CS3343 Software Engineering Practice

Project title:

Service Plan Searching Assistant - Search It(SI)

Analysis and Design Report

Group Member:

Ng Chi Kin

Wu Wei Jian

Tsui Ka Kin

Tam Kwan Leong

Chan Ching Yi

Table of content

[**Design constraints**](#_l0hzjcpc4dqv) **5**

[**2. Requirement specifications**](#_jhc0javj4j47) **6**

[**3. Use case diagram**](#_gah5dyd7vhdl) **8**

[**4. Use case specification**](#_i3y95t04c0e) **9**

[4.1 Register Account](#_sirlinf5wksn) 9

[4.2 Login](#_xqodlzired28) 10

[4.3 Display Plan List](#_sdv4zs5qclqj) 11

[4.4 Filter Plan](#_17joxz4l8f0o) 12

[4.5 View Plan Detail](#_1psqebv4f5db) 13

[4.6 Add To Shopping Cart](#_pjdgaar78qj0) 14

[4.7 View Shopping Cart](#_f60zzqhc3zg) 15

[4.8 Check Out](#_t6h9fjvsca5e) 16

[4.9 Delete Shopping Cart Record](#_pl18c5n58ws) 17

[4.10 Check Order](#_ui9u2flcn7a) 18

[4.11 Check Plan](#_9o26g826r6wd) 19

[4.12 Update Plan](#_utb2pd874wx0) 20

[4.13 Add Plan](#_okv8ntyx96yh) 21

[4.14 Delete Plan](#_27p2jau6n8it) 22

[**5. Class diagram**](#_wlhxa2ad35eh) **23**

[**6. Class description**](#_sw8b2fw6vvur) **24**

[6.1 Model.user package](#_uzuq2eklfejs) 24

[6.1.1 Role](#_eyrq3i1mf5sq) 24

[6.1.2 Officer](#_s88ab7vy4bb5) 25

[6.1.3 User](#_edsy6uokthqe) 25

[6.2 Controller](#_ty7p4dpnspq7) 26

[6.3 Helper](#_uu5issg1zhfi) 28

[6.4 Cache](#_u0fws3rklb5e) 28

[6.5 Register](#_meejlkrkgy2x) 28

[6.7 Manager](#_1j33bfw4c4a) 29

[6.7.1 ServicePlanManager](#_v9bitpdf87jv) 31

[6.7.2 AccountManager](#_du9b3b1atb8) 31

[6.7.3 Response](#_vypw4aeo2e5l) 31

[**6.8 Login**](#_yd3359za4ru9) **32**

[6.8.1 SearchingService](#_rjwocmw8lex8) 33

[6.8.2 LoginService](#_aw5jwihl17ev) 34

[6.9 Order](#_g8cv5nw2ptzh) 34

[6.9.1 ManageOrder](#_ahhhc4br25r6) 35

[6.9.2 Order](#_rs7vjpamc67a) 35

[6.10 ShoppingCart](#_knumy6jzeg39) 36

[6.10.1 ManageShoppingCart](#_doceplodtgyn) 37

[6.10.2 ShoppingCart](#_ehf5af9dfn8a) 37

[6.11 service\_plan](#_2kqeo07q4hw1) 38

[6.11.1 ServicePlan](#_fbrbghr9vjhy) 39

[6.11.2 MassageService](#_ukdozsi48gjw) 39

[6.11.3 AirtimeService](#_erobhvpsvpel) 40

[6.11.4 FullSpeedShareService](#_pv12hxps1j2x) 40

[6.11.5 ExtraOffer](#_zi3nz4lu5h2i) 40

[6.11.6 ServicePlanStorage](#_ke4fi2pbgpo9) 41

[**7. any patterns used in the class diagram**](#_bjgiztoo5ckf) **42**

# Design constraints

There were few constraints and limitation during the development, they are operating system platform, user interface, and database.

For the operating system platform, this program supports Mac OS, Windows, and Linux platform since the program was designed with Java language, the Java environment is available for the above platform. On the other hand, the program would able support mobile platform and web platforms due to programming language limitation, such as iPhone, Android, Chrome and Firefox. Concerning the database design, we don’t have a database for this program since this is a small program, we don’t want the program to complicated. Therefore, the data would be saved in runtime.

In addition, the command line interface may not good as the graphical user interface because the graphical user interface development is time-consuming. Sometimes, the command line interface may difficult to follow, a user needs to read the description before he/she performs an action. In contrast, the graphical user interface is iconic and easy to follow. Also, it makes the program more user-friendly and it provides a good user experience with a graphical user interface. To accelerate the development progress, a console program was chosen despite the graphical user interface could provide the best user experience.

In conclusion, there were few constraints and limitation of this program, such as available platform and user interface. For the user interface, we think that a command line interface should be enough for the project scope. The above constraints and limitation wouldn’t be the main concern for this project.

# 2. Requirement specifications

About the user requirement, a client wants to view all the plans before the selection. However, if the number of the plan is very large, they want to use different condition to filter the plans. Moreover, they want the system having a shopping cart to save their favorite plan that orders it together. After adding some plan to the shopping cart, a management function is required to manage their shopping cart before the checkout function.

In the admin side, they want to manage their plan that adds a new plan, update the plan detail or delete the plan. checking the order list should be included in the function since they want to view the order list to facilitate them to follow up on orders.

In the system specifications, we set some specifications in the system

1. Input

a. Function selection

Description: the program provided some function for the user, and it selects by user input

Format: Number (e.g. 1,2,3..) or English alphabet(e.g. C=Checkout)

Exception: If the data type is incorrect in the current selection, a message will be displayed to remind the user.

b. Plan selection

Description: the program provided view plan detail function that to display the detail information of the selected plan.

Format: String(P + Number e.g. P001, P002)

Exception: If the input is invalid, a message will be displayed to remind the user.

c. Add or Delete shopping cart record function

Description: the program provided add or delete shopping cart record function to manage the added record.

Format: Number (e.g. 1,2,3..)

Exception: If the input is invalid, a message will be displayed to remind the user.

d. Manage the plan function

Description: the program provided manage the plan.

Format: String (e.g. description) or Number (e.g price)

Exception: If the input is invalid, a message will be displayed to remind the user.

e. Account registration

Description: system allow a user to register a new account and allow each string as the username and password

Format: English alphabet (e.g. a,b,c… ) or Number (e.g 1,2,3…)

2. System

a. Account

Description: User must be login to unlock system function.

b. User type

Description: Client account cannot manage the plan and check order list.

c. Result display

Description: system should give some response to a user when they input. The respond must match the user input.

3. search result

a. Shopping cart

Format:

The number of shopping cart record + ”. ” + Plan Name

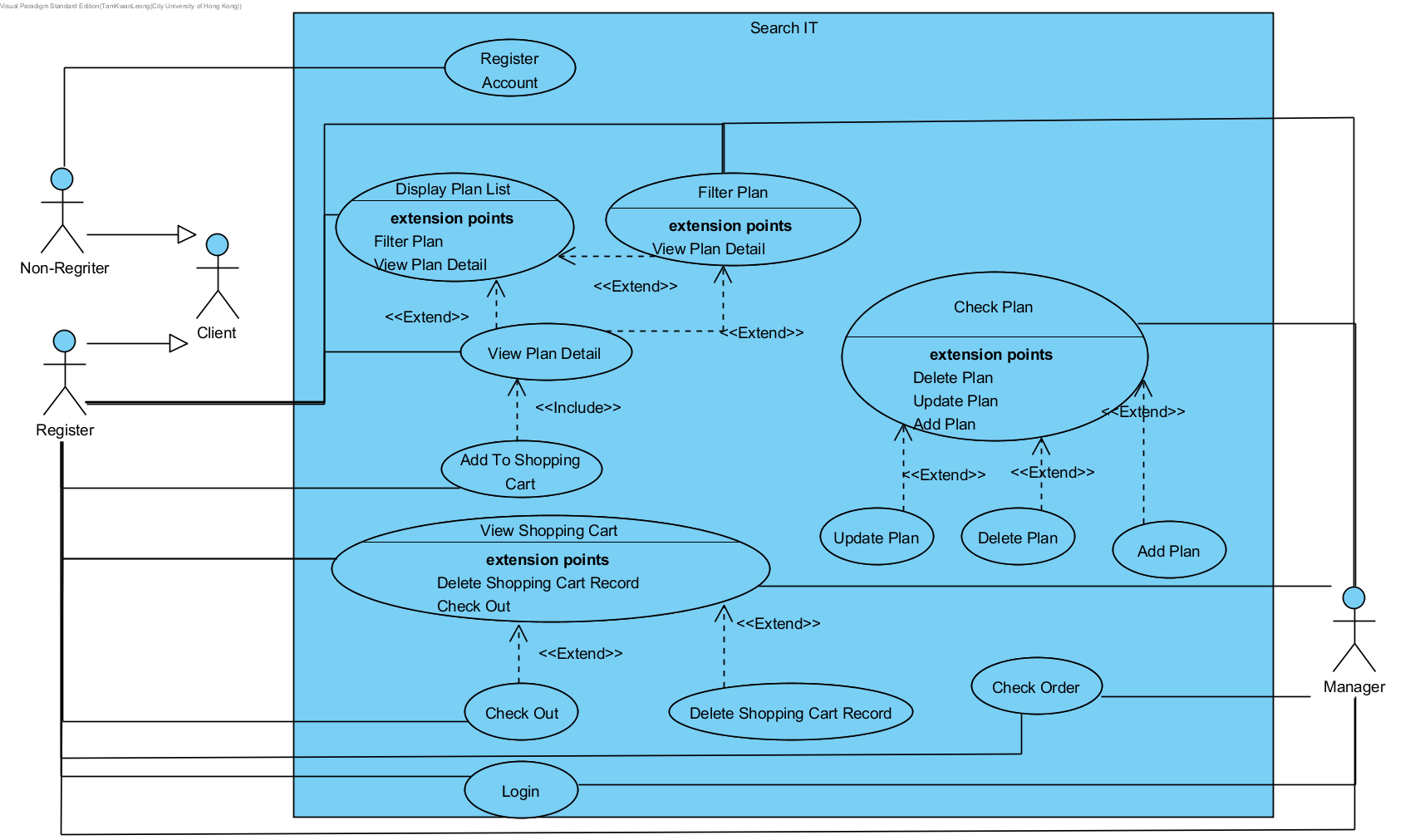
“Duration: ” + Plan Duration and Unit

b. Show all plan

Format:

Plan ID + “: ” + Plan Name + “-” +Fee

# 3. Use case diagram



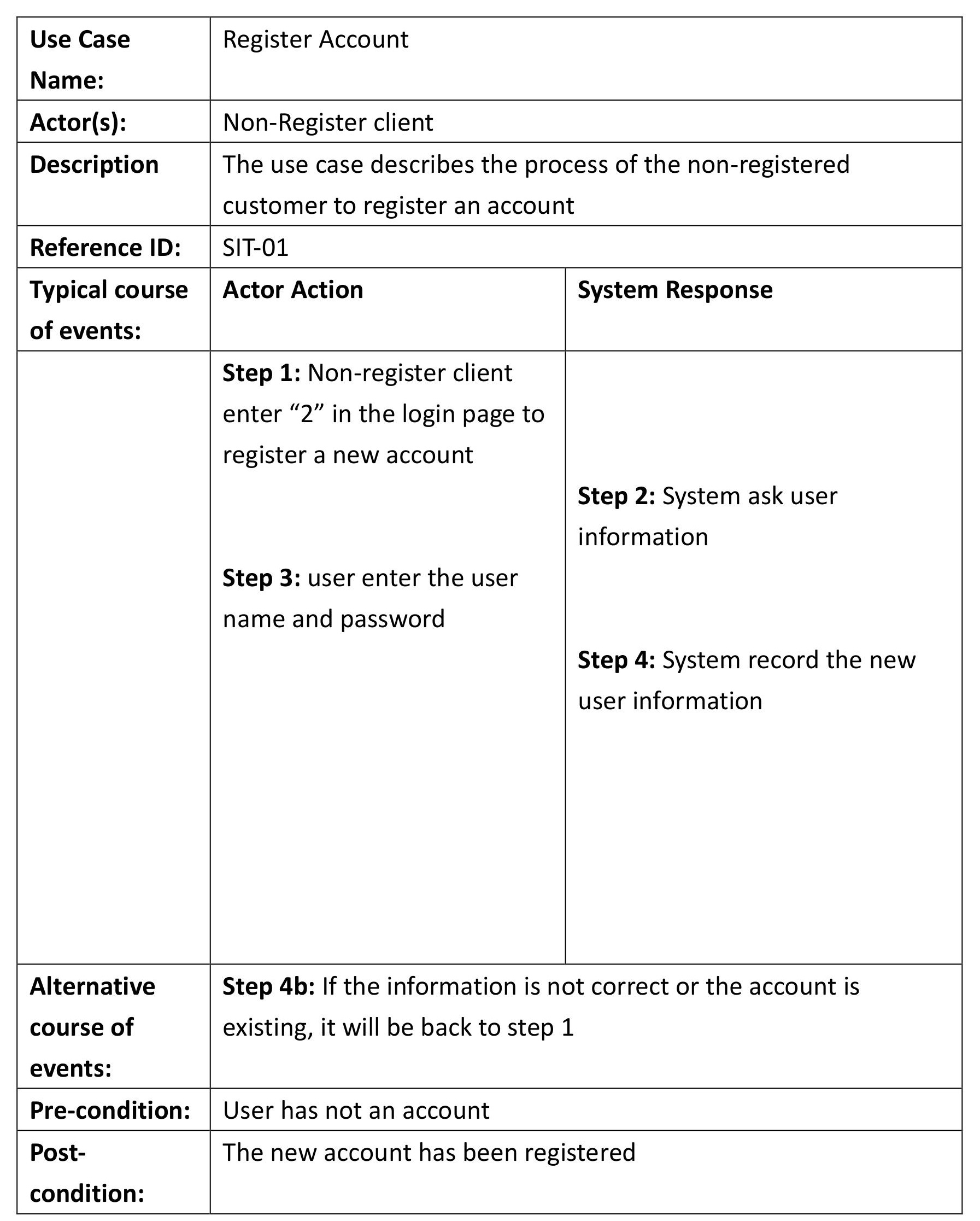
This is the use case diagram of Search IT system. There are 3 actors include non-registered client, registered client, and manager. The system can be divided into client-side and manager side.

In the client side, the system allows a non-registered client to register a new account for unlocking different function. About the registered client, it allows them to display the plan list and filter it through different condition such as price. After that, they can view the plan detail and add the selected plan to the shopping cart. Moreover, the client can check their added record through the view the shopping cart function and manage their shopping cart by checkout function and delete shopping cart record function. After the checkout, the shopping cart record will be submitted to the order list.

In the managed side, they have the function of the client. They also can check the order list that to view the client’s submit record. In addition, the system allows them to check the plan and manage it's through update plan function, delete plan function and add plan function.

# 4. Use case specification

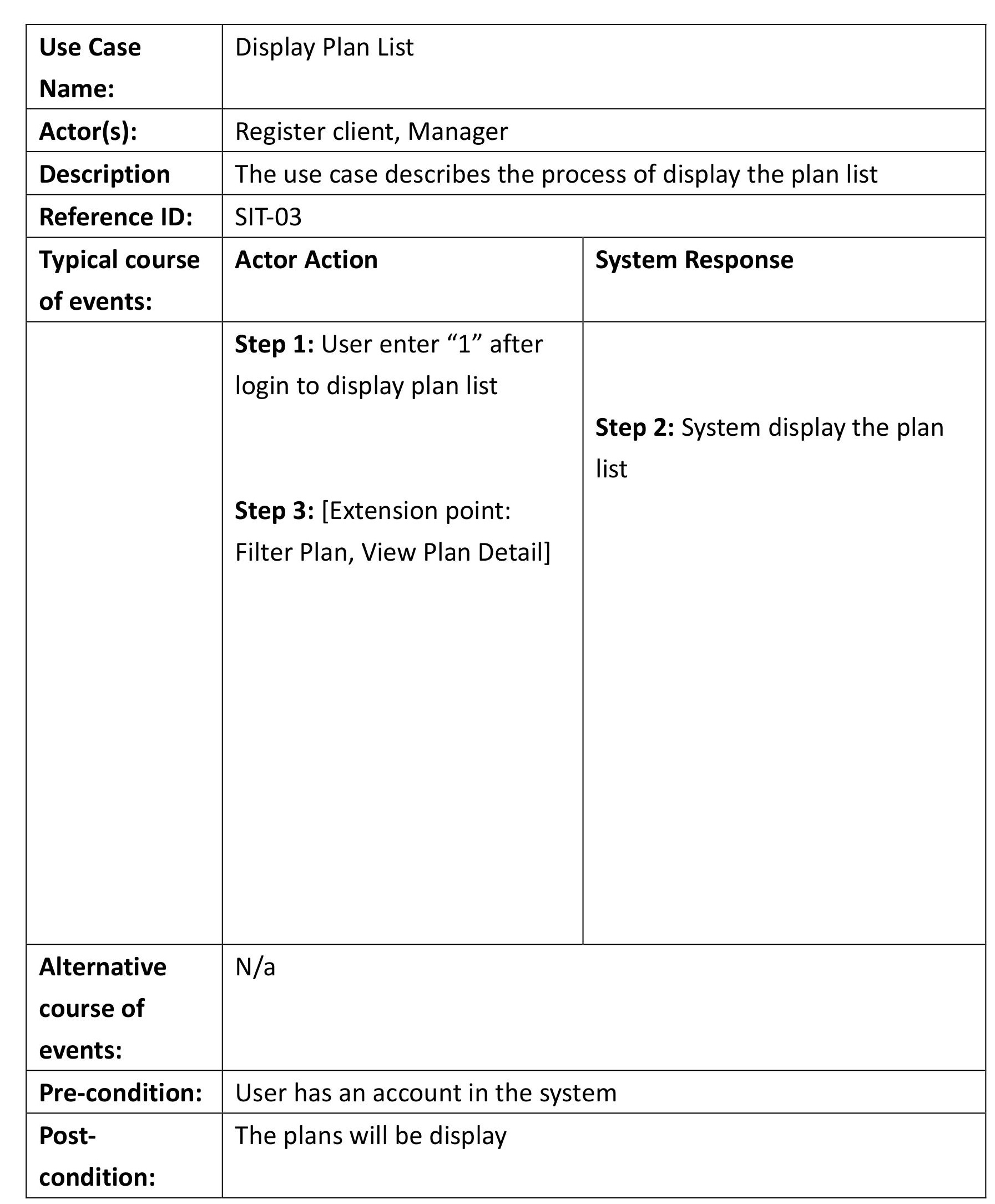
### 4.1 Register Account



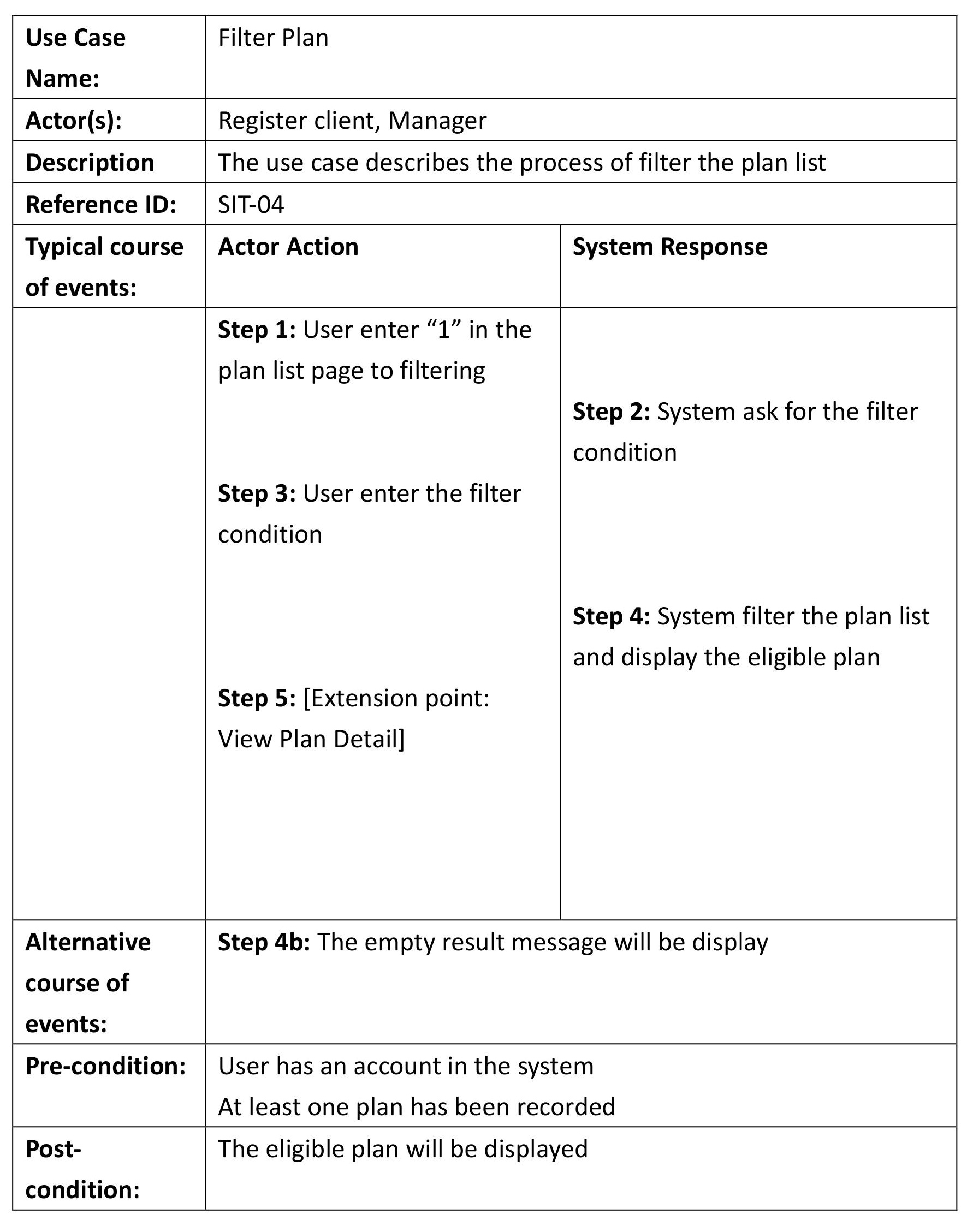
### 4.2 Login



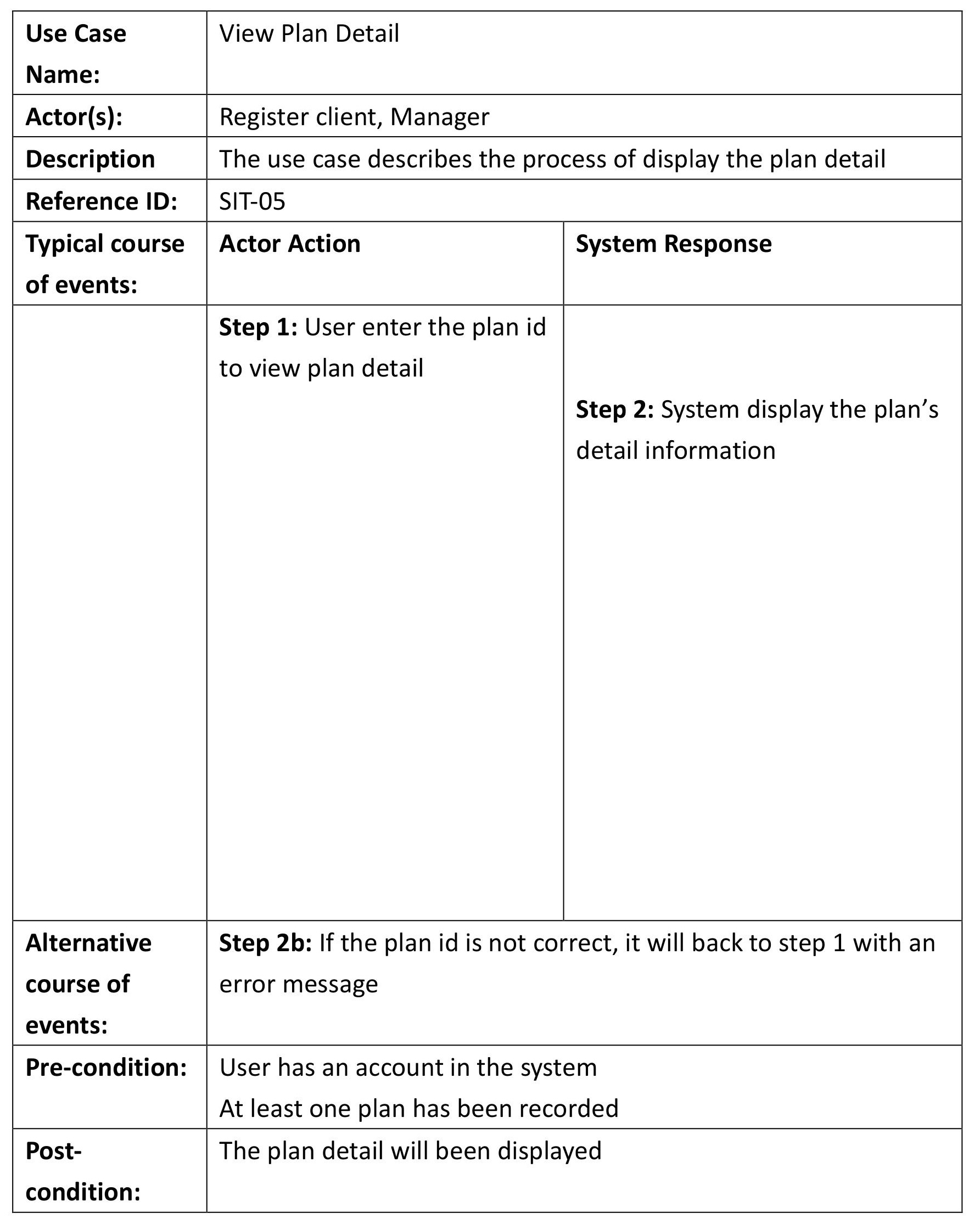
### 4.3 Display Plan List



### 4.4 Filter Plan



### 4.5 View Plan Detail



### 4.6 Add To Shopping Cart

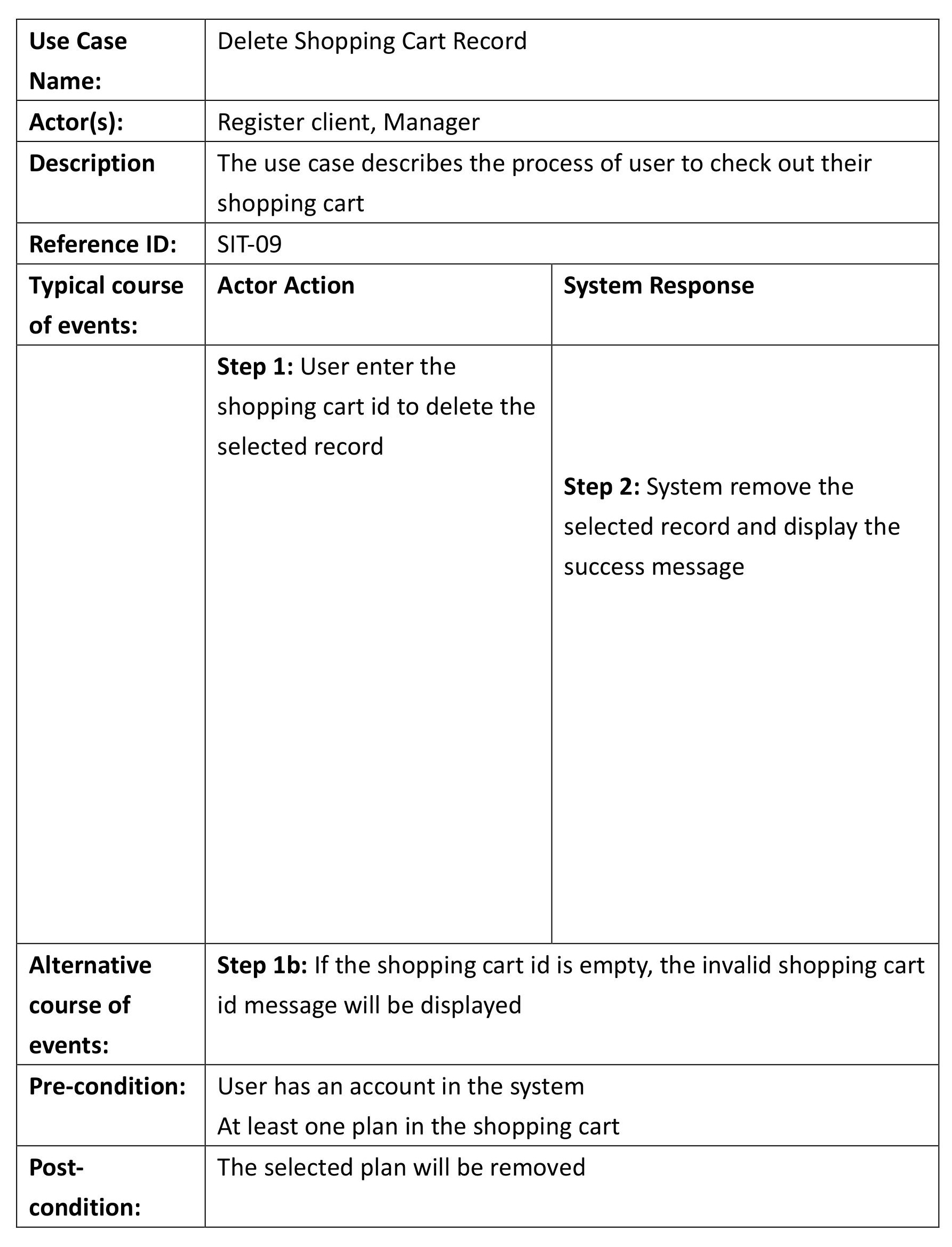


### 4.7 View Shopping Cart

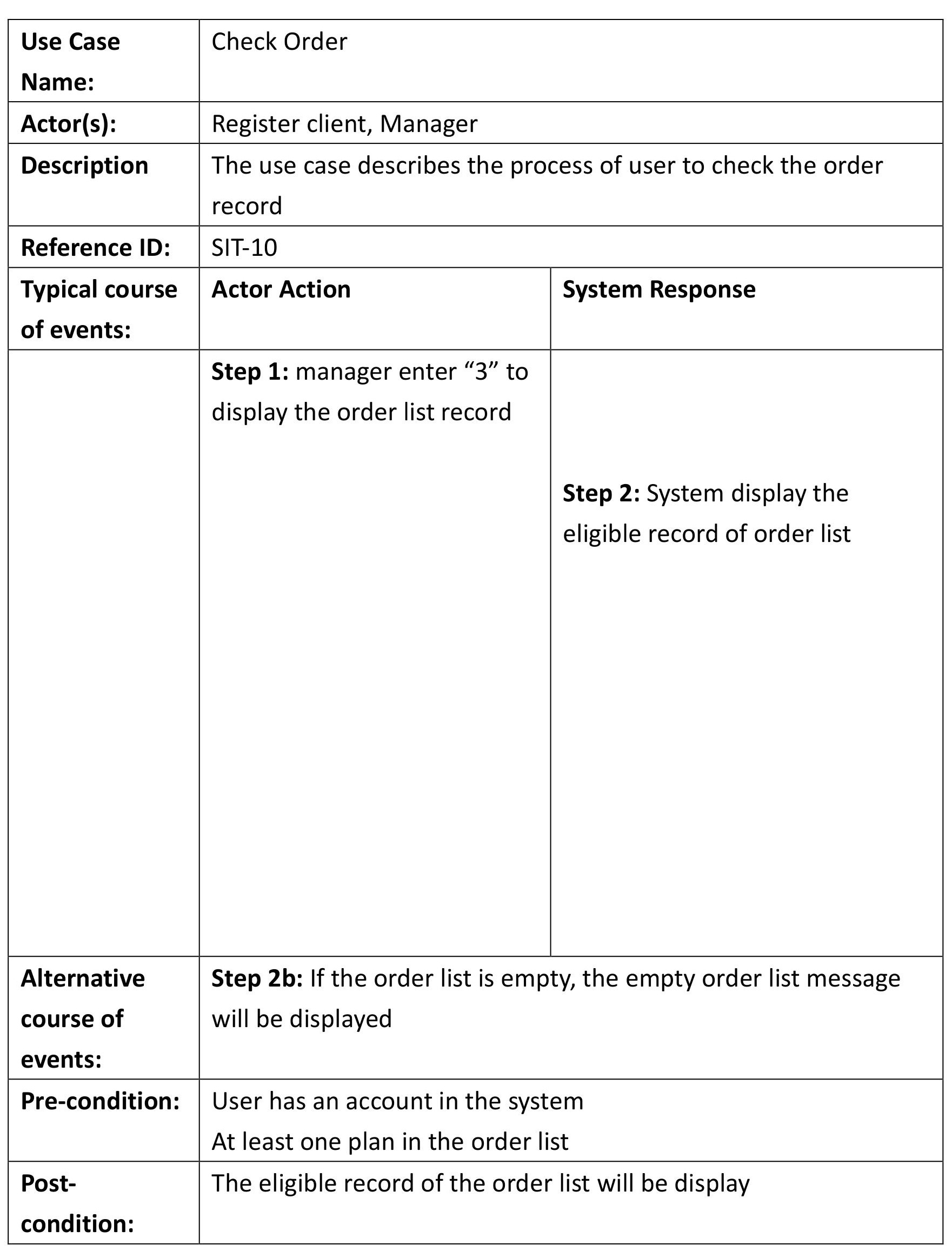


### 4.8 Check Out

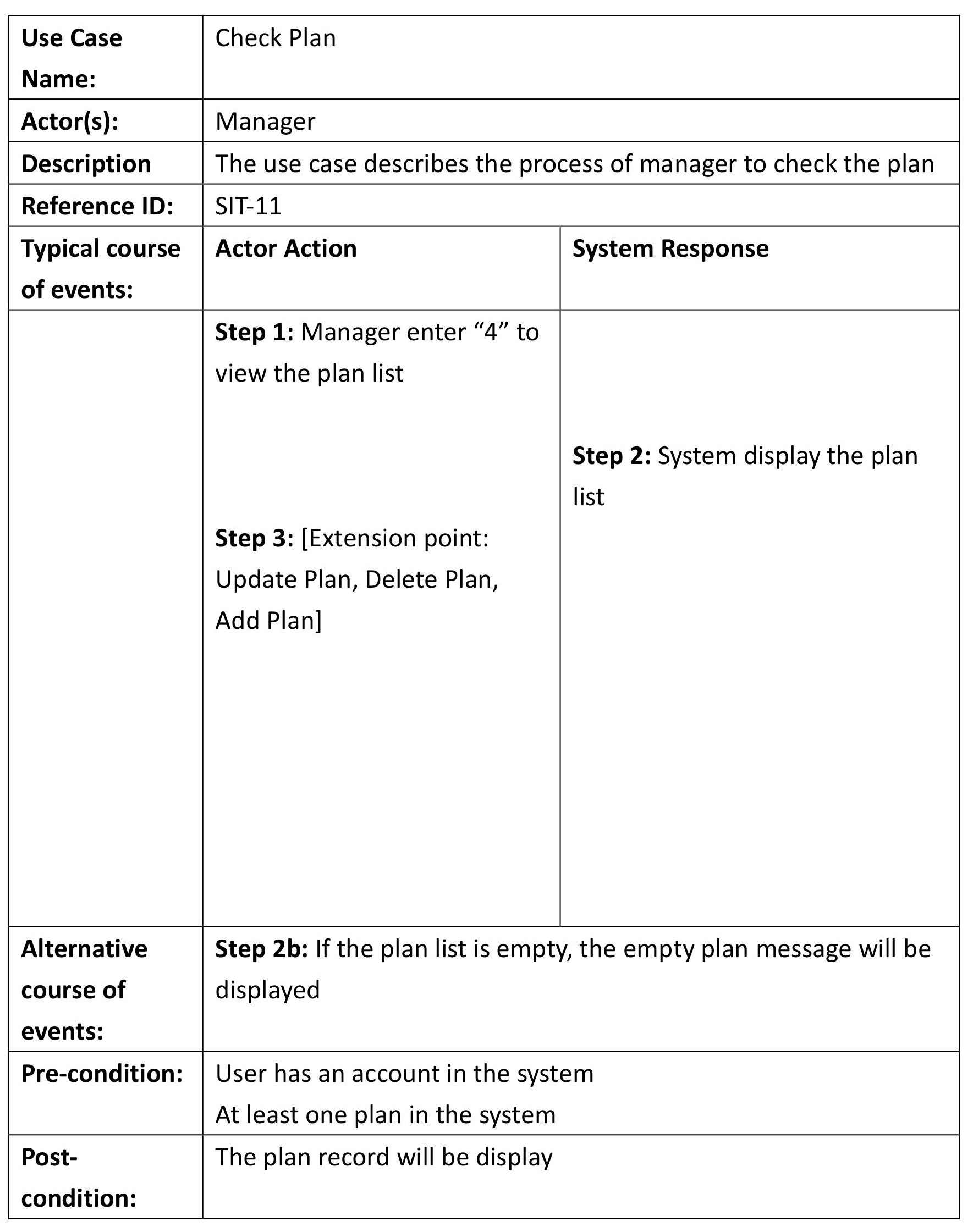
### 4.9 Delete Shopping Cart Record



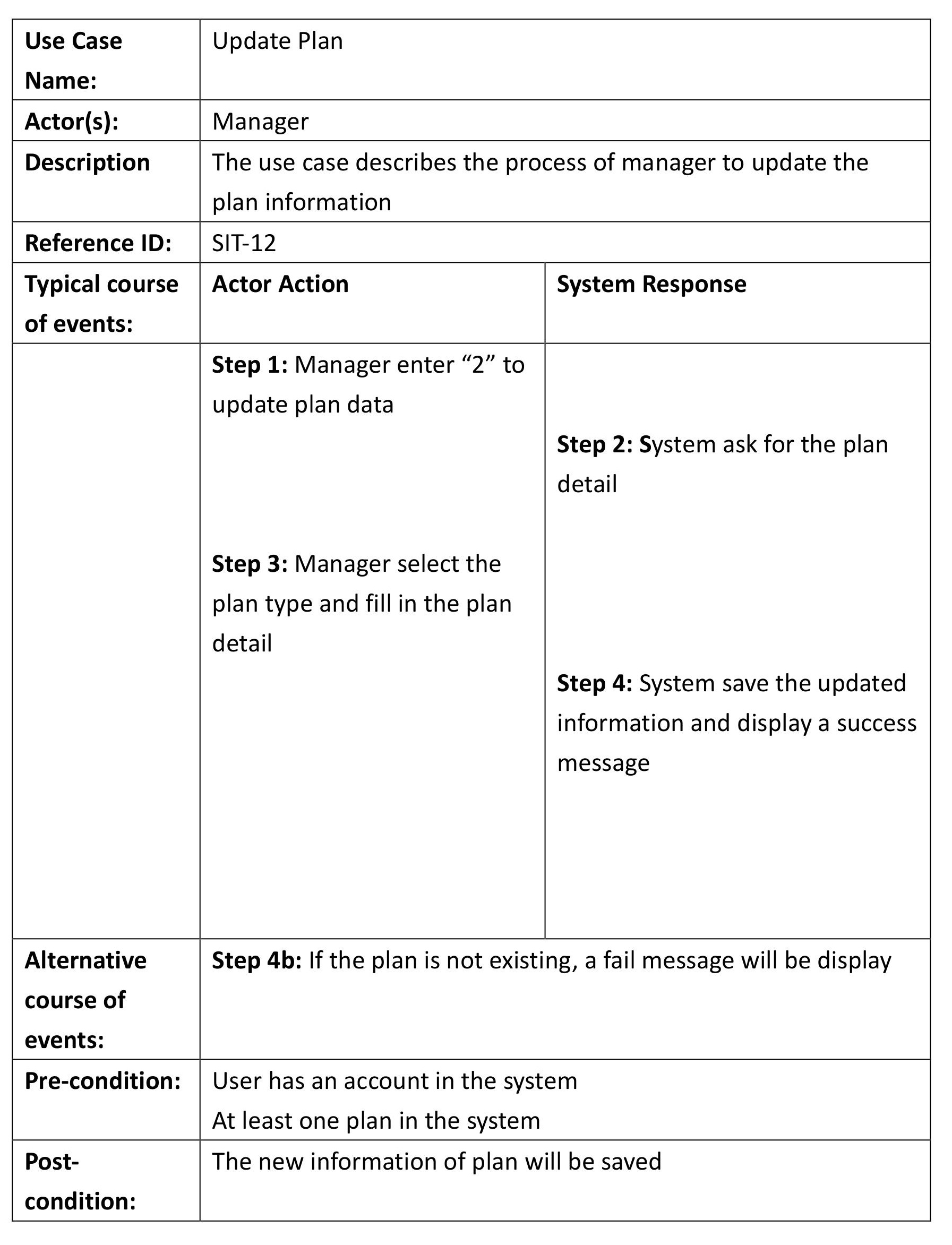
### 4.10 Check Order



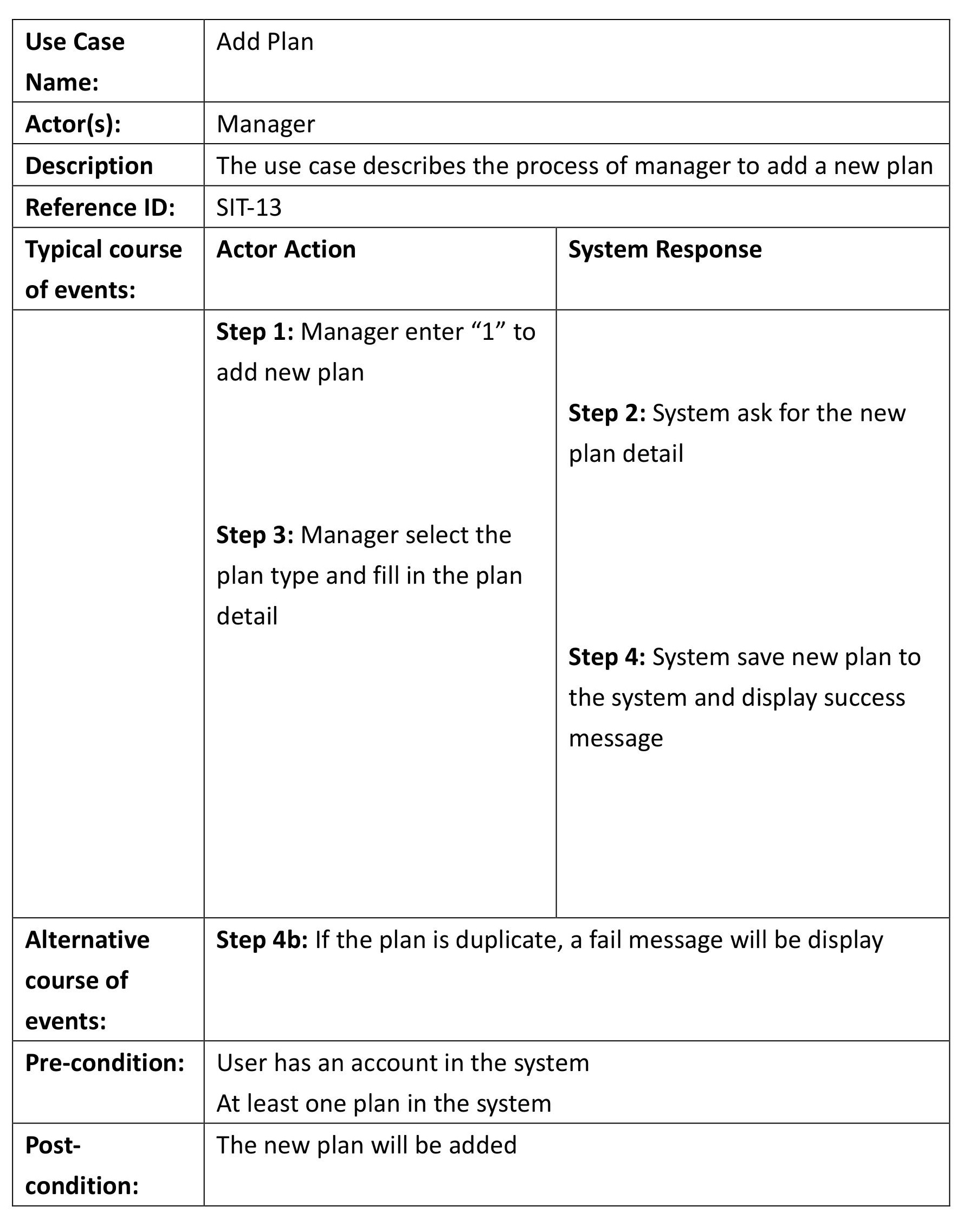
### 4.11 Check Plan



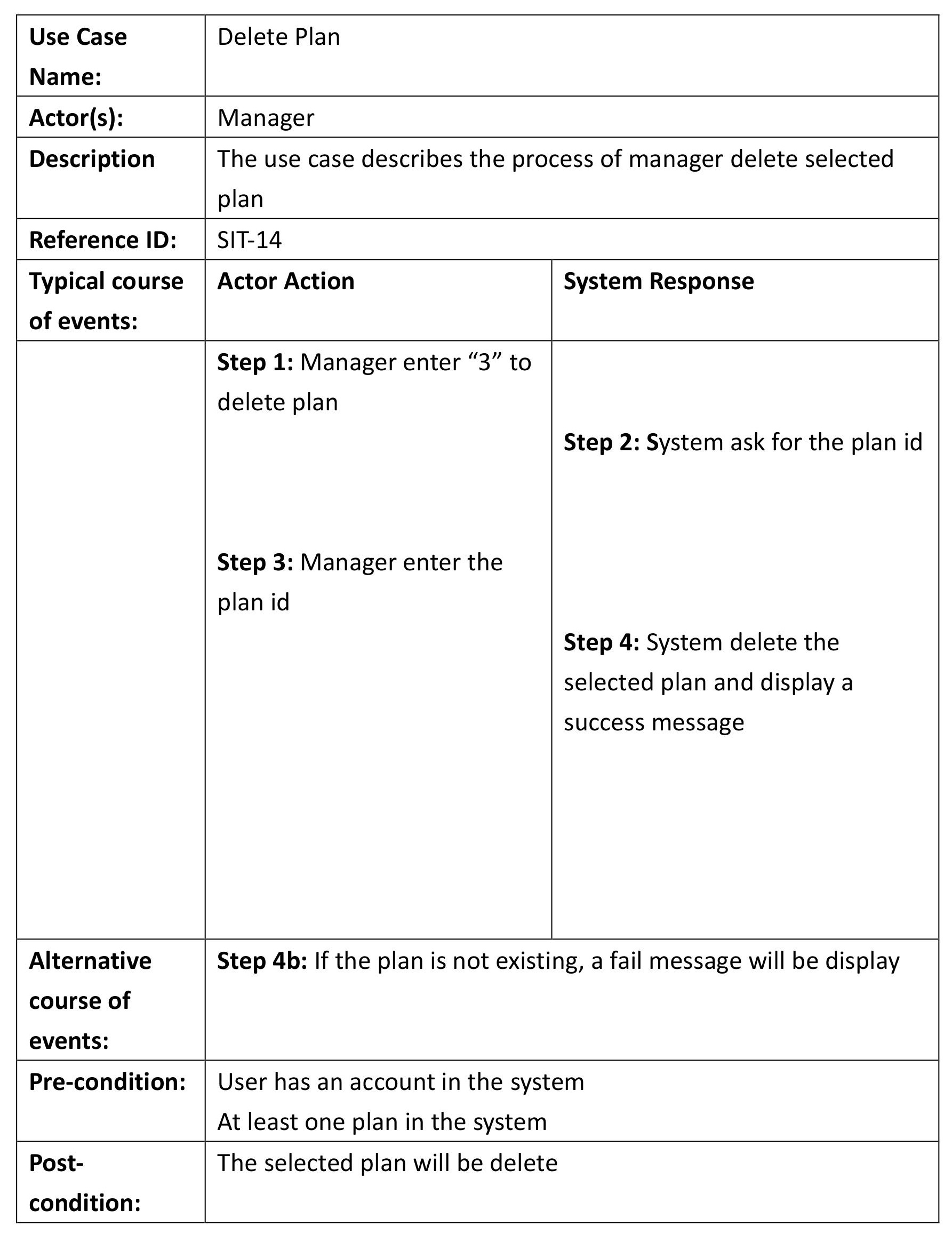
### 4.12 Update Plan



### 4.13 Add Plan



### 4.14 Delete Plan



# 5. Class diagram

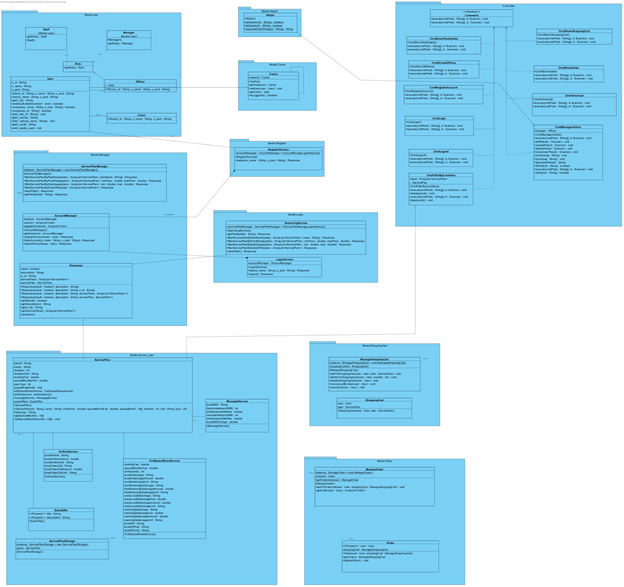
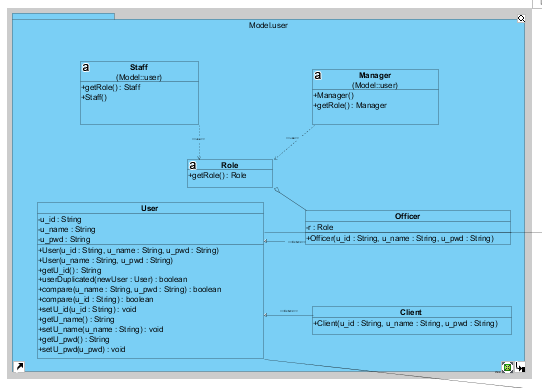


Figure 5.1 class diagram

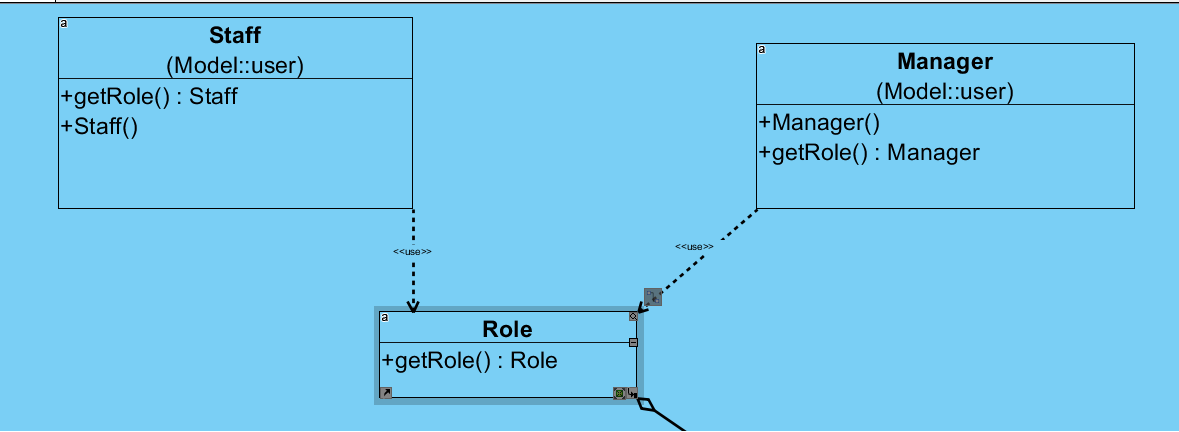
# 6. Class description

## 6.1 Model.user package



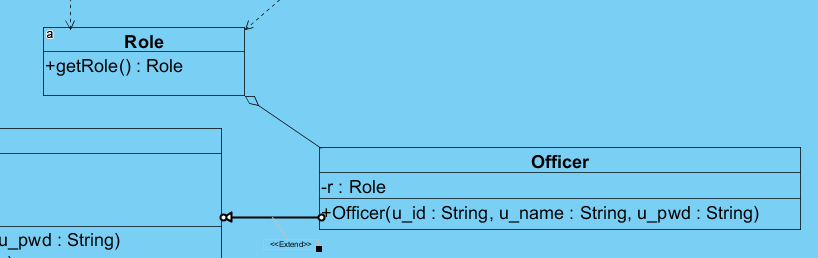
Fou officer, there is two roles of the officer which are a manager and a staff.

### 6.1.1 Role



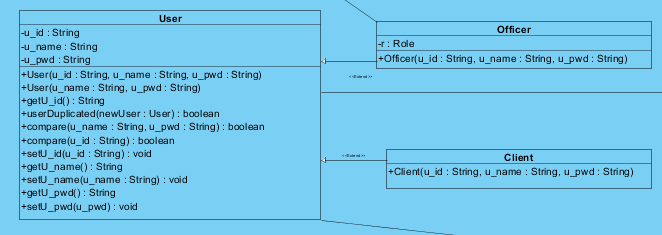
Role is an interface and it contains getRole class.Staff and manager is a type of roles. When a new role has been created, the new role only needs to implement the role class.

### 6.1.2 Officer



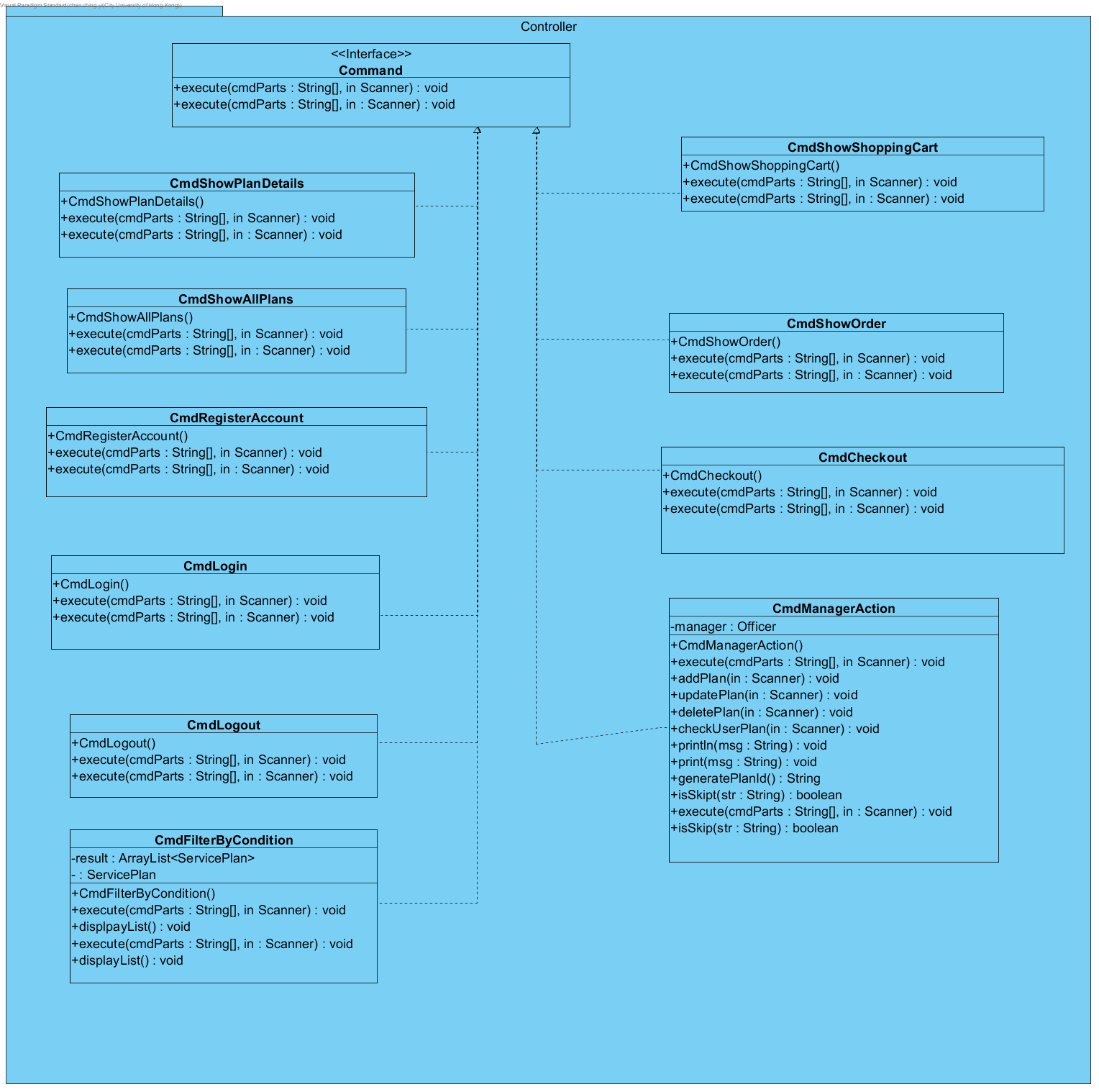
The role class is designed for officer class to determine the user type and setting the permission. The officer class created for adding officer account. It also can pick the officer’s type.

### 6.1.3 User



For login, there are two types of the user which are Officer and Client. The user class handles different type of user. Also, it contains the variables and function which is existing in every user type. Compare function is designed for login. Also, userDuplicatated is for checking a username that it is existing.

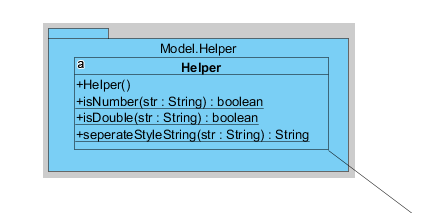
## 6.2 Controller



It groups all command action. There are CmdShowAllPlans, CmdShowOrder, CmdRegisterAccount, CmdCheckout, CmdLogin, CmdManagerAction, CmdLogout, CmdFilterByCondition.

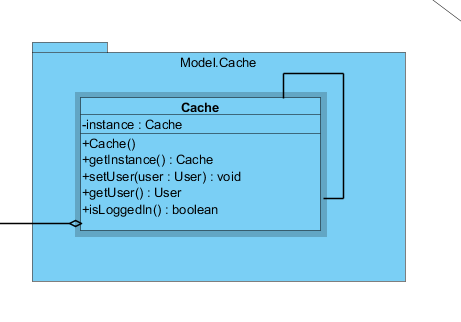
* CmdRegister calls RegisterService class for registering an account.
* CmdFilterByCondition is handling any action call with service plan search.
* CmdMsnsgerAction is handling any manager function such as add plan, delete plan, update plan etc.
* CmdLogout is handling logout function
* CmdShowAllPlan is displaying all plan
* CmdShowOrder is displaying all order which takes by the user or for officer check the existing order.

## 6.3 Helper



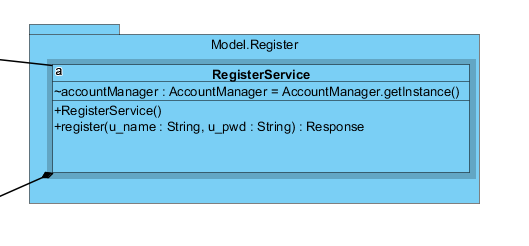
This is the supplementary class for checking the value valid or not. It includes isDouble and isNumber function.

## 6.4 Cache



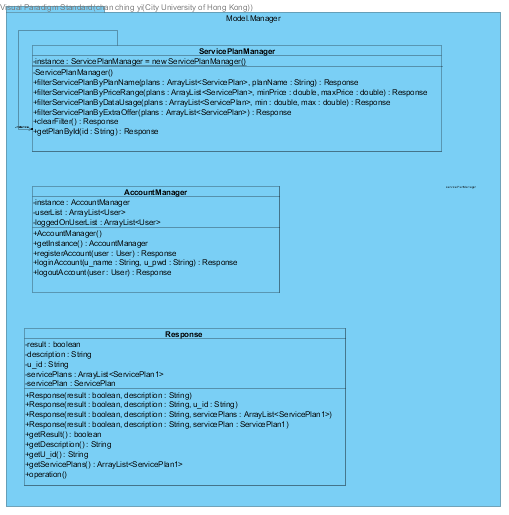
The Cache class stores the user data, when they are signed in.

## 6.5 Register



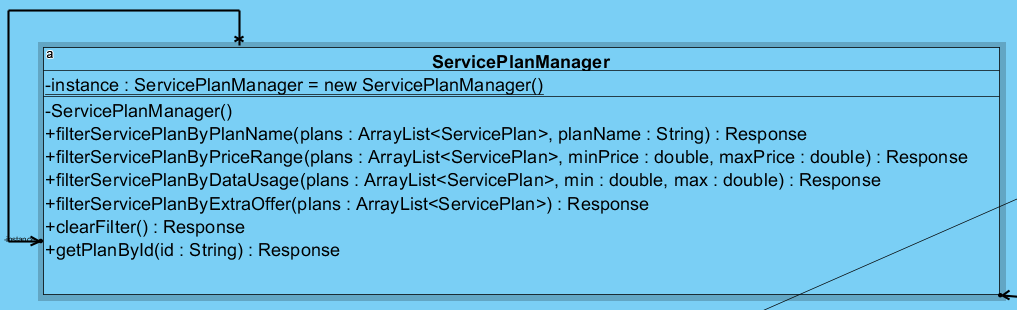
When the system received a signal that they want to create an account. The RegisterService function will occur by CmdRegister. First, it will create a user model. Then, add it to the User list. When the creation process is successful, it will return a response object.

## 6.7 Manager



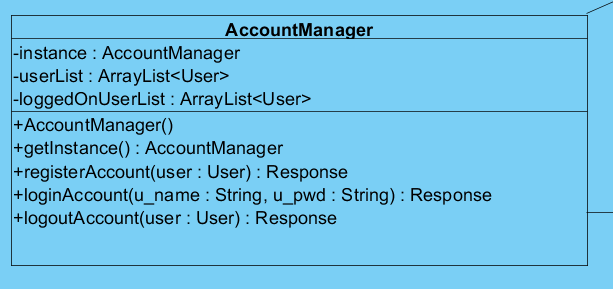
In the manager package, it has three classes which are ServicePlanManager, AccountManager, and Response. ServicePlanManager is handling ….. Response class is handling the response result of any action such as account registration, service plan handling, order taking, etc. AccountManager is handling any action with the account such as login, logout, account storing, etc.

### 6.7.1 ServicePlanManager

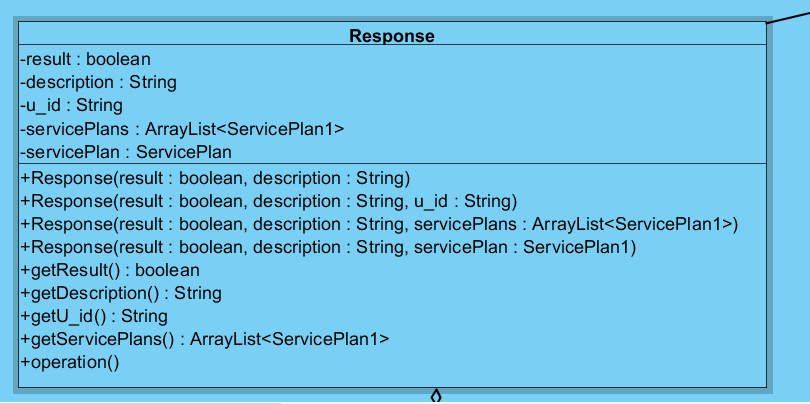


### 

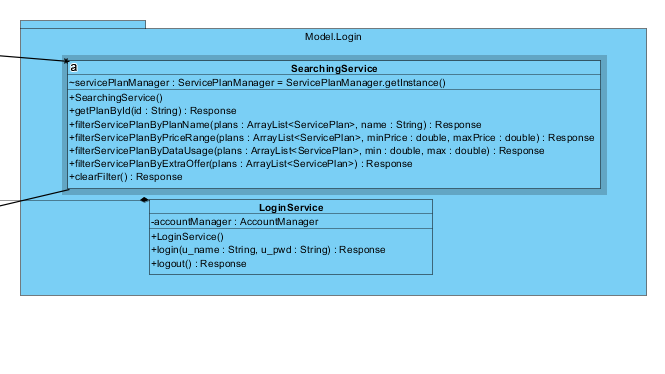
### 6.7.2 AccountManager



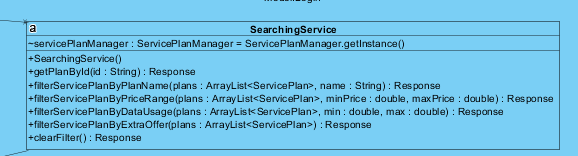
### 6.7.3 Response



# 6.8 Login

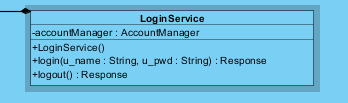


### 6.8.1 SearchingService



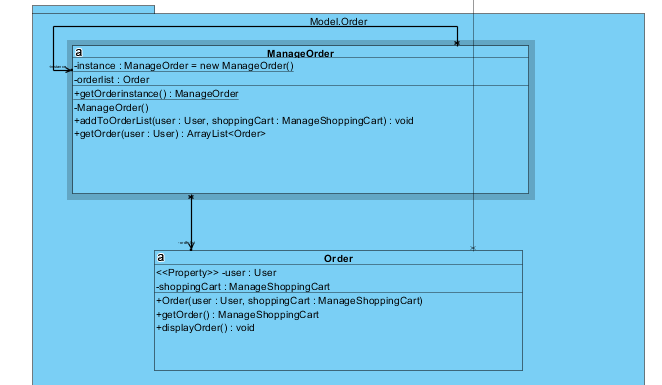
This class is handling service plan searching function with some criteria such as plan name, price range, data usage.

### 6.8.2 LoginService

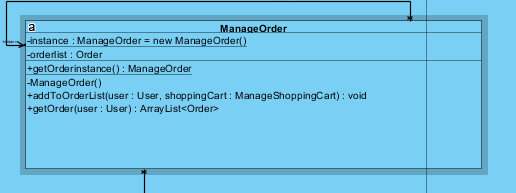


This class is handling the login process. When a login is successful, the login log will log to the Cache class. Also, it will return a response object, when the login and logout process had been occurred.

## 6.9 Order

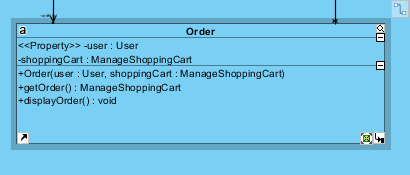


### 6.9.1 ManageOrder



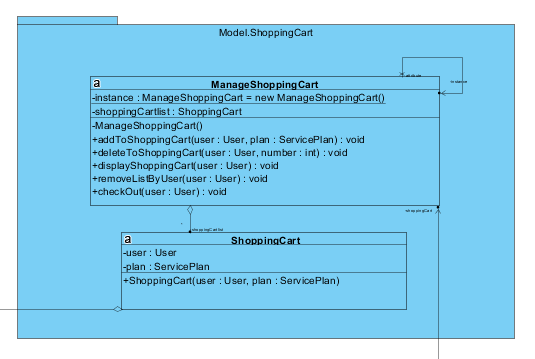
The Order class is handling order making. The order list stored an order or few of order when they are purchased. Officer and user can check their order which is not purchased.

### 6.9.2 Order

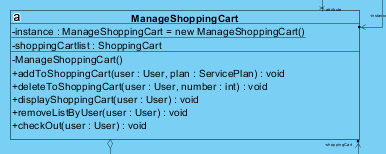


This is an order object, it includes some information of an order such as user information, selected service plans. We also can display the selected service plan.

## 6.10 ShoppingCart

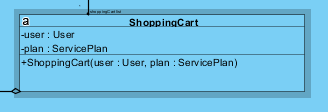


### 6.10.1 ManageShoppingCart



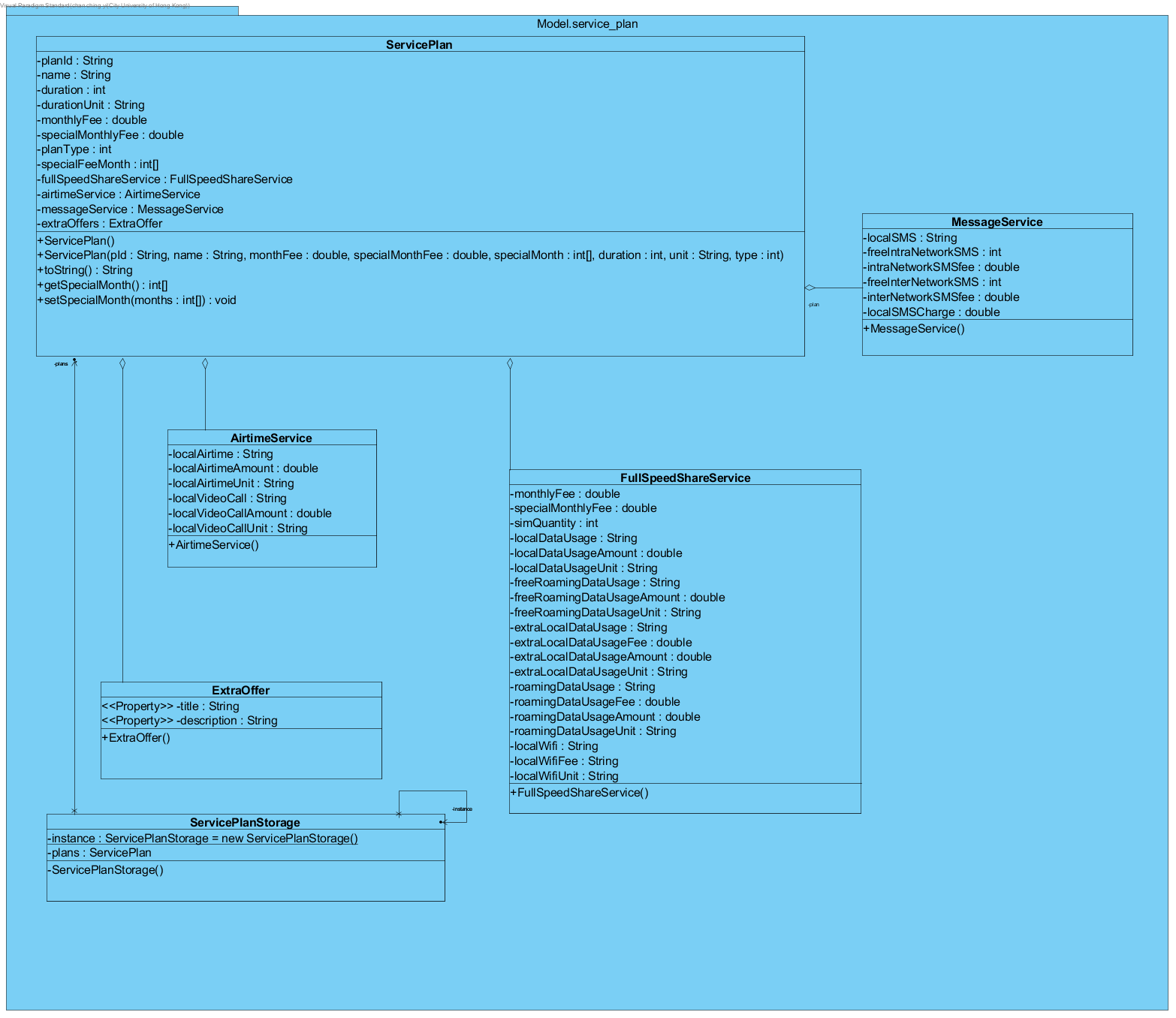
ManageShoppingCart uses for shopping cart management. It includes adding an item to the shopping cart, deleting an item in the shopping cart, listing the shopping cart items, clearing shopping cart, and check out the shopping cart.

### 6.10.2 ShoppingCart

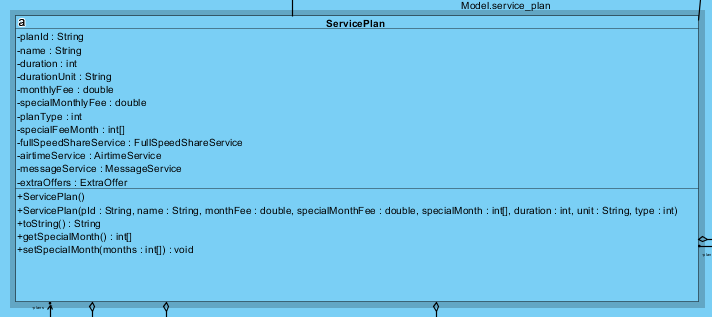


It is a shopping cart object. It includes a user object and service plan.

### 6.11 service\_plan



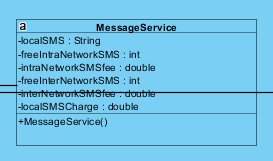
### 6.11.1 ServicePlan



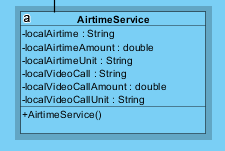
This is a service plan object, it includes the plan id, name, duration of years, monthly of fee, plan type, etc. also, it collects different service function information.

In figure 6.11.2, 6.11.3, and 6.11.4, they are the service plan information of different criteria such as message service, airtime service, and full speed service. When we are searching for a service plan, we can get the service plan details.

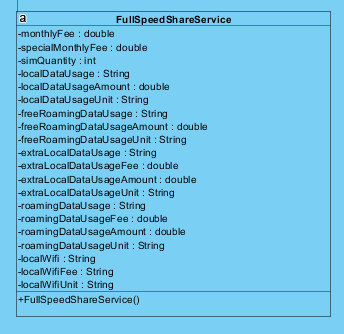
### 6.11.2 MassageService



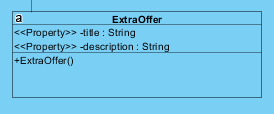
### 6.11.3 AirtimeService



### 6.11.4 FullSpeedShareService

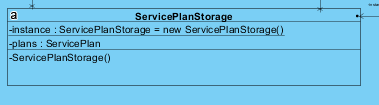


### 6.11.5 ExtraOffer



For providing a special discount or an offer to the user, the manager can add an extra offer in the system. Then, the officer can promote it and a user can search it in the system.

### 6.11.6 ServicePlanStorage



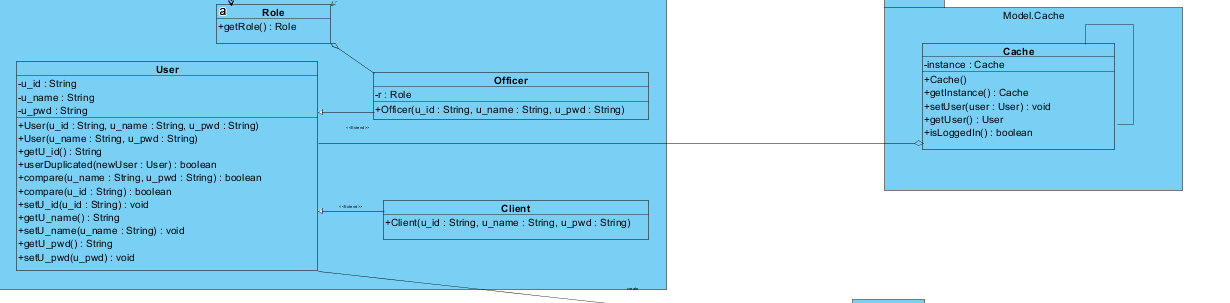
It is a container for storing all service plan.

# 7. any patterns used in the class diagram

1. Singleton pattern

Singleton pattern is a software design pattern that restricts instantiation of a class to an object. This is useful when only one object is needed to coordinate the operation of the whole system. This concept is sometimes extended to systems that operate more efficiently when only one object existing or systems that limit instantiation to a certain number of objects. The term comes from the mathematical concept of bachelors.

In the project, the example of applying singleton pattern is [Cache] model class.

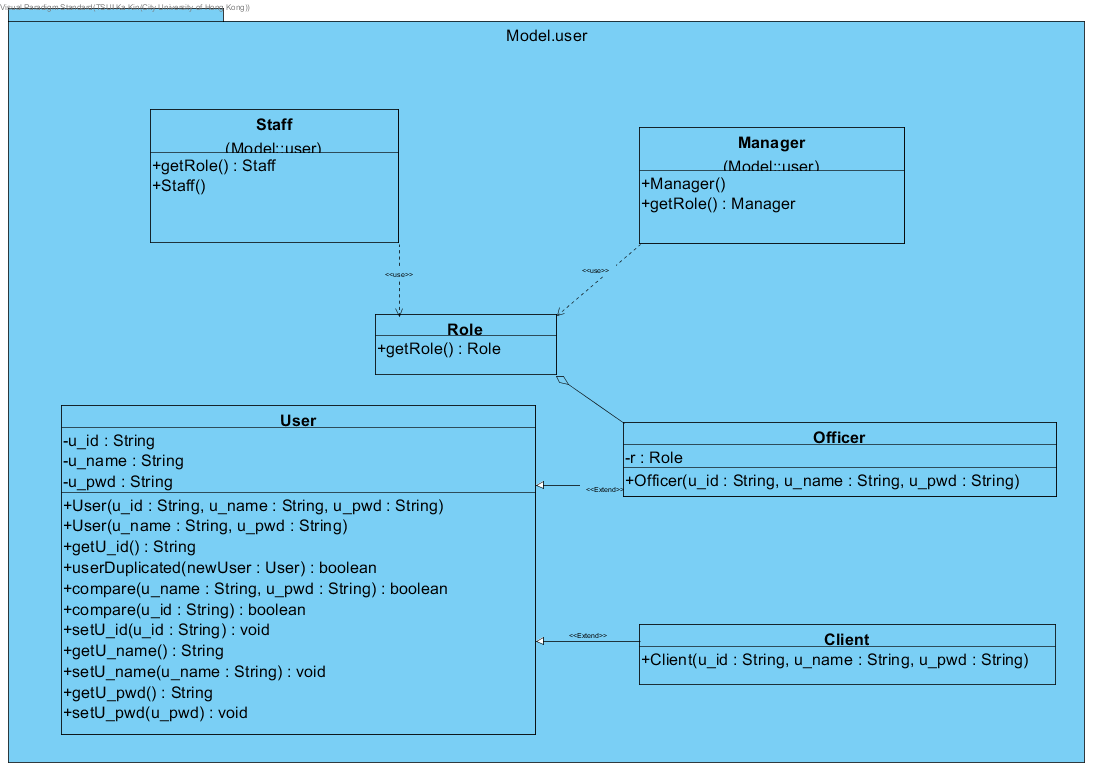


[Cache] is used for saving signed in the state of a user to check do you have any user who has logged in.

1. State pattern

State pattern is a behavioral software design pattern that allows an object to change its behavior when its internal state changes. This pattern is close to the concept of limited state machine, but it must be remembered that this pattern is not a software implementation of limited state machine. The state mode can be interpreted as a policy mode, which can switch current policies by calling the methods defined in the mode interface.

For the [Officer] model class, it contains a [Role] interface class, which is applying state pattern, to divide into different roles. In the [Role], there is a method getRole, and then the [Staff] and [Manager] also implemented the getRole.



1. Command pattern

In object-oriented programming, command pattern is a behavior design pattern in which objects are used to encapsulate all the information needed to perform operations or trigger events later. This information includes the method name, the object that owns the method, and the value of the method parameter.

The four terms associated with command mode are command, receiver, caller, and client. The command object knows the receiver and calls the method of the receiver. The parameter values of the receiver method are stored in the command. Receiver objects that execute these methods are also stored in command objects through aggregation. Then, when the EXECUTE () method in the command is called, the receiver performs the work. The caller object knows how to execute the command and has the option to account for the execution of the command. The caller does not know any specific commands, it only knows the command interface. Caller objects, command objects and recipient objects are held by client objects. The client decides which recipient objects to assign to command objects and which commands to assign to the caller. The client decides which commands to execute at which point. To execute a command, it passes the command object to the calling program object.

In the project, there is a [Controller] package to storage the command interface and classes as the invoker and command. Then, the main class should be a client to receive actions from the user. After that, the main class would perform different command via the invoker by the client’s input. Finally, the command would update data storage or return a result to the receiver, and then turn to terminate itself.

