

# Functional Requirements

**Group Name:** Group 6\_b

Jessica (JI) Lessev	13049136
Thabang (TM) Letageng	13057937
Michelle Swanepoel	13066294
Prenolan Govender	13102380
Fako (FJA) Peleha	12230830
Lutfiyya Razak	10198408
Ephiphania Munava	10624610
Maria Qumalo	29461775

**Git repository link:**

[https://github.com/u12230830/COS301\\_6b](https://github.com/u12230830/COS301_6b)

**Date:** 27 February 2015

## Contents

<b>BuzzSpace Handling</b>	<b>5</b>
1 Create BuzzSpace	5
2 Read BuzzSpace	6
3 Update BuzzSpace	6
4 Delete BuzzSpace	7
<b>Thread Handling</b>	<b>9</b>
1 Create Thread	9
2 Read Thread	12
3 Update Thread	16
4 Delete Thread	20
5 Summarise Thread	24
6 Hide Thread	26
7 Close Thread	29
8 Move Thread	31
<b>Post Handling</b>	<b>34</b>
1 Create Post	34
2 Read Post	35
3 Update Post	35
4 Delete Post	36
5 Mark solution post	37
6 Vote and Evaluate Posts	38
7 Post Tagging	40
<b>Filtering</b>	<b>43</b>

1 Filter threads and buzzspaces	43
2 Filter threads and buzzspaces	48
Authorisation	49
1 Login	49
2 Logout	49
3 Registration	53
Reporting	55
1 Thread Evaluation Report	55
2 Student Post Evaluation Report	57
3 Audit Report	60
Profile Handling	62
1 Create Profile	62
2 Modify Profile	63
3 View Profile	65
4 Create Private Message	67
5 Read Private Message	67
6 Update Private Message	67
7 Delete Private Message	67
Domain Model	68

The following modules will be discussed, each with their own set of use cases:

- BuzzSpace Handling
- Thread Handling
- Posts Handling
- Filtering
- Authorisation
- Reporting
- Profile Handling

# BuzzSpace Handling

## Use Cases

### 1 Create BuzzSpace

#### Description:

This use case encapsulates the ability for a user to create a new BuzzSpace that will be associated with a specific active module. Each BuzzSpace will have a root thread to start with. The creator of the BuzzSpace will set properties of the BuzzSpace (i.e. which status is needed for a user to do what).

#### 1.1 Prioritization:

Critical

#### 1.2 Conditions and Data Structures:

##### Pre-Conditions:

- The user wanting to create a new BuzzSpace should be logged in to the system and should be registered with the module for which he/she wants to create the BuzzSpace.
- The user should have a high enough status to be allowed to create a BuzzSpace (i.e. lecturer status).
- The module for which the BuzzSpace is being created must be an active module at that time.

##### Post-Conditions:

- A message is displayed to inform the creator of the BuzzSpace that the creation has been successful. Users can now start more threads on this BuzzSpace.
- If this service has been rejected, an exception is thrown and an error message will be displayed to the user.

## **Requests and Results Data Structures:**

### **1.3 Required Functionality:**

### **1.4 Process Specifications:**

## **2 Read BuzzSpace**

### **Description:**

This use case includes the ability to view BuzzSpaces, this is, to view the different threads in the BuzzSpace (and thus have access to them), and also to view the properties of the specific BuzzSpace (i.e. the permissions that were set, date of creation and the creator of this thread).

### **2.1 Prioritization:**

Critical

### **2.2 Conditions and Data Structures:**

#### **Pre-Conditions:**

None

#### **Post-Conditions:**

- Threads can now be viewed and accessed, and the properties can also be viewed.

## **Requests and Results Data Structures:**

### **2.3 Required Functionality:**

### **2.4 Process Specifications:**

## **3 Update BuzzSpace**

### **Description:**

This use case gives the ability for a user with a high enough status (lecturer status) to change the permissions needed for certain actions of this BuzzSpace.

### **3.1 Prioritization:**

Important

### **3.2 Conditions and Data Structures:**

#### **Pre-Conditions:**

- The user must be logged in and registered for the module that the BuzzSpace is in.
- The user must have lecturer status to be able to update threads.

#### **Post-Conditions:**

- A message is displayed to let the user know that the update has been done successfully
- If the service was rejected, an exception is thrown and an error message is displayed to let the user know.

#### **Requests and Results Data Structures:**

### **3.3 Required Functionality:**

### **3.4 Process Specifications:**

## **4 Delete BuzzSpace**

#### **Description:**

This use case includes the ability to delete a BuzzSpace when the module is not active anymore, or the BuzzSpace is not used anymore. Only the creator of the BuzzSpace will be able to delete a BuzzSpace, except if he changed the property to let anyone of a certain status be able to delete it.

#### **4.1 Prioritization:**

Critical

### **4.2 Conditions and Data Structures:**

#### **Pre-Conditions:**

- The user must be logged in
- The user must be the creator of the thread or must have a high enough status (if the permission for this was customized).

#### **Post-Conditions:**

- The BuzzSpace is deleted and archived for future reference.

**Requests and Results Data Structures:**

**4.3 Required Functionality:**

**4.4 Process Specifications:**



# Thread Handling

## Use Cases

### 1 Create Thread

#### **Description:**

1. User activates Create Thread function by selecting the Create Thread option.
2. System responds by presenting user with a form to create thread topic etc
3. User fills in form topic of thread, tags, etc and submits the form
4. System reviews the submitted information and verifies users status and privileges, decorates the thread as needed and checks that all requirements (rules) are followed by the user based on his/her status
5. System displays either an acknowledgment or error message based on the previous checkpoint.

#### **1.1 Prioritization:**

This Use case is considered Important

#### **1.2 Conditions and Data Structures:**

Participants: Initiated by User and communicates with system

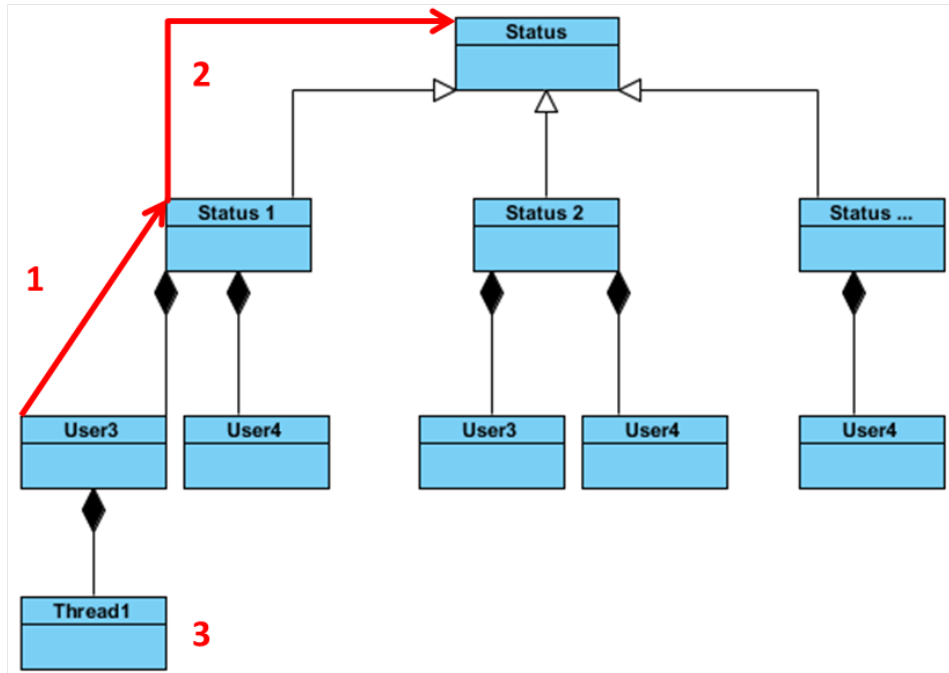
#### **Pre-Conditions:**

1. The User is logged in
  - a. The user has a specific status

#### **Post-Conditions:**

1. The user fills in the form to create a new thread
  - a. User must have a specific status in order to perform certain functions when creating a thread

## Requests and Results Data Structures:

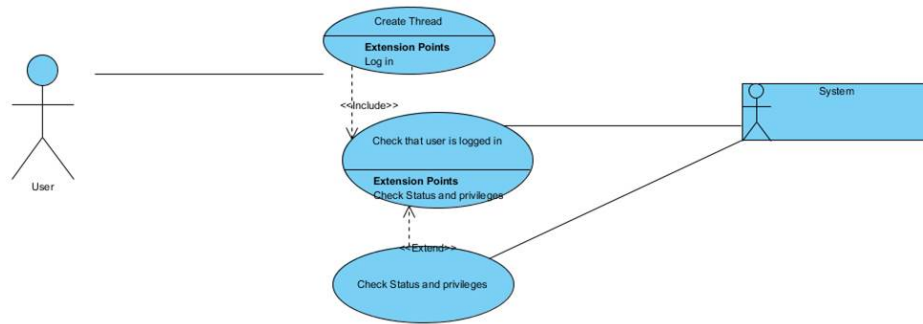


Status can be an abstract class and each sub-class will be a generalization of this class as each new status IS-A status. Each user HAS-A status and thus each new user will be a composition of each specific status class. Each thread HAS-A user that started it and thus each new thread will be a composition of each user.

Steps:

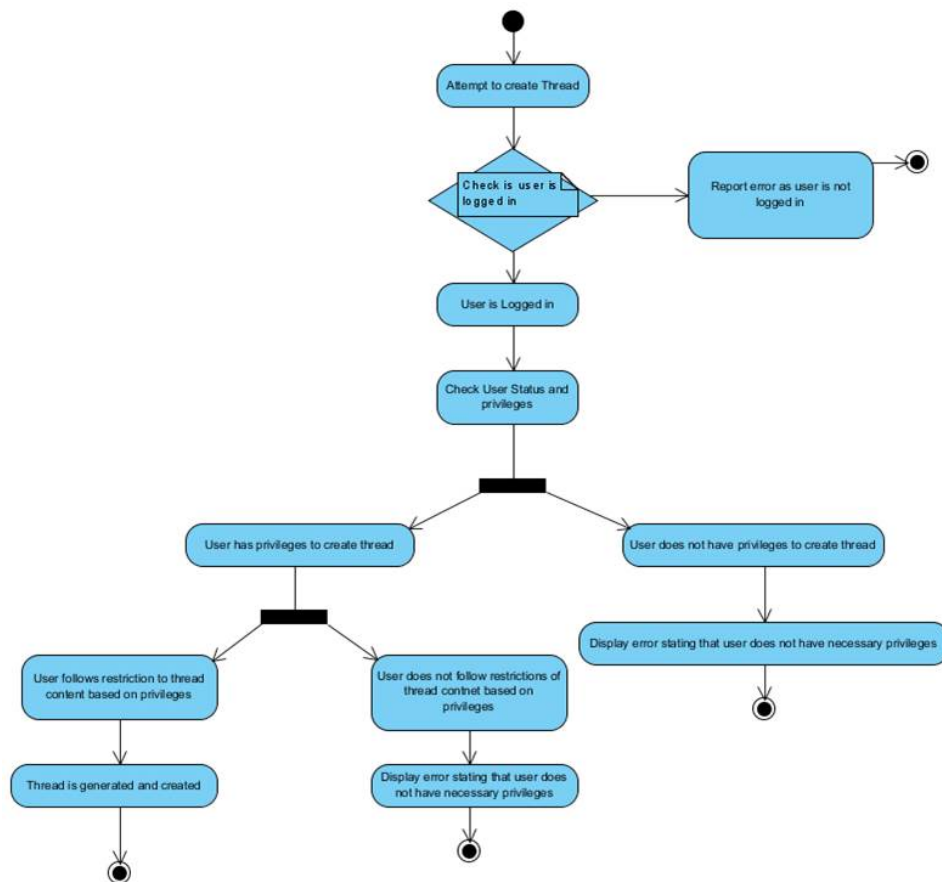
1. If the user attempts to create a thread, the system will first check if the user is logged in.
  - 1.1. If so, it will go to number 1 above and check the users status
    - 1.1.1. The system verifies the status and at number 2 checks that the user has the privileges to create threads of the appropriate size, contents etc.
      - 1.1.1.1 If the user has the necessary privileges then the thread is created at 3
      - 1.1.1.2 Otherwise an error message is displayed telling the user that he/she does not have the necessary privileges
  2. If the user is not logged in, an error message is displayed telling the user that he/she is not logged in and it will redirect to the logging page.

### 1.3 Required Functionality:

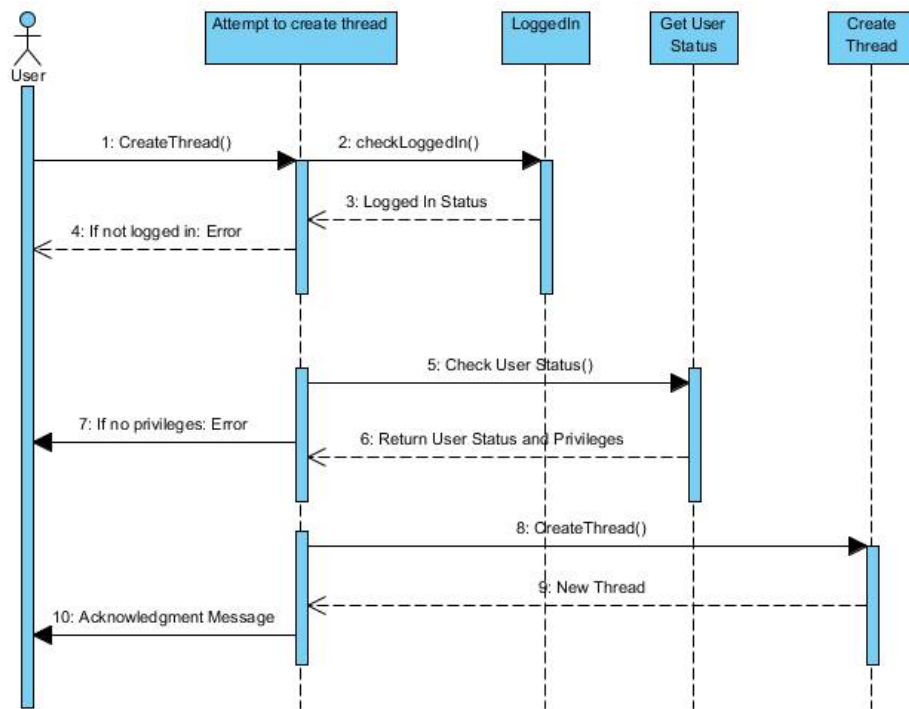


### 1.4 Process Specifications:

Activity Diagram



Sequence Diagram



## 2 Read Thread

### Description:

1. User activates Read Thread function by selecting the A Thread they wish to read.
2. System responds by presenting user with the selected thread of posts
3. User reads the posts and based on privileges can make changes such as update, delete, reply, tag, rate etc

### 2.1 Prioritization:

This Use case is considered Important

### 2.2 Conditions and Data Structures:

Participants:

Initiated by User and communicates with system

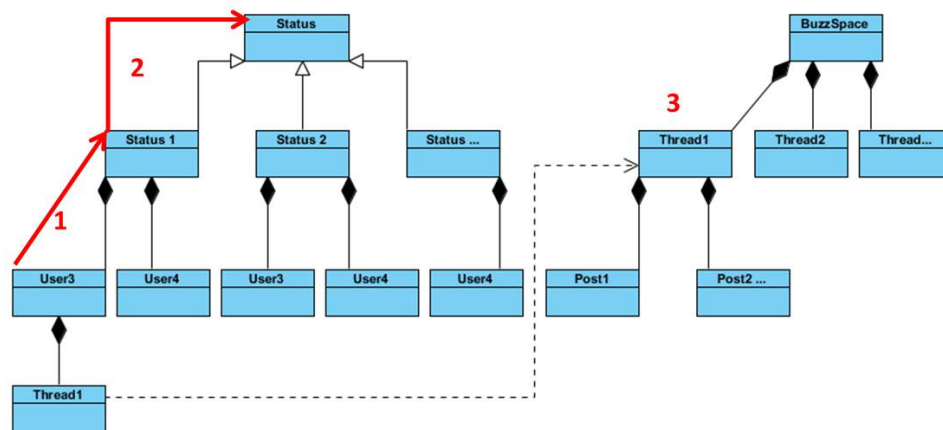
### Pre-Conditions:

1. The user must click on a thread they wish to read
  - 1.1. If User is logged in and
    - 1.1.1. User has status with high privileges then the user has options to change thread
      - 1.1.1.1. Calls other use cases
    - 1.1.2. Otherwise user is automatically logged in as Guest and can only view posts

### Post-Conditions:

1. System displays selected thread with posts
2. If user is logged on and has privileges then an attempt to make changes such as update, delete, reply, tag, rate etc will be acknowledged.

### Requests and Results Data Structures:



Status can be an abstract class and each sub-class will be a generalization of this class as each new status IS-A status.

Each user HAS-A status and thus each new user will be a composition of each specific status class.

Each thread HAS-A user that started it and thus each new thread will be a composition of each user.

Each thread HAS-A BuzzSpace and thus each new thread will be a composition of each BuzzSpace.

Each Post HAS-A Thread and thus each new post will be a composition of each Thread

### Steps:

1. If the user attempts to enter a thread to read then the system will first

check if the user is logged in.

1.1. If so it will go to number 3 and open the selected thread to read.

1.1.1. If user attempts to alter a post it will go to number 1 above and check the users status

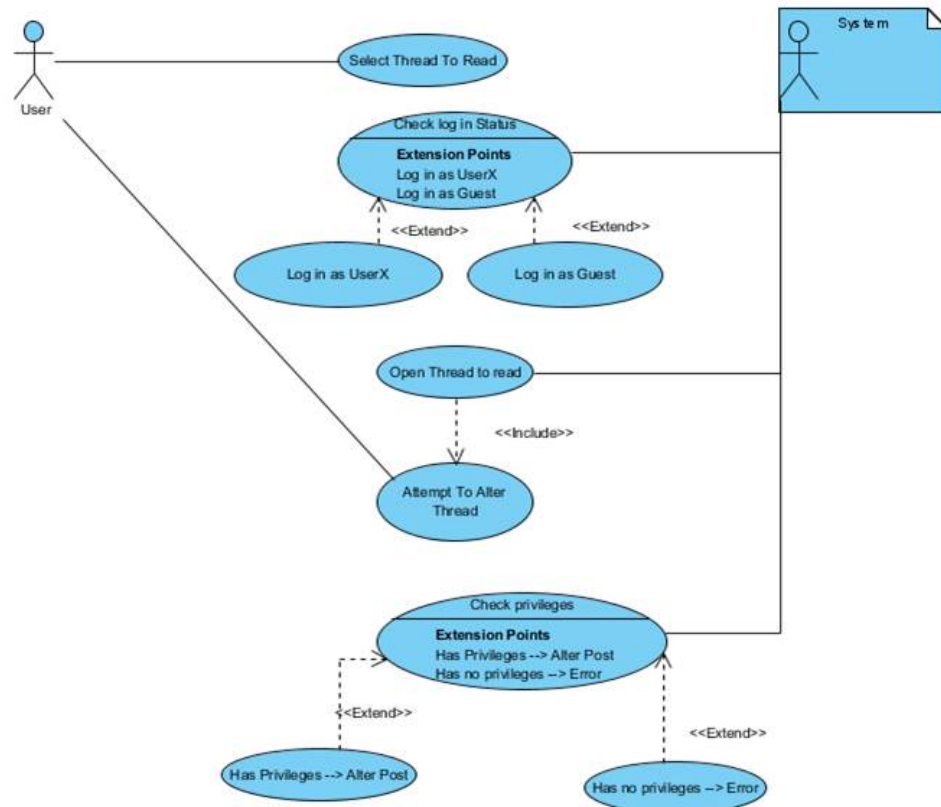
1.1.1.1. The system verifies the status and at number 2 checks that the user has the privileges to create threads of the appropriate size, contents etc.

1.1.1.2. If the user has the necessary privileges then the post is altered at 3

1.1.1.3. Otherwise an error message is displayed telling the user that he/she does not have the necessary privileges

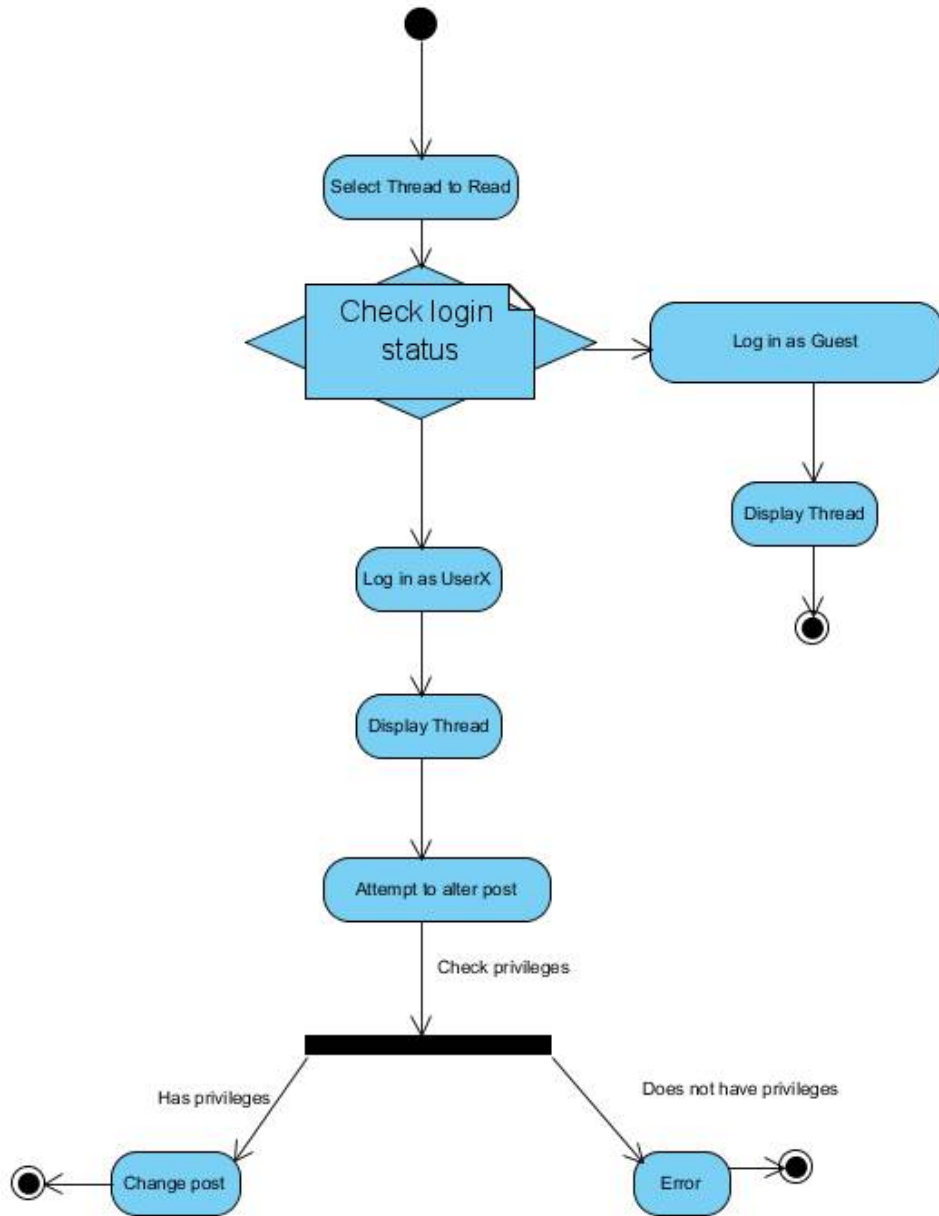
1.2. If not the user will be automatically logged on as Guest

## 2.3 Required Functionality:

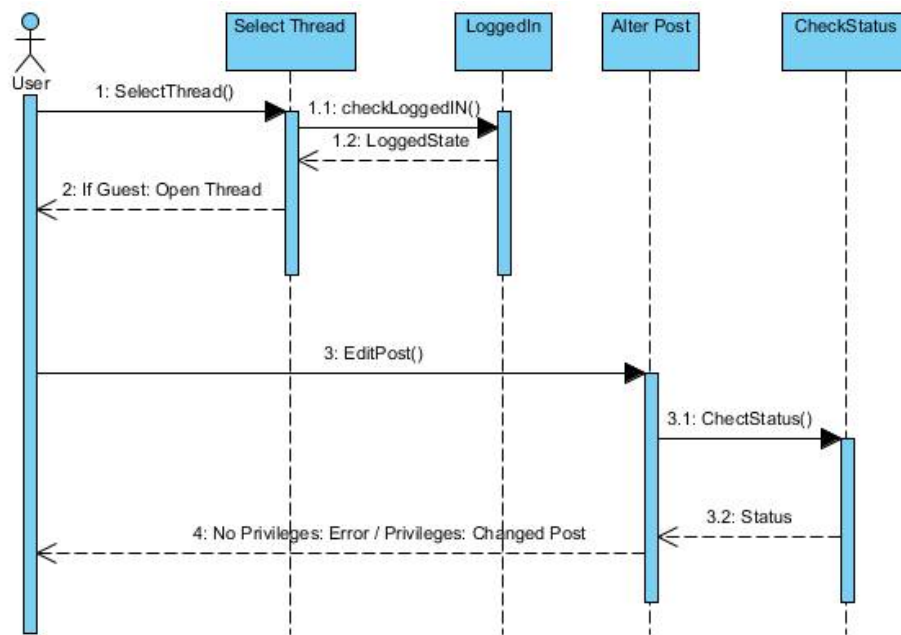


## 2.4 Process Specifications:

### Activity Diagram



Sequence Diagram



### 3 Update Thread

#### Description:

1. User activates Update Thread function by selecting the Update Thread option by choosing how they wish to update the thread.
2. System checks to see whether user is logged in
3. System reviews the submitted information and verifies users status and privileges, decorates the thread as needed and checks that all requirements (rules) are followed by the user based on his/her status
4. System displays either an acknowledgment or error message based on the previous checkpoint.

#### 3.1 Prioritization:

This Use case is considered Important

#### 3.2 Conditions and Data Structures:

Participants:

Initiated by User and communicates with system



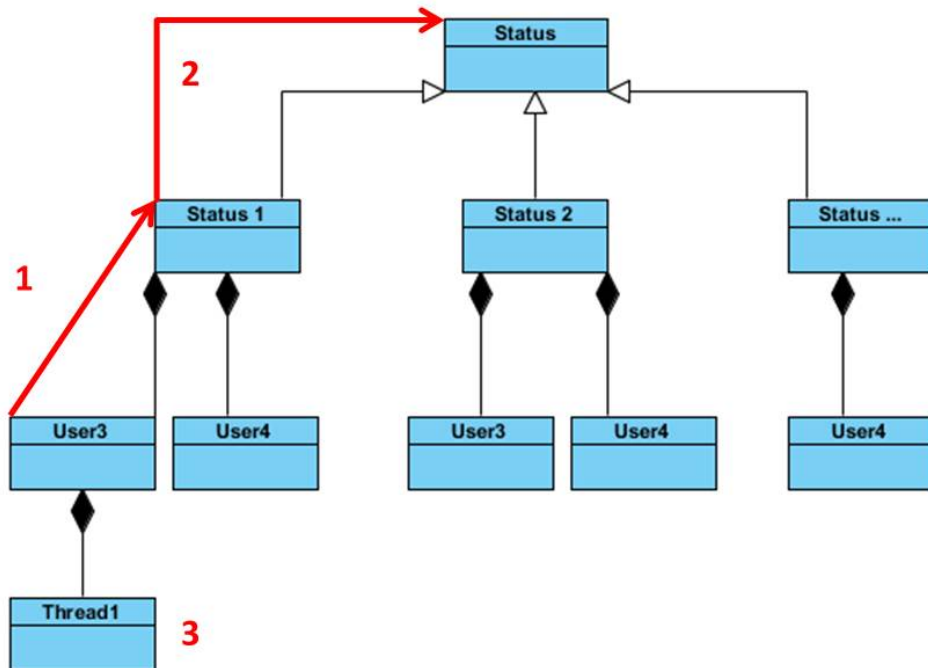
### Pre-Conditions:

1. User chooses the way in which to update a thread
2. The User is logged in
  - 2.1. The user has a specific status
  - 2.2. User must have a specific status in order to perform certain functions when updating a thread

### Post-Conditions:

1. System displays an acknowledgment message and updates the new thread if user follows status based rules
2. Systems display an error message and requests user to restart if user attempts to perform a function that is not within the scope of their status privileges.

### Requests and Results Data Structures:



Status can be an abstract class and each sub-class will be a generalization of this class as each new status IS-A status.  
Each user HAS-A status and thus each new user will be a composition of

each specific status class.

Each thread HAS-A user that started it and thus each new thread will be a composition of each user.

Steps:

1. If the user attempts to update a thread, the system will first check if the user is logged in.

1.1. If so, it will go to number 1 above and check the users status

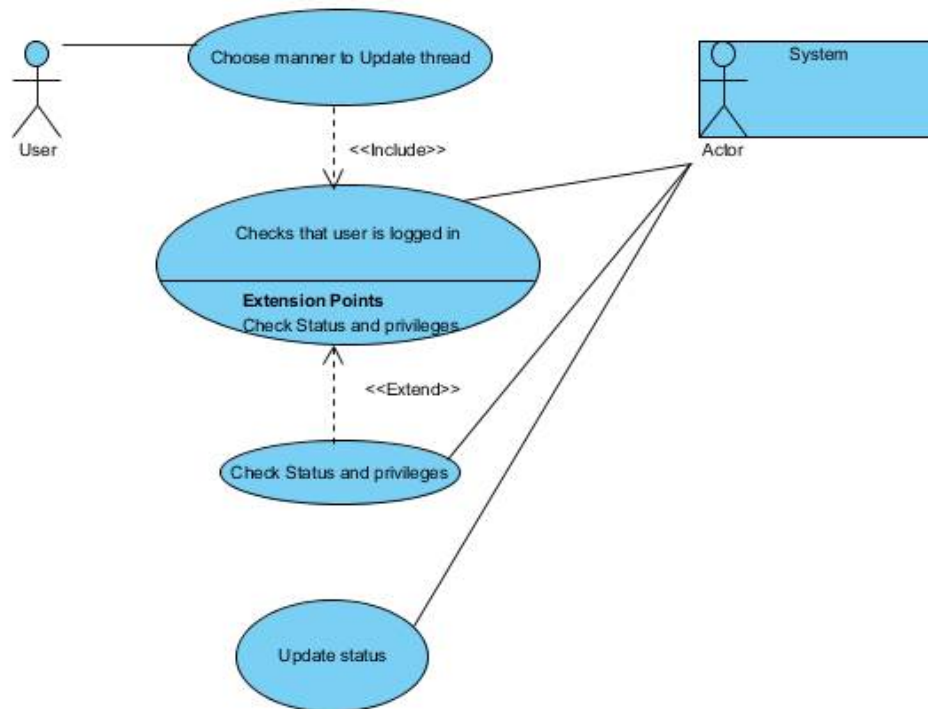
1.1.1. The system verifies the status and at number 2 checks that the user has the privileges to update threads in the specific way chosen.

1.1.1.1. If the user has the necessary privileges then the thread is updated at 3

1.1.1.2. Otherwise an error message is displayed telling the user that he/she does not have the necessary privileges

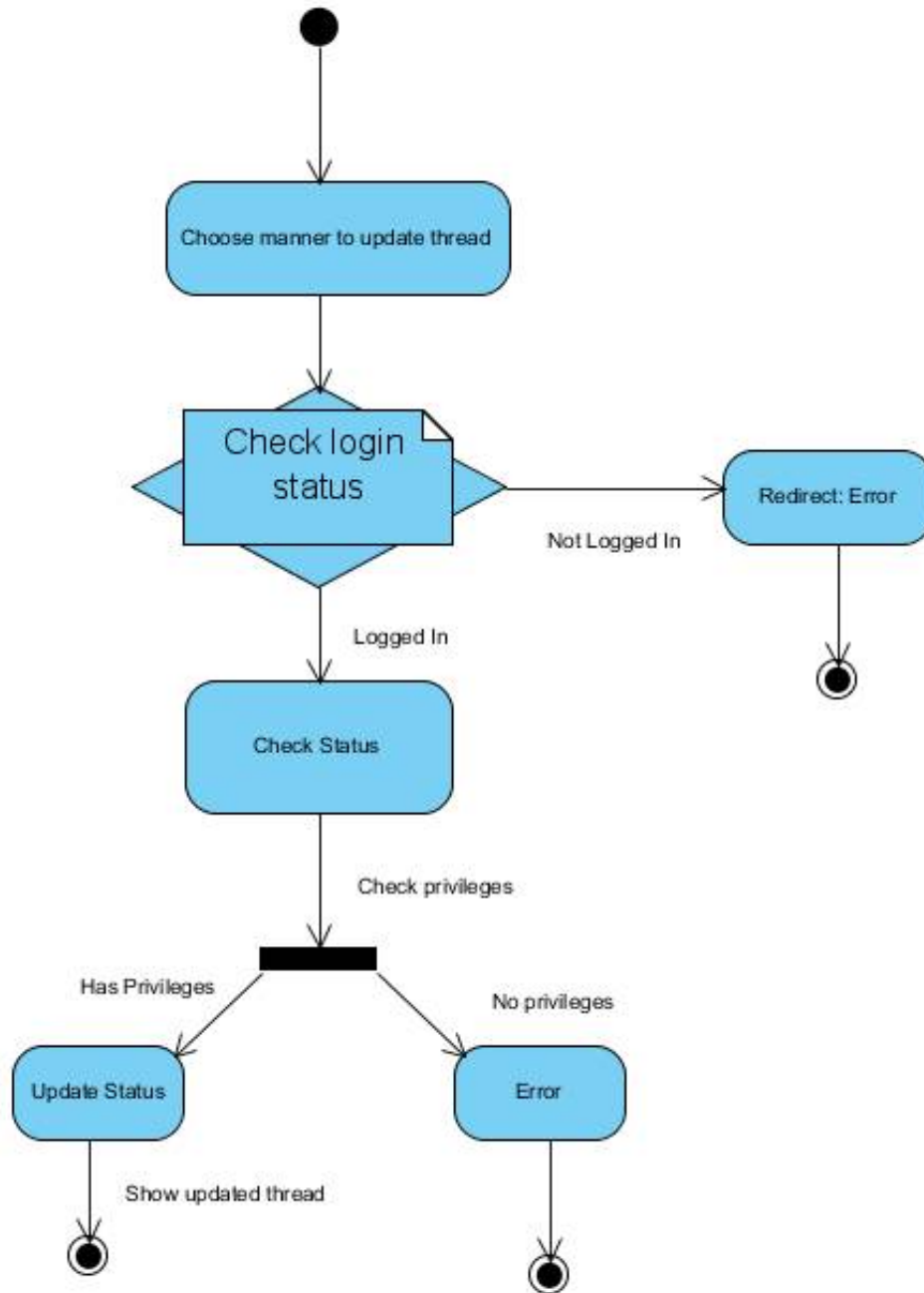
1.2. If the user is not logged in, an error message is displayed telling the user that he/she is not logged in and it will redirect to the logging page.

### 3.3 Required Functionality:

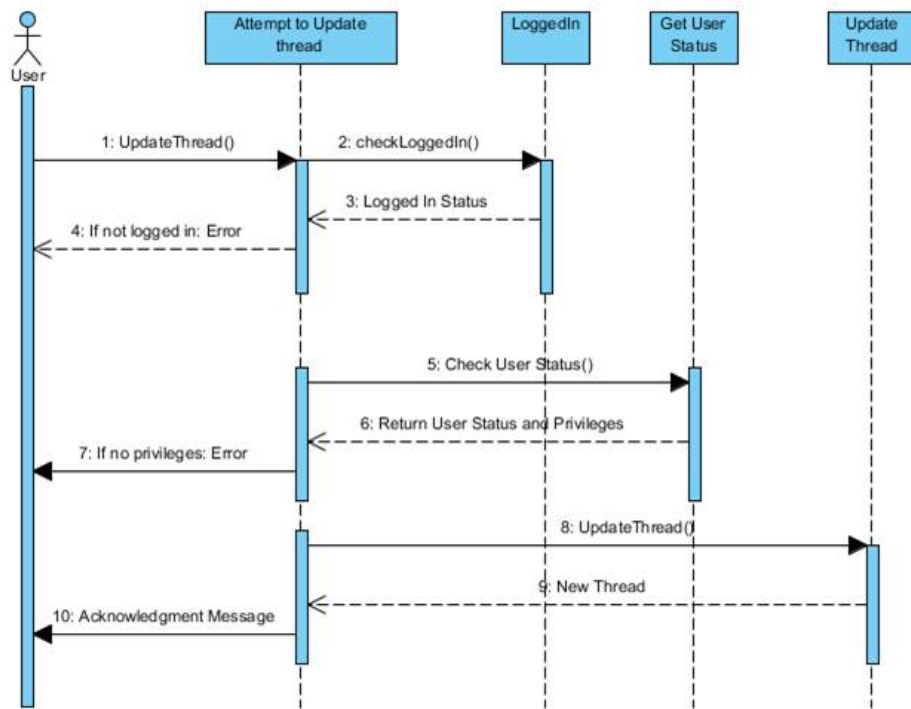


### 3.4 Process Specifications:

Activity Diagram



Sequence Diagram



## 4 Delete Thread

### Description:

- 1 User activates Delete Thread function by selecting the Delete Thread option.
- 2 System checks to see whether user is logged in
- 3 System reviews the submitted information and verifies users status and privileges
- 4 System displays either an acknowledgement and removes the thread or displays an error message based on the pervious checkpoint.

### 4.1 Prioritization:

This Use case is considered Important

## 4.2 Conditions and Data Structures:

Participants:

Initiated by User and communicates with system

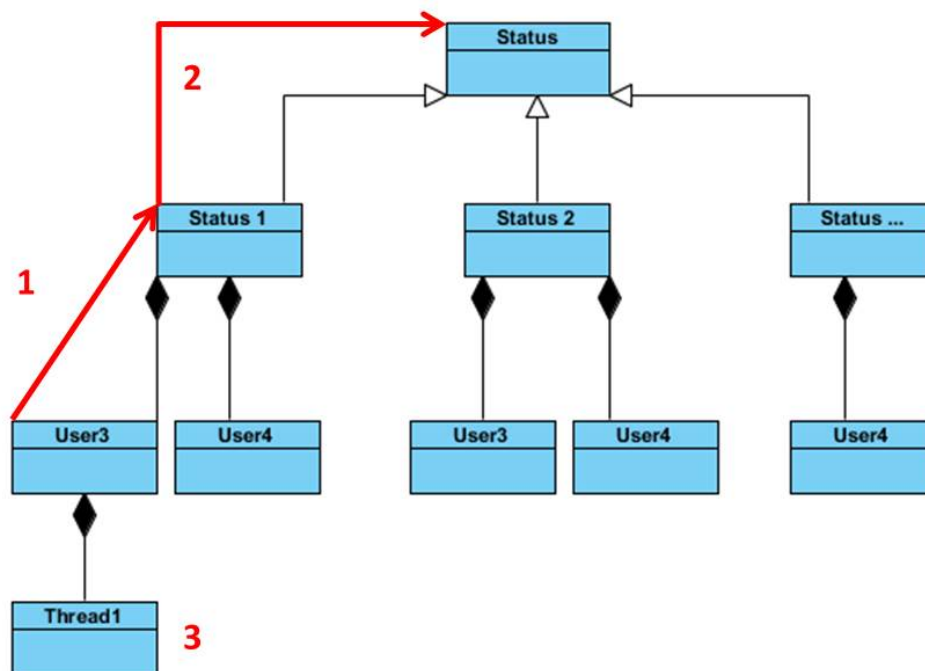
### Pre-Conditions:

1. User selects Delete Thread
2. The User is logged in
  - 2.1. The user has a specific status
  - 2.2. User must have a specific status in order to delete a thread

### Post-Conditions:

1. System displays an acknowledgment message and removes the thread if user follows status based rules
2. Systems display an error message and requests user to restart if user attempts to perform a function that is not within the scope of their status privileges.

### Requests and Results Data Structures:



Status can be an abstract class and each sub-class will be a generalization

of this class as each new status IS-A status.

Each user HAS-A status and thus each new user will be a composition of each specific status class.

Each thread HAS-A user that started it and thus each new thread will be a composition of each user.

Steps:

1. If the user attempts to delete a thread, the system will first check if the user is logged in.

1.1. If so, it will go to number 1 above and check the users status

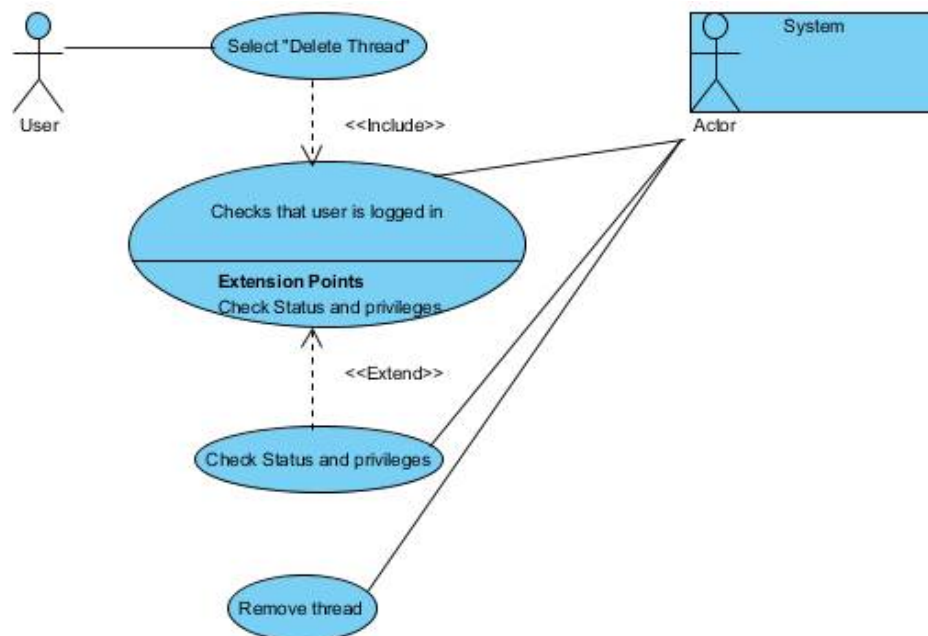
1.1.1. The system verifies the status and at number 2 checks that the user has the privileges to delete thread.

1.1.1.1. If the user has the necessary privileges then the thread at 3 is removed from the tree

1.1.1.2. Otherwise an error message is displayed telling the user that he/she does not have the necessary privileges

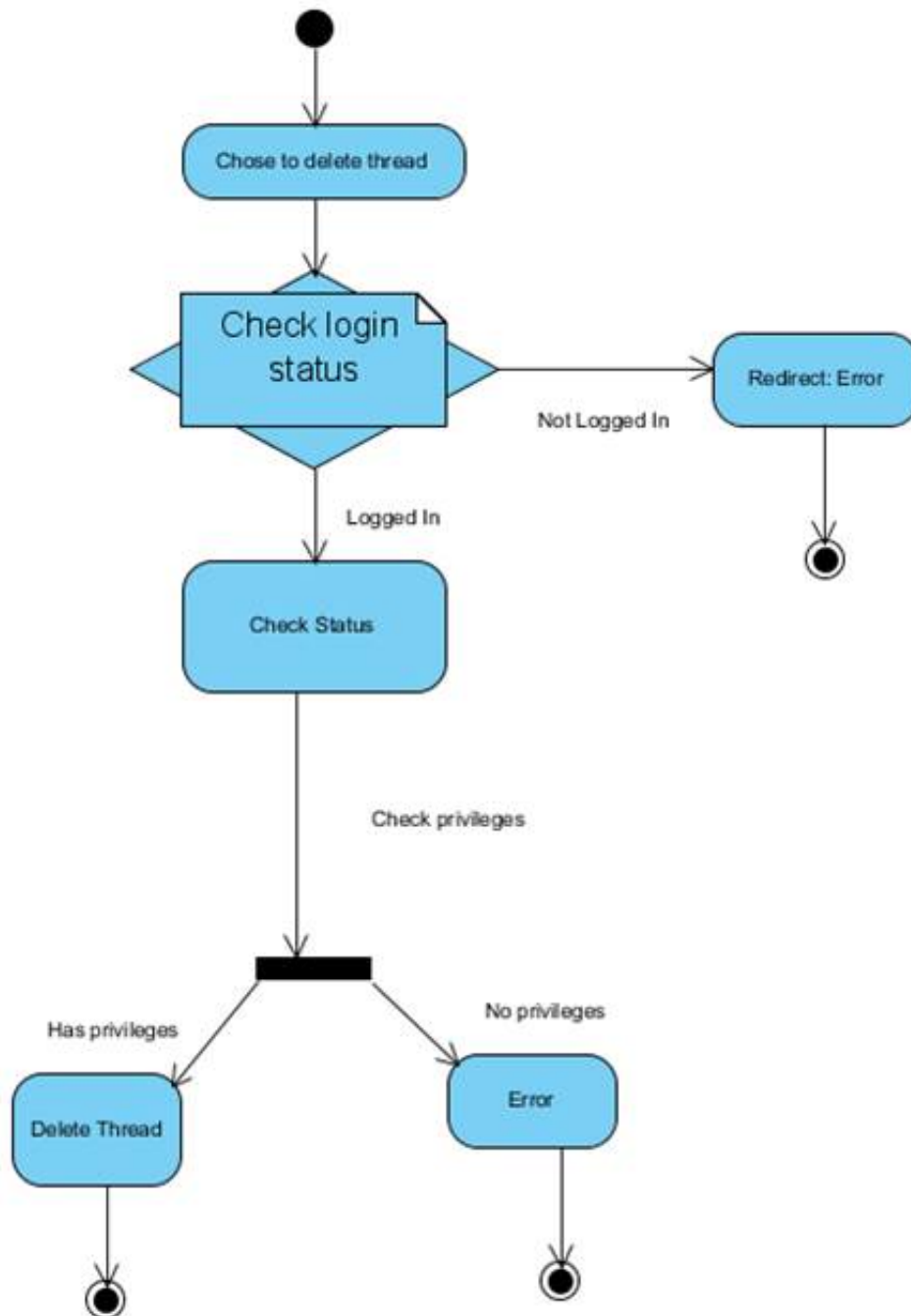
1.2. If the user is not logged in, an error message is displayed telling the user that he/she is not logged in and it will redirect to the logging page.

### 4.3 Required Functionality:

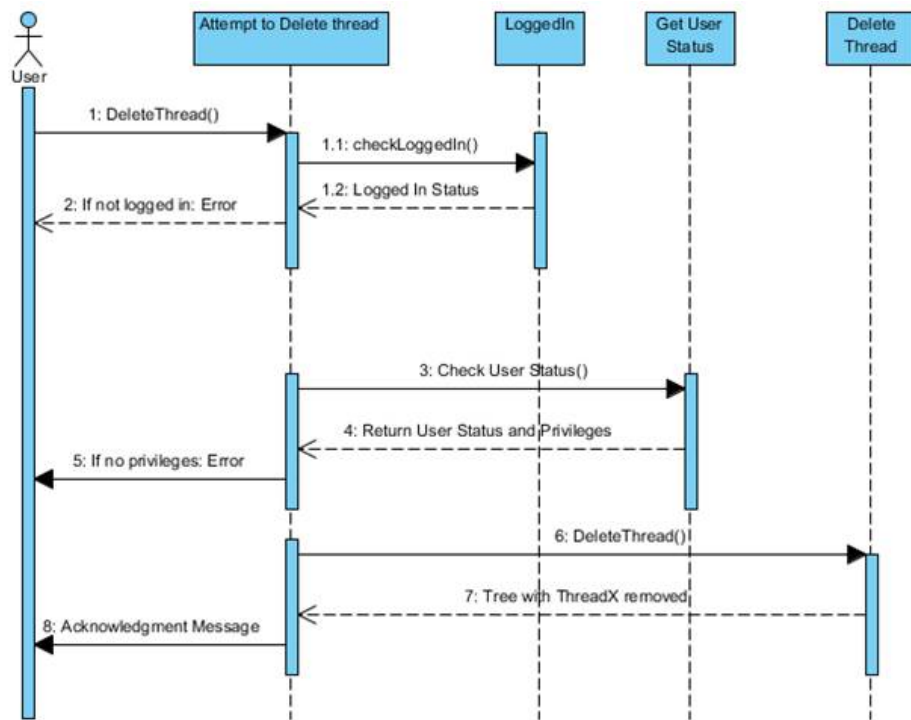


#### 4.4 Process Specifications:

Activity Diagram



Sequence Diagram



## 5 Summarise Thread

### Description:

Since a thread can, in theory, grow infinitely long, It is therefor necessary to show only the important aspects of the thread. These are the root post in the thread, the declared answer in the thread and the thread's social tags.

### 5.1 Prioritization:

This feature is Nice-To-Have. The system will not be immensely devalued if this feature is not added.

### 5.2 Conditions and Data Structures:

#### Pre-Conditions:

1. The thread must be closed (No longer open for discussion).
2. User must be logged on as the system administrator to summarise a



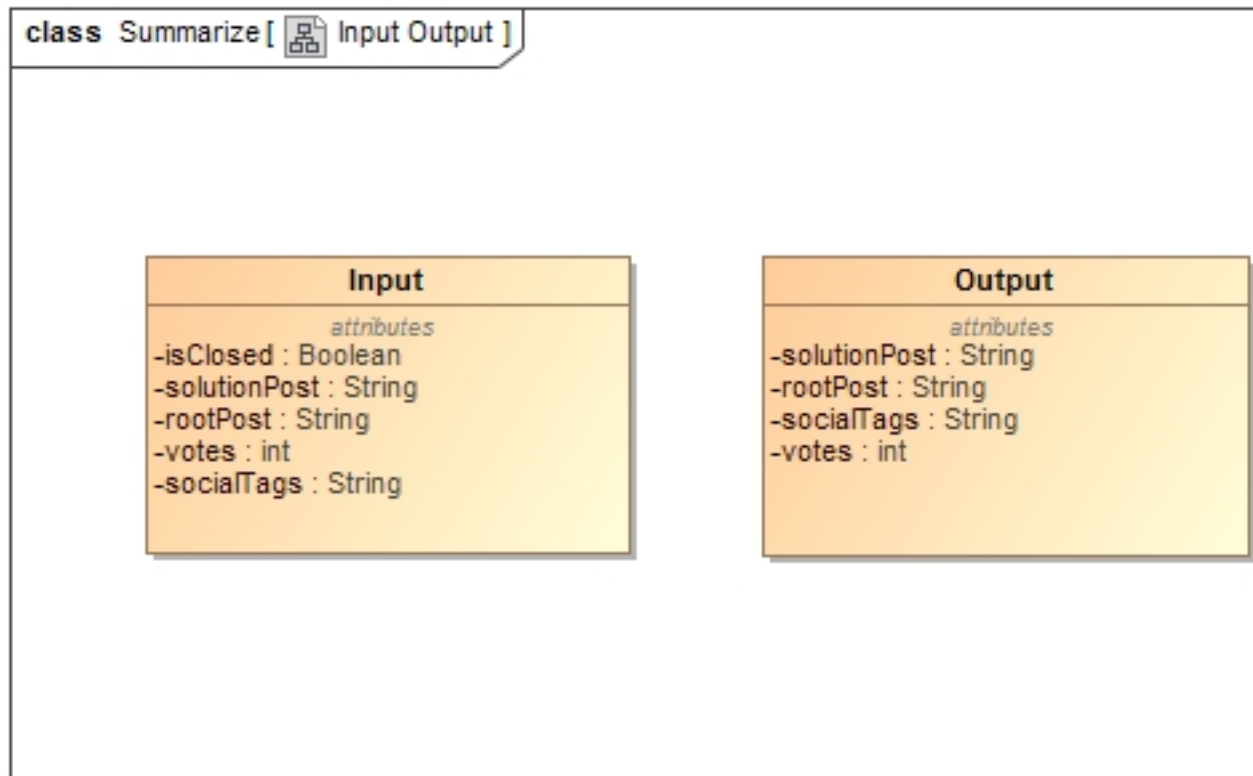
thread.

3. The thread must be more than 3 posts long.

### Post-Conditions:

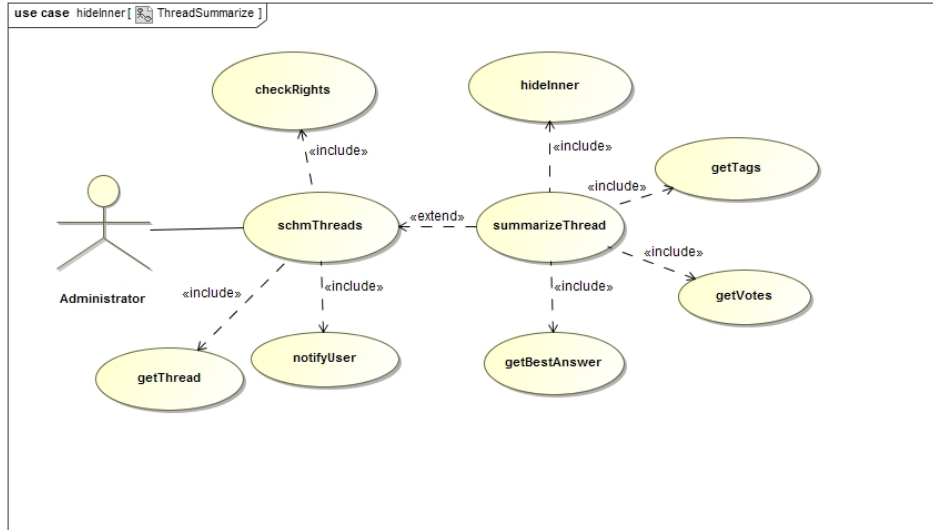
1. The thread is summarised successfully if all pre-conditions are met.
2. The system displays an error message if at least one of the pre-conditions are violated.

### Requests and Results Data Structures:



### 5.3 Required Functionality:

Summarizing a thread



### 5.4 Process Specifications:

## 6 Hide Thread

### Description:

Hiding a thread is when the System administrator makes the thread invisible to the user but not to the system. It is necessary for cases like when a thread is under plagiarism investigations or when the discussion is considered inappropriate. It will still exist in the Buzz space but users will not see it and, in turn, add onto it.

### 6.1 Prioritization:

It is considered critical to the system. It remains key that the system maintains its credibility and integrity at all times. This avoids conflicts and legal issues that may arise.

## 6.2 Conditions and Data Structures:

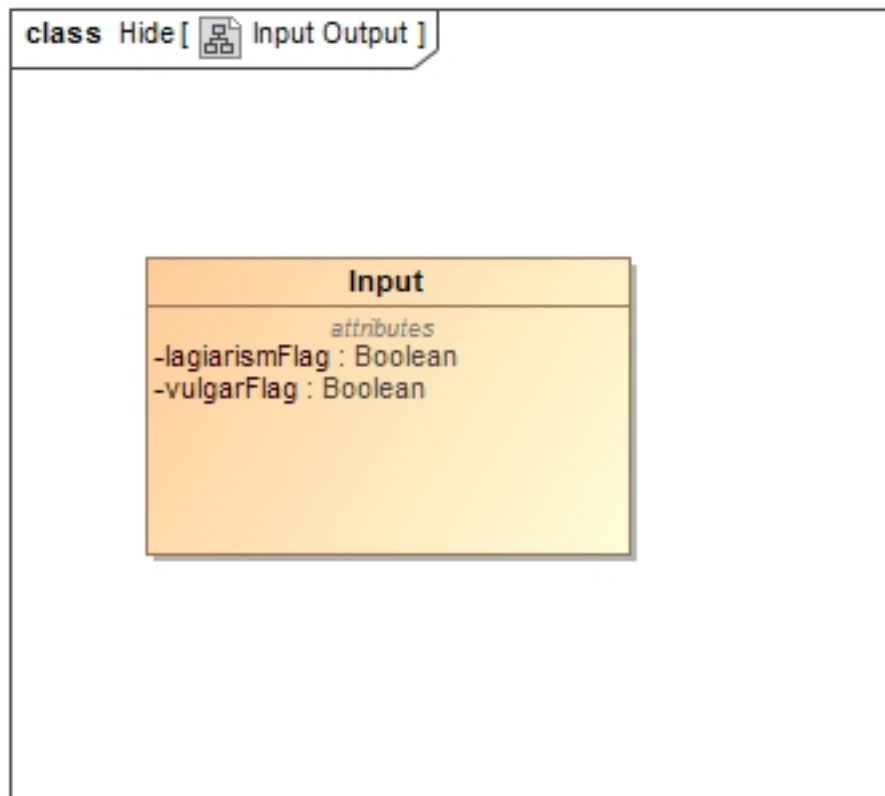
### Pre-Conditions:

1. The user must be logged on as a system administrator.
2. The thread must have been flagged by the system for plagiarism or other questionable characteristics such as vulgar language or slander.

### Post-Conditions:

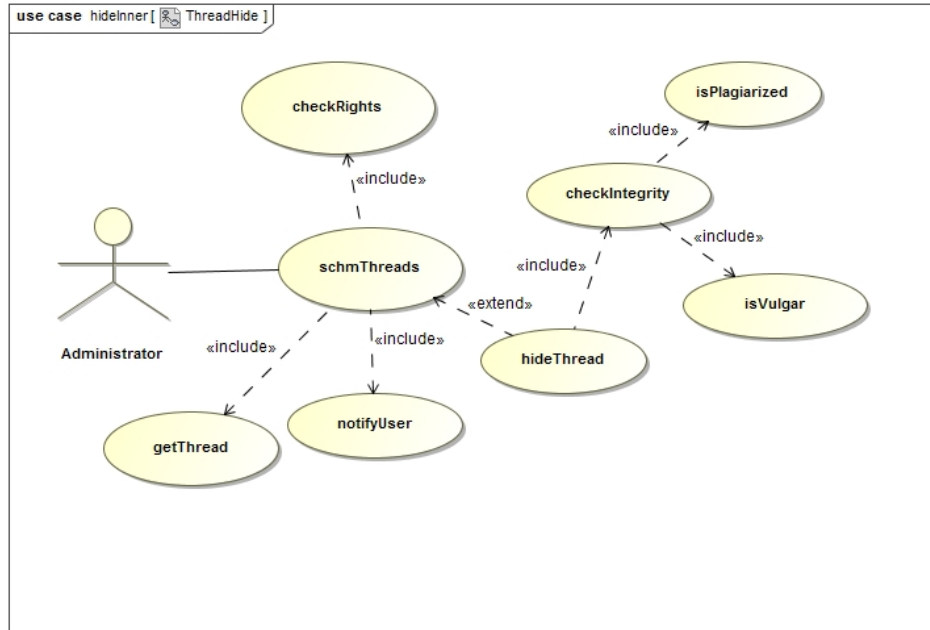
1. The thread is successfully hidden from the user if all pre-conditions are met.
2. The thread is still visible to the system.
3. A notification is sent to the owner of the thread that it is pending investigation.
4. The thread remains unhidden and an error message is shown if at least one of the pre-conditions aren't met.

## Requests and Results Data Structures:



### 6.3 Required Functionality:

Hiding a thread.



### 6.4 Process Specifications:

## 7 Close Thread

### Description:

Once a best answer has been agreed upon by the thread owner and users, it is no longer necessary for the thread to be open to discussion. It is therefore preferable for it to be available for viewing and not commenting on. It is considered closed when this happens.

### 7.1 Prioritization:

This feature is considered important. The system can cope without it but it is not preferable because threads can grow to be very long.

## 7.2 Conditions and Data Structures:

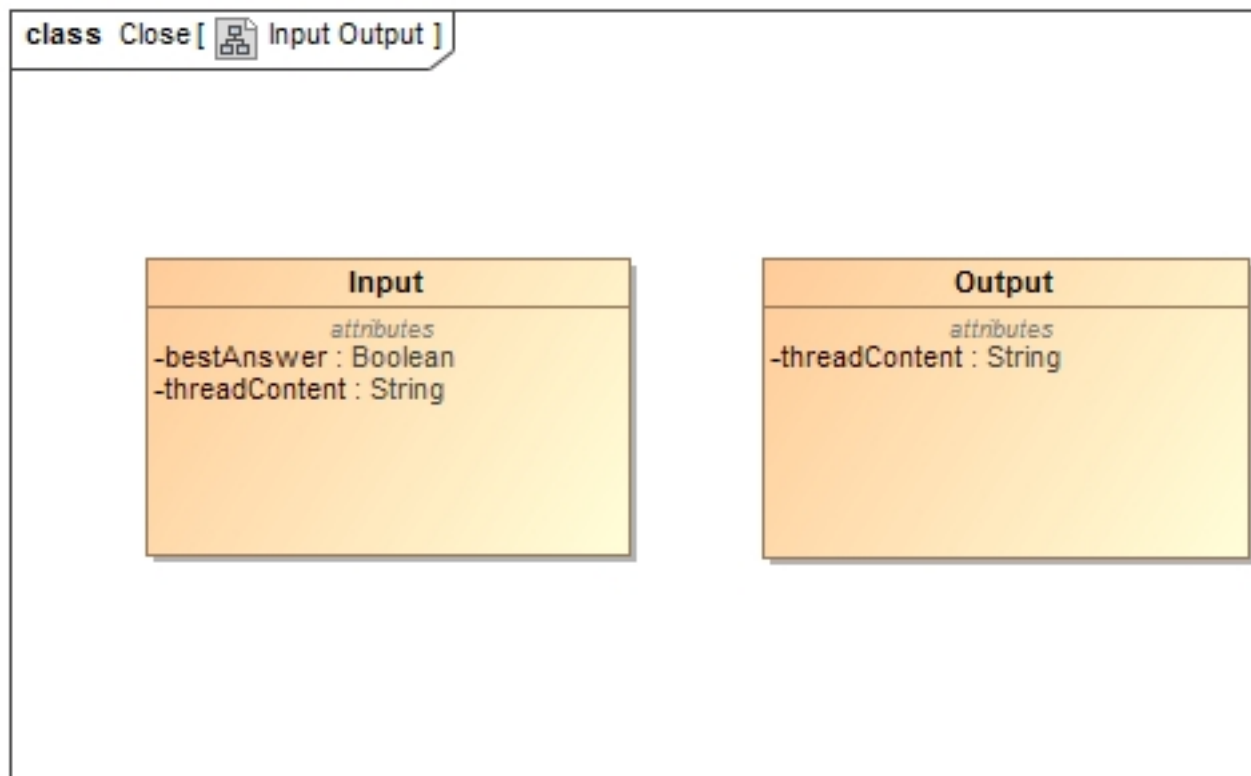
### Pre-Conditions:

1. The user must be logged in as an administrator.
2. The a best answer has to have been chosen by the thread owner based on his or her discretion and votes from the users.

### Post-Conditions:

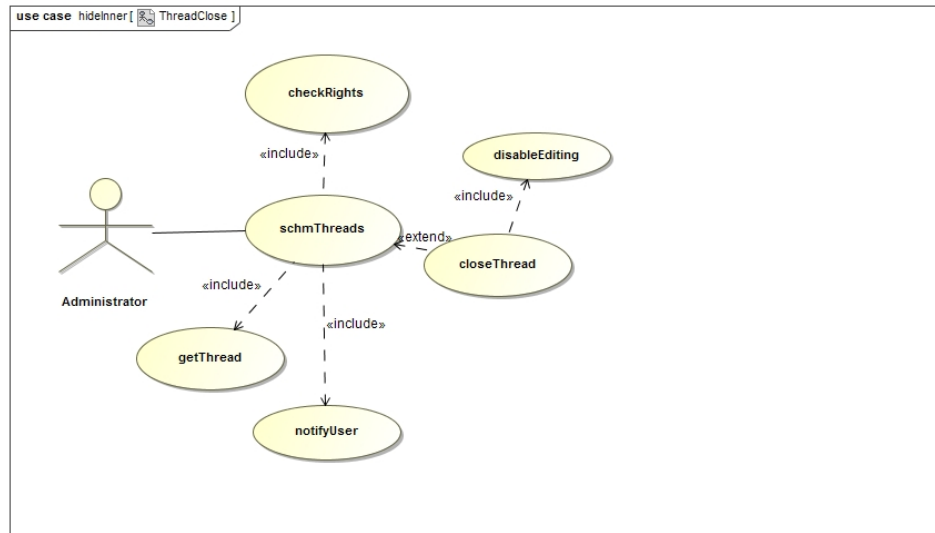
1. An error message is shown if at least one of the pre-conditions have been violated.
2. The thread is closed successfully if the pre-conditions have been met or open otherwise.
3. User is notified that thread is closed.

### Requests and Results Data Structures:



### 7.3 Required Functionality:

Closing a thread.



### 7.4 Process Specifications:

## 8 Move Thread

### Description:

Moving a thread can entail either deleting it from its original Buzz space to a different one or duplicating it. The former is necessary when a thread is not relevant to the Buzz space in question while the latter is important when it is relevant to both.

### 8.1 Prioritization:

### 8.2 Conditions and Data Structures:

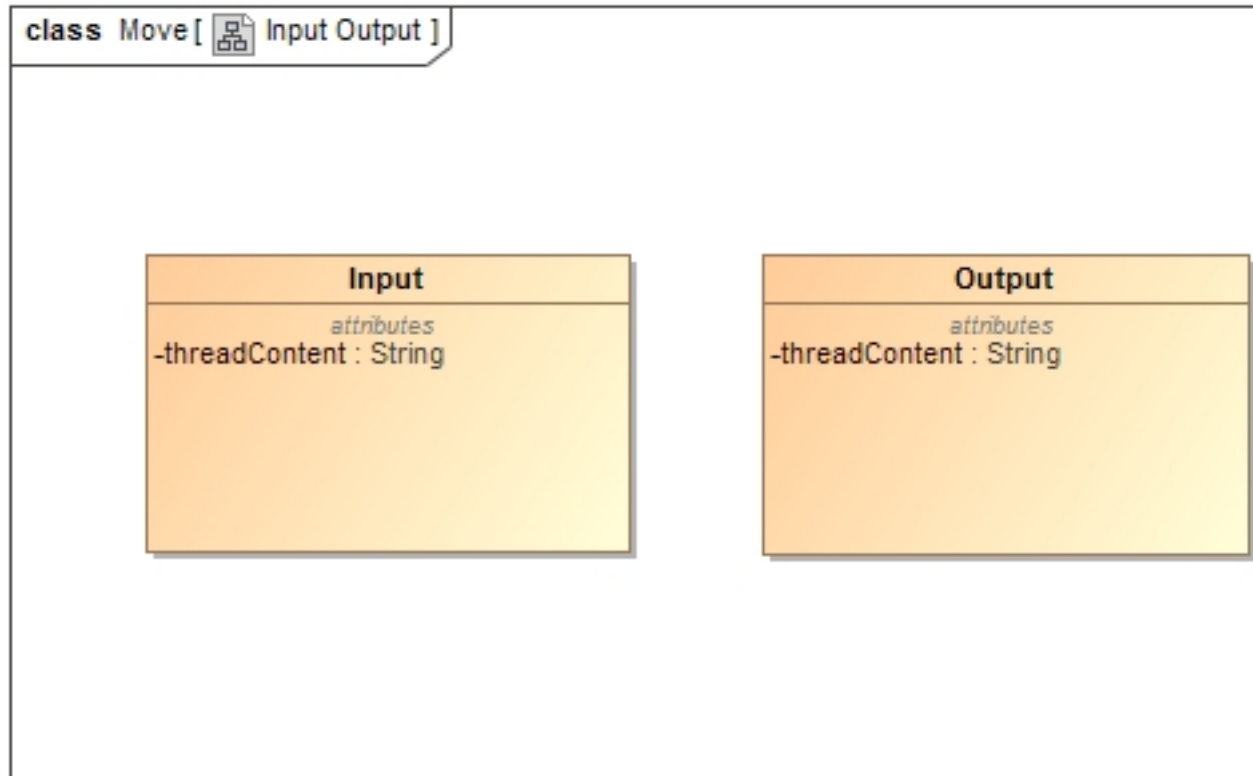
#### Pre-Conditions:

1. The user must be logged in as an administrator.
2. The thread owner must have given consent for the contents of the thread to be moved.

### Post-Conditions:

1. An error message is displayed if the user is not logged on as admin.
2. The thread is successfully moved if both pre-conditions are met.
3. The thread owner is notified if the thread is successfully moved.

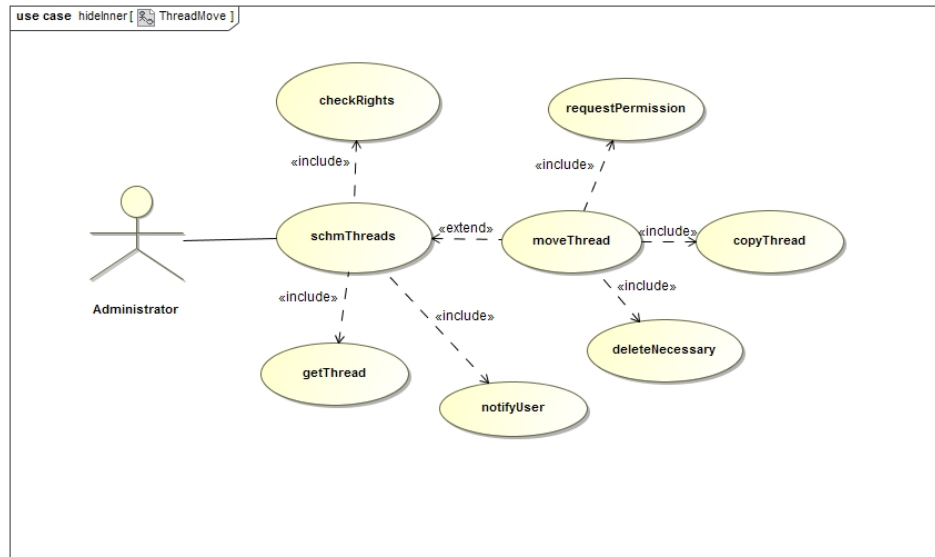
### Requests and Results Data Structures:





### 8.3 Required Functionality:

Moving a thread.



### 8.4 Process Specifications:

# Post Handling

## Use Cases

### 1 Create Post

#### Description:

This use case refers to the ability to create a new post on a certain thread.

#### 1.1 Prioritization:

Critical

#### 1.2 Conditions and Data Structures:

##### Pre-Conditions:

- The user must be logged in and registered to the specific BuzzSpace in which the post is created
- The user must adhere to the specifications of which user status levels can create a post on which level at which length.

##### Post-Conditions:

- If the user did not adhere to the specifications of which user status levels can create a post on which level at which length, an exception will be thrown and an error message will be displayed to the user.
- if plagiarism was detected on the post, then the service is rejected and an exception is thrown with an error message is displayed to the user.
- If netiquette rules were broken, then the service is rejected and an exception is thrown with an error message is displayed to the user.
- If the user is not logged in or registered for the module, then the service is rejected and an exception is thrown with an error message is displayed to the user.
- If the post was successful, a message is displayed to the user to let him/her know it was successful, and the message can now be read.

## **Requests and Results Data Structures:**

### **1.3 Required Functionality:**

### **1.4 Process Specifications:**

## **2 Read Post**

### **Description:**

This use case gives the functionality to read posts. A post will be marked as read/unread. Anyone can read any posts, if a user is not logged in, a guest account will be used to read posts.

### **2.1 Prioritization:**

Critical

### **2.2 Conditions and Data Structures:**

#### **Pre-Conditions:**

None

#### **Post-Conditions:**

- If the user was logged in under his/her name, the post(s) that was/were read will be marked as unread.
- If a user was using the guest account, none of the posts will be marked/unmarked.

## **Requests and Results Data Structures:**

### **2.3 Required Functionality:**

### **2.4 Process Specifications:**

## **3 Update Post**

### **Description:**

This use case refers to the ability to update posts that have already been posted. The user will have a specified time frame in which he/she will have the option to do this.

### **3.1 Prioritization:**

Nice-to-have

### **3.2 Conditions and Data Structures:**

#### **Pre-Conditions:**

- The user must be logged in.
- The user must be the original creator of the post.

#### **Post-Conditions:**

- The posts will be updated accordingly.

#### **Requests and Results Data Structures:**

### **3.3 Required Functionality:**

### **3.4 Process Specifications:**

## **4 Delete Post**

#### **Description:**

This use case includes the ability to delete a post. Only the poster, creator of thread and other people with lecturer status will be able to do this. This option will only be available if the user is logged in.

#### **4.1 Prioritization:**

Important

### **4.2 Conditions and Data Structures:**

#### **Pre-Conditions:**

- The user must be registered for the module.
- The user must be either the creator or a user with lecturer status.

#### **Post-Conditions:**

- The service will be rejected if the user is not registered for the specific module, or if the user is not the creator of the thread.
- If the deletion was successful, a message will be displayed to the user to notify him/her and the creator of the thread and/or the creator of thread will be informed of the deletion.
- Deleted post will be archived for future reference.

## Requests and Results Data Structures:

### 4.3 Required Functionality:

### 4.4 Process Specifications:

## 5 Mark solution post

### Description:

This use case allows a user or an owner of a thread to choose or mark a post in that thread as being the best answer, which should then elevate that post in the thread and should make in turn make the post easily visible as one enters the thread.

### 5.1 Prioritization:

**Nice-To-Have:** Marking a post as the best answer.

### 5.2 Conditions and Data Structures:

#### Pre-Conditions:

A user must have the privileges set by the administration of a space in order to mark a post as the best answer. The user is required to be logged in.

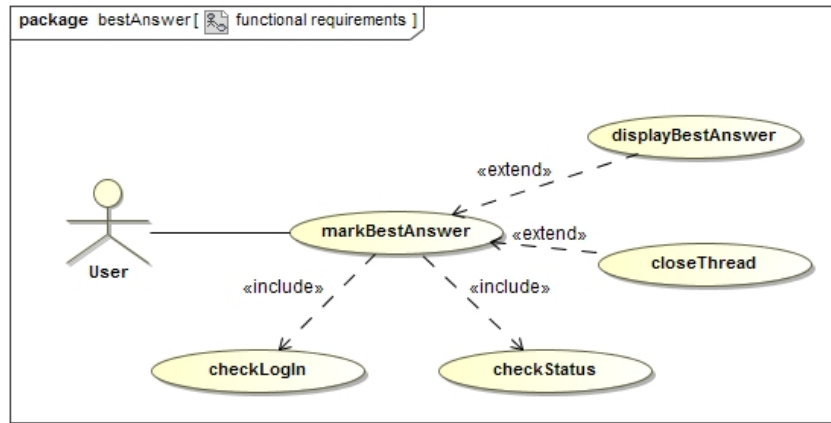
#### Post-Conditions:

The thread should effectively display that an answer has been selected and optionally close the thread. The poster of the best answer should acquire an increase in progress to the next status level.

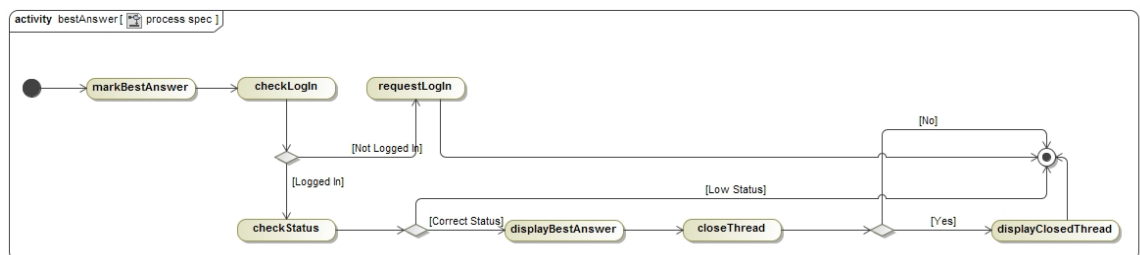
## Requests and Results Data Structures:

Input	Output
<i>attributes</i> <b>+bestAnswer : Boolean = false</b>	<i>attributes</i> <b>+solution : Boolean = true</b>

### 5.3 Required Functionality:



### 5.4 Process Specifications:



## 6 Vote and Evaluate Posts

### Description:

This use case refers to the ability of a user to either vote up or down a post; a higher vote count on a post indicates the usefulness of a post. It also refers to a user being able to evaluate or give feedback on another users post.

### 6.1 Prioritization:

**Important:** Voting on and evaluation of posts.

### 6.2 Conditions and Data Structures:

#### Pre-Conditions:

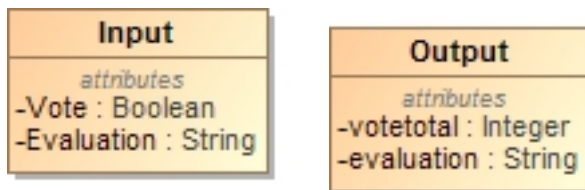
A post has to have been created for a user to be able to vote on or evaluate it. Any user can vote on or evaluate a post as long as the user is logged in and a part of a specific Buzz Space. An exception to this rule is when the

administration of a Buzz Space set specific rules as to which users (perhaps based on status) can vote on or evaluate posts.

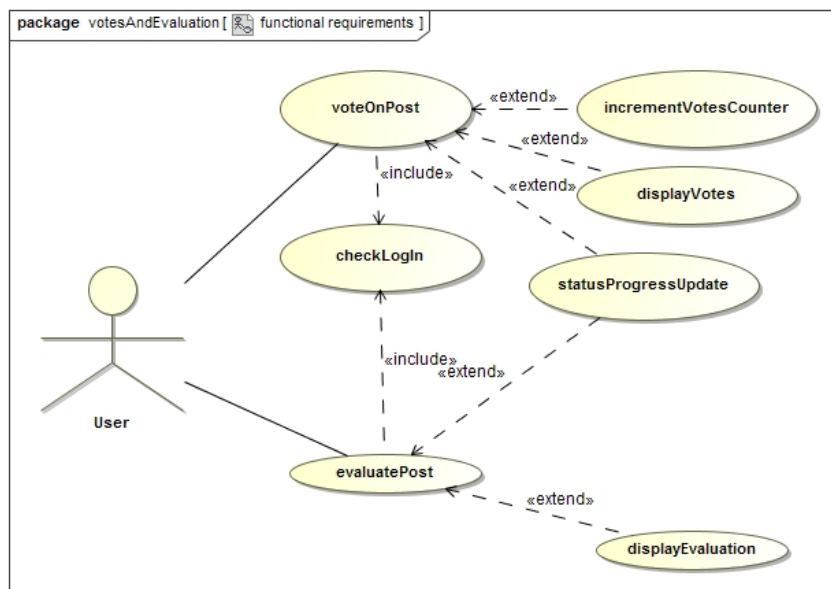
### Post-Conditions:

A vote or evaluation has to be visible to every user (or only some based on the administration of that space) after it has been made on a post. A positive vote or evaluation has to reflect upon the posters progress to the next status level.

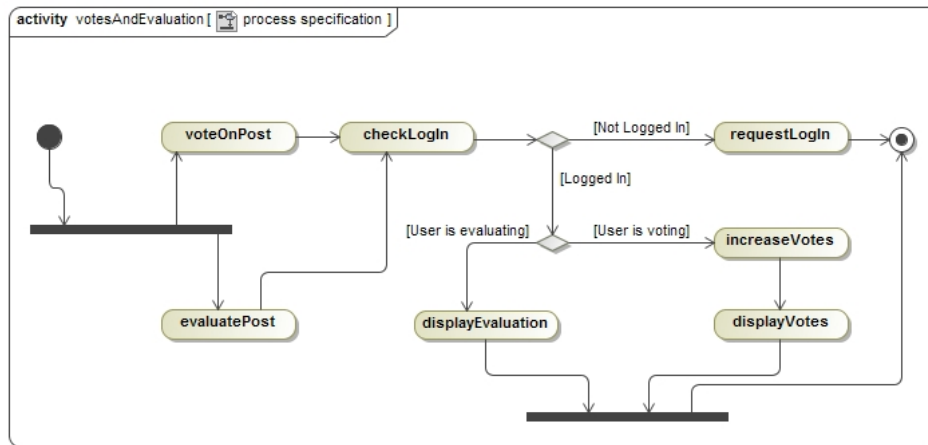
### Requests and Results Data Structures:



### 6.3 Required Functionality:



## 6.4 Process Specifications:



## 7 Post Tagging

### Description:

**Brief:**One very important functional requirement is the ability for the user to structure the information and content according to his own preference and needs. Social tagging is the feature that will enable him to do so

### 7.1 Prioritization:

**Case-Priority - Important:** The user should be able to tag topics of interest in the post they created, as well as posts created by other users. Social tagging saves time and keeps user up to date with latest news and post regarding tagged topics

Users can:

View posts according to tags.

Sort posts according to tags.

Follow different authors of posts and index topics and subjects according to tags.

### 7.2 Conditions and Data Structures:

#### Pre-Conditions:

A post must not have been previously marked as a tag. If there are any keywords that look like "links" and have a hash symbol in front of them within the post, that post may be seen as already tagged.



## Post-Conditions:

A post becomes a tagged post if it has words with in it that have a hash symbol in front of them.

words that are tags must change colour and become links.

The user can now index topics, follow different authors, sort and view posts according to the tagged topics.

## Requests and Results Data Structures:

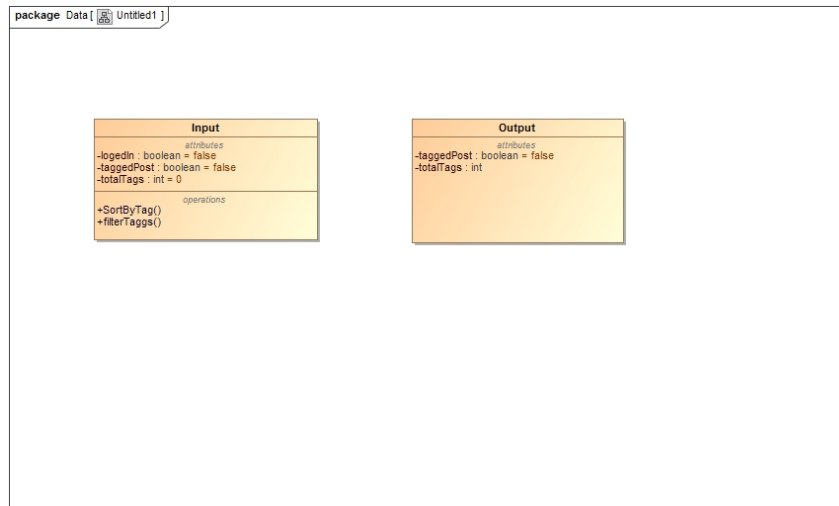


Figure 1: Input and Output classes showing potential attributes and operations \*loggedIn: boolean = true.

## 7.3 Required Functionality:

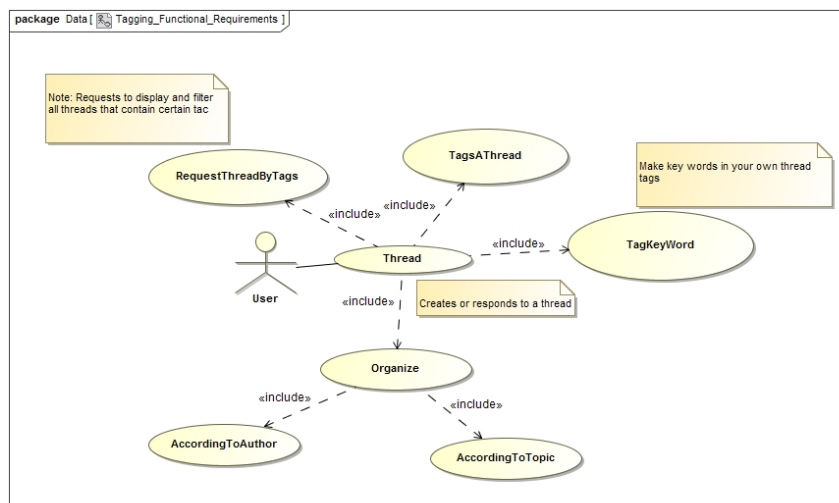


Figure 2: A UML use case diagram showing the required functionality for tagging posts.

#### 7.4 Process Specifications:

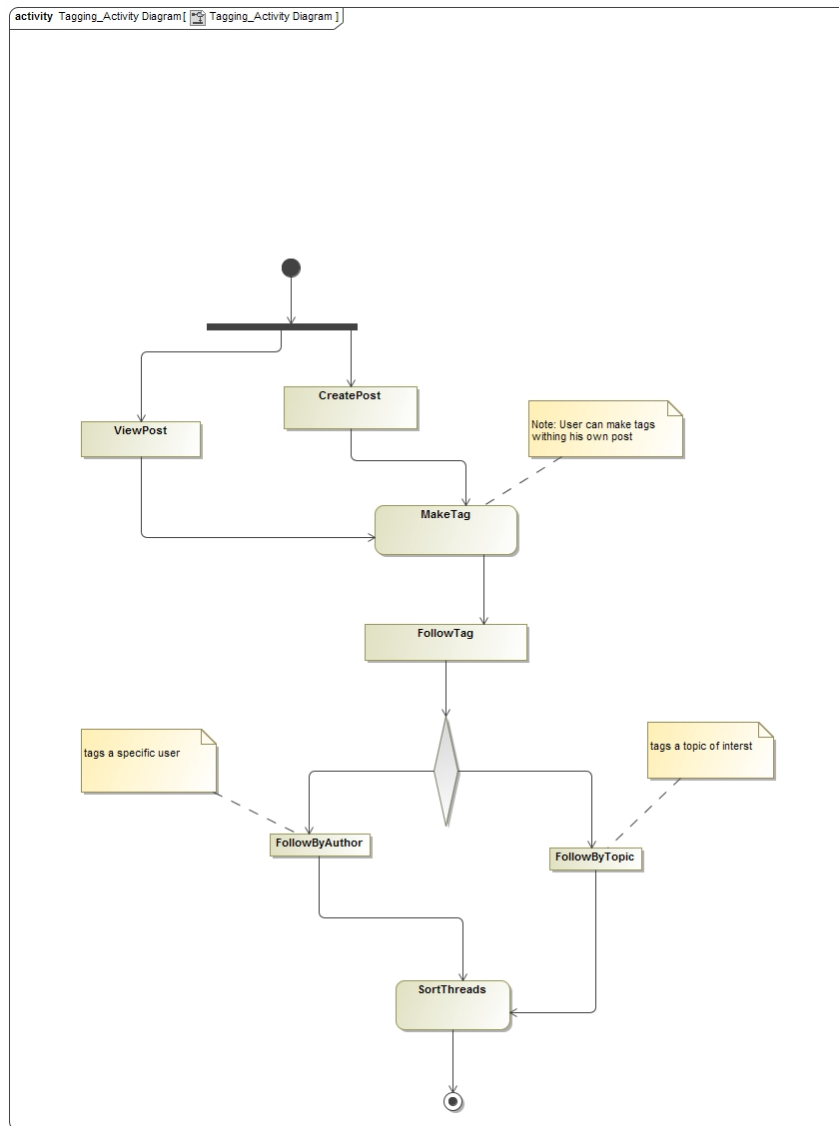


Figure 3: An activity diagram showing the process in which tags are made and organized.

# Filtering

## Use Cases

### 1 Filter threads and buzzspaces

#### **Description:**

This module provides a search and filter functionality. It allows the user to search the current Buzz Space they are in and if need be, filter the results. Users can filter the Buzz Space by tags, date, user or by user-rating. They can also search through the Buzz Space by entering a search-phrase in a search box, alternatively they can both search and filter i.e filter the search-results by tags, date, user or user-rating.

#### **1.1 Prioritization:**

Since the number of threads can be infinitely high, this functionality is important when one wishes to view specific threads

#### **1.2 Conditions and Data Structures:**

##### **Pre-Conditions:**

1. A specific Buzz Space has to have been created for a user to be able to search and/or filter.
2. The user has to at least choose a single filter option and/or enter a search phrase

##### **Post-Conditions:**

1. Upon clicking the submit button, the search and/or filter results will be automatically displayed for the user to see.

## Requests and Results Data Structures:

### Filter

Input
-searchPhrase : String = null -searchAndFilter : boolean = false +filterby : StructuredExpression

Output
-results : String

### Search

Input
-searchPhrase : String = null -searchAndFilter : boolean = false +filterby : StructuredExpression

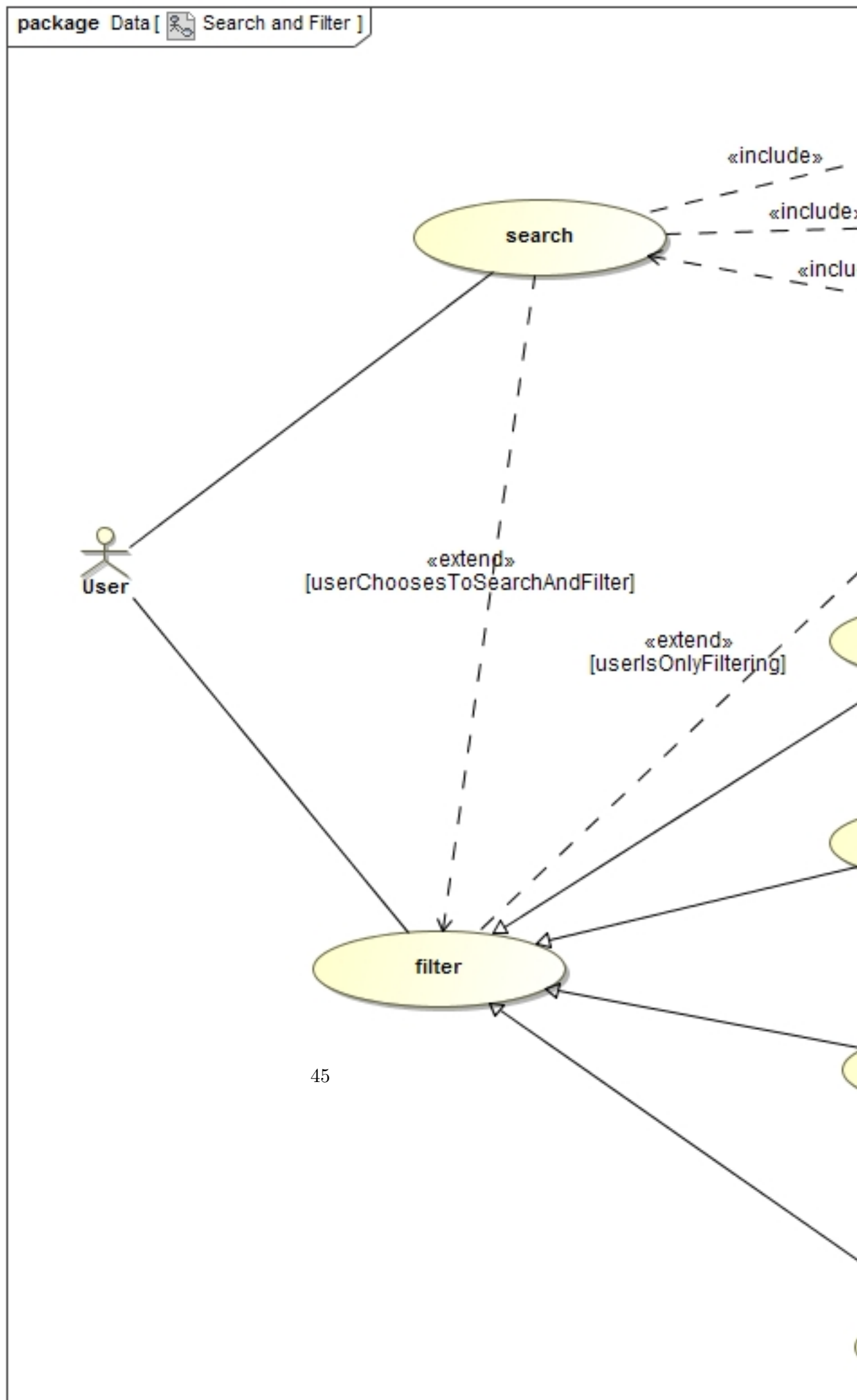
Output
-results : String

### Search and Filter

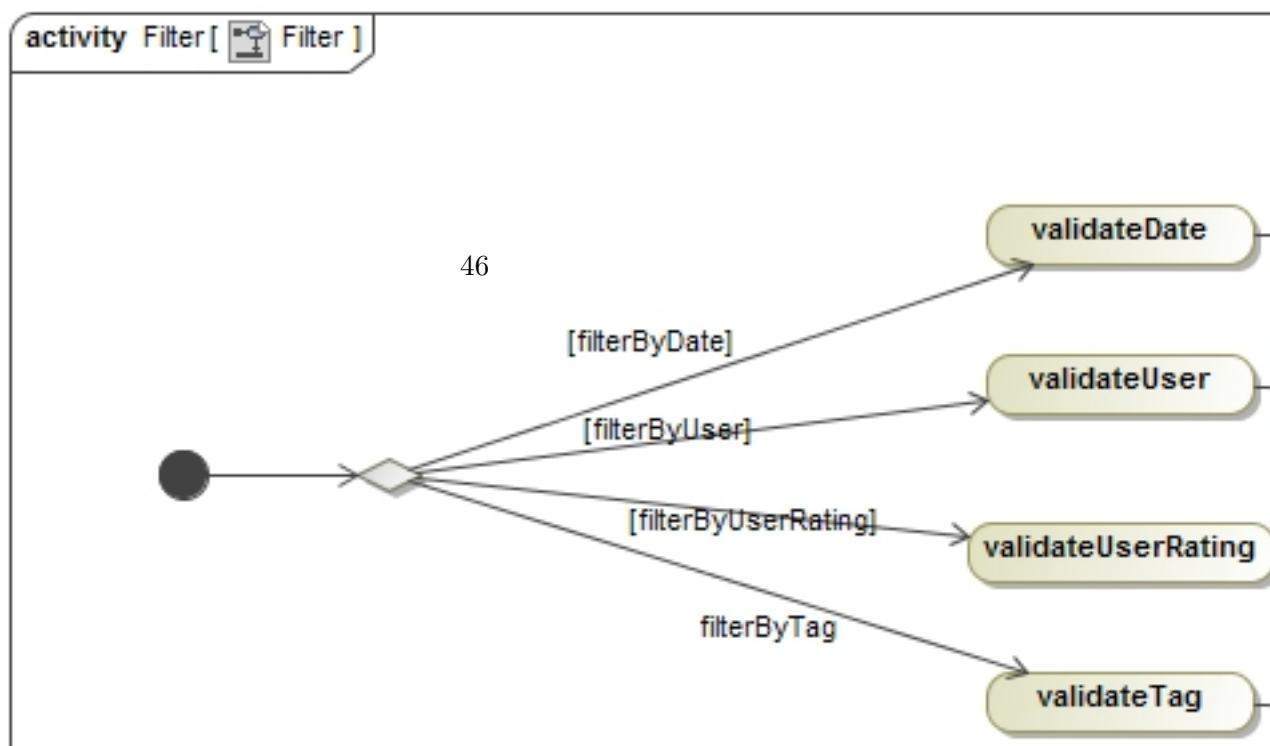
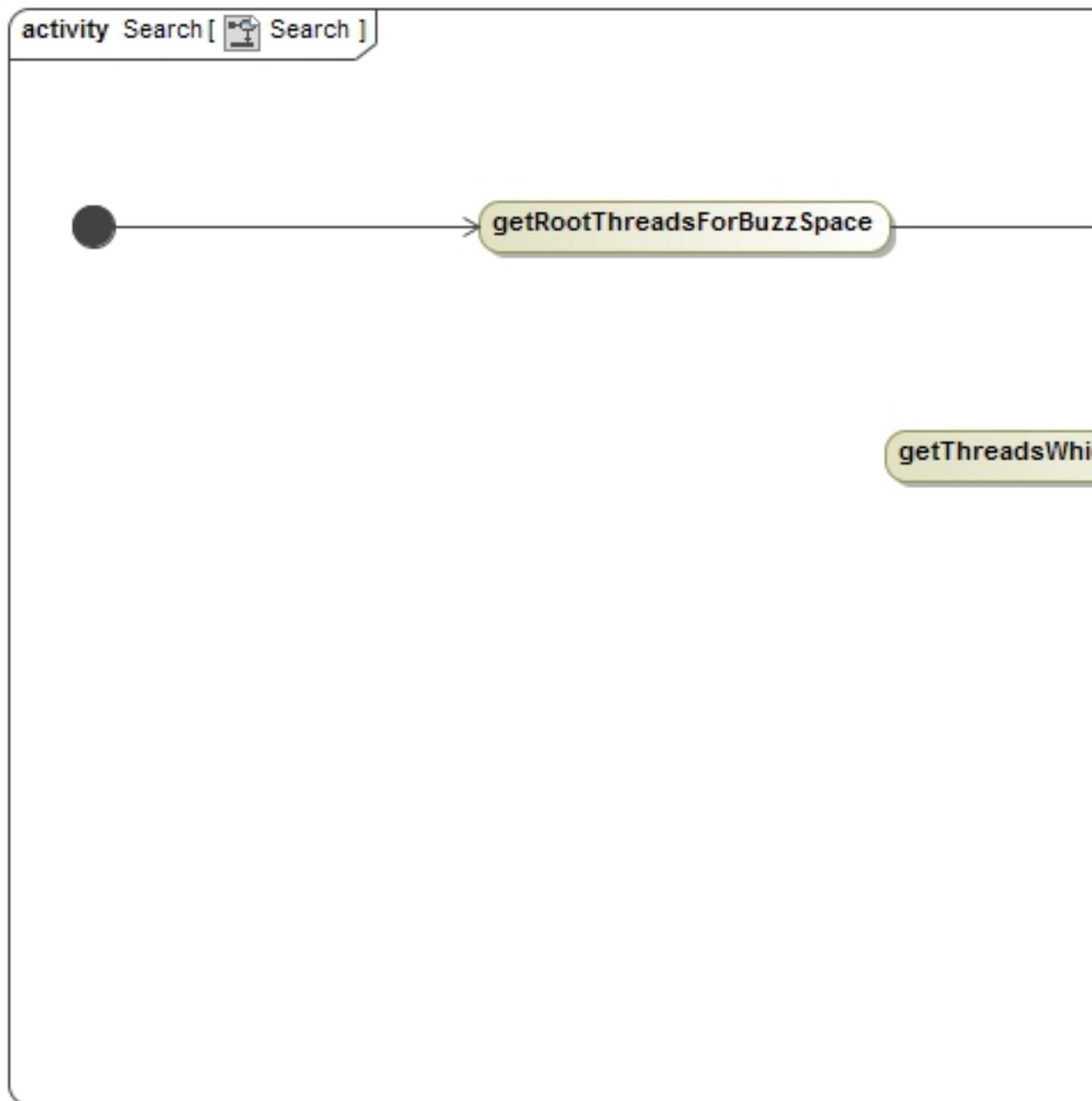
Input
-searchPhrase : String = null -searchAndFilter : boolean = true +filterby : StructuredExpression


Output
-results : String

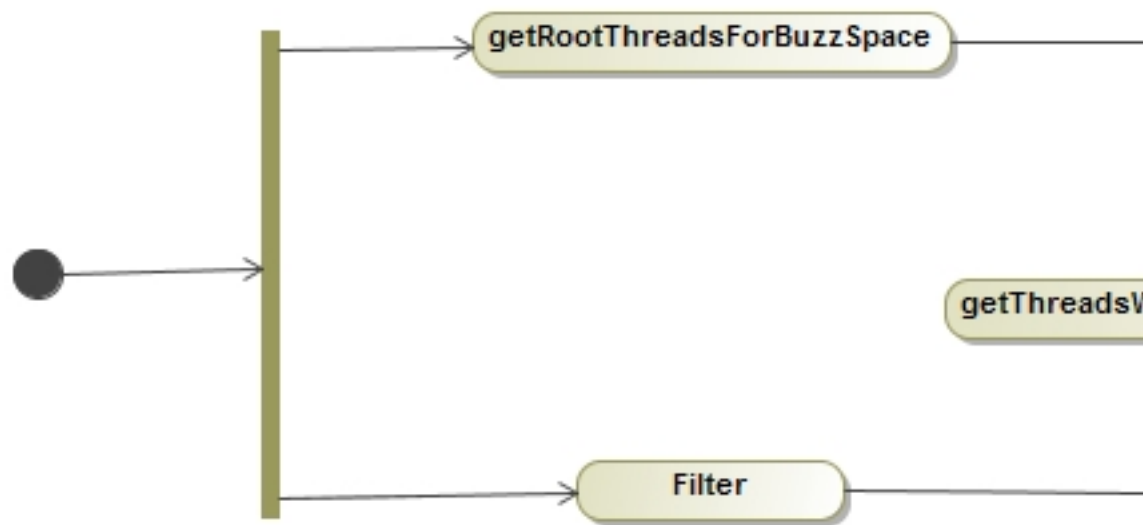
### 1.3 Required Functionality:



## 1.4 Process Specifications:



activity Search and Filter [  Search and Filter ]



## **2 Filter threads and buzzspaces**

**Description:**

**2.1 Prioritization:**

**2.2 Conditions and Data Structures:**

**Pre-Conditions:**

**Post-Conditions:**

**Requests and Results Data Structures:**

**2.3 Required Functionality:**

**2.4 Process Specifications:**



# Authorisation

## Use Cases

### 1 Login

#### Description:

This use case defines the ability for a user to log into her/his account to activate certain functionalities of the system.

#### 1.1 Prioritization:

Critical

#### 1.2 Conditions and Data Structures:

##### Pre-Conditions:

- The user must be a registered student.

##### Post-Conditions:

- If the user is not registered, an exception will be thrown and an error message will be shown to the user.
- If the login was successful, a message will be shown to inform the user that he/she is now logged in. More features will now be available to the user.

##### Requests and Results Data Structures:

#### 1.3 Required Functionality:

#### 1.4 Process Specifications:

### 2 Logout

#### Description:

1. User activates Logout function by selecting the Logout option.
2. System responds by presenting user with standard Guest interface with no profile or privileges

## 2.1 Prioritization:

This Use case is considered Important

## 2.2 Conditions and Data Structures:

Participants:

Initiated by User and communicates with system

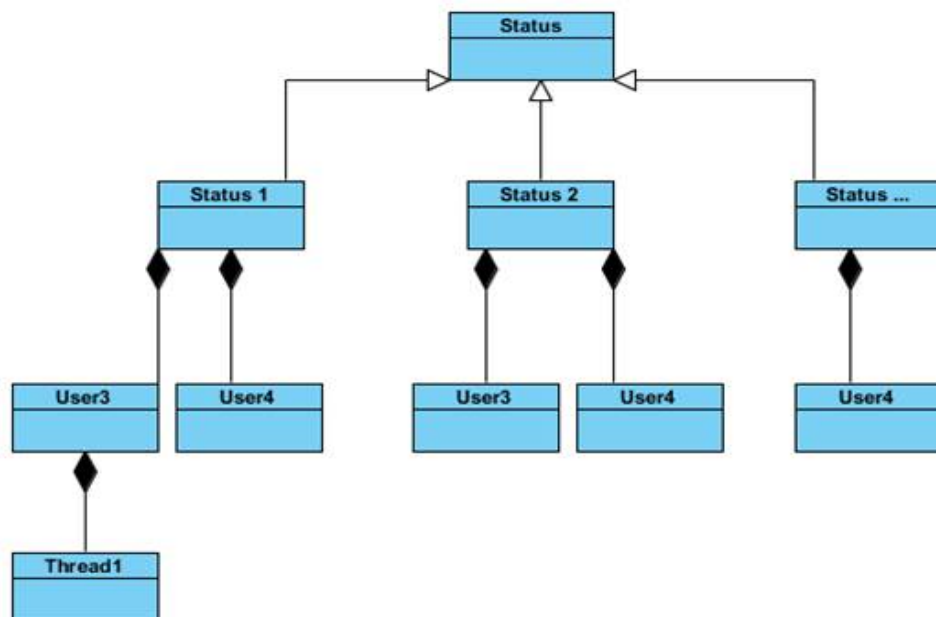
### Pre-Conditions:

1. The user must click on Logout function

### Post-Conditions:

1. System displays user with standard guest interface
2. User has no privileges and status and cannot perform any function that a logged in user can

### Requests and Results Data Structures:



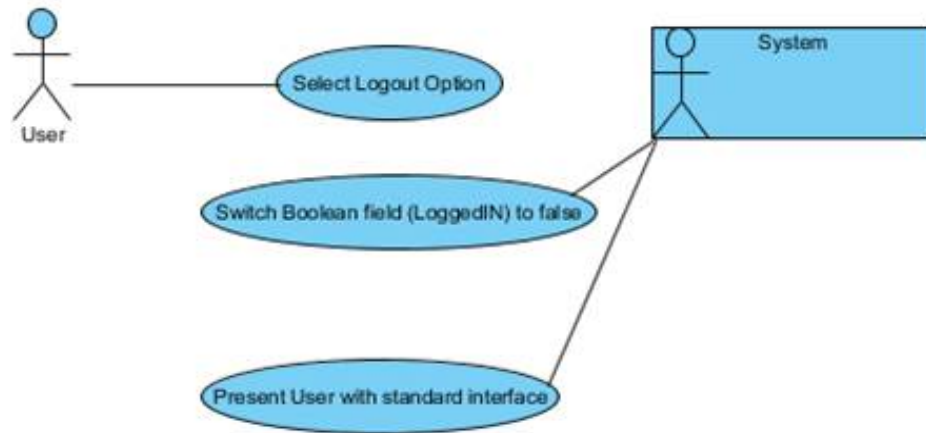
Steps:

1. User selects the logout option
  - 1.1. Boolean field in the specific user class e.g User3 will be switched to

false

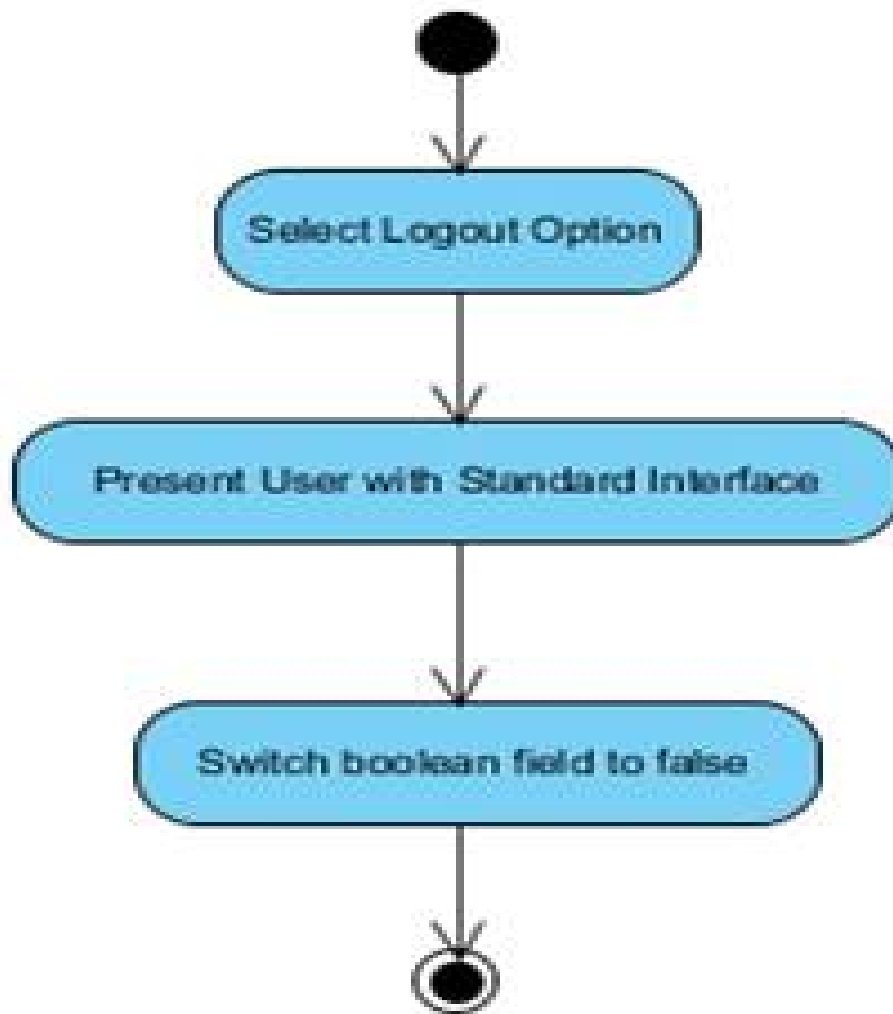
2. User will be presented with Guest interface

### 2.3 Required Functionality:

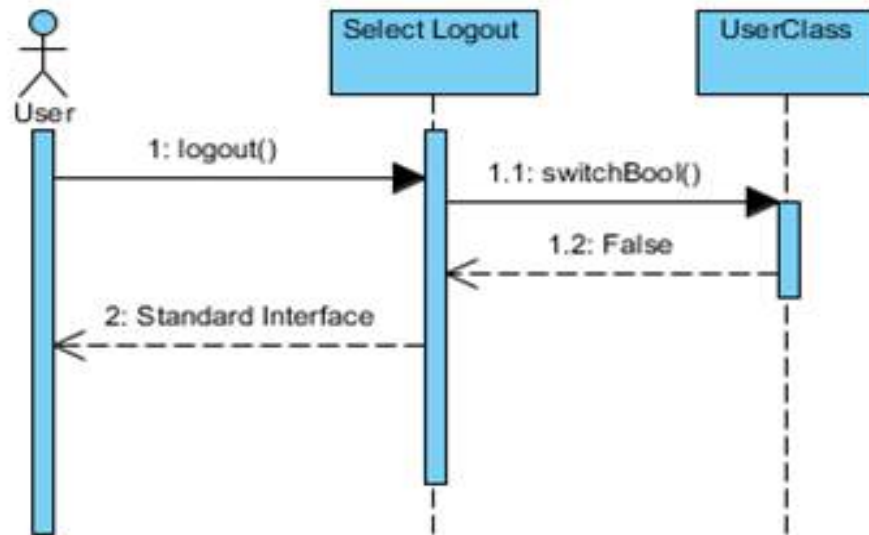


## 2.4 Process Specifications:

Activity Diagram



Sequence Diagram



### 3 Registration

#### Description:

This use case gives the ability to register to a specific BuzzSpace.

#### 3.1 Prioritization:

Critical

#### 3.2 Conditions and Data Structures:

##### Pre-Conditions:

- The user must be a registered student of the module associated with the BuzzSpace

##### Post-Conditions:

- If the user is not a registered student for the module, an exception is thrown and an error message is shown to the user.
- If the service was successful, a message is shown to the user to inform him/her that it was successful. The user will now be able to operate on the BuzzSpace.

**Requests and Results Data Structures:**

**3.3 Required Functionality:**

**3.4 Process Specifications:**

# Reporting

## Use Cases

### 1 Thread Evaluation Report

#### Description:

This use case gives a user with certain privileges the ability to calculate statistical information about threads and to generate reports. The use case include the sourcing of all thread components, the calculating of statistics, construction of report data objects and the rendering of a report onto a user interface or PDF file.

#### 1.1 Prioritization:

**Important:** Thread Evaluation Report

#### 1.2 Conditions and Data Structures:

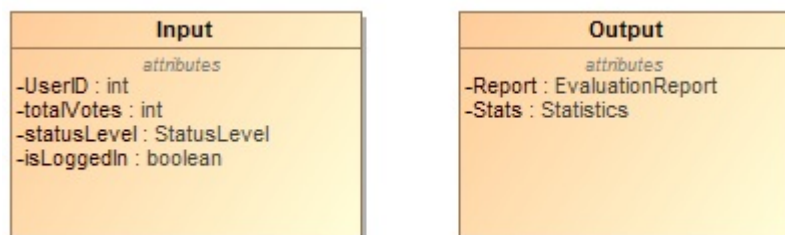
##### Pre-Conditions:

A user has to be a registered user of a Buzz Space and must also be logged in to be able to generate a report. The ThreadID has to exist.

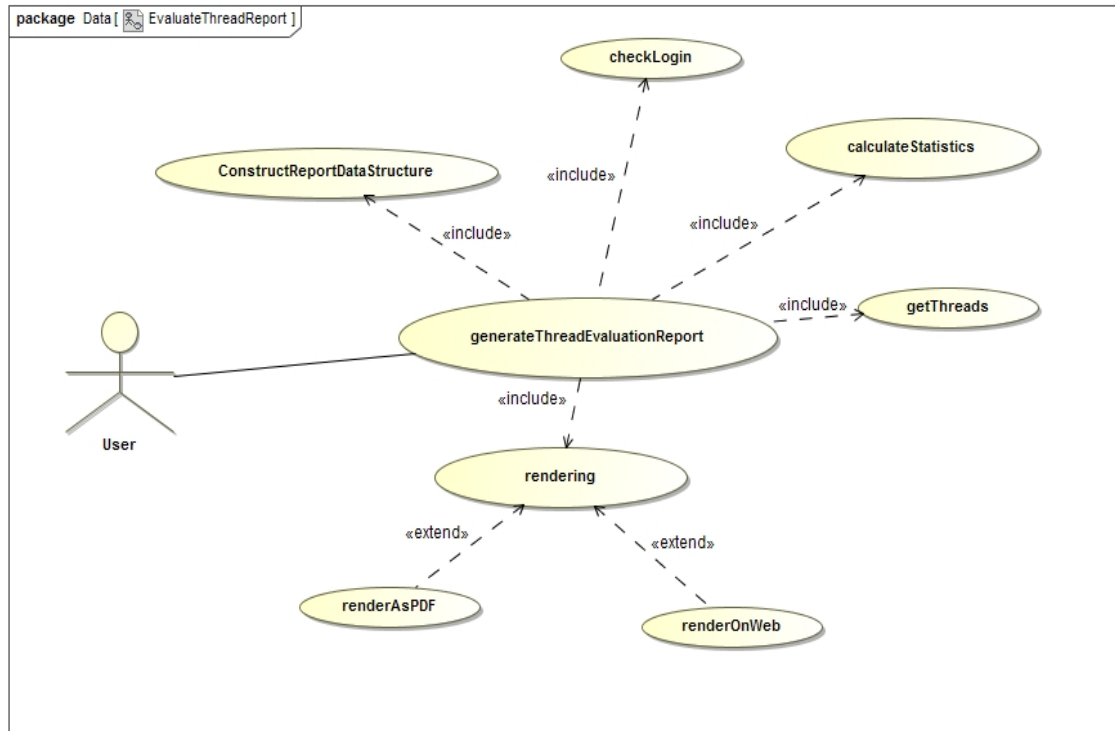
##### Post-Conditions:

A user should get statistical information about threads and generate a threads evaluation report.

##### Requests and Results Data Structures:

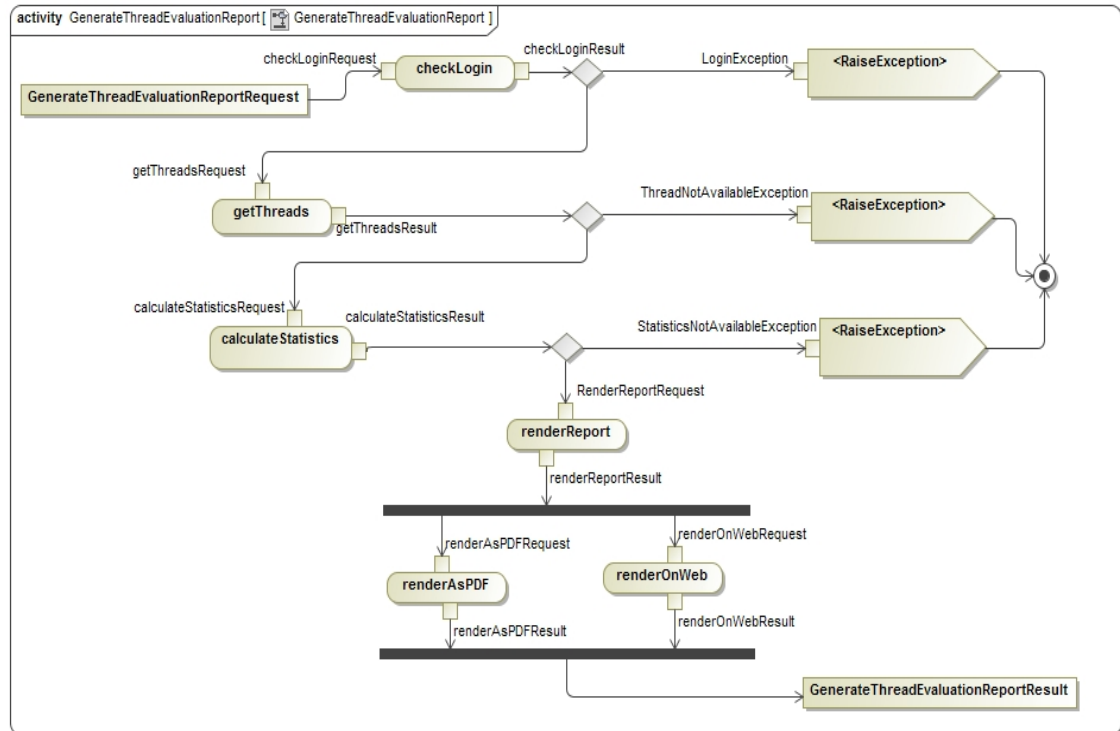


### 1.3 Required Functionality:





## 1.4 Process Specifications:



## 2 Student Post Evaluation Report

### Description:

This use case gives a user with certain privileges the ability to calculate statistical information about students and posts. They can also generate reports. The use case include the sourcing of all student and post components, the calculating of statistics, construction of report data objects and the rendering of a report onto a user interface or PDF file. Users request sufficient information to identify the student and the post. The response object has a timestamp.

### 2.1 Prioritization:

**Important:** Student and Post Evaluation Report

## 2.2 Conditions and Data Structures:

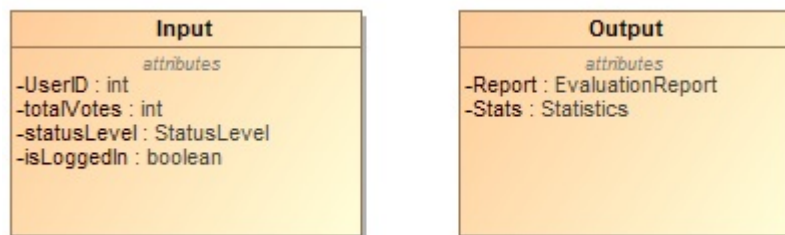
### Pre-Conditions:

A user has to be a registered user of a Buzz Space and must also be logged in to be able to generate a report. The UserID and post has to exist within a given Buzz Space.

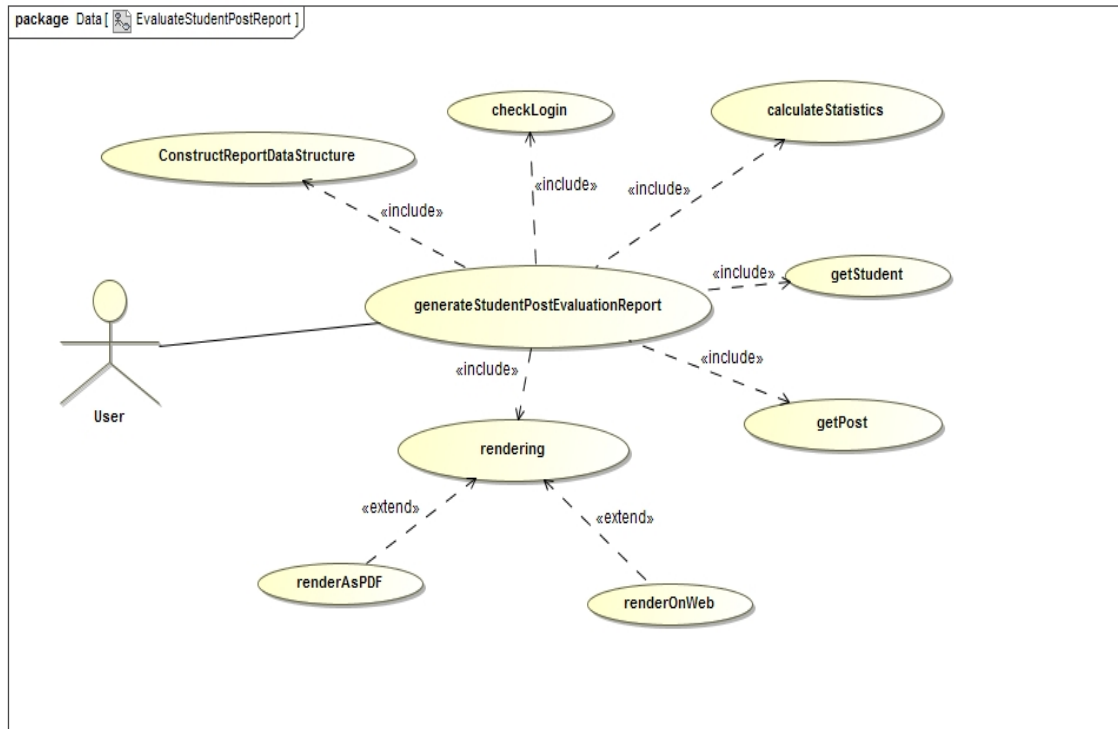
### Post-Conditions:

A user should get statistical information about students and posts as well as a student post evaluation report.

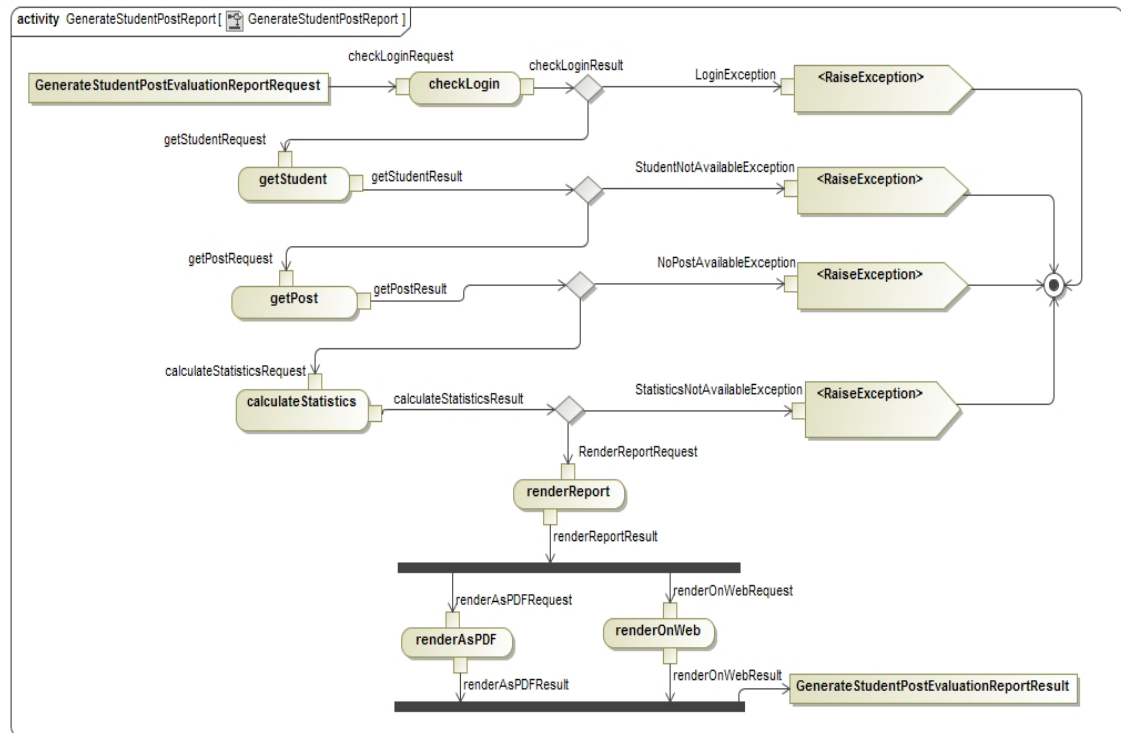
### Requests and Results Data Structures:



## 2.3 Required Functionality:



## 2.4 Process Specifications:



## 3 Audit Report

### Description:

This use case gives a lecturer the ability to generate an audit report for any thread, post or student, which will contain:

- A data/time stamp
- User contributions
- ThreadID for a specific thread
- UserID for a specific user
- Action which was performed by a user
- All thread creations, modifications, deletions or updates
- All user profile creations, modifications, deletions or updates
- All post creations, modifications, deletions
- All tag creations and usage

- The number of votes per post
- Users status levels and privileges
- Any request to open/close a thread
- Any request to generate any report including Thread Evaluation Report and Student Post Report

### **3.1 Prioritization:**

**Important:** Audit Report

### **3.2 Conditions and Data Structures:**

#### **Pre-Conditions:**

A lecturer has to be a registered user of a Buzz Space and must also be logged in to be able to generate a report.

#### **Post-Conditions:**

A lecturer should get a detailed audit report.

# Profile Handling

## Use Cases

### 1 Create Profile

#### Description:

**Description:** Profile creation allows the user to register and login to the system to view and interact with the BuzzSpace.

#### 1.1 Prioritization:

Critical

#### 1.2 Conditions and Data Structures:

##### Pre-Conditions:

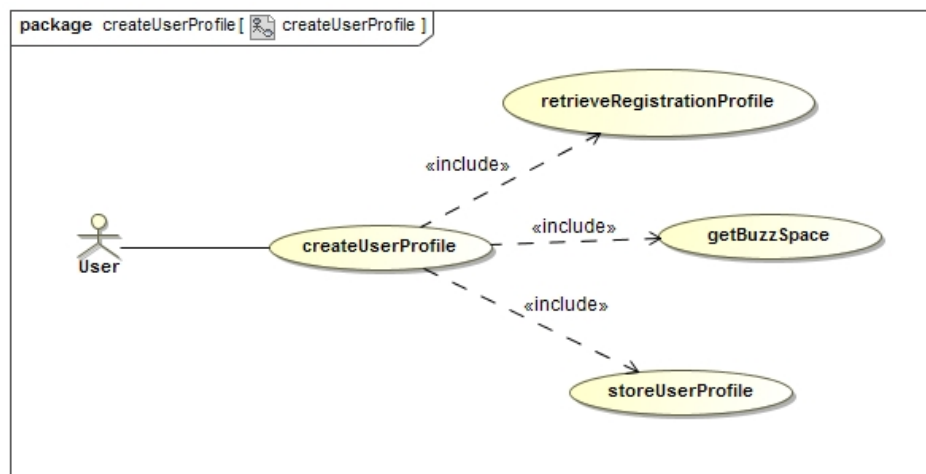
- User is logged into the system as a guest (There can only be one profile per user).

##### Post-Conditions:

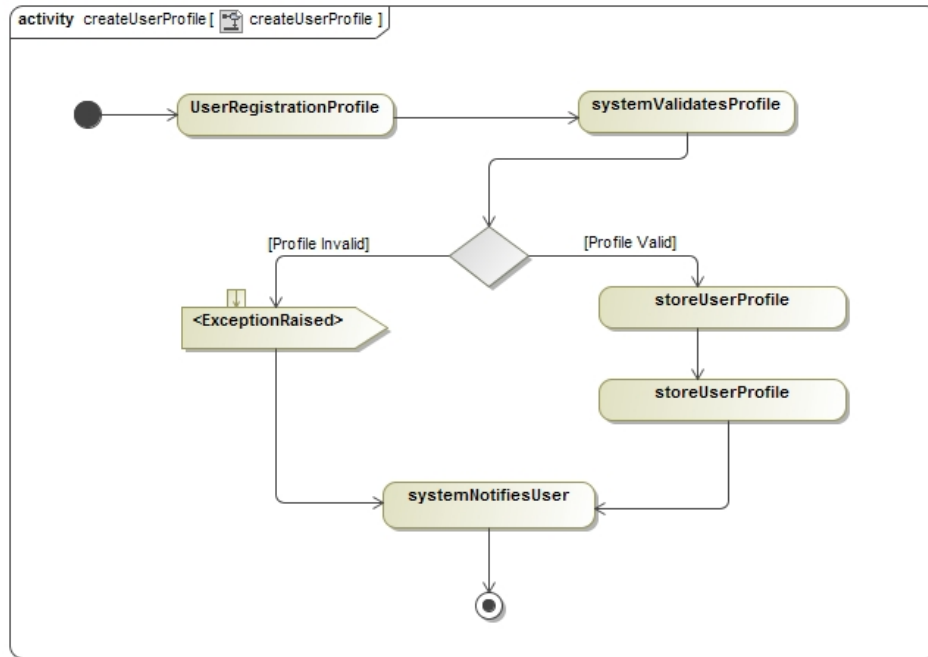
- Validates a user logged into the system and has a new profile.

##### Requests and Results Data Structures:

#### 1.3 Required Functionality:



## 1.4 Process Specifications:



## 2 Modify Profile

### Description:

**Description:** Profile modification allows the user logged into the system to modify the user profile that was previously created.

### 2.1 Prioritization:

Important

### 2.2 Conditions and Data Structures:

#### Pre-Conditions:

- User logged into the system.
- User has an already existing profile.

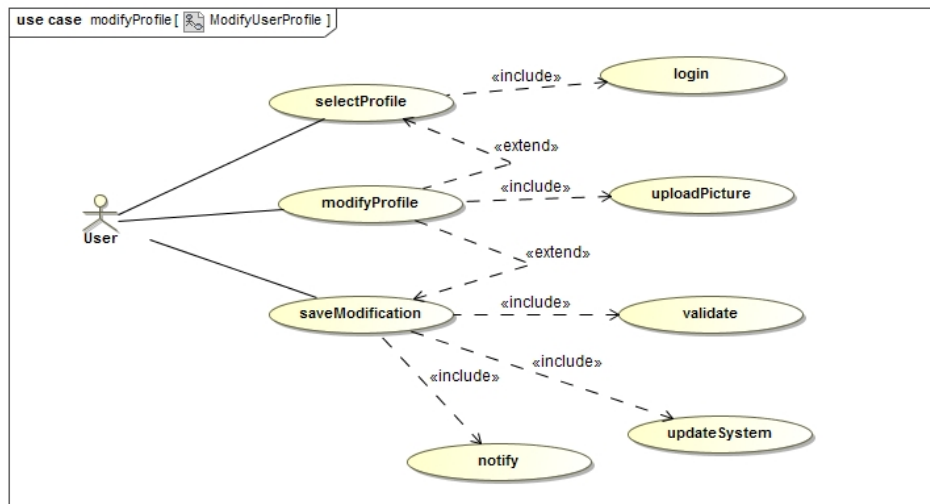
#### Post-Conditions:

- User modifies existing profile: User is logged in and user profile has been modified.

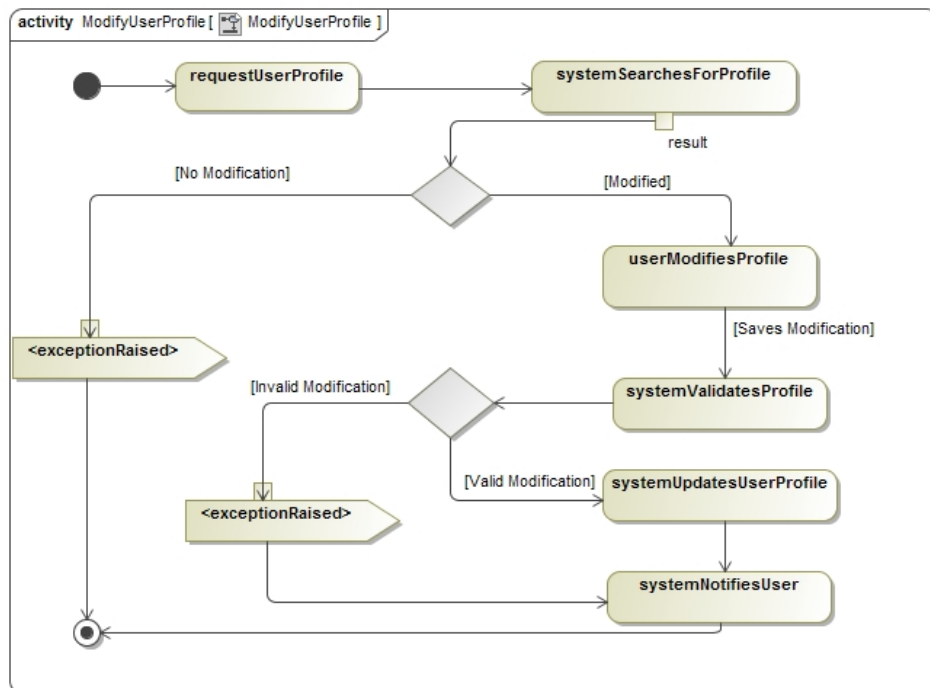
- User keeps already existing profile: User had logged into the system and the profile had not been modified.

## Requests and Results Data Structures:

### 2.3 Required Functionality:



### 2.4 Process Specifications:





### 3 View Profile

#### Description:

**Description:** Profile view allows the user to view ones or other users profiles on the BuzzSpace.

#### 3.1 Prioritization:

Nice To Have

#### 3.2 Conditions and Data Structures:

##### Pre-Conditions:

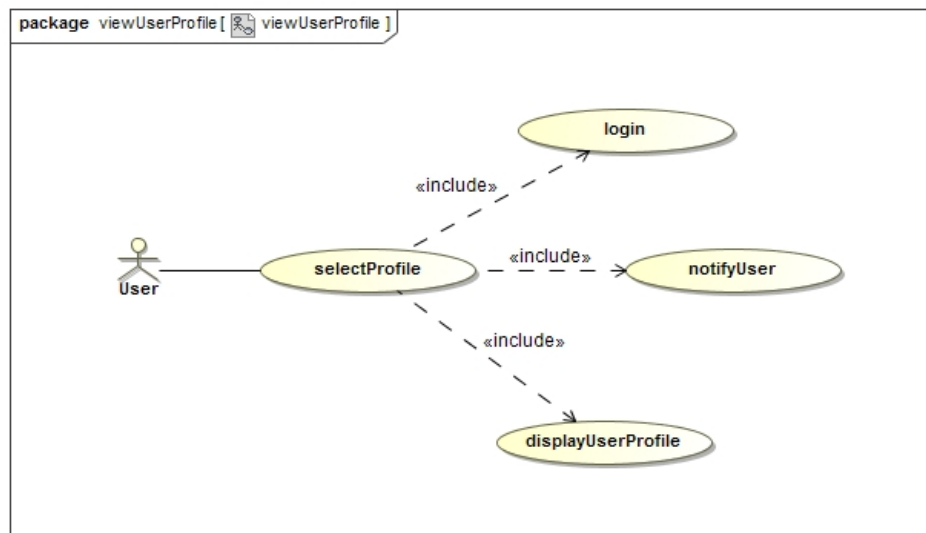
- User is logged into the system as a user not a guest.

##### Post-Conditions:

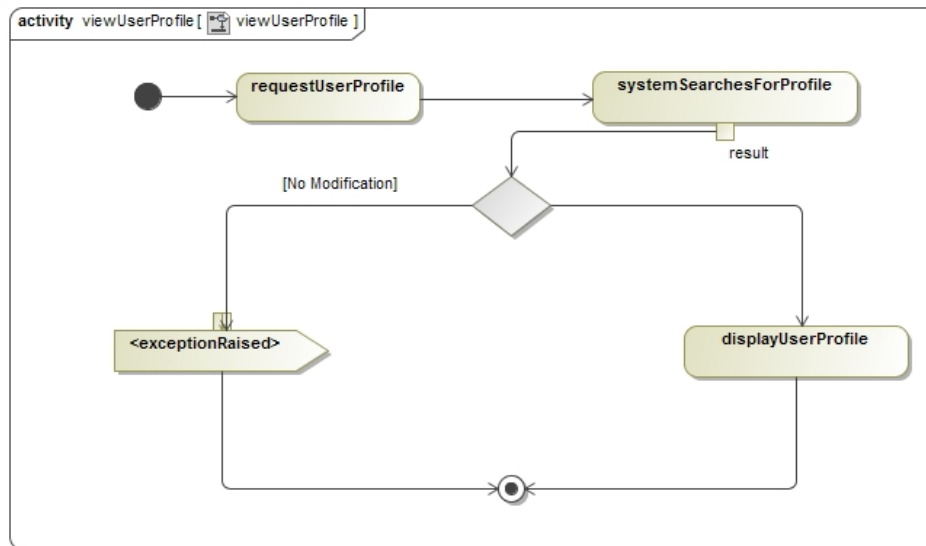
- The user is logged into the system and has a profile displayed.

#### Requests and Results Data Structures:

#### 3.3 Required Functionality:



### 3.4 Process Specifications:



## 4 Create Private Message

Description:

4.1 Prioritization:

4.2 Conditions and Data Structures:

Pre-Conditions:

Post-Conditions:

Requests and Results Data Structures:

4.3 Required Functionality:

4.4 Process Specifications:

## 5 Read Private Message

Description:

5.1 Prioritization:

5.2 Conditions and Data Structures:

Pre-Conditions:

Post-Conditions:

Requests and Results Data Structures:

5.3 Required Functionality:

5.4 Process Specifications:

## 6 Update Private Message

Description:

6.1 Prioritization:

6.2 Conditions and Data Structures:

Pre-Conditions:

Post-Conditions:

Requests and Results Data Structures:

6.3 Required Functionality:

6.4 Process Specifications:

## 7 Delete Private Message

Description: 67

7.1 Prioritization:

7.2 Conditions and Data Structures:

Pre-Conditions:

Post-Conditions:

Requests and Results Data Structures:

7.3 Required Functionality:

7.4 Process Specifications:

# Domain Model