutList
Low Coupling	- Controller doesn't need to know how to get a workout
High Cohesion	- Workout has only one responsibility. Set user name.
Controller	- Controller represents a handler of all system events, used by user.

6.1.8 UC1 Main2 - Operation 2

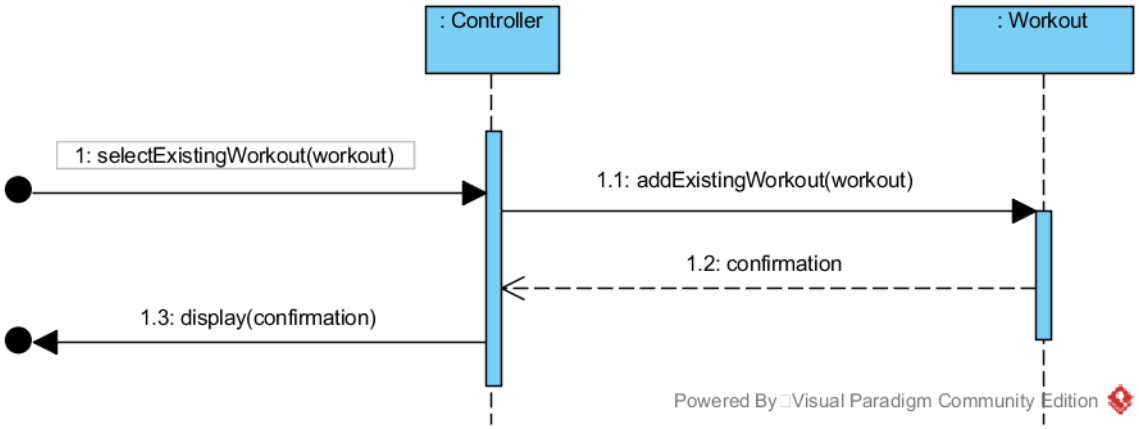


Figure 6-15 UC1 Main2 Sequence Diagram for Operation 2

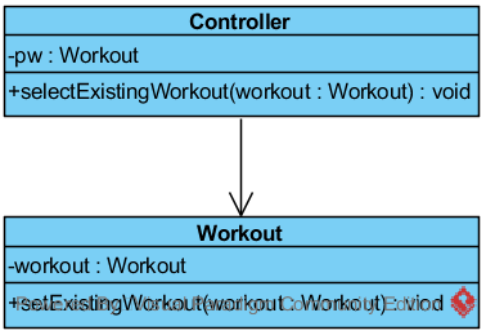


Figure 6-16 UC1 Main2 Class Diagram Operation 2

Table 6-8 UC1 GRASP Main2 Operation 2

Creator	- None
Information Expert	- Controller knows workout - Workout knows confirmation
Low Coupling	- Controller doesn't need to know how to add an existing workout
High Cohesion	- Controller has only one responsibility. Add an existing workout
Controller	- Controller represents a handler of all system events, used by user.

6.1.9 UC1 Main2 - Operation 3

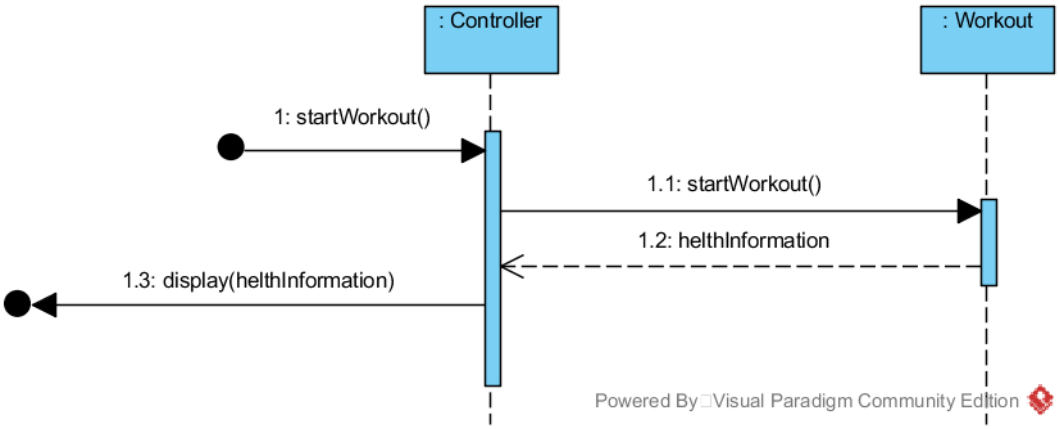


Figure 6-17 UC1 Main2 Sequence Diagram for Operation 3

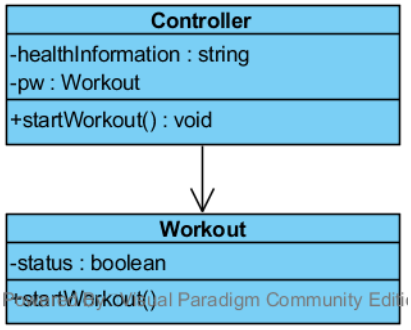


Figure 6-18 UC1 Main2 Class Diagram Operation 3

Table 6-9 UC1 GRASP Main2 Operation 3

Creator	- None
Information Expert	- Workout knows healthInformation
Low Coupling	- Controller doesn't need to know how to start a workout
High Cohesion	- Controller has only one responsibility. Start a workout.
Controller	- Controller represents a handler of all system events, used by user.

6.1.10 UC1 Main2 - Operation 4

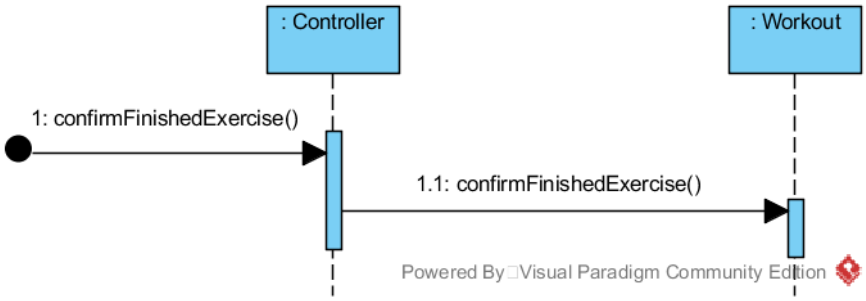


Figure 6-19 UC1 Main2 Sequence Diagram for Operation 4

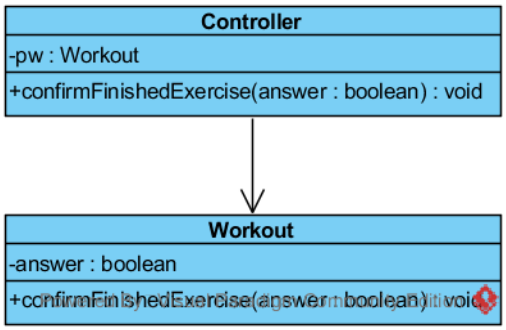


Figure 6-20 UC1 Main2 Class Diagram Operation 4

Table 6-10 UC1 GRASP Main2 Operation 4

Creator	- None
Information Expert	- Controller knows answer
Low Coupling	- Controller doesn't need to know how to confirm a finished exercise
High Cohesion	- Controller hast only one responsibility. Confirm a finished exercise.
Controller	- Controller represents a handler of all system events, used by user.

6.1.11 UC1 Main2 - Operation 5

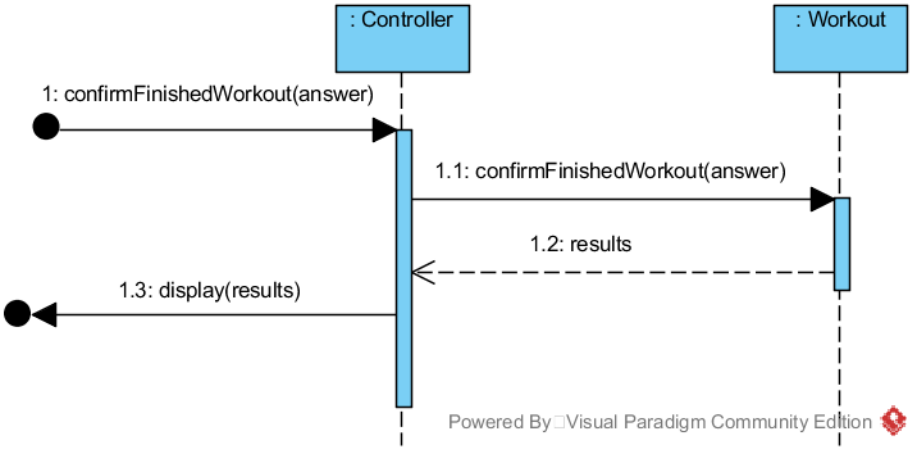


Figure 6-21 UC1 Main2 Sequence Diagram for Operation 5

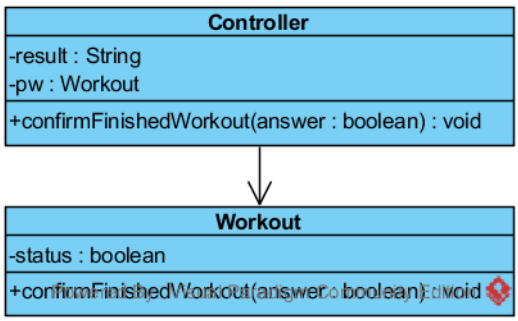


Figure 6-22 UC1 Main2 Class Diagram Operation 5

Table 6-11 UC1 GRASP Main2 Operation 5

Creator	- None
Information Expert	- Controller knows answer - Workout knows result
Low Coupling	- Controller doesn't need to know how to confirm a finished workout
High Cohesion	- Controller has only one responsibility. Confirm a finished workout.
Controller	- Controller represents a handler of all system events, used by user.

6.1.12 Combined Design Class Diagram for Use Case 1

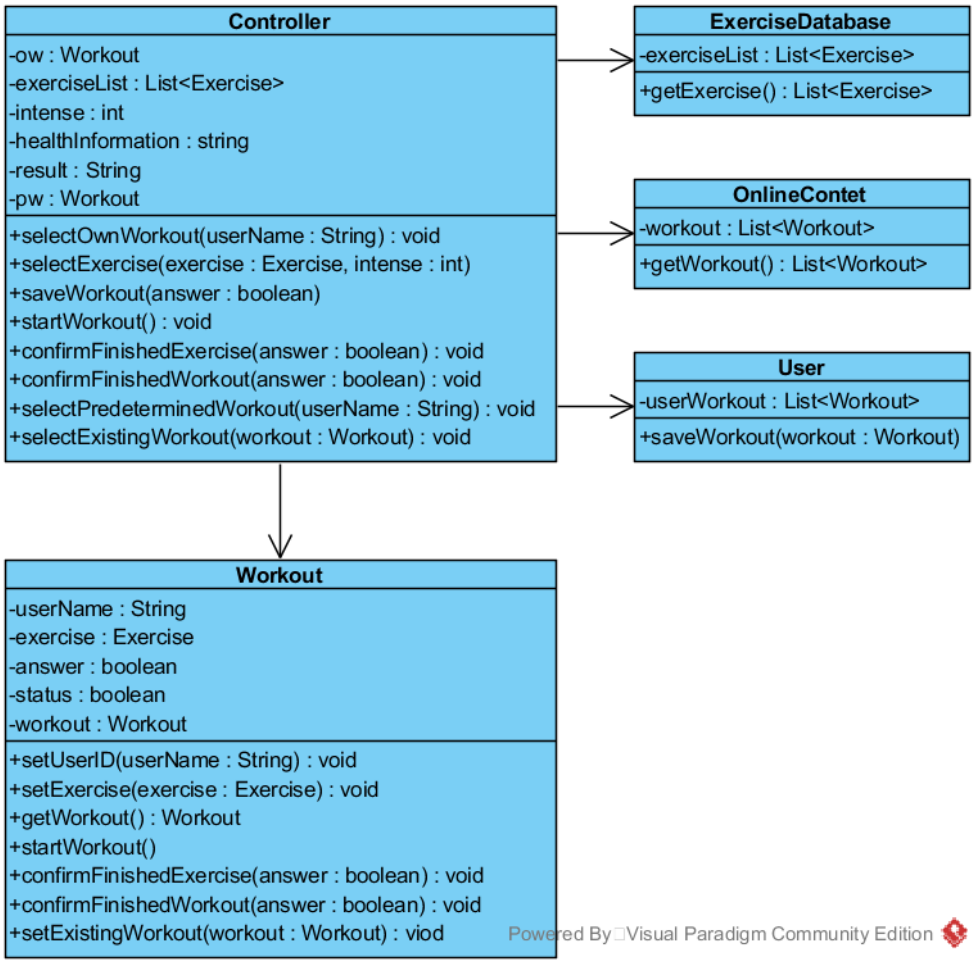


Figure 6-23 Combined DCD for UC1

6.1.13 Combined Design Sequence Diagram for Use Case 1 Main 1

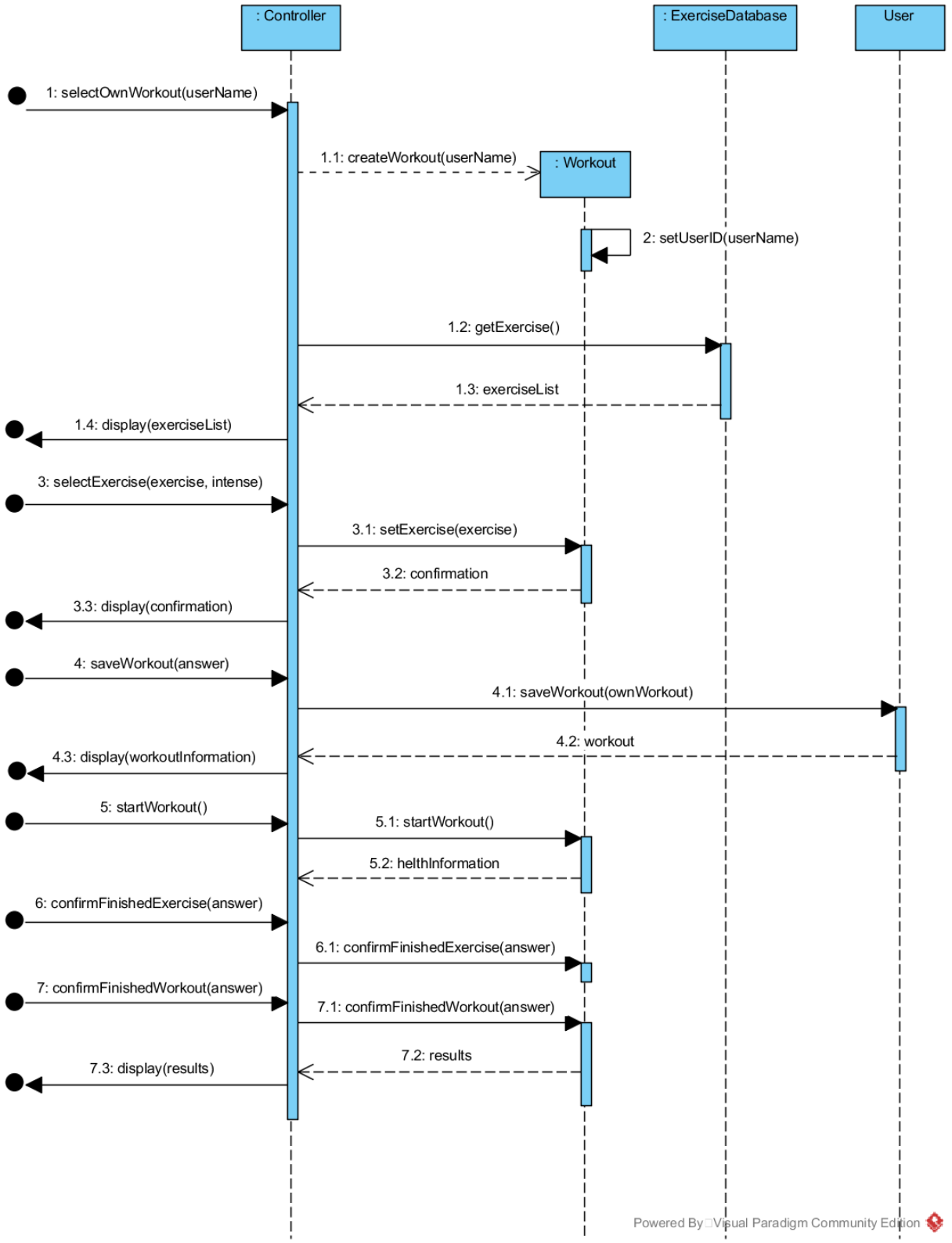


Figure 6-24 Combined DSD for UC1 Main 1

6.1.14 **Combined Design Sequence Diagram for Use Case 1 Main 2**

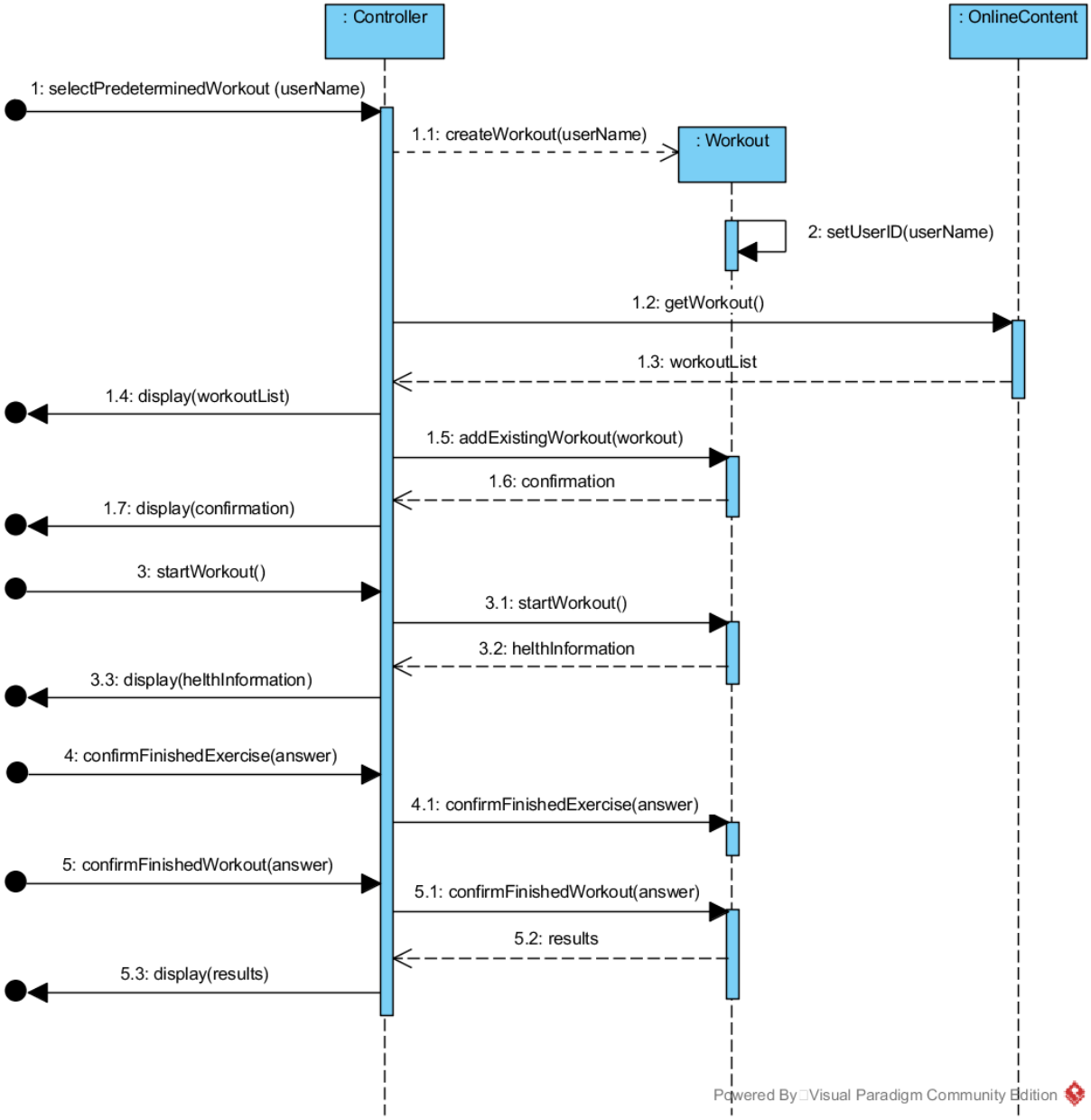


Figure 6-25 Combined DSD for UC1 Main 2

6.2 Use Case 2 - Do Diet Program Realization

6.2.1 UC2 Operation 1

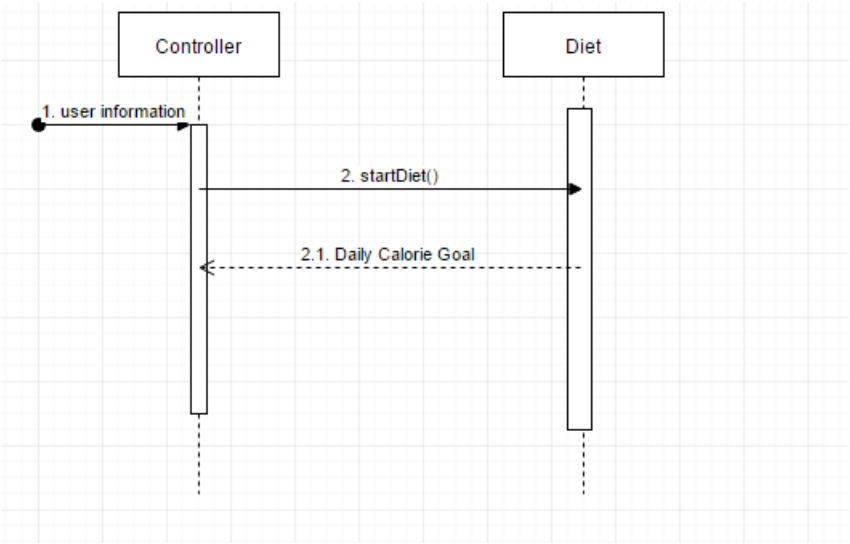


Figure 6-26 UC2 Sequence Diagram for Operation 1

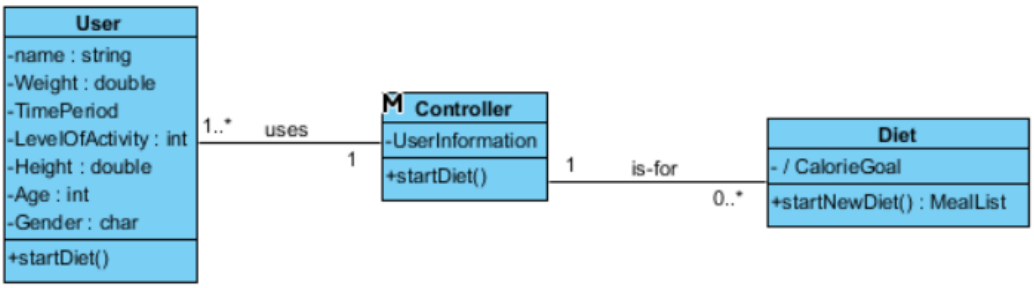


Figure 6-27 UC2 Class Diagram Operation 1

Table 6-12 UC2 GRASP Operation 1

Creator	-	User
Information Expert	-	Diet knows user information from user
Low Coupling	-	User does not need to know how to calculate daily calorie goal
High Cohesion	-	User has only one responsibility to startDiet().
Controller	-	Controlled by user

6.2.2 UC2 Operation 2

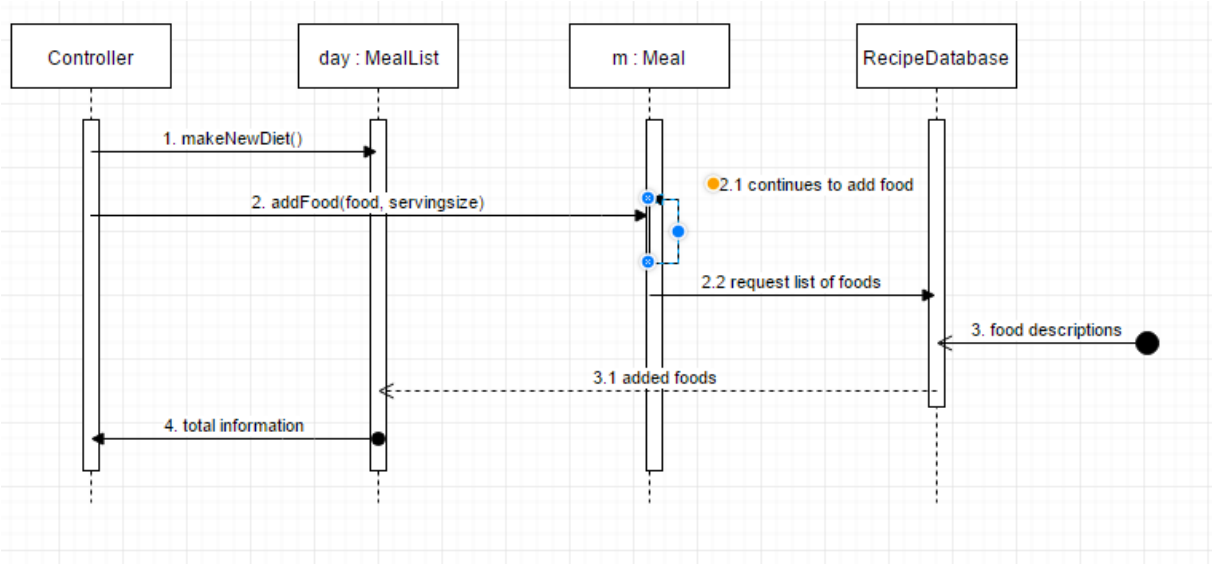


Figure 6-28 UC2 Sequence Diagram for Operation 2

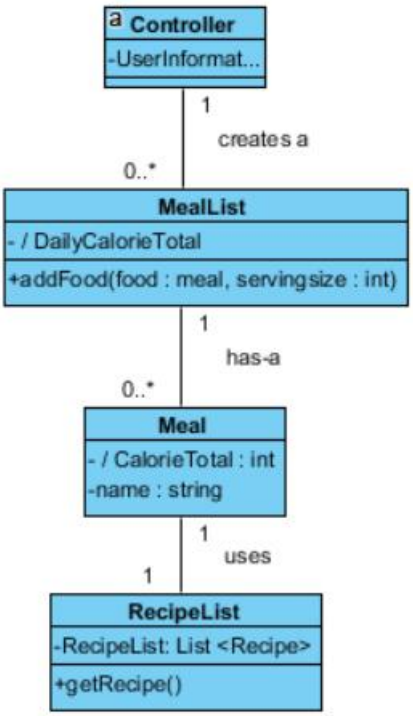


Figure 6-29 UC2 Class Diagram Operation 2

Table 6-13 UC2 GRASP Operation 2

Creator	- User creates a Diet for the day.
Information Expert	- MealList knows nutrition information of Meals from RecipeDatabase.
Low Coupling	- User does not need to know how to get list of foods.
High Cohesion	- MealList has only one responsibility to display information. Organized by time of day.
Controller	- Controlled by user

6.2.3 Combined Design Class Diagram for Use Case 2

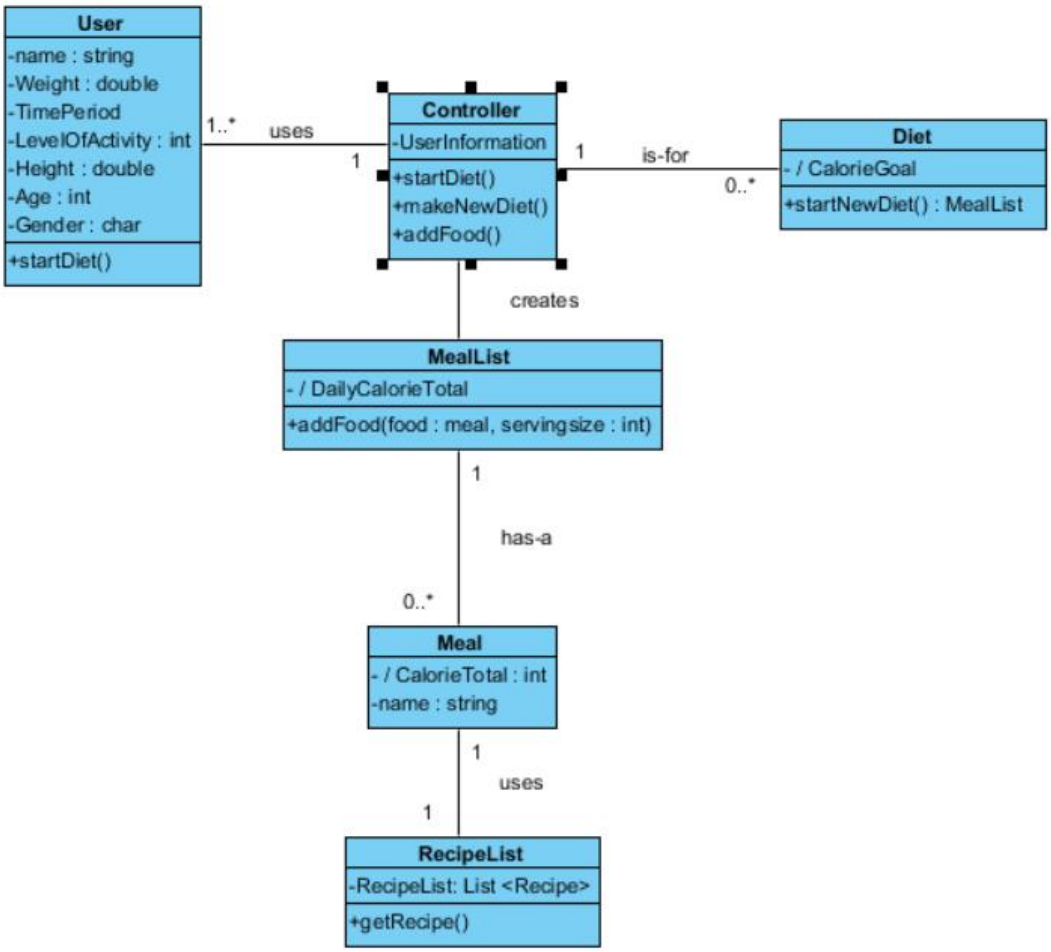


Figure 6-30 Combined DCD for UC2

6.2.4 Combined Design Sequence Diagram for Use Case 2

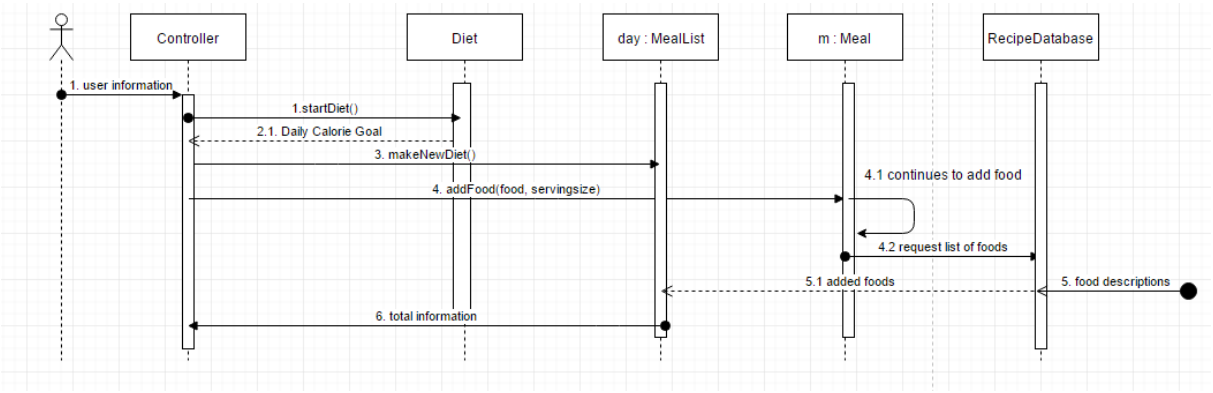


Figure 6-31 Combined DSD for UC2

6.3 Use Case 3 - Interact with the Social Media Realization

6.3.1 UC3 Operation 1

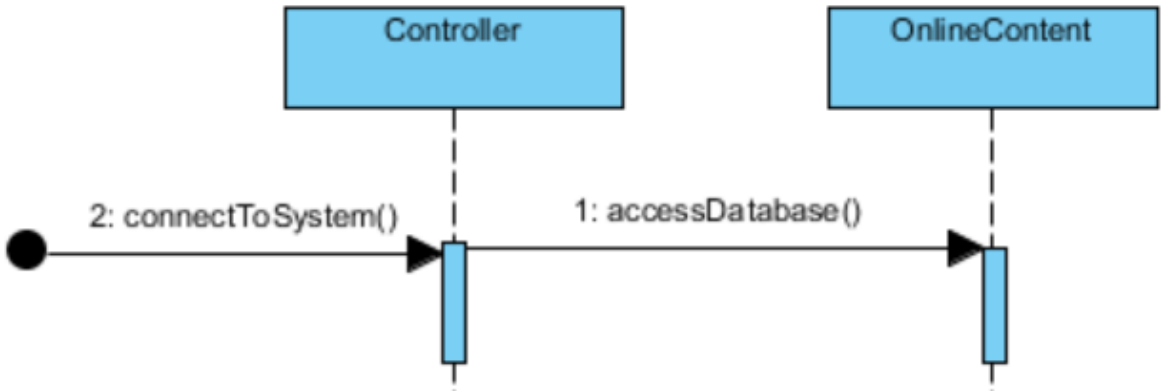


Figure 6-32 UC3 Sequence Diagram for Operation 1

User logs in and connect to system. Database knows user's account information and grant access to database, where all the social media content is.

Figure 6-33 UC3 Class Diagram Operation 1

Table 6-14 UC3 GRASP Operation 1

Creator	- User
Information Expert	- None
Low Coupling	- None
High Cohesion	- Controller has single responsibility to log in
Controller	- None

6.3.2 UC3 Operation 2

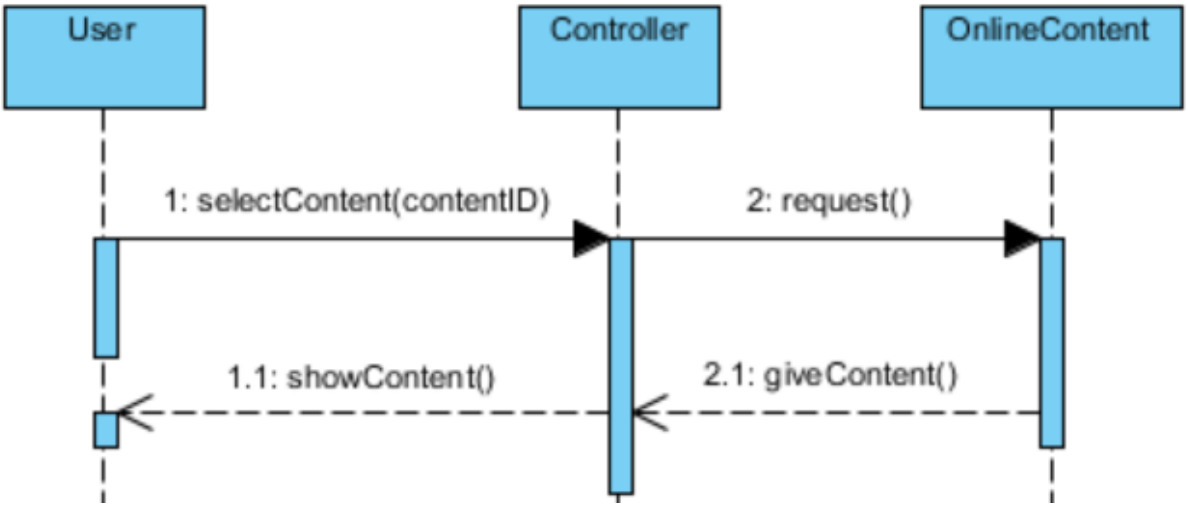


Figure 6-34 UC3 Sequence Diagram for Operation 2

User selects the content he wants and system send the request to database. Database sends back to selected content.

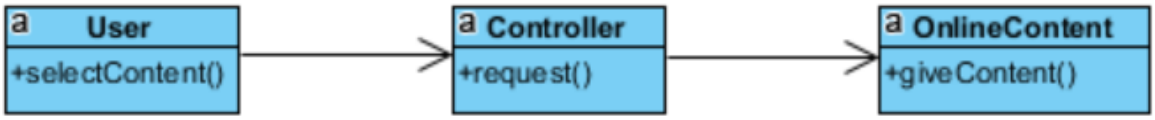


Figure 6-35 UC3 Class Diagram Operation 2

Table 6-15 UC3 GRASP Operation 2

Creator	- User
Information Expert	- OnlineContent knows content
Low Coupling	- User does not need to know how to get content.
High Cohesion	- User has single responsibility to select content
Controller	- User

6.3.3 UC3 Operation 3

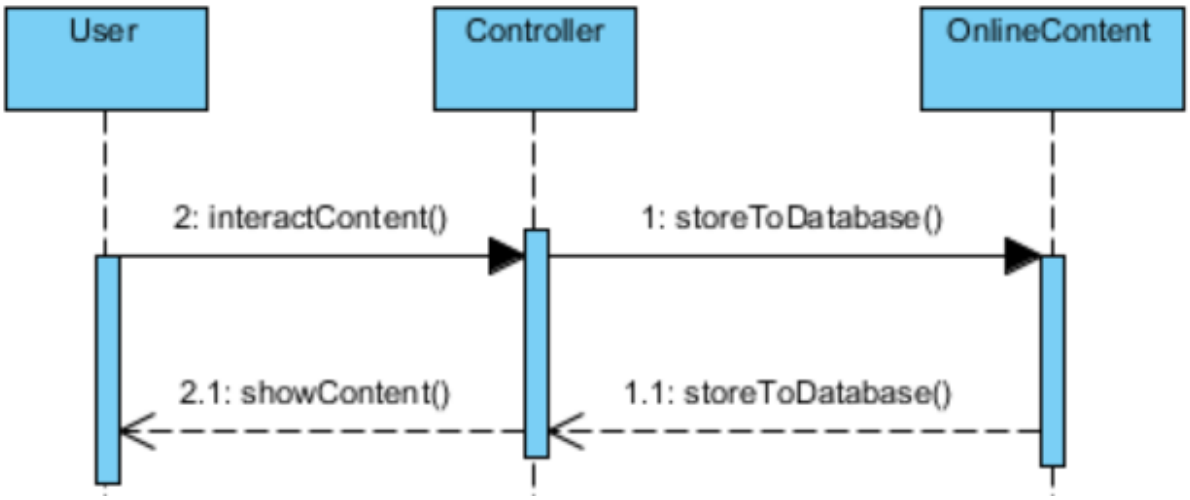


Figure 6-36 UC3 Sequence Diagram for Operation 3

User interact with the content, for example user can vote or leave a comment to selected content. Interaction will be stored to database for others to see.

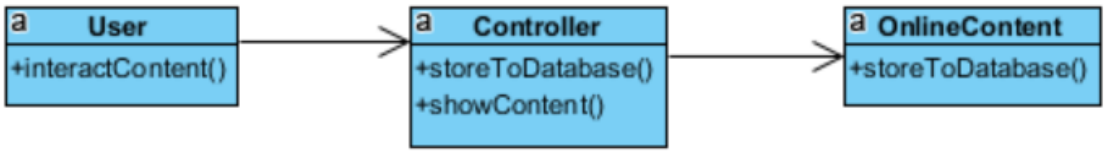


Figure 6-37 UC3 Class Diagram Operation 3

Table 6-16 UC3 GRASP Operation 2

Creator	- User
Information Expert	- None
Low Coupling	- User does not need to know how to store information to online content
High Cohesion	- Controller has single responsibility to interact with content
Controller	- User

6.3.4 Combined Design Class Diagram for Use Case 3

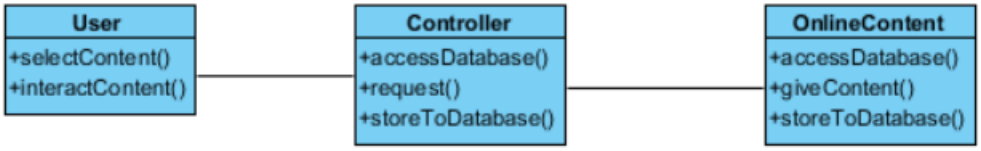


Figure 6-38 Combined DCD for UC3

6.3.5 Combined Design Sequence Diagram for Use Case 3

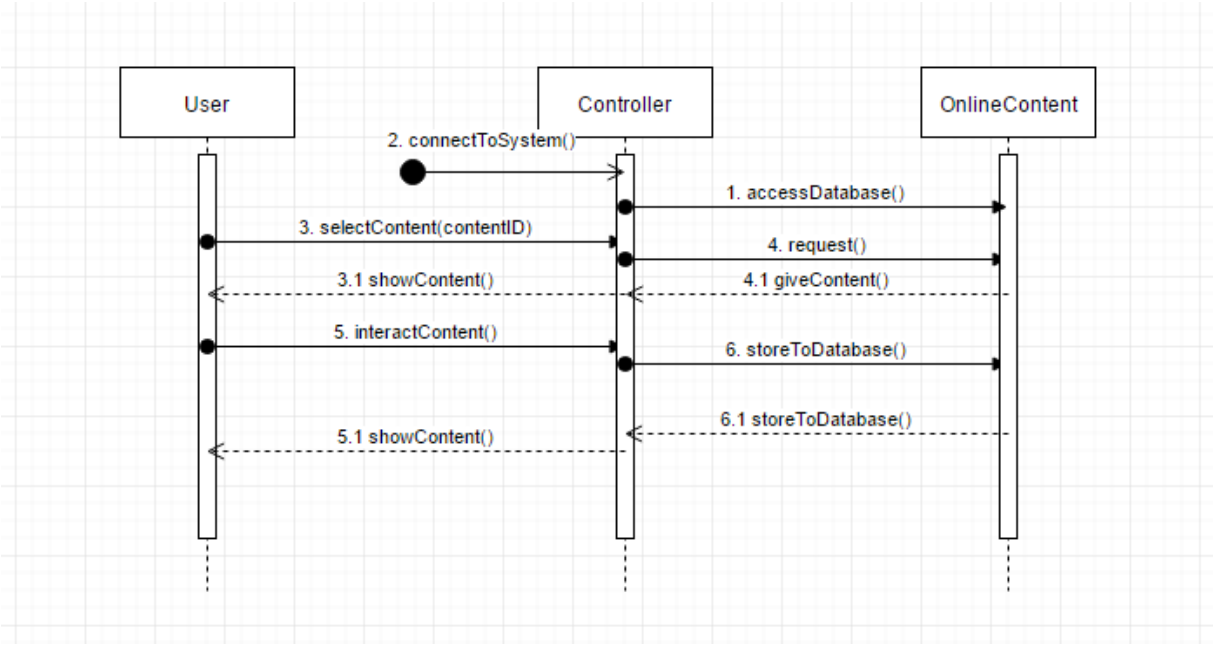


Figure 6-39 Combined DSD for UC3

6.4 Use Case 4 - Interact with Content Realization

6.4.1 UC4 Operation 1

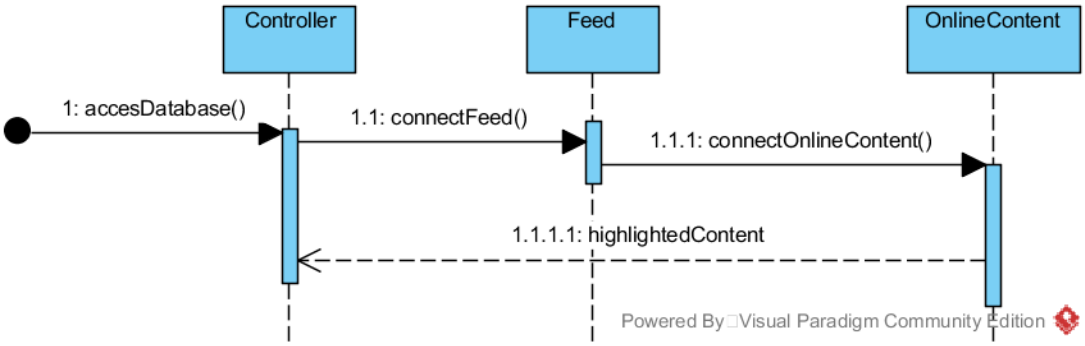


Figure 6-40 UC4 Sequence Diagram for Operation 1

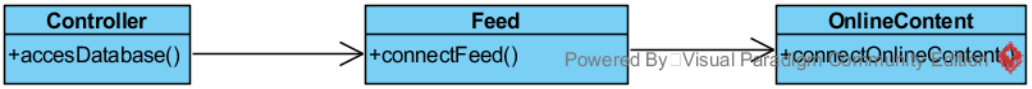


Figure 6-41 UC4 Class Diagram Operation 1

Table 6-17 UC4 GRASP Operation 1

Creator	- None
Information Expert	- OnlineContent knows about highlightedContent
Low Coupling	- Responsibilities are delegated so that the dependencies remain low
High Cohesion	- User has only one responsibility, connectFeed() - Feed has only one responsibility, connectOnline()
Controller	- Controller represents a handler of all system events, used by user

6.4.2 UC4 Operation 2

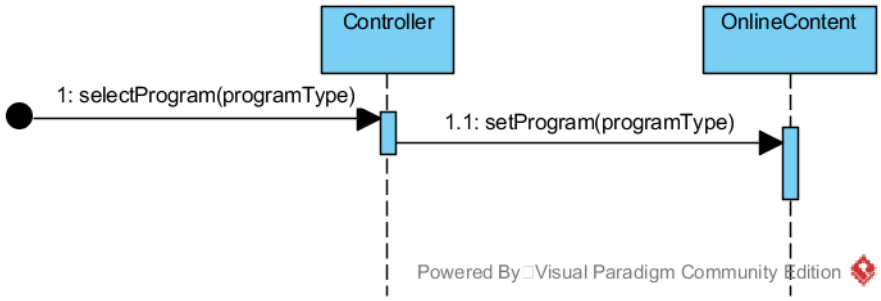


Figure 6-42 UC4 Sequence Diagram for Operation 2

Figure 6-43 UC4 Class Diagram Operation 2

Table 6-18 UC4 GRASP Operation 2

Creator	- None
Information Expert	- OnlineContent knows about the programtypes
Low Coupling	- Responsibilities are delegated so that the dependencies remain low
High Cohesion	- Controller has only one responsibility to selectProgram()
Controller	- Controller represents a handler of all system events, used by user

6.4.3 UC4 Operation 3

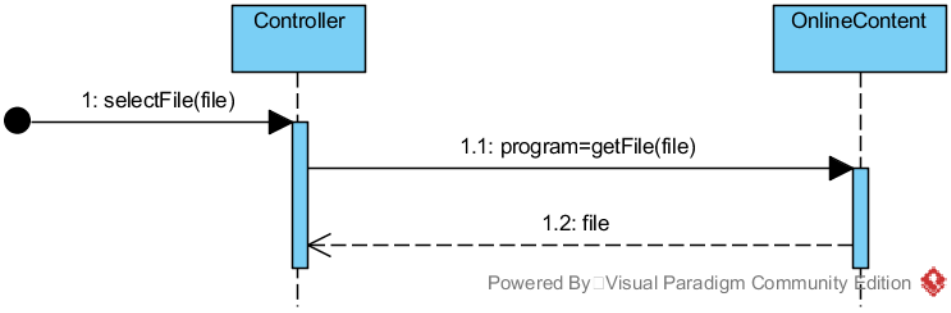


Figure 6-44 UC4 Sequence Diagram for Operation 3

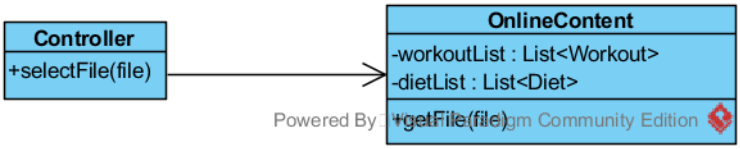


Figure 6-45 UC4 Class Diagram Operation 4

Table 6-19 UC4 GRASP Operation 3

Creator	- None
Information Expert	- OnlineContent knows about the file
Low Coupling	- None
High Cohesion	- None
Controller	- Controller represents a handler of all system events, used by user

6.4.4 UC4 Operation 4

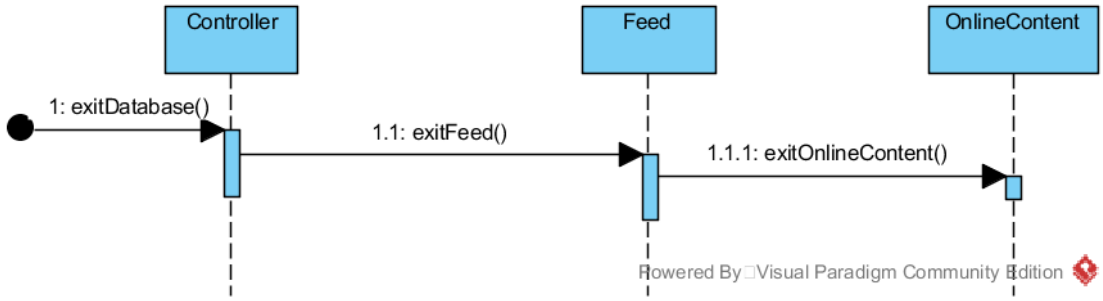


Figure 6-46 UC4 Sequence Diagram for Operation 5

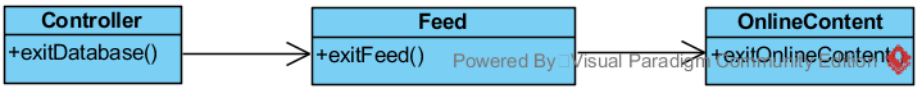


Figure 6-47 UC4 Class Diagram Operation 4

Table 6-20 UC4 GRASP Operation 4

Creator	- None
Information Expert	- None
Low Coupling	- Responsibilities are delegated so that the dependencies remain low
High Cohesion	- None
Controller	- Controller represents a handler of all system events, used by user

6.4.5 Combined Design Class Diagram for Use Case 4

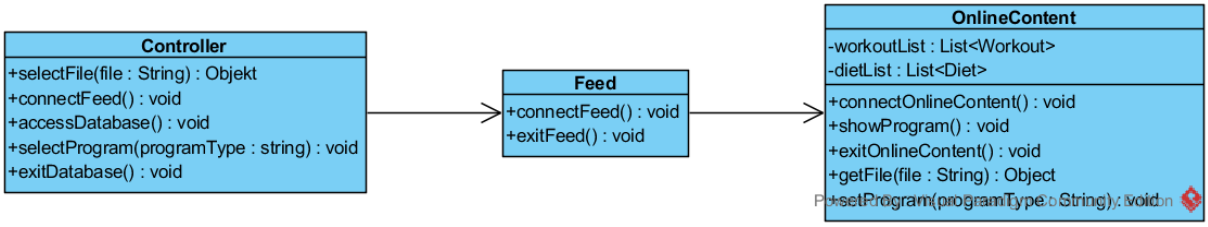


Figure 6-48 Combined DCD for UC4

6.4.6 Combined Design Sequence Diagram for Use Case 4

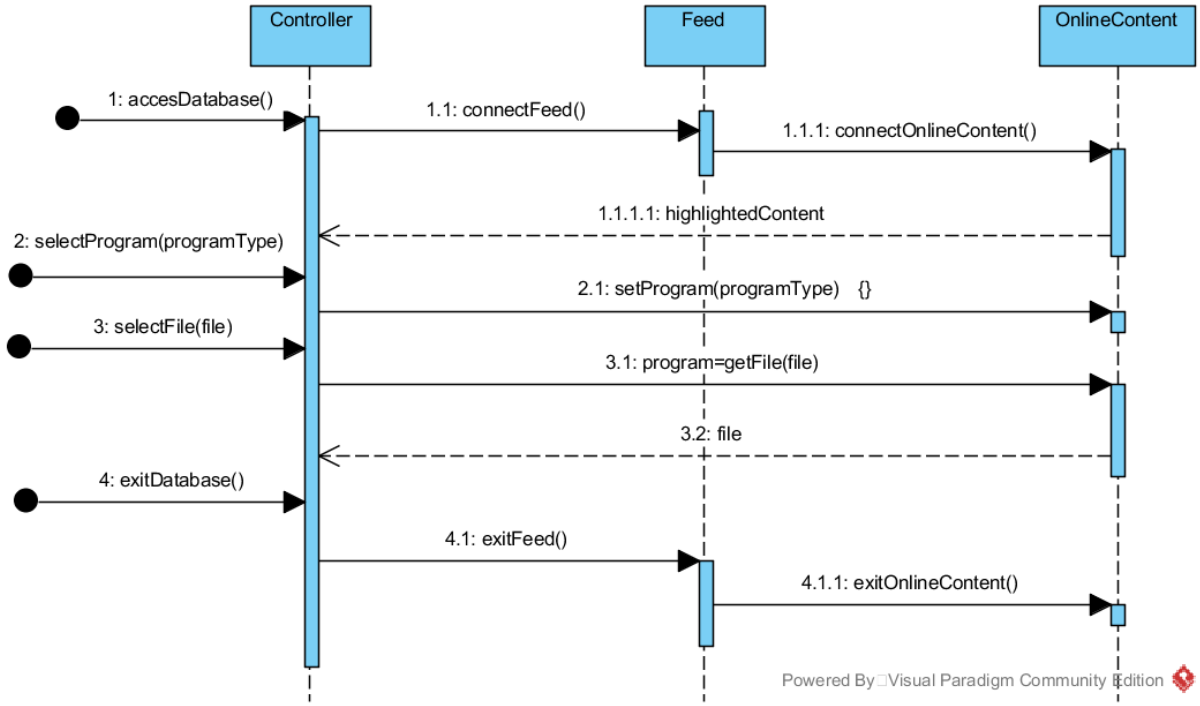


Figure 6-49 Combined DSD for UC4

6.5 Use Case 5 – Manage users

6.5.1 UC5 Operation 1

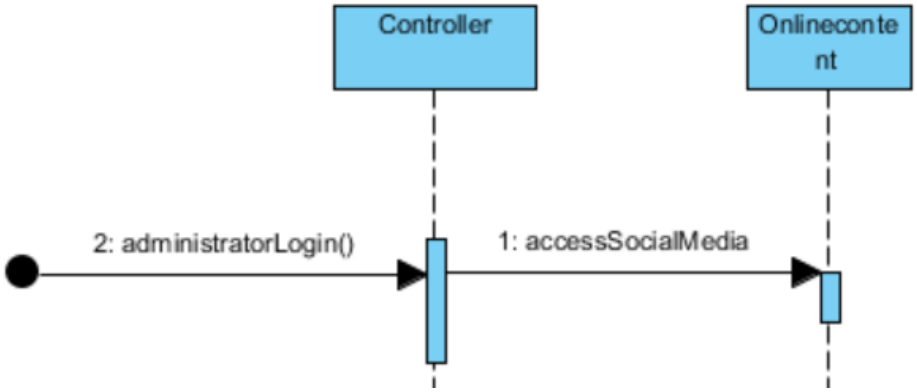


Figure 6-43 UC5 Sequence Diagram for Operation 1

Social media administrator logs into system and get access to social media content that he is supervising. Database recognize social media administrator's account.

Figure 6-44 UC5 Class Diagram Operation 1

Table 6-22 UC5 GRASP Operation 1

Creator	- None
Information Expert	- OnlineContent knows administrator account information
Low Coupling	- Responsibilities are delegated so that the dependencies remain low
High Cohesion	- None
Controller	- None

6.5.2 UC5 Operation 2

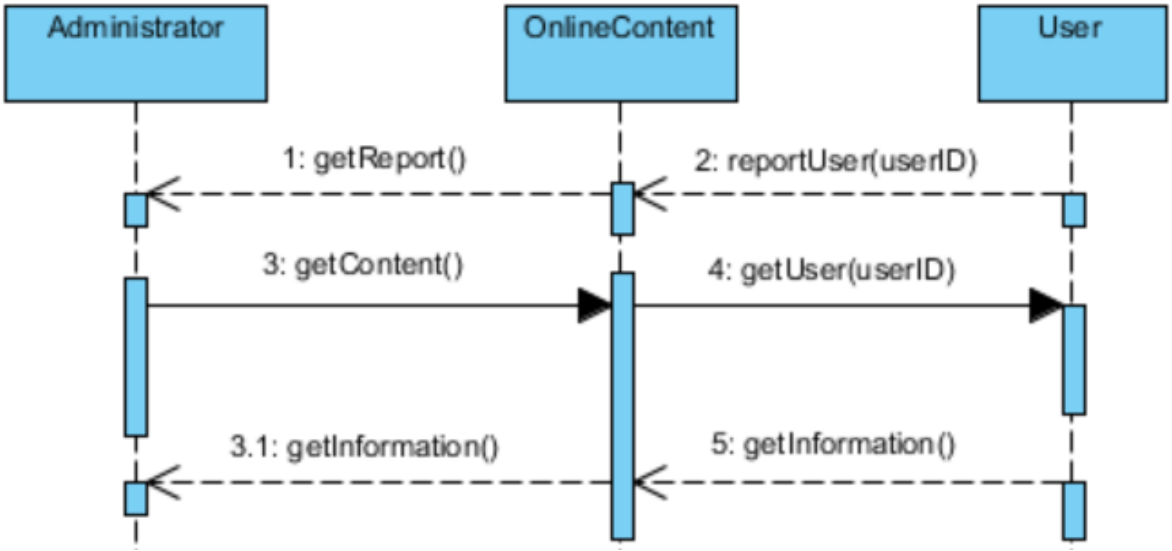


Figure 6-44 UC5 Sequence Diagram for Operation 2

Social media administrator gets notifications about reported users. Social media administrator checks the users account and receive information about these users from database.

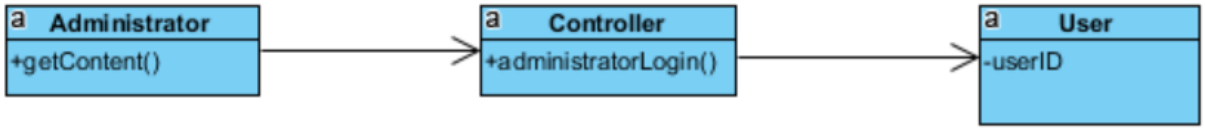


Figure 6-45 UC5 Class Diagram Operation 2

Table 6-23 UC5 GRASP Operation 2

Creator	- Social media administrator
Information Expert	- Database knows social media administrator
Low Coupling	- Social media administrator does not need to know how to get users information or reports
High Cohesion	- Social media administrator has single responsibility to select user
Controller	- Social media administrator

6.5.3 UC5 Operation 3

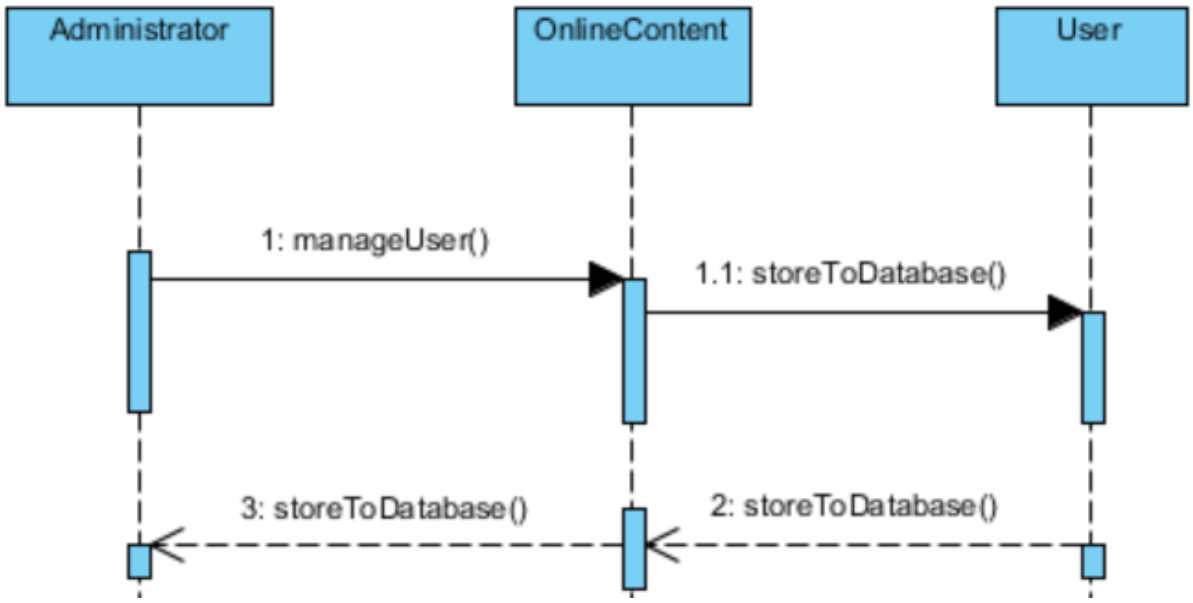


Figure 6-45 UC5 Sequence Diagram for Operation 3

Social media administrator manages users. For example, social media administrator can delete reported user's account for misbehaving. Information about this will be stored to database.

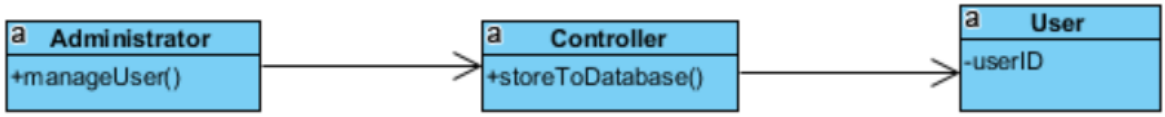


Figure 6-46 UC5 Class Diagram Operation 3

Table 6-24 UC5 GRASP Operation 3

Creator	- Social media administrator
Information Expert	- Database knows social media administrator
Low Coupling	- Social media administrator does not need to know how to store information to database
High Cohesion	- Social media administrator has single responsibility to manage user
Controller	- Social media administrator

6.5.4 Combined Design Class Diagram for Use Case 5

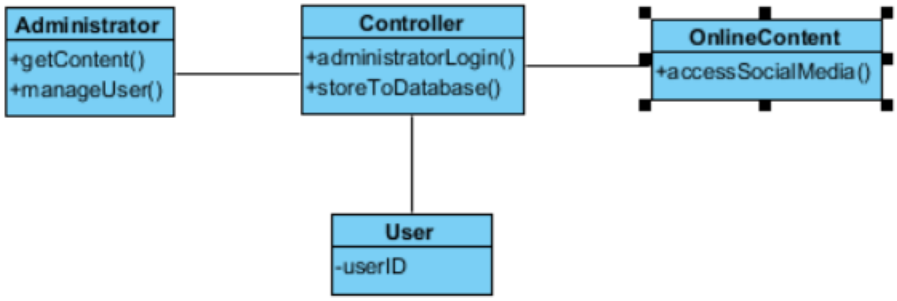


Figure 6-50 Combined DCD for UC5

6.5.5 Combined Design Sequence Diagram for Use Case 5

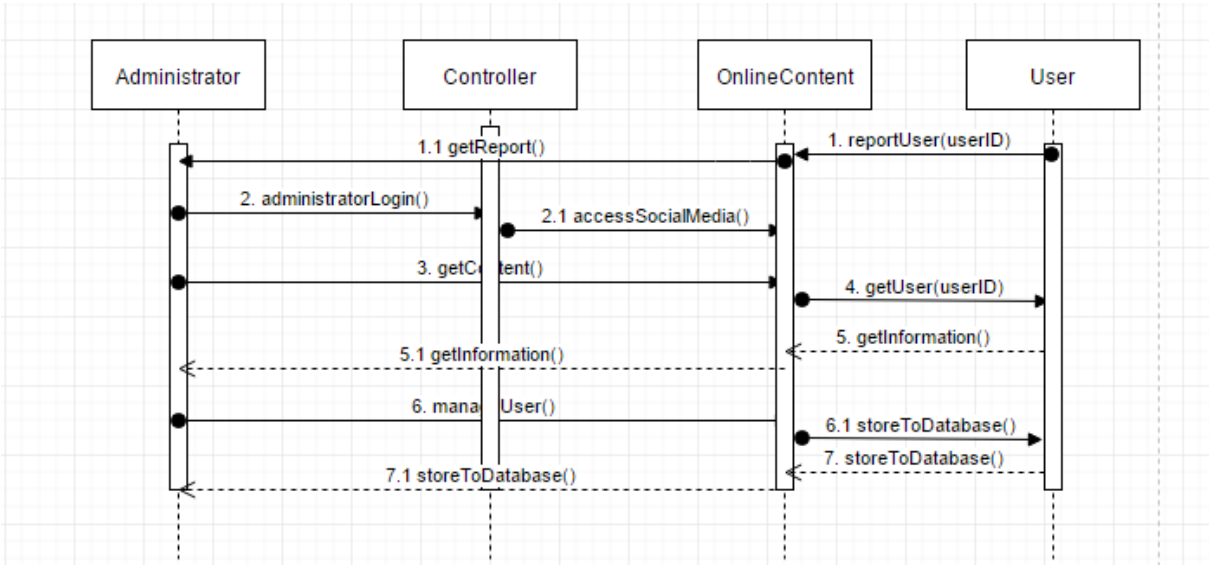


Figure 6-51 Combined DSD for UC5

6.6 Design Class Diagram of the System

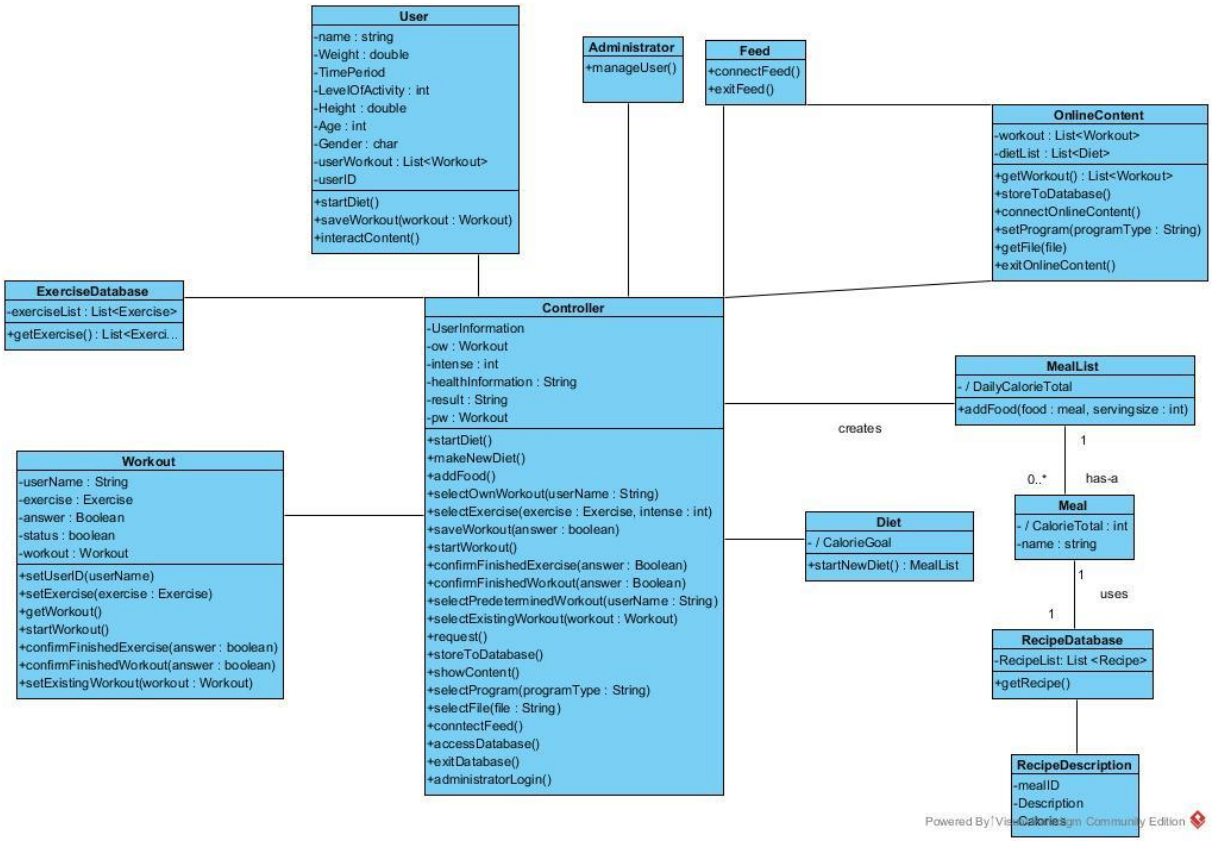


Figure 6-52 DCD of the system

7 Architecture

7.1 Introduction

This SAD summarizes the architecture from multiple views. These include logical, process, deployment, data, and use case views. Each of these views are chosen to explain why we chose to construct our system in this way from multiple perspectives. Also they will allow readers to quickly understand the major functions of our system.

In this report we chose to exclude process view because we decided that our system in its cur-rent form does not require complicated dynamic processes.

7.2 Architectural Decisions (Technical Memos)

Factor	Measures and quality scenarios	Variability (current flexibility and future evolution)	Impact of factor (and its variability) on stakeholders, architecture and other factors	Priority for Success	Difficulty or Risk
Reliability – Recoverability					
Recovery from remote service failure	When a remote service fails, user should reconnect within 1 minute of its detected non-availability.	Current flexibility – User has to manually reconnect to the remote service. Evolution – the remote service will reconnect itself. So the user has more convenience when errors occur.	High impact on the large-scale design. User really dislikes it when remote services fail, as it prevents them from using e.g. the online content system.	H	M
Recovery from remote Own exercises/diets database failure	As above	Current flexibility – Local user-side use of cached “most recent” exercise/diet info is acceptable (and desirable) until reconnection is possible. Evolution – within 3 years, user-side mass storage and replication solutions will be cheap and effective, allowing permanent complete replication and thus local usage.	As above	H	M
Supportability – Adaptability					
Support many third-party services (gyroscope, pedometer, heart beat	If there are new beneficial third-party-services there will be a noti-	Current flexibility – as described by factor. Evolution possibility - We will extend our system: Multilanguage support, add more features on diet	Optional for user. Therefore small impact on design.	M	L

sensor).	fication for the user within short time period	system.			

Technical Memo

Issue: Reliability – Recovery from Remote Service Failure

Solution Summary: failover from remote to local and local service by user input.

Factors

- Robust recovery from remote service failure (e.g. Calorie calculator).
- Robust recovery from remote database failure (e.g. Current exercises and diets)

Solution

Where possible, offer local implementations of remote services, usually with simplified or constrained behavior. For example, the current exercises and/or diet database will be a small cache of the most recently started or ongoing ones. The local product stored and forwarded at reconnection.

Motivation

Fitness application users want to rely on their application as much as possible, they want to be able to exercise using Aegle without having sudden technical issues. Therefore, if the Aegle offers this level of reliability and recovery, it will be a very attractive product. The design also supports the evolution point of future users willing and able to continue their exercise without having to stop because of temporary system failure of the application.

Unresolved Issues

none

Alternatives Considered

None

Technical Memo

Issue: Adaptability – Third-Party Services

Solution Summary: Protected Variation using interfaces and Adapters

Factors

- Support many, changeable third-party services (gyroscope, pedometer, heart beat sensor)

Solution

Keep Aegle up-to-date with newest third-party services. Check possible new implementations of third-party services and use them if they are beneficial for the user.

Motivation

Keeping third-party services up-to-date provide high safety for the user. The probability of data loss remains low and offers the user a convenient usage of the app. Also with recent updates user are attracted and motivated in using this app because it distinguishes itself from other apps with its unique features.

Unresolved Issues

none

Alternatives Considered

Fix one service rather than many services. But this way is reducing our adaptability.

7.3 Logical View

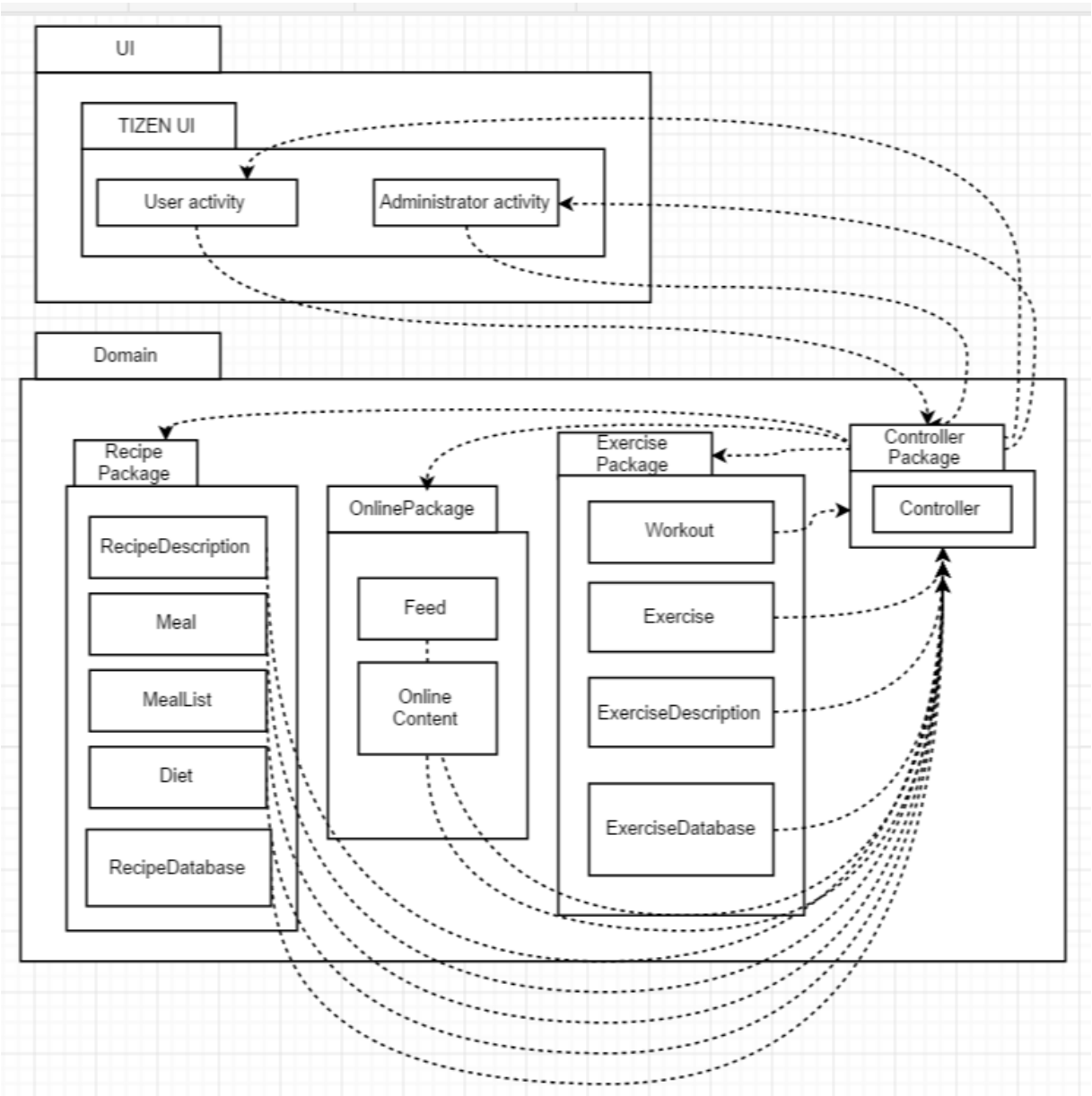


Figure 7-1 Logical View for the system

7.4 Process view

Because the application doesn't have more than one process at the same time, the process view is inapplicable.

7.5 Deployment view

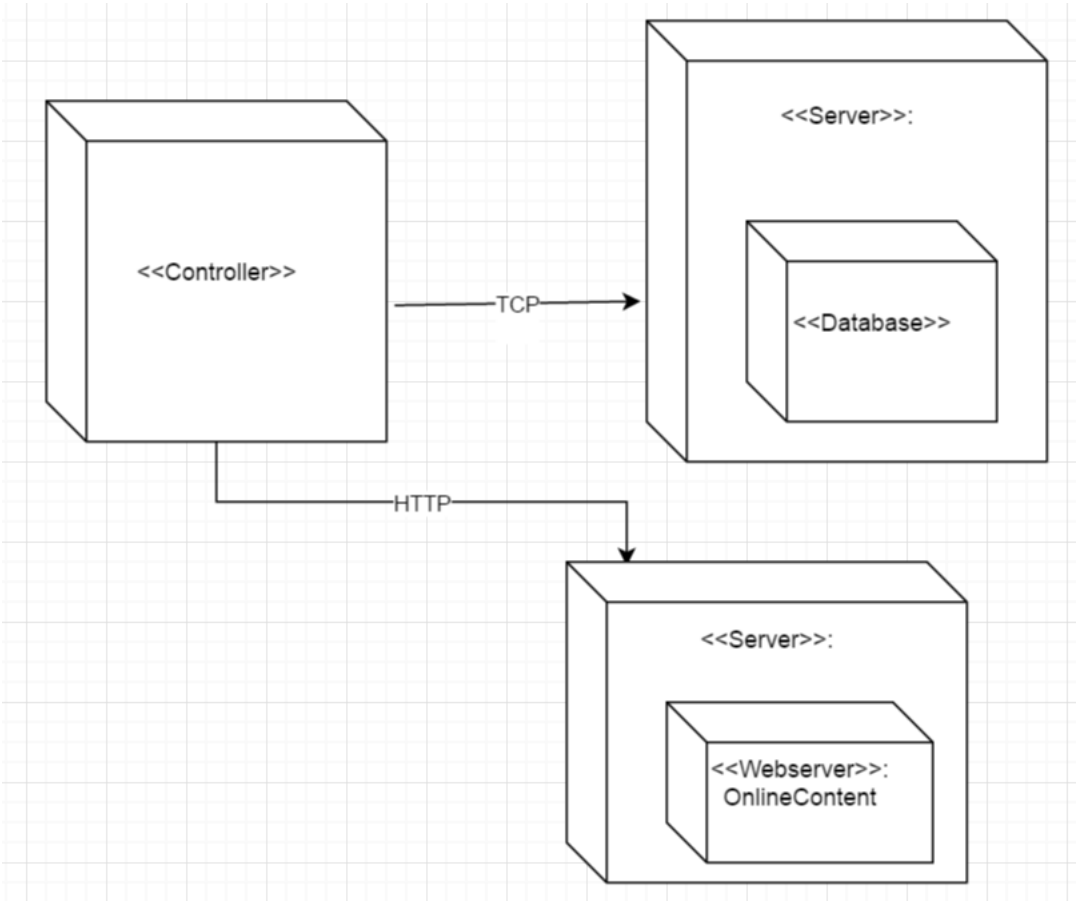


Figure 7-2 Development viewe of then system

7.6 Data view

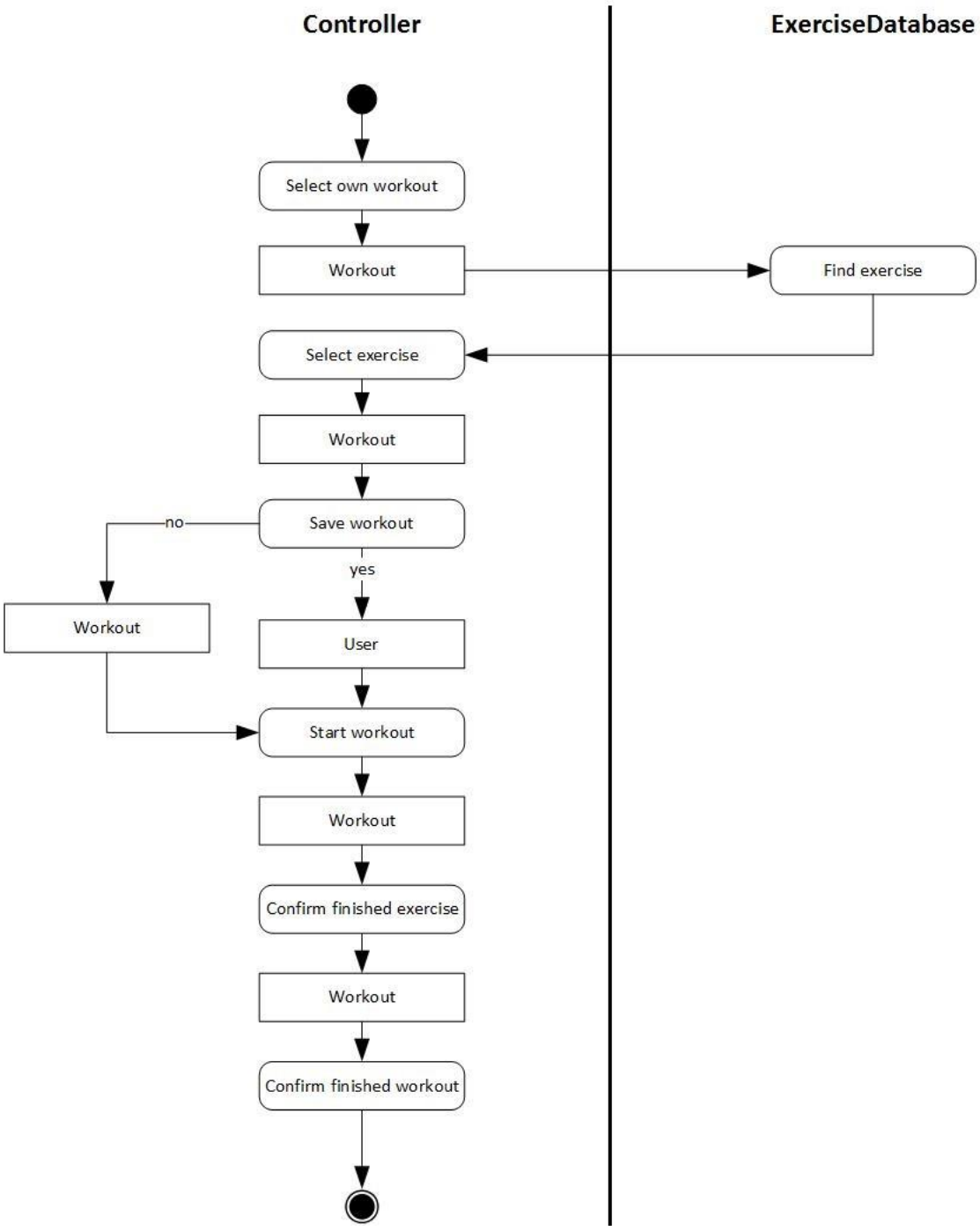


Figure 7-3 Data View of UC1 Main1

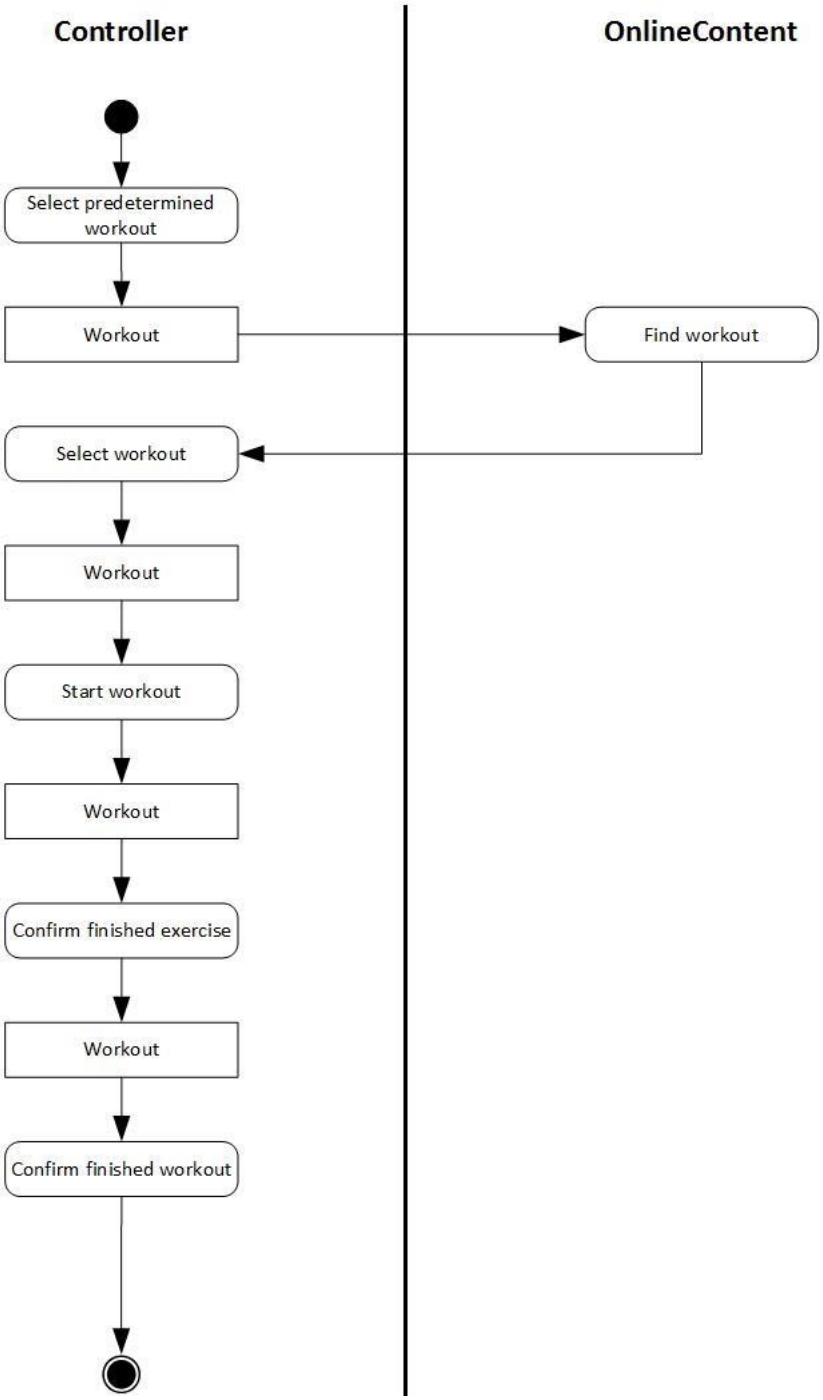


Figure 7-4 Data View of UC1 Main 2

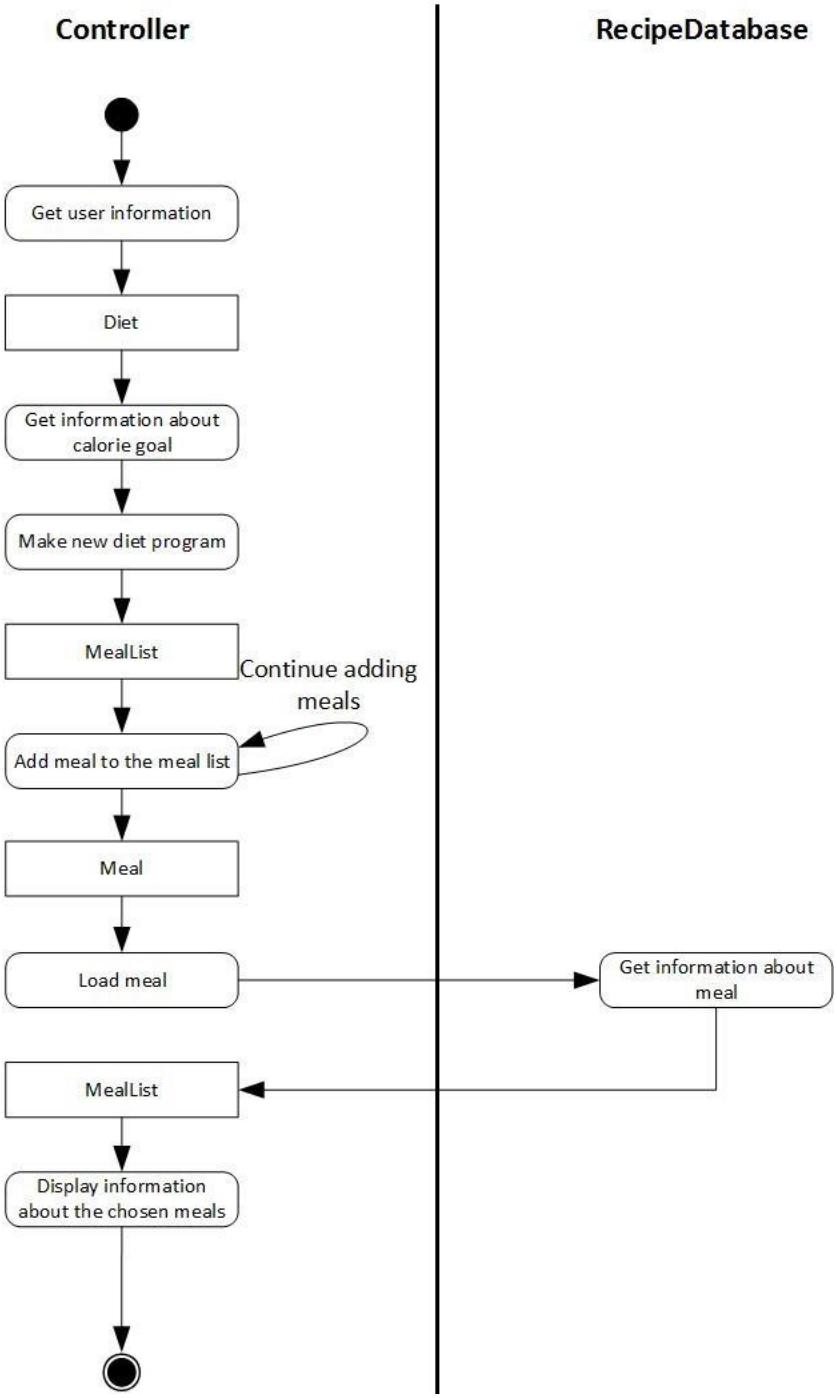


Figure 7-5 Data View of UC2

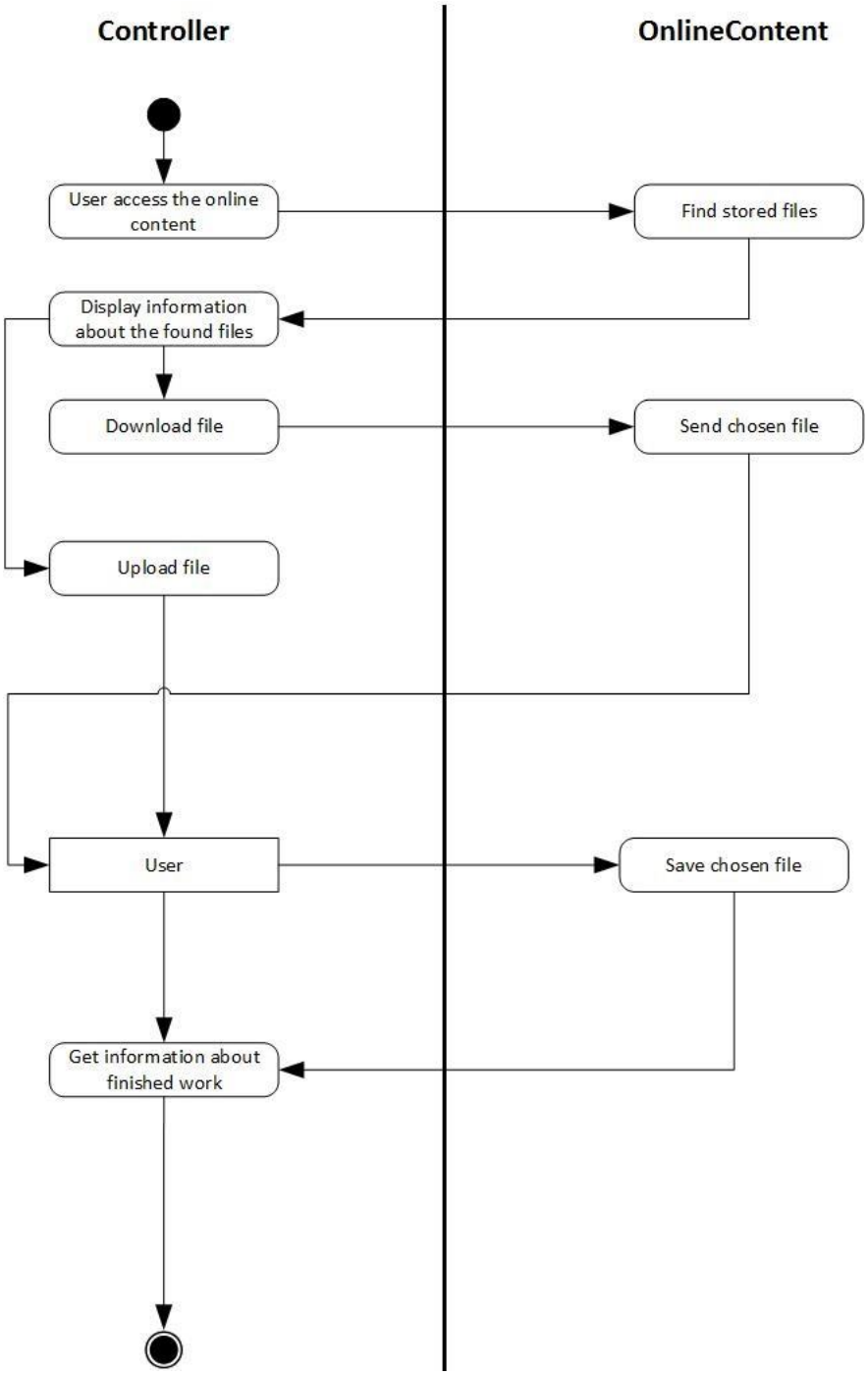


Figure 7-6 Data View of UC3

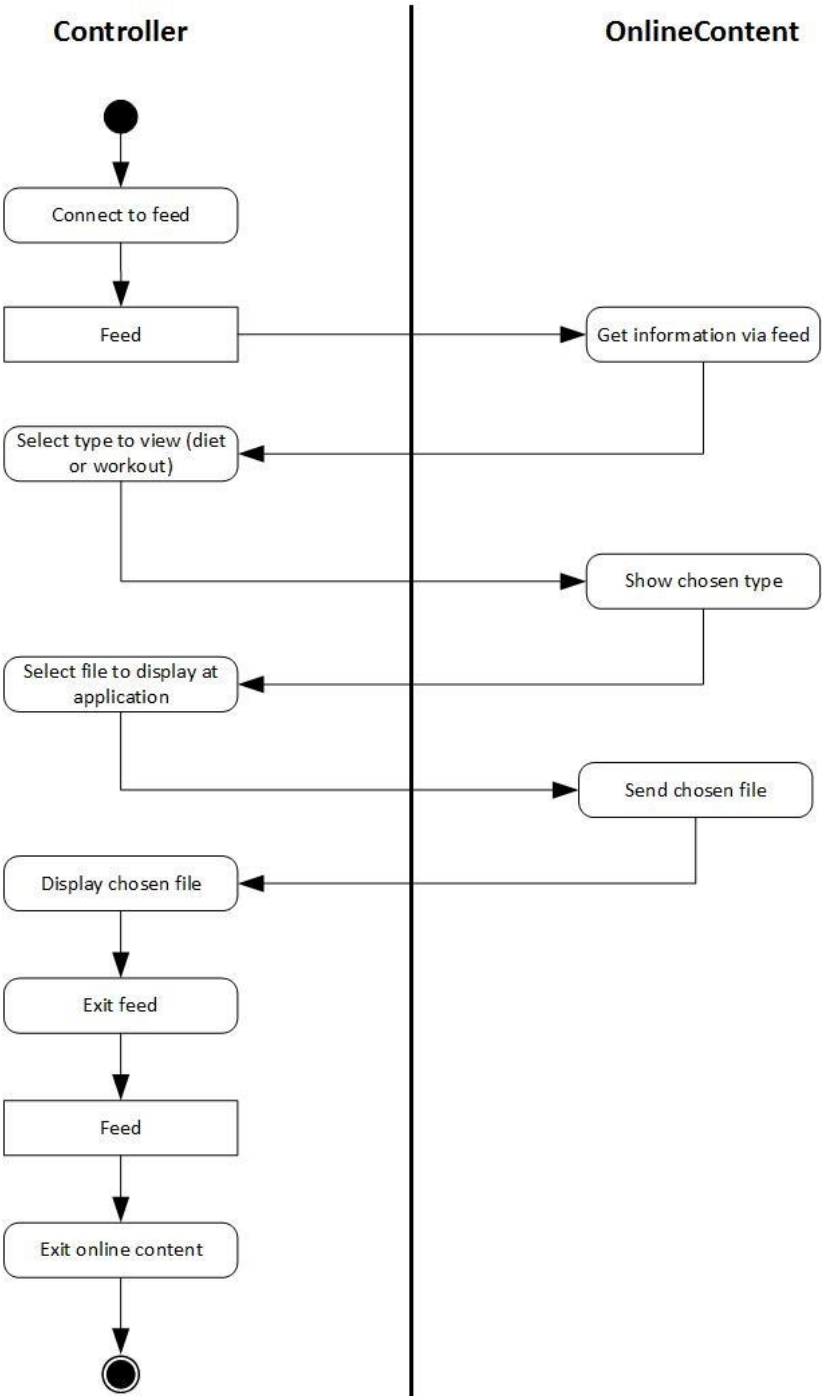


Figure 7-7 Data View of UC4

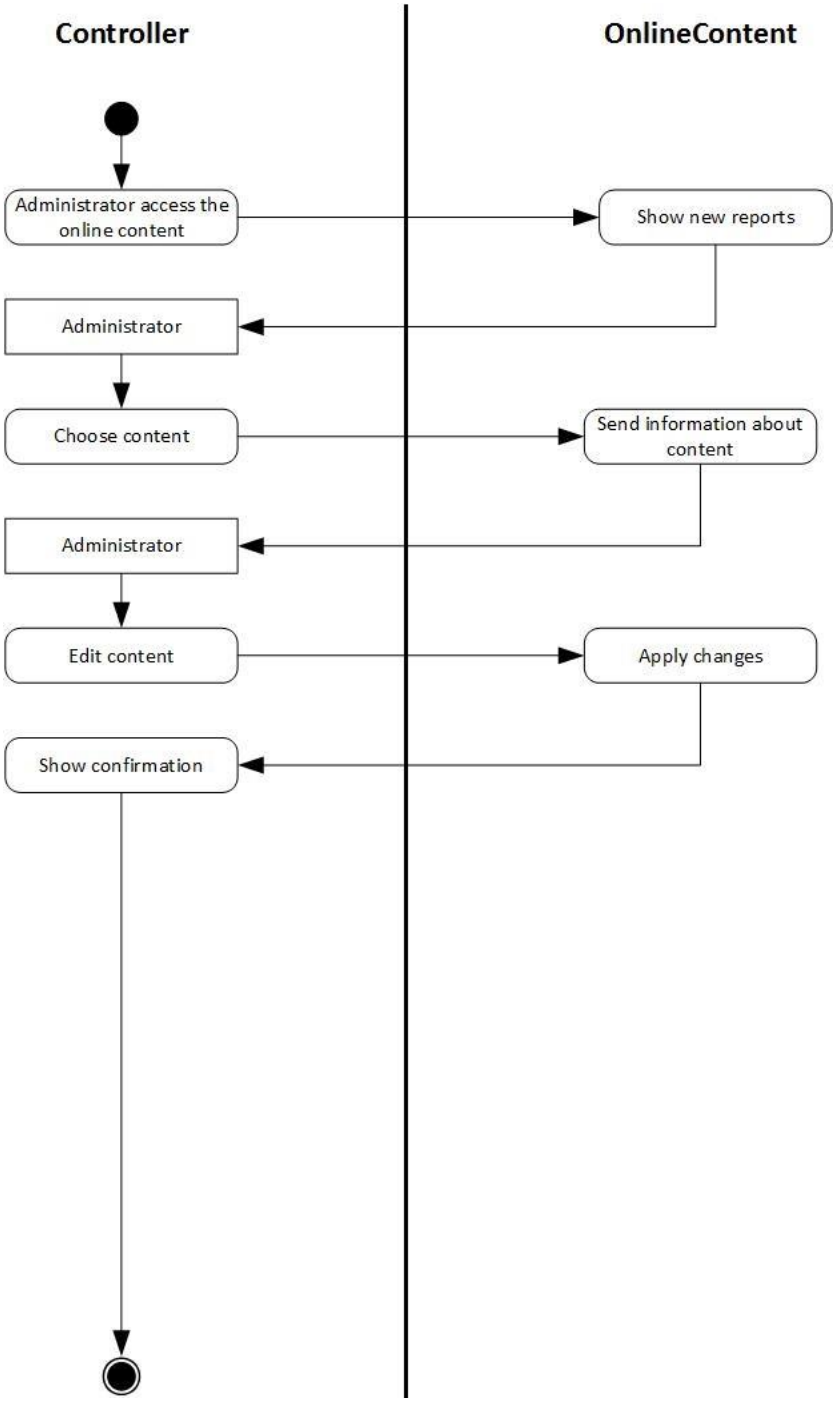


Figure 7-8 Data View of UC5

7.7 Use case view

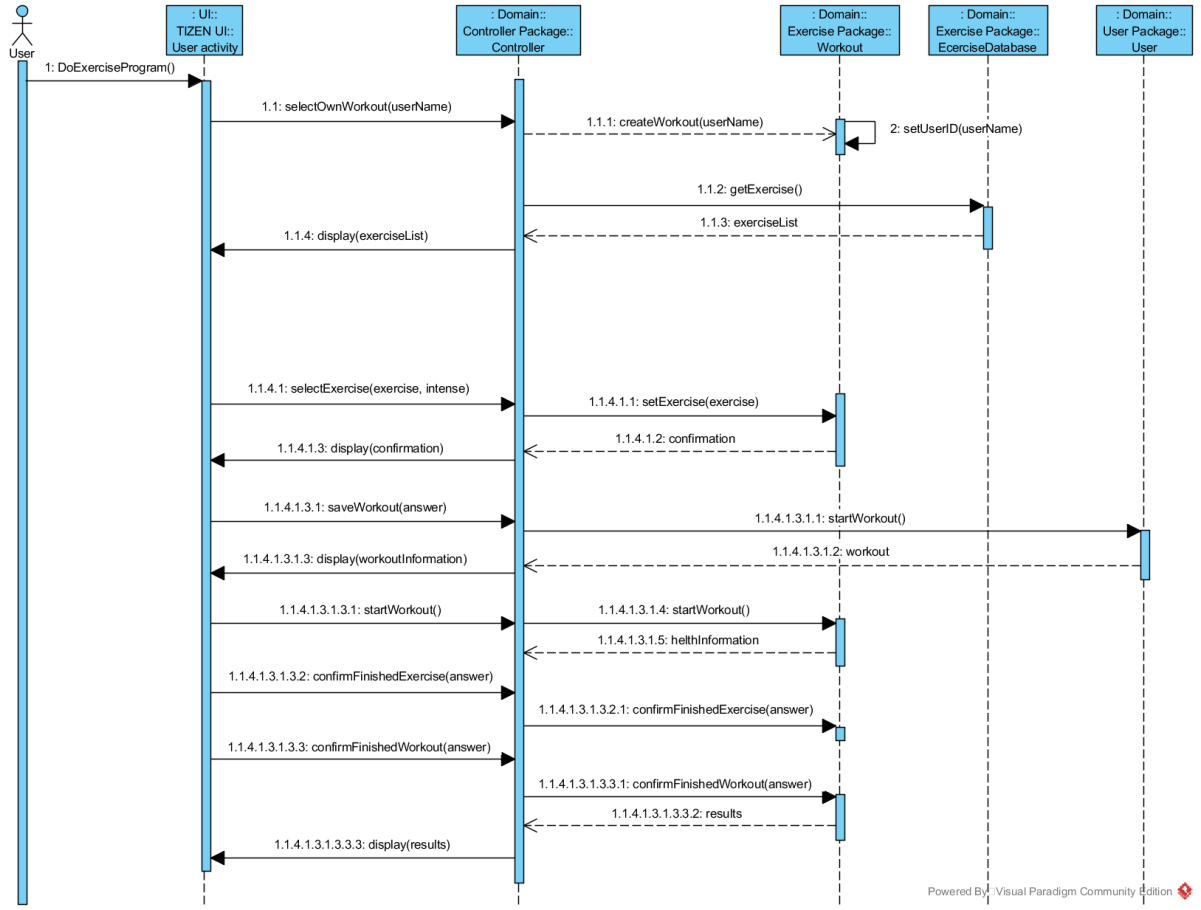


Figure 7-9 Use Case View UC1 Main1

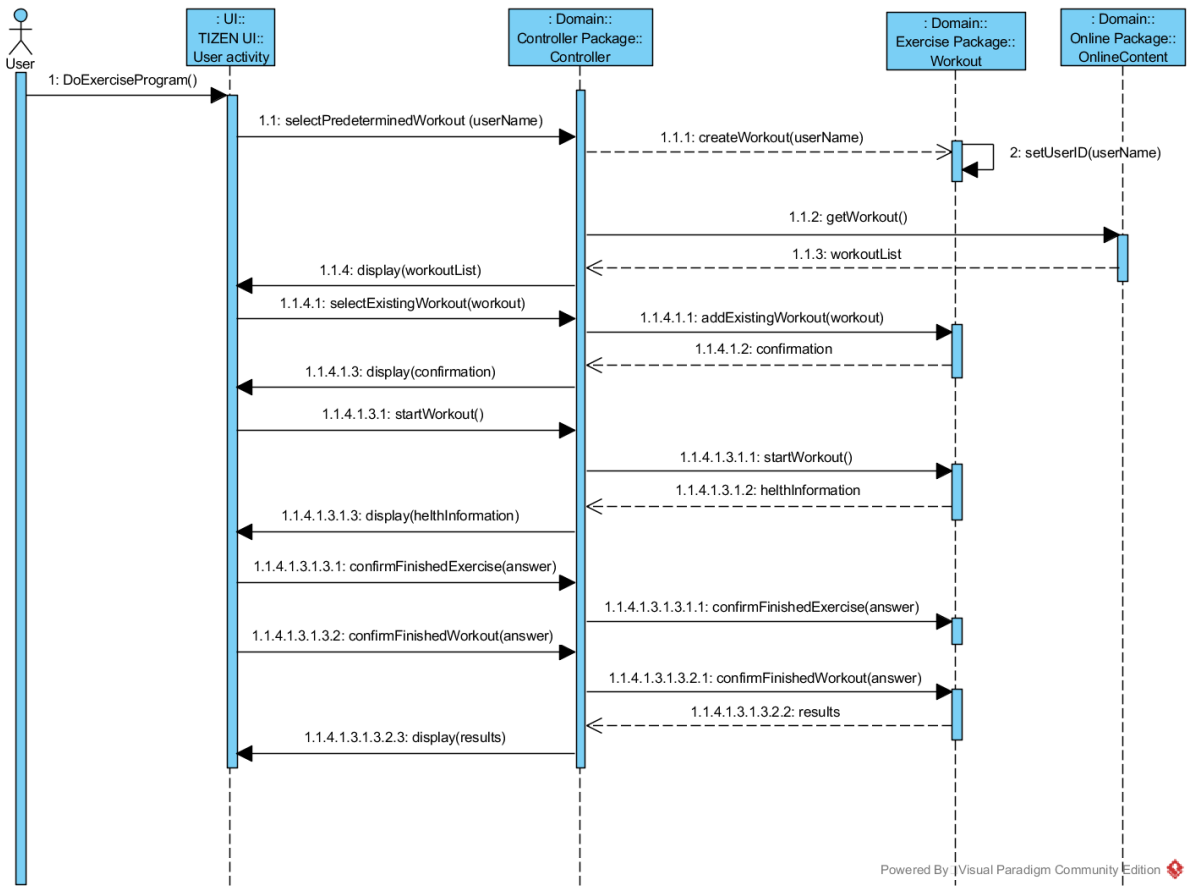


Figure 7-10 Use Case View UC1 Main 2

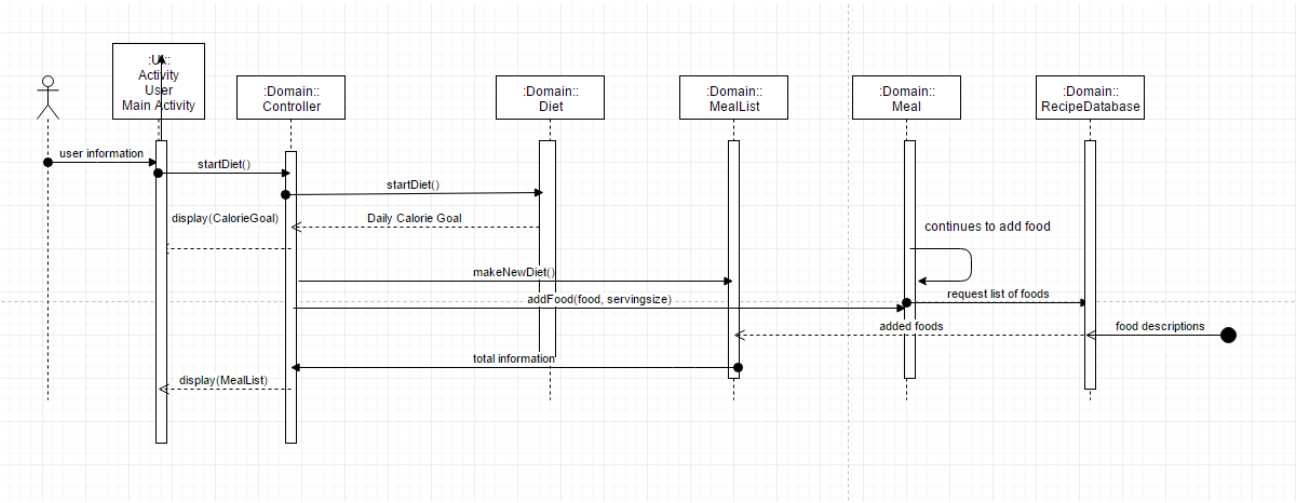


Figure 7-11 Use Case View UC2

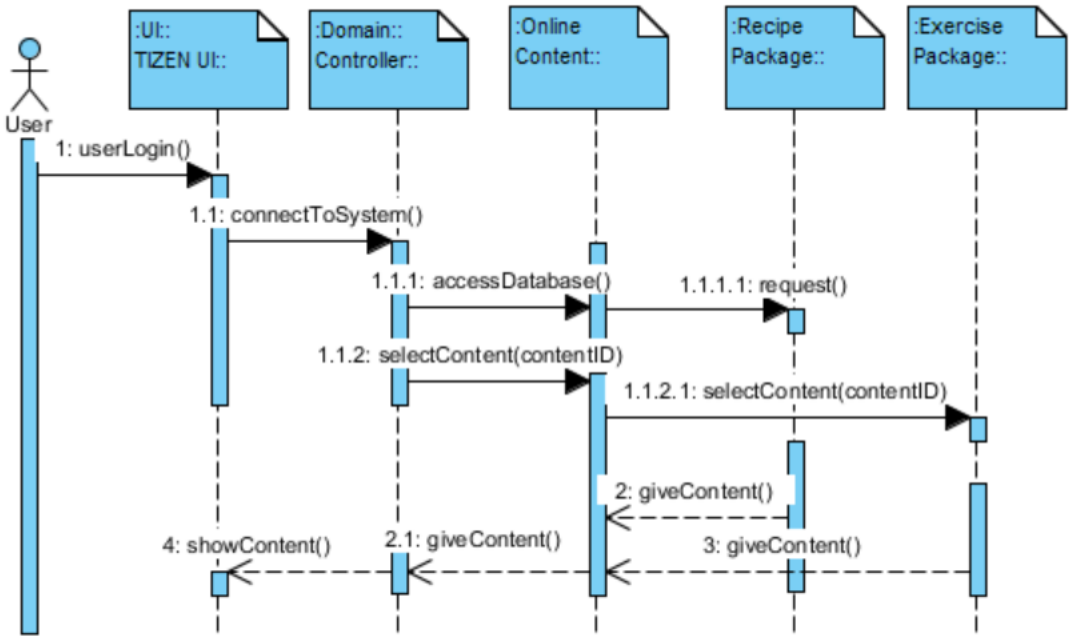


Figure 7-12 Use Case View UC3

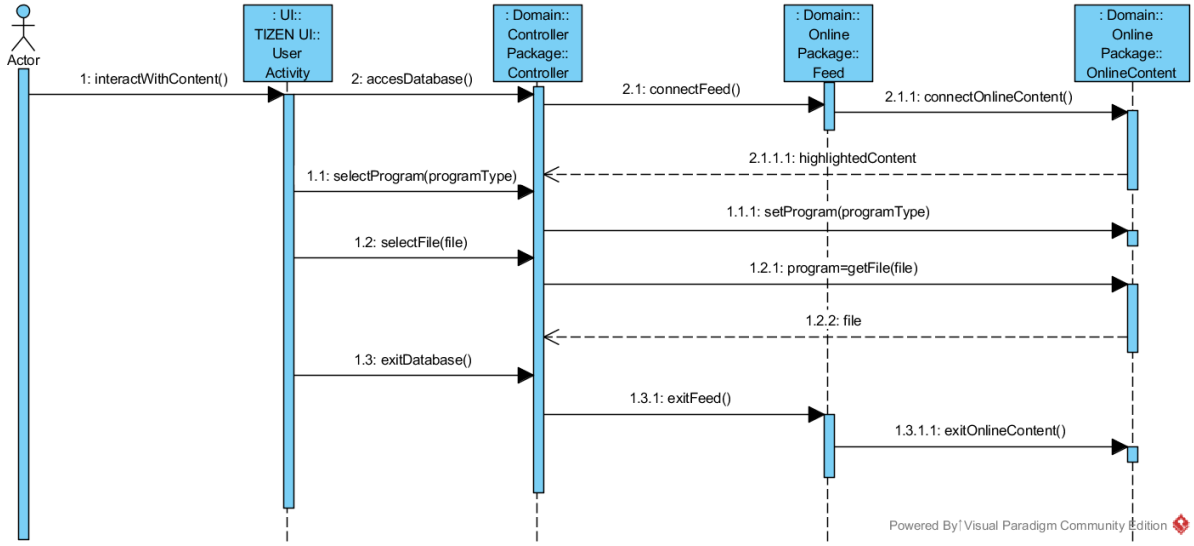


Figure 7-13 Use Case View UC4

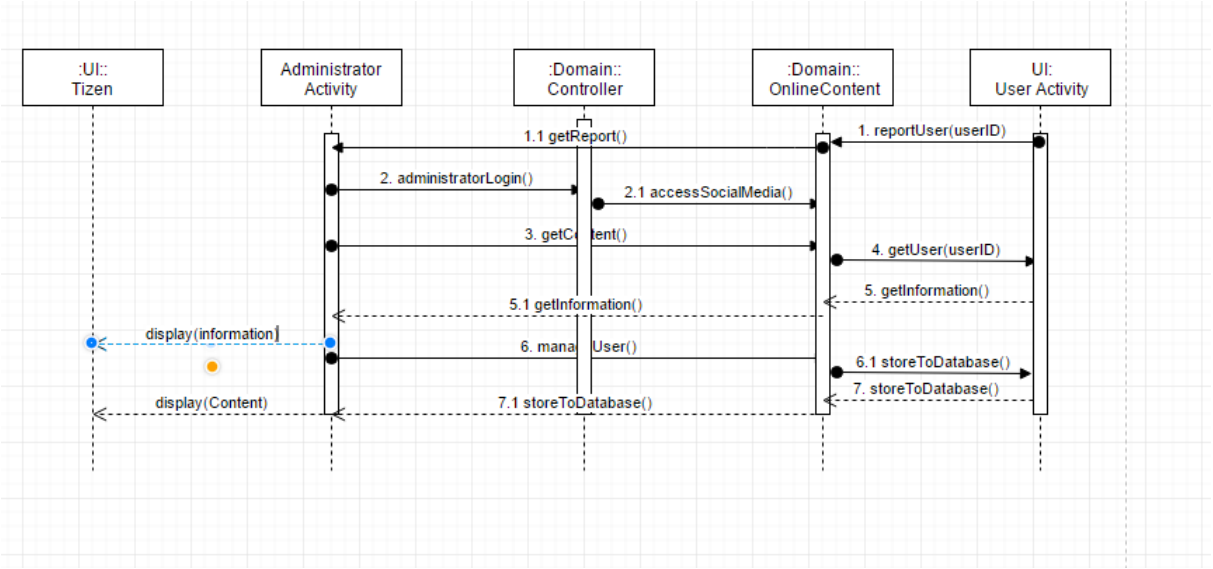


Figure 7-14 Use Case View UC5

8 Conclusion

8.1 Vision

Aegle is next generation social, all-in-one smart fitness companion application. Aegle is designed to have flexibility to be useful for anyone interested in fitness, containing multiple user interface mechanisms supported by variety of devices intuitively. The system will provide convenient way for users to start and maintain a regular exercise program, along with a proper diet.

Aegle is also integrated with its own social media platform. Users will be able to share their personal achievements and milestones, along with the ability to fit in with others in your own niche of the healthy community. The greatest advantage of the social media platform is the ability for users to share their workout programs and recipes. Other users can try them out and leave reviews. The most popular workouts and recipes will then be showcased and recommended to users. Doing this will ensure that Aegle will always have the best information available to our users.

8.2 Objectives

The goal behind Aegle is to give a complete beginner the ability to step right in and begin their journey to a healthier lifestyle the moment they download the application. Aegle is all-in-one fitness application that helps users to start and maintain healthy lifestyle. Including variety of exercises and diets along with Aegle's own unique social media platform. The community of Aegle's social media platform provides accurate information, reviews and discussions about the content.

8.3 Summarize

8.3.1 Design

We have accomplished 90% of our design. Some parts were left out of the report in purpose. Reason for that is lack of understanding subjects deeply enough to make it in time.

8.3.2 Implementation

We have accomplished 30% of our implementation.

8.4 Additional work

If there is additional work required, we would be focusing on refining

8.5 Lessons learnt through this project

We have learned about the actual process of inception and elaboration while working this project. We were able to analyze our system in the perspective of the user and administrator through finding requirements. Working on this project has given us first-hand experience on how to apply GRASP patterns and design a software. On the whole, this project gave us good opportunity to study and practice about object oriented analysis and design, how to make various UML diagrams and apply them to our project.

9 References

- Larman, Craig. Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and Iterative Development. Upper Saddle River, NJ: Prentice Hall PTR, 2010. Print.

- MD Mifflin, ST St Jeor, et al. A new predictive equation for resting energy expenditure in healthy individuals. J Am Diet Assoc 2005:51:241-247.

- Patrick, Megan Lane. "What Does 100 Calories Look Like?" SparkPeople. N.p., n.d. Web. 05 June 2016.

- Smit, Ellen. "News and Research Communications." U.S. Adults Get failing Grade in Healthy Lifestyle Behavior. Oregon State University, 21 Mar. 2016. Web. 23 Mar. 2016. <http://oregonstate.edu/ua/ncs/archives/2016/mar/us-adults-get-failing-grade-healthy-lifestyle-behavior>.

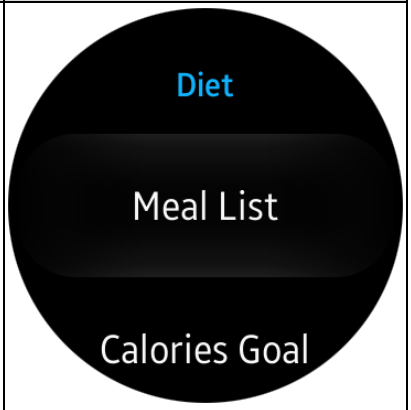
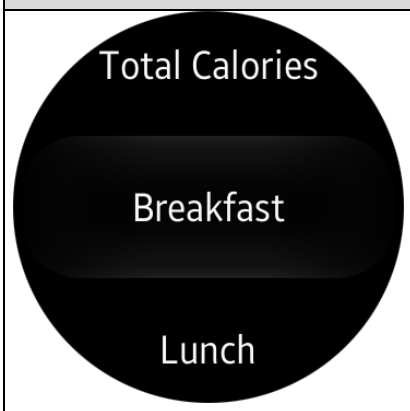
10 Appendix

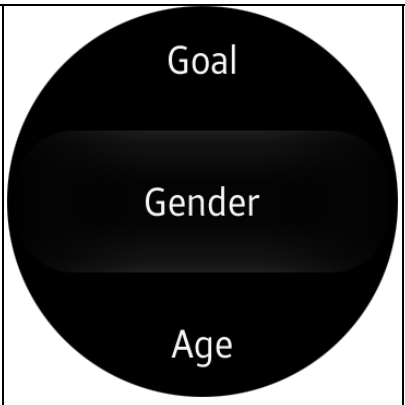
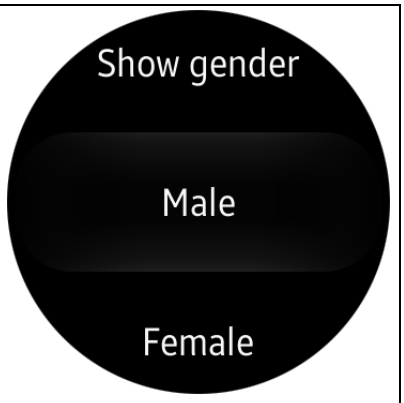
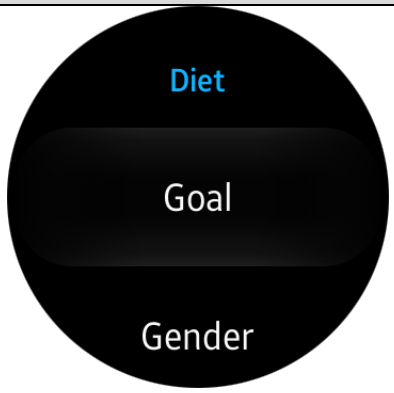
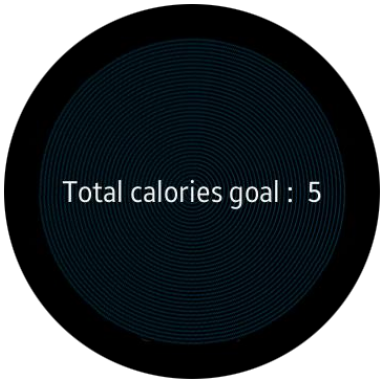
10.1 Glossary

Term	Definition and Information
Aegle	Name of the software system. Aegle is the name of the Greek Goddess of Radiance and Good Health
Social Media	Aegle will contain an integrated social media service designed to be used for sharing content such as exercise programs, meal plans, and recipes.
(General) User	Normal user of the application
Certified User	A verified health expert who holds greater weight when sharing content
Administrator	Administrator who manages user accounts. Moderates the community. Verifies certified users.
Exercise Program/Routine	A set exercise routine meant to be followed regularly.
Diet Program/Meal Plan	A meal plan meant to be followed regularly. Is highly specific mapping out nutrition and calorie content of each individual meal.
User Base	The total of the users who make up the community. The users are the ones who primarily create and share content.
Content	The information uploaded, downloaded, and reviewed by users such as exercise programs, diet programs, recipes, general advice.
Database	System that holds all content
Recipe	Either an individual food item such as banana or a more complicated meal such as a sandwich.
Meal List	A menu that displays all the food items and recipes for the day organized by each meal.
UC	Short form of Use Case.
Workout	Includes a list of exercises for the user.
ExerciseDatabase	Holds the exercises
Exercise	A description how to move the body to get fit.
GRASP	Consist of guidelines for assigning responsibility to classes and objects

RecipeDatabase	Holds the meals
User	Someone who uses the system
Calorie goal	Recommended calorie intake based on the Miffin St Jeor equation

10.2 Live Demo

Step 1	Step 2	Step 3
		
Default menu	Default menu	**Diet menu**
Step 4	**Step 5**	**Step 6**
		
Meal list menu	Meal menu	Food added
Step 7	**Step 8**	**Step 9**

		
Total calories	Calorie goal menu	Gender menu
Step 10	**Step 11**	**Step 12**
		
Show gender	Calorie goal menu	Calorie goal

10.3 Source code

10.3.1 Main

```
#include "main.h"

char *main_menu_names[] = {
	"Workout", "Diet", "Community", "Settings",/* "Genlist", "Image",
"PageControl", "Popup", "Progress",
	"Nocontents", "Radio", "Scroller",
	"(Eext) Datetime", "(Eext) Genlist", "(Eext) More Option",
	"(Eext) ProgressBar", "(Eext) Rotary Selector", "(Eext) Scroller",
"(Eext) Slider", "(Eext) Spinner",*/
	NULL
};

typedef struct _item_data
{
	int index;
	Elm_Object_Item *item;
} item_data;

static void
win_delete_request_cb(void *data, Evas_Object *obj, void *event_info)
{
	/* To make your application go to background,
		Call the elm_win_lower() instead
		Evas_Object *win = (Evas_Object *) data;
		elm_win_lower(win); */
	ui_app_exit();
}

static void
gl_selected_cb(void *data, Evas_Object *obj, void *event_info)
{
	Elm_Object_Item *it = (Elm_Object_Item *)event_info;
	elm_genlist_item_selected_set(it, EINA_FALSE);
}

static char *
_gl_menu_title_text_get(void *data, Evas_Object *obj, const char *part)
{
	char buf[1024];

	snprintf(buf, 1023, "%s", "Aegle");
	return strdup(buf);
}

static char *
_gl_menu_text_get(void *data, Evas_Object *obj, const char *part)
{
	char buf[1024];
	item_data *id = (item_data *)data;
	int index = id->index;

	if (!strcmp(part, "elm.text")) {
```

```
            snprintf(buf, 1023, "%s", main_menu_names[index]);
            return strdup(buf);
        }
        return NULL;
}

static void
_gl_menu_del(void *data, Evas_Object *obj)
{
        // FIXME: Unrealized callback can be called after this.
        // Accessing Item_Data can be dangerous on unrealized callback.
        item_data *id = (item_data *)data;
        if (id) free(id);
}

static Eina_Bool
naviframe_pop_cb(void *data, Elm_Object_Item *it)
{
        ui_app_exit();
        return EINA_FALSE;
}

static void
create_list_view(appdata_s *ad)
{
        Evas_Object *genlist;
        Evas_Object *circle_genlist;
        Evas_Object *btn;
        Evas_Object *nf = ad->nf;
        Elm_Object_Item *nf_it;
        Elm_Genlist_Item_Class *itc = elm_genlist_item_class_new();
        Elm_Genlist_Item_Class *ttc = elm_genlist_item_class_new();
        Elm_Genlist_Item_Class *ptc = elm_genlist_item_class_new();
        item_data *id;
        int index = 0;

        /* Genlist */
        genlist = elm_genlist_add(nf);
        elm_genlist_mode_set(genlist, ELM_LIST_COMPRESS);
        evas_object_smart_callback_add(genlist,  "selected",  gl_selected_cb,
NULL);

        circle_genlist    =    eext_circle_object_genlist_add(genlist,    ad-
>circle_surface);
        eext_circle_object_genlist_scroller_policy_set(circle_genlist,
ELM_SCROLLER_POLICY_OFF, ELM_SCROLLER_POLICY_AUTO);
        eext_rotary_object_event_activated_set(circle_genlist, EINA_TRUE);

        /* Genlist Title Item style */
        ttc->item_style = "title";
        ttc->func.text_get = _gl_menu_title_text_get;
        ttc->func.del = _gl_menu_del;

        /* Genlist Item style */
        itc->item_style = "default";
        itc->func.text_get = _gl_menu_text_get;
        itc->func.del = _gl_menu_del;

        /* Genlist Padding Item style */
```

```
	ptc->item_style = "padding";
	ptc->func.del = _gl_menu_del;

	/* Title Items Here */
	elm_genlist_item_append(genlist,	ttc,	NULL,	NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);

	/* Main Menu Items Here */
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item	=	elm_genlist_item_append(genlist,	itc,	id,	NULL,
ELM_GENLIST_ITEM_NONE, NULL, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item	=	elm_genlist_item_append(genlist,	itc,	id,	NULL,
ELM_GENLIST_ITEM_NONE, ui_cb, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item	=	elm_genlist_item_append(genlist,	itc,	id,	NULL,
ELM_GENLIST_ITEM_NONE, NULL, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item	=	elm_genlist_item_append(genlist,	itc,	id,	NULL,
ELM_GENLIST_ITEM_NONE, NULL, ad);


	elm_genlist_item_append(genlist,	ptc,	NULL,	NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);

	elm_genlist_item_class_free(itc);
	elm_genlist_item_class_free(ttc);
	elm_genlist_item_class_free(ptc);

	/* This button is set for devices which doesn't have H/W back key. */
	btn = elm_button_add(nf);
	elm_object_style_set(btn, "naviframe/end_btn/default");
	nf_it = elm_naviframe_item_push(nf, NULL, btn, NULL, genlist, "emp-
ty");
	elm_naviframe_item_pop_cb_set(nf_it, naviframe_pop_cb, ad->win);
}

static void
create_base_gui(appdata_s *ad)
{
	/*
	 * Widget Tree
	 * Window
	 *  - conform
	 *   - layout main
	 *    - naviframe */

	/* Window */
	ad->win = elm_win_util_standard_add(PACKAGE, PACKAGE);
	elm_win_conformant_set(ad->win, EINA_TRUE);
	elm_win_autodel_set(ad->win, EINA_TRUE);

	if (elm_win_wm_rotation_supported_get(ad->win)) {
		int rots[4] = { 0, 90, 180, 270 };
```

```
            elm_win_wm_rotation_available_rotations_set(ad->win, (const int
*)(&rots), 4);
      }

      evas_object_smart_callback_add(ad->win,              "delete,request",
win_delete_request_cb, NULL);

      /* Conformant */
      ad->conform = elm_conformant_add(ad->win);
      evas_object_size_hint_weight_set(ad->conform,        EVAS_HINT_EXPAND,
EVAS_HINT_EXPAND);
      elm_win_resize_object_add(ad->win, ad->conform);
      evas_object_show(ad->conform);

      // Eext Circle Surface Creation
      ad->circle_surface = eext_circle_surface_conformant_add(ad->conform);

      /* Indicator */
      /* elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW); */

      /* Base Layout */
      ad->layout = elm_layout_add(ad->conform);
      evas_object_size_hint_weight_set(ad->layout,         EVAS_HINT_EXPAND,
EVAS_HINT_EXPAND);
      elm_layout_theme_set(ad->layout, "layout", "application", "default");
      evas_object_show(ad->layout);

      elm_object_content_set(ad->conform, ad->layout);

      /* Naviframe */
      ad->nf = elm_naviframe_add(ad->layout);
      create_list_view(ad);
      elm_object_part_content_set(ad->layout,   "elm.swallow.content",   ad-
>nf);
      eext_object_event_callback_add(ad->nf,             EEXT_CALLBACK_BACK,
eext_naviframe_back_cb, NULL);
      eext_object_event_callback_add(ad->nf,             EEXT_CALLBACK_MORE,
eext_naviframe_more_cb, NULL);

      /* Show window after base gui is set up */
      evas_object_show(ad->win);
}

static bool
app_create(void *data)
{
      /* Hook to take necessary actions before main event loop starts
         Initialize UI resources and application's data
         If this function returns true, the main loop of application starts
         If this function returns false, the application is terminated */
      appdata_s *ad = (appdata_s *)data;

      elm_app_base_scale_set(1.8);
      create_base_gui(ad);

      return true;
}

static void
```

```
app_control(app_control_h app_control, void *data)
{
	/* Handle the launch request. */
}

static void
app_pause(void *data)
{
	/* Take necessary actions when application becomes invisible. */
}

static void
app_resume(void *data)
{
	/* Take necessary actions when application becomes visible. */
}

static void
app_terminate(void *data)
{
	/* Release all resources. */
}

static void
ui_app_lang_changed(app_event_info_h event_info, void *user_data)
{
	/*APP_EVENT_LANGUAGE_CHANGED*/
	char *locale = NULL;
	system_settings_get_value_string(SYSTEM_SETTINGS_KEY_LOCALE_LANGUAGE,
&locale);
	elm_language_set(locale);
	free(locale);
	return;
}

static void
ui_app_orient_changed(app_event_info_h event_info, void *user_data)
{
	/*APP_EVENT_DEVICE_ORIENTATION_CHANGED*/
	return;
}

static void
ui_app_region_changed(app_event_info_h event_info, void *user_data)
{
	/*APP_EVENT_REGION_FORMAT_CHANGED*/
}

static void
ui_app_low_battery(app_event_info_h event_info, void *user_data)
{
	/*APP_EVENT_LOW_BATTERY*/
}

static void
ui_app_low_memory(app_event_info_h event_info, void *user_data)
{
	/*APP_EVENT_LOW_MEMORY*/
}
```

```
int
main(int argc, char *argv[])
{
	appdata_s ad = {0,};
	int ret = 0;

	ui_app_lifecycle_callback_s event_callback = {0,};
	app_event_handler_h handlers[5] = {NULL, };

	event_callback.create = app_create;
	event_callback.terminate = app_terminate;
	event_callback.pause = app_pause;
	event_callback.resume = app_resume;
	event_callback.app_control = app_control;

	ui_app_add_event_handler(&handlers[APP_EVENT_LOW_BATTERY],
APP_EVENT_LOW_BATTERY, ui_app_low_battery, &ad);
	ui_app_add_event_handler(&handlers[APP_EVENT_LOW_MEMORY],
APP_EVENT_LOW_MEMORY, ui_app_low_memory, &ad);
	ui_app_add_event_handler(&handlers[APP_EVENT_DEVICE_ORIENTATION_CHANG
ED], APP_EVENT_DEVICE_ORIENTATION_CHANGED, ui_app_orient_changed, &ad);
	ui_app_add_event_handler(&handlers[APP_EVENT_LANGUAGE_CHANGED],
APP_EVENT_LANGUAGE_CHANGED, ui_app_lang_changed, &ad);
	ui_app_add_event_handler(&handlers[APP_EVENT_REGION_FORMAT_CHANGED],
APP_EVENT_REGION_FORMAT_CHANGED, ui_app_region_changed, &ad);
	ui_app_remove_event_handler(handlers[APP_EVENT_LOW_MEMORY]);

	ret = ui_app_main(argc, argv, &event_callback, &ad);
	if (ret != APP_ERROR_NONE) {
		dlog_print(DLOG_ERROR, LOG_TAG, "app_main() is failed. err =
%d", ret);
	}

	return ret;
}
```

10.3.2 User Interface

```
#include "main.h"
#include <string.h>
#include <stdio.h>

Evas_Object *entry;
int calories=0, gender=0, age=0, weight=0, height=0, calories_goal=0;

//new String array
char *meal_time_names[] = {
	"Total Calories", "Breakfast", "Lunch","Dinner", "Snack",
	NULL
};

char *add_number[] = {
	"Show total", "+10", "+5", "+1",
	NULL
};
```

```
char *add_number2[] = {
      "Show total", "+50", "+10", "+5", "+1",
      NULL
};

char *gender_name[] = {
      "Show gender", "Male", "Female",
      NULL
};

char name[50];


int getCalories()
{
      return calories;
}



char *button_menu_names[] = {
      "Meal List", "Calories Goal",
      NULL
};

char *meal_names[] = {
      "100g chicken breast", "1 large apple","1 banana", "1 hard boiled
egg", "1 shin ramen",
      NULL
};

char *user_info[] = {
      "Goal", "Gender", "Age", "Weight", "Height",
      NULL
};



typedef struct _item_data
{
      int index;
      Elm_Object_Item *item;
} item_data;

static void
gl_selected_cb(void *data, Evas_Object *obj, void *event_info)
{
      Elm_Object_Item *it = (Elm_Object_Item *)event_info;
      elm_genlist_item_selected_set(it, EINA_FALSE);
}

static char *
_gl_menu_title_text_get(void *data, Evas_Object *obj, const char *part)
{
      char buf[1024];

      snprintf(buf, 1023, "%s", "Diet");
      return strdup(buf);
```

```
}

static char *
_gl_menu_text_get(void *data, Evas_Object *obj, const char *part)
{
	char buf[1024];
	item_data *id = (item_data *)data;
	int index = id->index;

	if (!strcmp(part, "elm.text")) {
		snprintf(buf, 1023, "%s", button_menu_names[index]);
		return strdup(buf);
	}
	return NULL;
}

static char *
_gl_menu_text_get1(void *data, Evas_Object *obj, const char *part)
{
	char buf[1024];
	item_data *id = (item_data *)data;
	int index = id->index;

	if (!strcmp(part, "elm.text")) {
		snprintf(buf, 1023, "%s", meal_time_names[index]);
		return strdup(buf);
	}
	return NULL;
}

static char *
_gl_menu_text_get2(void *data, Evas_Object *obj, const char *part)
{
	char buf[1024];
	item_data *id = (item_data *)data;
	int index = id->index;

	if (!strcmp(part, "elm.text")) {
		snprintf(buf, 1023, "%s", meal_names[index]);
		return strdup(buf);
	}
	return NULL;
}

static char *
_gl_menu_text_get3(void *data, Evas_Object *obj, const char *part)
{
	char buf[1024];
	item_data *id = (item_data *)data;
	int index = id->index;

	if (!strcmp(part, "elm.text")) {
		snprintf(buf, 1023, "%s", user_info[index]);
		return strdup(buf);
	}
	return NULL;
}
```

```
static char *
_gl_menu_text_get4(void *data, Evas_Object *obj, const char *part)
{
	char buf[1024];
	item_data *id = (item_data *)data;
	int index = id->index;

	if (!strcmp(part, "elm.text")) {
		snprintf(buf, 1023, "%s", add_number[index]);
		return strdup(buf);
	}
	return NULL;
}

static char *
_gl_menu_text_get5(void *data, Evas_Object *obj, const char *part)
{
	char buf[1024];
	item_data *id = (item_data *)data;
	int index = id->index;

	if (!strcmp(part, "elm.text")) {
		snprintf(buf, 1023, "%s", gender_name[index]);
		return strdup(buf);
	}
	return NULL;
}

static char *
_gl_menu_text_get6(void *data, Evas_Object *obj, const char *part)
{
	char buf[1024];
	item_data *id = (item_data *)data;
	int index = id->index;

	if (!strcmp(part, "elm.text")) {
		snprintf(buf, 1023, "%s", add_number2[index]);
		return strdup(buf);
	}
	return NULL;
}

static void
_gl_menu_del(void *data, Evas_Object *obj)
{
	// FIXME: Unrealized callback can be called after this.
	// Accessing Item_Data can be dangerous on unrealized callback.
	item_data *id = (item_data *)data;
	if (id) free(id);
}

static void
_popup_hide_cb(void *data, Evas_Object *obj, void *event_info)
{
	if(!obj) return;
	elm_popup_dismiss(obj);
}

static void
```

```
_popup_hide_finished_cb(void *data, Evas_Object *obj, void *event_info)
{
	if(!obj) return;
	evas_object_del(obj);
}

static void _block_clicked_cb(void *data, Evas_Object *obj, void
*event_info)
{
	if(!obj) return;
	elm_popup_dismiss(obj);
}

static void _timeout_cb(void *data, Evas_Object *obj, void *event_info)
{
	if(!obj) return;
	elm_popup_dismiss(obj);
}

static void _popup_toast_cb1(void *data, Evas_Object *obj, void
*event_info)
{
	Evas_Object *popup;
	appdata_s *ad = (appdata_s *)data;

	popup = elm_popup_add(ad->win);
	elm_object_style_set(popup, "toast/circle");
	elm_popup_orient_set(popup, ELM_POPUP_ORIENT_BOTTOM);
	evas_object_size_hint_weight_set(popup, EVAS_HINT_EXPAND,
EVAS_HINT_EXPAND);
	eext_object_event_callback_add(popup, EEXT_CALLBACK_BACK,
_popup_hide_cb, NULL);
	evas_object_smart_callback_add(popup, "dismissed",
_popup_hide_finished_cb, NULL);
	elm_object_part_text_set(popup,"elm.text", "100g chicken breast add-
ed!");
	calories+=165;
	evas_object_smart_callback_add(popup, "block,clicked",
_block_clicked_cb, NULL);

	elm_popup_timeout_set(popup, 2.0);
	evas_object_smart_callback_add(popup, "timeout", _timeout_cb, NULL);

	evas_object_show(popup);
}

static void _popup_toast_cb2(void *data, Evas_Object *obj, void
*event_info)
{
	Evas_Object *popup;
	appdata_s *ad = (appdata_s *)data;

	popup = elm_popup_add(ad->win);
	elm_object_style_set(popup, "toast/circle");
	elm_popup_orient_set(popup, ELM_POPUP_ORIENT_BOTTOM);
	evas_object_size_hint_weight_set(popup, EVAS_HINT_EXPAND,
EVAS_HINT_EXPAND);
	eext_object_event_callback_add(popup, EEXT_CALLBACK_BACK,
_popup_hide_cb, NULL);
```

```
	evas_object_smart_callback_add(popup, "dismissed",
_popup_hide_finished_cb, NULL);
	elm_object_part_text_set(popup,"elm.text", "1 large apple added!");
	calories+=100;
	evas_object_smart_callback_add(popup, "block,clicked",
_block_clicked_cb, NULL);

	elm_popup_timeout_set(popup, 2.0);
	evas_object_smart_callback_add(popup, "timeout", _timeout_cb, NULL);

	evas_object_show(popup);
}

static void _popup_toast_cb3(void *data, Evas_Object *obj, void
*event_info)
{
	Evas_Object *popup;
	appdata_s *ad = (appdata_s *)data;

	popup = elm_popup_add(ad->win);
	elm_object_style_set(popup, "toast/circle");
	elm_popup_orient_set(popup, ELM_POPUP_ORIENT_BOTTOM);
	evas_object_size_hint_weight_set(popup, EVAS_HINT_EXPAND,
EVAS_HINT_EXPAND);
	eext_object_event_callback_add(popup, EEXT_CALLBACK_BACK,
_popup_hide_cb, NULL);
	evas_object_smart_callback_add(popup, "dismissed",
_popup_hide_finished_cb, NULL);
	elm_object_part_text_set(popup,"elm.text", "1 banana added!");
	calories+=125;
	evas_object_smart_callback_add(popup, "block,clicked",
_block_clicked_cb, NULL);

	elm_popup_timeout_set(popup, 2.0);
	evas_object_smart_callback_add(popup, "timeout", _timeout_cb, NULL);

	evas_object_show(popup);
}

static void _popup_toast_cb4(void *data, Evas_Object *obj, void
*event_info)
{
	Evas_Object *popup;
	appdata_s *ad = (appdata_s *)data;

	popup = elm_popup_add(ad->win);
	elm_object_style_set(popup, "toast/circle");
	elm_popup_orient_set(popup, ELM_POPUP_ORIENT_BOTTOM);
	evas_object_size_hint_weight_set(popup, EVAS_HINT_EXPAND,
EVAS_HINT_EXPAND);
	eext_object_event_callback_add(popup, EEXT_CALLBACK_BACK,
_popup_hide_cb, NULL);
	evas_object_smart_callback_add(popup, "dismissed",
_popup_hide_finished_cb, NULL);
	elm_object_part_text_set(popup,"elm.text", "1 hard boiled egg add-
ed!");
	calories+=72;
	evas_object_smart_callback_add(popup, "block,clicked",
_block_clicked_cb, NULL);
```

```
	elm_popup_timeout_set(popup, 2.0);
	evas_object_smart_callback_add(popup, "timeout", _timeout_cb, NULL);

	evas_object_show(popup);
}

static void _popup_toast_cb5(void *data, Evas_Object *obj, void
*event_info)
{
	Evas_Object *popup;
	appdata_s *ad = (appdata_s *)data;

	popup = elm_popup_add(ad->win);
	elm_object_style_set(popup, "toast/circle");
	elm_popup_orient_set(popup, ELM_POPUP_ORIENT_BOTTOM);
	evas_object_size_hint_weight_set(popup, EVAS_HINT_EXPAND,
EVAS_HINT_EXPAND);
	eext_object_event_callback_add(popup, EEXT_CALLBACK_BACK,
_popup_hide_cb, NULL);
	evas_object_smart_callback_add(popup, "dismissed",
_popup_hide_finished_cb, NULL);
	elm_object_part_text_set(popup,"elm.text", "1 shin ramen added!");
	calories+=240;
	evas_object_smart_callback_add(popup, "block,clicked",
_block_clicked_cb, NULL);

	elm_popup_timeout_set(popup, 2.0);
	evas_object_smart_callback_add(popup, "timeout", _timeout_cb, NULL);

	evas_object_show(popup);
}

static void _popup_calorie_cb(void *data, Evas_Object *obj, void
*event_info)
{
	Evas_Object *popup;
	appdata_s *ad = (appdata_s *)data;
	sprintf(name, "Total calories:  %d", getCalories());
	popup = elm_popup_add(ad->win);
	elm_object_style_set(popup, "toast/circle");
	elm_popup_orient_set(popup, ELM_POPUP_ORIENT_BOTTOM);
	evas_object_size_hint_weight_set(popup, EVAS_HINT_EXPAND,
EVAS_HINT_EXPAND);
	eext_object_event_callback_add(popup, EEXT_CALLBACK_BACK,
_popup_hide_cb, NULL);
	evas_object_smart_callback_add(popup, "dismissed",
_popup_hide_finished_cb, NULL);
	elm_object_part_text_set(popup,"elm.text", name);
	evas_object_smart_callback_add(popup, "block,clicked",
_block_clicked_cb, NULL);

	elm_popup_timeout_set(popup, 2.0);
	evas_object_smart_callback_add(popup, "timeout", _timeout_cb, NULL);

	evas_object_show(popup);
}

static void show_goal_cb(void *data, Evas_Object *obj, void *event_info)
```

```
{
            Evas_Object *popup;
            appdata_s *ad = (appdata_s *)data;
            calories_goal = (int)10*weight+6.25*height-5*age;
            if(gender==0)
            {
                  calories_goal+=5;
            }
            else
            {
                  calories_goal-=161;
            }
            sprintf(name, "Total calories goal :  %d", calories_goal);
            popup = elm_popup_add(ad->win);
            elm_object_style_set(popup, "toast/circle");
            elm_popup_orient_set(popup, ELM_POPUP_ORIENT_BOTTOM);
            evas_object_size_hint_weight_set(popup,        EVAS_HINT_EXPAND,
EVAS_HINT_EXPAND);
            eext_object_event_callback_add(popup,        EEXT_CALLBACK_BACK,
_popup_hide_cb, NULL);
            evas_object_smart_callback_add(popup,               "dismissed",
_popup_hide_finished_cb, NULL);
            elm_object_part_text_set(popup,"elm.text", name);
            evas_object_smart_callback_add(popup,           "block,clicked",
_block_clicked_cb, NULL);

            elm_popup_timeout_set(popup, 2.0);
            evas_object_smart_callback_add(popup,  "timeout",  _timeout_cb,
NULL);

            evas_object_show(popup);
}

static void show_gender_cb(void *data, Evas_Object *obj, void *event_info)
{
      Evas_Object *popup;
      appdata_s *ad = (appdata_s *)data;
      char *male="male";
      char *female="female";
      if(gender==0)
      {
            sprintf(name, "Gender:  %s", male);
      }
      else
      {
            sprintf(name, "Gender:  %s", female);
      }
      popup = elm_popup_add(ad->win);
      elm_object_style_set(popup, "toast/circle");
      elm_popup_orient_set(popup, ELM_POPUP_ORIENT_BOTTOM);
      evas_object_size_hint_weight_set(popup,              EVAS_HINT_EXPAND,
EVAS_HINT_EXPAND);
      eext_object_event_callback_add(popup,              EEXT_CALLBACK_BACK,
_popup_hide_cb, NULL);
      evas_object_smart_callback_add(popup,                     "dismissed",
_popup_hide_finished_cb, NULL);
      elm_object_part_text_set(popup,"elm.text", name);
      evas_object_smart_callback_add(popup,                 "block,clicked",
_block_clicked_cb, NULL);
```

```
	elm_popup_timeout_set(popup, 2.0);
	evas_object_smart_callback_add(popup, "timeout", _timeout_cb, NULL);

	evas_object_show(popup);
}

static void _add_male_cb(void *data, Evas_Object *obj, void *event_info)
{
	gender=0;
}

static void _add_female_cb(void *data, Evas_Object *obj, void *event_info)
{
	gender=1;
}

static void _user_info_entry_cb1(void *data, Evas_Object *obj, void
*event_info)
{
	appdata_s *ad = (appdata_s *)data;
	Evas_Object *genlist;
	Evas_Object *circle_genlist;
	Evas_Object *nf = ad->nf;
	Elm_Object_Item *nf_it;
	Elm_Genlist_Item_Class *itc = elm_genlist_item_class_new();
	Elm_Genlist_Item_Class *ttc = elm_genlist_item_class_new();
	Elm_Genlist_Item_Class *ptc = elm_genlist_item_class_new();
	item_data *id;
	int index = 0;

	genlist = elm_genlist_add(nf);
	elm_genlist_mode_set(genlist, ELM_LIST_COMPRESS);
	evas_object_smart_callback_add(genlist, "selected", gl_selected_cb,
NULL);

	circle_genlist = eext_circle_object_genlist_add(genlist, ad-
>circle_surface);
	eext_circle_object_genlist_scroller_policy_set(circle_genlist,
ELM_SCROLLER_POLICY_OFF, ELM_SCROLLER_POLICY_AUTO);
	eext_rotary_object_event_activated_set(circle_genlist, EINA_TRUE);

	ttc->item_style = "title";
	ttc->func.text_get = _gl_menu_title_text_get;
	ttc->func.del = _gl_menu_del;

	itc->item_style = "default";
	itc->func.text_get = _gl_menu_text_get5;
	itc->func.del = _gl_menu_del;

	ptc->item_style = "padding";
	ptc->func.del = _gl_menu_del;

	elm_genlist_item_append(genlist, ttc, NULL, NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);



	id = calloc(sizeof(item_data), 1);
```

```
	id->index = index++;
	id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, show_gender_cb, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, _add_male_cb, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, _add_female_cb, ad);



	elm_genlist_item_append(genlist, ptc, NULL, NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);

	elm_genlist_item_class_free(ttc);
	elm_genlist_item_class_free(itc);
	elm_genlist_item_class_free(ptc);

	nf_it = elm_naviframe_item_push(nf, "Button", NULL, NULL, genlist,
"empty");
}

static void show_age_cb(void *data, Evas_Object *obj, void *event_info)
{
	Evas_Object *popup;
	appdata_s *ad = (appdata_s *)data;

	sprintf(name, "Age :  %d", age);
	popup = elm_popup_add(ad->win);
	elm_object_style_set(popup, "toast/circle");
	elm_popup_orient_set(popup, ELM_POPUP_ORIENT_BOTTOM);
	evas_object_size_hint_weight_set(popup, EVAS_HINT_EXPAND,
EVAS_HINT_EXPAND);
	eext_object_event_callback_add(popup, EEXT_CALLBACK_BACK,
_popup_hide_cb, NULL);
	evas_object_smart_callback_add(popup, "dismissed",
_popup_hide_finished_cb, NULL);
	elm_object_part_text_set(popup,"elm.text", name);
	evas_object_smart_callback_add(popup, "block,clicked",
_block_clicked_cb, NULL);

	elm_popup_timeout_set(popup, 2.0);
	evas_object_smart_callback_add(popup, "timeout", _timeout_cb, NULL);

	evas_object_show(popup);
}

static void _add_10_years_cb(void *data, Evas_Object *obj, void
*event_info)
{
	age+=10;
}

static void _add_5_years_cb(void *data, Evas_Object *obj, void *event_info)
{
```

```
	age+=5;
}

static void _add_1_year_cb(void *data, Evas_Object *obj, void *event_info)
{
	age+=1;
}

static void _user_info_entry_cb2(void *data, Evas_Object *obj, void
*event_info)
{
	appdata_s *ad = (appdata_s *)data;
		Evas_Object *genlist;
		Evas_Object *circle_genlist;
		Evas_Object *nf = ad->nf;
		Elm_Object_Item *nf_it;
		Elm_Genlist_Item_Class *itc = elm_genlist_item_class_new();
		Elm_Genlist_Item_Class *ttc = elm_genlist_item_class_new();
		Elm_Genlist_Item_Class *ptc = elm_genlist_item_class_new();
		item_data *id;
		int index = 0;

		genlist = elm_genlist_add(nf);
		elm_genlist_mode_set(genlist, ELM_LIST_COMPRESS);
		evas_object_smart_callback_add(genlist, "selected",
gl_selected_cb, NULL);

		circle_genlist = eext_circle_object_genlist_add(genlist, ad-
>circle_surface);
		eext_circle_object_genlist_scroller_policy_set(circle_genlist,
ELM_SCROLLER_POLICY_OFF, ELM_SCROLLER_POLICY_AUTO);
		eext_rotary_object_event_activated_set(circle_genlist,
EINA_TRUE);

		ttc->item_style = "title";
		ttc->func.text_get = _gl_menu_title_text_get;
		ttc->func.del = _gl_menu_del;

		itc->item_style = "default";
		itc->func.text_get = _gl_menu_text_get4;
		itc->func.del = _gl_menu_del;

		ptc->item_style = "padding";
		ptc->func.del = _gl_menu_del;

		elm_genlist_item_append(genlist, ttc, NULL, NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);



		id = calloc(sizeof(item_data), 1);
		id->index = index++;
		id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, show_age_cb, ad);
		id = calloc(sizeof(item_data), 1);
		id->index = index++;
		id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, _add_10_years_cb, ad);
		id = calloc(sizeof(item_data), 1);
```

```
            id->index = index++;
            id->item  =  elm_genlist_item_append(genlist,  itc,  id,  NULL,
ELM_GENLIST_ITEM_NONE, _add_5_years_cb, ad);
            id = calloc(sizeof(item_data), 1);
            id->index = index++;
            id->item  =  elm_genlist_item_append(genlist,  itc,  id,  NULL,
ELM_GENLIST_ITEM_NONE, _add_1_year_cb, ad);



            elm_genlist_item_append(genlist,      ptc,      NULL,      NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);

            elm_genlist_item_class_free(ttc);
            elm_genlist_item_class_free(itc);
            elm_genlist_item_class_free(ptc);

            nf_it = elm_naviframe_item_push(nf, "Button", NULL, NULL, gen-
list, "empty");
}

static void show_weight_cb(void *data, Evas_Object *obj, void *event_info)
{
      Evas_Object *popup;
      appdata_s *ad = (appdata_s *)data;

      sprintf(name, "Weight :  %d", weight);
      popup = elm_popup_add(ad->win);
      elm_object_style_set(popup, "toast/circle");
      elm_popup_orient_set(popup, ELM_POPUP_ORIENT_BOTTOM);
      evas_object_size_hint_weight_set(popup,              EVAS_HINT_EXPAND,
EVAS_HINT_EXPAND);
      eext_object_event_callback_add(popup,              EEXT_CALLBACK_BACK,
_popup_hide_cb, NULL);
      evas_object_smart_callback_add(popup,                    "dismissed",
_popup_hide_finished_cb, NULL);
      elm_object_part_text_set(popup,"elm.text", name);
      evas_object_smart_callback_add(popup,                 "block,clicked",
_block_clicked_cb, NULL);

      elm_popup_timeout_set(popup, 2.0);
      evas_object_smart_callback_add(popup, "timeout", _timeout_cb, NULL);

      evas_object_show(popup);
}

static void _add_10_kilo_cb(void *data, Evas_Object *obj, void *event_info)
{
      weight+=10;
}

static void _add_5_kilo_cb(void *data, Evas_Object *obj, void *event_info)
{
      weight+=5;
}

static void _add_1_kilo_cb(void *data, Evas_Object *obj, void *event_info)
{
      weight+=1;
```

```
}

static void _user_info_entry_cb3(void *data, Evas_Object *obj, void
*event_info)
{
      appdata_s *ad = (appdata_s *)data;
            Evas_Object *genlist;
            Evas_Object *circle_genlist;
            Evas_Object *nf = ad->nf;
            Elm_Object_Item *nf_it;
            Elm_Genlist_Item_Class *itc = elm_genlist_item_class_new();
            Elm_Genlist_Item_Class *ttc = elm_genlist_item_class_new();
            Elm_Genlist_Item_Class *ptc = elm_genlist_item_class_new();
            item_data *id;
            int index = 0;

            genlist = elm_genlist_add(nf);
            elm_genlist_mode_set(genlist, ELM_LIST_COMPRESS);
            evas_object_smart_callback_add(genlist,              "selected",
gl_selected_cb, NULL);

            circle_genlist  =  eext_circle_object_genlist_add(genlist,  ad-
>circle_surface);
            eext_circle_object_genlist_scroller_policy_set(circle_genlist,
ELM_SCROLLER_POLICY_OFF, ELM_SCROLLER_POLICY_AUTO);
            eext_rotary_object_event_activated_set(circle_genlist,
EINA_TRUE);

            ttc->item_style = "title";
            ttc->func.text_get = _gl_menu_title_text_get;
            ttc->func.del = _gl_menu_del;

            itc->item_style = "default";
            itc->func.text_get = _gl_menu_text_get4;
            itc->func.del = _gl_menu_del;

            ptc->item_style = "padding";
            ptc->func.del = _gl_menu_del;

            elm_genlist_item_append(genlist,      ttc,      NULL,      NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);



            id = calloc(sizeof(item_data), 1);
            id->index = index++;
            id->item  =  elm_genlist_item_append(genlist,  itc,  id,  NULL,
ELM_GENLIST_ITEM_NONE, show_weight_cb, ad);
            id = calloc(sizeof(item_data), 1);
            id->index = index++;
            id->item  =  elm_genlist_item_append(genlist,  itc,  id,  NULL,
ELM_GENLIST_ITEM_NONE, _add_10_kilo_cb, ad);
            id = calloc(sizeof(item_data), 1);
            id->index = index++;
            id->item  =  elm_genlist_item_append(genlist,  itc,  id,  NULL,
ELM_GENLIST_ITEM_NONE, _add_5_kilo_cb, ad);
            id = calloc(sizeof(item_data), 1);
            id->index = index++;
```

```
            id->item  =  elm_genlist_item_append(genlist,  itc,  id,  NULL,
ELM_GENLIST_ITEM_NONE, _add_1_kilo_cb, ad);



            elm_genlist_item_append(genlist,      ptc,      NULL,      NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);

            elm_genlist_item_class_free(ttc);
            elm_genlist_item_class_free(itc);
            elm_genlist_item_class_free(ptc);

            nf_it = elm_naviframe_item_push(nf, "Button", NULL, NULL, gen-
list, "empty");

}

static void show_height_cb(void *data, Evas_Object *obj, void *event_info)
{
      Evas_Object *popup;
      appdata_s *ad = (appdata_s *)data;

      sprintf(name, "Height :  %d", height);
      popup = elm_popup_add(ad->win);
      elm_object_style_set(popup, "toast/circle");
      elm_popup_orient_set(popup, ELM_POPUP_ORIENT_BOTTOM);
      evas_object_size_hint_weight_set(popup,              EVAS_HINT_EXPAND,
EVAS_HINT_EXPAND);
      eext_object_event_callback_add(popup,              EEXT_CALLBACK_BACK,
_popup_hide_cb, NULL);
      evas_object_smart_callback_add(popup,                    "dismissed",
_popup_hide_finished_cb, NULL);
      elm_object_part_text_set(popup,"elm.text", name);
      evas_object_smart_callback_add(popup,                 "block,clicked",
_block_clicked_cb, NULL);

      elm_popup_timeout_set(popup, 2.0);
      evas_object_smart_callback_add(popup, "timeout", _timeout_cb, NULL);

      evas_object_show(popup);
}

static void _add_50_cm_cb(void *data, Evas_Object *obj, void *event_info)
{
      height+=50;
}

static void _add_10_cm_cb(void *data, Evas_Object *obj, void *event_info)
{
      height+=10;
}

static void _add_5_cm_cb(void *data, Evas_Object *obj, void *event_info)
{
      height+=5;
}

static void _add_1_cm_cb(void *data, Evas_Object *obj, void *event_info)
{
```

```
	height+=1;
}

static void _user_info_entry_cb4(void *data, Evas_Object *obj, void
*event_info)
{
	appdata_s *ad = (appdata_s *)data;
		Evas_Object *genlist;
		Evas_Object *circle_genlist;
		Evas_Object *nf = ad->nf;
		Elm_Object_Item *nf_it;
		Elm_Genlist_Item_Class *itc = elm_genlist_item_class_new();
		Elm_Genlist_Item_Class *ttc = elm_genlist_item_class_new();
		Elm_Genlist_Item_Class *ptc = elm_genlist_item_class_new();
		item_data *id;
		int index = 0;

		genlist = elm_genlist_add(nf);
		elm_genlist_mode_set(genlist, ELM_LIST_COMPRESS);
		evas_object_smart_callback_add(genlist, "selected",
gl_selected_cb, NULL);

		circle_genlist = eext_circle_object_genlist_add(genlist, ad-
>circle_surface);
		eext_circle_object_genlist_scroller_policy_set(circle_genlist,
ELM_SCROLLER_POLICY_OFF, ELM_SCROLLER_POLICY_AUTO);
		eext_rotary_object_event_activated_set(circle_genlist,
EINA_TRUE);

		ttc->item_style = "title";
		ttc->func.text_get = _gl_menu_title_text_get;
		ttc->func.del = _gl_menu_del;

		itc->item_style = "default";
		itc->func.text_get = _gl_menu_text_get6;
		itc->func.del = _gl_menu_del;

		ptc->item_style = "padding";
		ptc->func.del = _gl_menu_del;

		elm_genlist_item_append(genlist, ttc, NULL, NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);



		id = calloc(sizeof(item_data), 1);
		id->index = index++;
		id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, show_height_cb, ad);
		id = calloc(sizeof(item_data), 1);
		id->index = index++;
		id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, _add_50_cm_cb, ad);
		id = calloc(sizeof(item_data), 1);
		id->index = index++;
		id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, _add_10_cm_cb, ad);
		id = calloc(sizeof(item_data), 1);
		id->index = index++;
```

```
            id->item  =  elm_genlist_item_append(genlist,  itc,  id,  NULL,
ELM_GENLIST_ITEM_NONE, _add_5_cm_cb, ad);
            id = calloc(sizeof(item_data), 1);
            id->index = index++;
            id->item  =  elm_genlist_item_append(genlist,  itc,  id,  NULL,
ELM_GENLIST_ITEM_NONE, _add_1_cm_cb, ad);



            elm_genlist_item_append(genlist,      ptc,      NULL,      NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);

            elm_genlist_item_class_free(ttc);
            elm_genlist_item_class_free(itc);
            elm_genlist_item_class_free(ptc);

            nf_it = elm_naviframe_item_push(nf, "Button", NULL, NULL, gen-
list, "empty");

}

void
calorie_goal_cb(void *data, Evas_Object *obj, void *event_info)
{
      appdata_s *ad = (appdata_s *)data;
      Evas_Object *genlist;
      Evas_Object *circle_genlist;
      Evas_Object *nf = ad->nf;
      Elm_Object_Item *nf_it;
      Elm_Genlist_Item_Class *itc = elm_genlist_item_class_new();
      Elm_Genlist_Item_Class *ttc = elm_genlist_item_class_new();
      Elm_Genlist_Item_Class *ptc = elm_genlist_item_class_new();
      item_data *id;
      int index = 0;

      genlist = elm_genlist_add(nf);
      elm_genlist_mode_set(genlist, ELM_LIST_COMPRESS);
      evas_object_smart_callback_add(genlist,  "selected",  gl_selected_cb,
NULL);

      circle_genlist    =    eext_circle_object_genlist_add(genlist,    ad-
>circle_surface);
      eext_circle_object_genlist_scroller_policy_set(circle_genlist,
ELM_SCROLLER_POLICY_OFF, ELM_SCROLLER_POLICY_AUTO);
      eext_rotary_object_event_activated_set(circle_genlist, EINA_TRUE);

      ttc->item_style = "title";
      ttc->func.text_get = _gl_menu_title_text_get;
      ttc->func.del = _gl_menu_del;

      itc->item_style = "default";
      itc->func.text_get = _gl_menu_text_get3;
      itc->func.del = _gl_menu_del;

      ptc->item_style = "padding";
      ptc->func.del = _gl_menu_del;

      elm_genlist_item_append(genlist,        ttc,        NULL,        NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);
```

```
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, show_goal_cb, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, _user_info_entry_cb1, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, _user_info_entry_cb2, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, _user_info_entry_cb3, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, _user_info_entry_cb4, ad);


	elm_genlist_item_append(genlist, ptc, NULL, NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);

	elm_genlist_item_class_free(ttc);
	elm_genlist_item_class_free(itc);
	elm_genlist_item_class_free(ptc);

	nf_it = elm_naviframe_item_push(nf, "Button", NULL, NULL, genlist,
"empty");
}

void
mealList_cb(void *data, Evas_Object *obj, void *event_info)
{
	appdata_s *ad = (appdata_s *)data;
	Evas_Object *genlist;
	Evas_Object *circle_genlist;
	Evas_Object *nf = ad->nf;
	Elm_Object_Item *nf_it;
	Elm_Genlist_Item_Class *itc = elm_genlist_item_class_new();
	Elm_Genlist_Item_Class *ttc = elm_genlist_item_class_new();
	Elm_Genlist_Item_Class *ptc = elm_genlist_item_class_new();
	item_data *id;
	int index = 0;

	genlist = elm_genlist_add(nf);
	elm_genlist_mode_set(genlist, ELM_LIST_COMPRESS);
	evas_object_smart_callback_add(genlist, "selected", gl_selected_cb,
NULL);

	circle_genlist = eext_circle_object_genlist_add(genlist, ad-
>circle_surface);
	eext_circle_object_genlist_scroller_policy_set(circle_genlist,
ELM_SCROLLER_POLICY_OFF, ELM_SCROLLER_POLICY_AUTO);
```

```
	eext_rotary_object_event_activated_set(circle_genlist, EINA_TRUE);

	ttc->item_style = "title";
	ttc->func.text_get = _gl_menu_title_text_get;
	ttc->func.del = _gl_menu_del;

	itc->item_style = "default";
	itc->func.text_get = _gl_menu_text_get2;
	itc->func.del = _gl_menu_del;

	ptc->item_style = "padding";
	ptc->func.del = _gl_menu_del;

	elm_genlist_item_append(genlist,	ttc,	NULL,	NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);

	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item	=	elm_genlist_item_append(genlist,	itc,	id,	NULL,
ELM_GENLIST_ITEM_NONE, _popup_toast_cb1, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item	=	elm_genlist_item_append(genlist,	itc,	id,	NULL,
ELM_GENLIST_ITEM_NONE, _popup_toast_cb2, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item	=	elm_genlist_item_append(genlist,	itc,	id,	NULL,
ELM_GENLIST_ITEM_NONE, _popup_toast_cb3, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item	=	elm_genlist_item_append(genlist,	itc,	id,	NULL,
ELM_GENLIST_ITEM_NONE, _popup_toast_cb4, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item	=	elm_genlist_item_append(genlist,	itc,	id,	NULL,
ELM_GENLIST_ITEM_NONE, _popup_toast_cb5, ad);


	elm_genlist_item_append(genlist,	ptc,	NULL,	NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);

	elm_genlist_item_class_free(ttc);
	elm_genlist_item_class_free(itc);
	elm_genlist_item_class_free(ptc);

	nf_it	=	elm_naviframe_item_push(nf,	"Button",	NULL,	NULL,	genlist,
"empty");
}

void
mealTimes_cb(void *data, Evas_Object *obj, void *event_info)
{
	appdata_s *ad = (appdata_s *)data;
	Evas_Object *genlist;
	Evas_Object *circle_genlist;
	Evas_Object *nf = ad->nf;
	Elm_Object_Item *nf_it;
	Elm_Genlist_Item_Class *itc = elm_genlist_item_class_new();
	Elm_Genlist_Item_Class *ttc = elm_genlist_item_class_new();
```

```
      Elm_Genlist_Item_Class *ptc = elm_genlist_item_class_new();
      item_data *id;
      int index = 0;

      genlist = elm_genlist_add(nf);
      elm_genlist_mode_set(genlist, ELM_LIST_COMPRESS);
      evas_object_smart_callback_add(genlist,  "selected",  gl_selected_cb,
NULL);

      circle_genlist    =    eext_circle_object_genlist_add(genlist,    ad-
>circle_surface);
      eext_circle_object_genlist_scroller_policy_set(circle_genlist,
ELM_SCROLLER_POLICY_OFF, ELM_SCROLLER_POLICY_AUTO);
      eext_rotary_object_event_activated_set(circle_genlist, EINA_TRUE);

      ttc->item_style = "title";
      ttc->func.text_get = _gl_menu_title_text_get;
      ttc->func.del = _gl_menu_del;

      itc->item_style = "default";
      itc->func.text_get = _gl_menu_text_get1;
      itc->func.del = _gl_menu_del;

      ptc->item_style = "padding";
      ptc->func.del = _gl_menu_del;

      elm_genlist_item_append(genlist,        ttc,        NULL,        NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);


      id = calloc(sizeof(item_data), 1);
      id->index = index++;
      id->item    =    elm_genlist_item_append(genlist,    itc,    id,    NULL,
ELM_GENLIST_ITEM_NONE, _popup_calorie_cb, ad);
      id = calloc(sizeof(item_data), 1);
      id->index = index++;
      id->item    =    elm_genlist_item_append(genlist,    itc,    id,    NULL,
ELM_GENLIST_ITEM_NONE, mealList_cb, ad);
      id = calloc(sizeof(item_data), 1);
      id->index = index++;
      id->item    =    elm_genlist_item_append(genlist,    itc,    id,    NULL,
ELM_GENLIST_ITEM_NONE, mealList_cb, ad);
      id = calloc(sizeof(item_data), 1);
      id->index = index++;
      id->item    =    elm_genlist_item_append(genlist,    itc,    id,    NULL,
ELM_GENLIST_ITEM_NONE, mealList_cb, ad);
      id = calloc(sizeof(item_data), 1);
      id->index = index++;
      id->item    =    elm_genlist_item_append(genlist,    itc,    id,    NULL,
ELM_GENLIST_ITEM_NONE, mealList_cb, ad);


      elm_genlist_item_append(genlist,        ptc,        NULL,        NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);

      elm_genlist_item_class_free(ttc);
      elm_genlist_item_class_free(itc);
      elm_genlist_item_class_free(ptc);
```

```
	nf_it = elm_naviframe_item_push(nf, "Button", NULL, NULL, genlist,
"empty");
}

void
ui_cb(void *data, Evas_Object *obj, void *event_info)
{
	appdata_s *ad = (appdata_s *)data;
	Evas_Object *genlist;
	Evas_Object *circle_genlist;
	Evas_Object *nf = ad->nf;
	Elm_Object_Item *nf_it;
	Elm_Genlist_Item_Class *itc = elm_genlist_item_class_new();
	Elm_Genlist_Item_Class *ttc = elm_genlist_item_class_new();
	Elm_Genlist_Item_Class *ptc = elm_genlist_item_class_new();
	item_data *id;
	int index = 0;

	genlist = elm_genlist_add(nf);
	elm_genlist_mode_set(genlist, ELM_LIST_COMPRESS);
	evas_object_smart_callback_add(genlist, "selected", gl_selected_cb,
NULL);

	circle_genlist = eext_circle_object_genlist_add(genlist, ad-
>circle_surface);
	eext_circle_object_genlist_scroller_policy_set(circle_genlist,
ELM_SCROLLER_POLICY_OFF, ELM_SCROLLER_POLICY_AUTO);
	eext_rotary_object_event_activated_set(circle_genlist, EINA_TRUE);

	ttc->item_style = "title";
	ttc->func.text_get = _gl_menu_title_text_get;
	ttc->func.del = _gl_menu_del;

	itc->item_style = "default";
	itc->func.text_get = _gl_menu_text_get;
	itc->func.del = _gl_menu_del;

	ptc->item_style = "padding";
	ptc->func.del = _gl_menu_del;

	elm_genlist_item_append(genlist, ttc, NULL, NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);

	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, mealTimes_cb, ad);
	id = calloc(sizeof(item_data), 1);
	id->index = index++;
	id->item = elm_genlist_item_append(genlist, itc, id, NULL,
ELM_GENLIST_ITEM_NONE, calorie_goal_cb, ad);

	elm_genlist_item_append(genlist, ptc, NULL, NULL,
ELM_GENLIST_ITEM_NONE, NULL, NULL);

	elm_genlist_item_class_free(ttc);
	elm_genlist_item_class_free(itc);
	elm_genlist_item_class_free(ptc);
```

```
	nf_it = elm_naviframe_item_push(nf, "Button", NULL, NULL, genlist,
"empty");
}
```

10.4 Content of Figures

10.5 Content of Tables