Iskolivery: A Crowdsourced Courier Service

Analysis Model

Submitted to:

Asst. Prof. Ma. Rowena C. Solamo
Faculty Member
Department of Computer Science
College of Engineering
University of the Philippines, Diliman

Submitted by:
Fernandez, Aleksei Dominic C.
Legaspi, Bridget Noelle C.
Teves, Marc

In partial fulfillment of Academic Requirements
for the course
CS 191 Software Engineering
of the
1st Semester, AY 2018-2019

Unique Reference:

The documents are stored in https://github.com/marcteves/cs191-project/wiki/Project-Deliverables referenced with Group 1 - Iskolivery - Analysis Model.pdf.

Purpose:

This document is provided to derive the Analysis Model from the Use Case Model and Specifications.

Audience:

Students of University of the Philippines Diliman

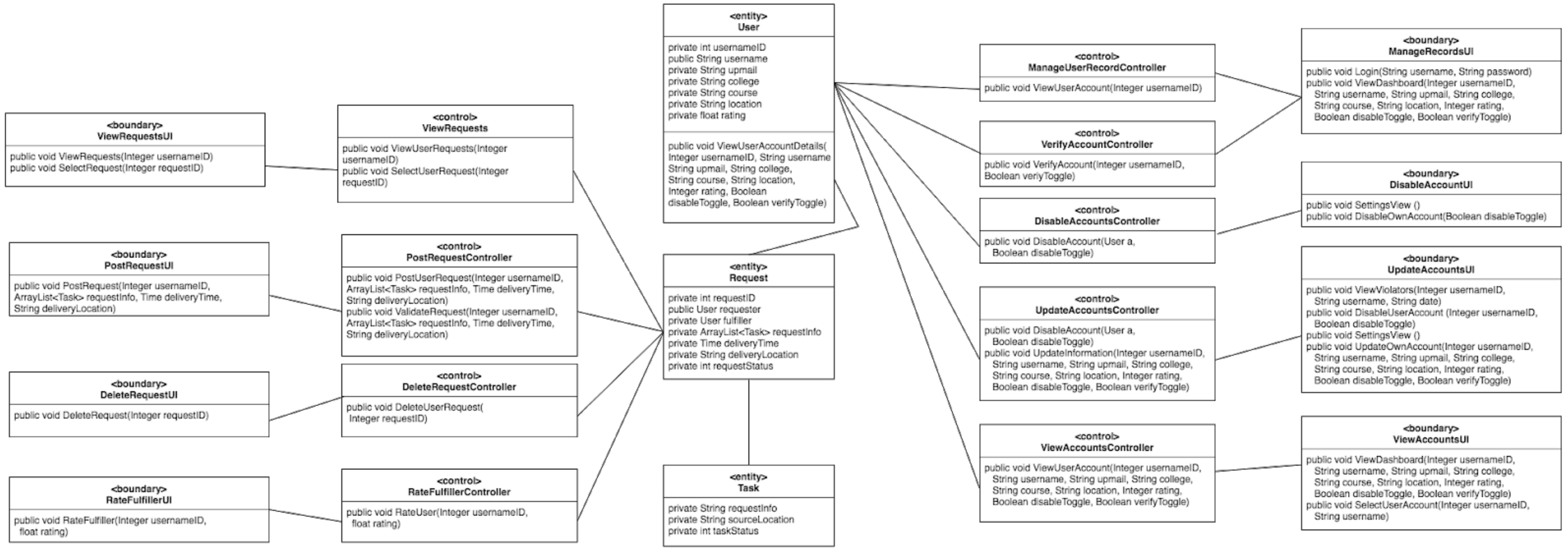
Revision Control:

Revision Date	***Person Responsible***	***Version Number***	***Modification***
10/03/2018	Bridget Noelle Legaspi	1.0	Initial document, behavioral models for Use Case 1-4, Analysis Model
10/04/2018	Bridget Noelle Legaspi	2.0	Added boundary and control classes for Use Case 1-4
10/04/2018	Marc Teves	2.1	Added boundary and control classes for Use Cases 5 and 7
10/04/2018	Aleksei Fernandez	2.2	Added boundary and control classes for Use Cases 6, 8, and 9

System Name: Iskolivery: A Crowdsourced Courier Service

Description: Iskolivery connects people who need items or services (eg. photocopying, buying school supplies, car ride, tutorial, etc.) to the people who can deliver these items or services for them. In the application, requesters will be able to post a request and fulfillers may accept these requests. Requesters can choose to add a "bounty" (or incentive) to hasten the fulfillment of their requests.

Analysis Model:


<entity>
User
private int usernameID
public String username
private String upmail
private String college
private String course
private String location
private float rating
public void ViewUserAccountDetails(
Integer usernameID, String username
String upmail, String college,
String course, String location,
Integer rating, Boolean
disableToggle, Boolean verifyToggle)
<boundary>
ViewRequestsUI
public void ViewRequests(Integer usernameID)
public void SelectRequest(Integer requestID)
<control>
ViewRequests
public void ViewUserRequests(Integer
usernameID)
public void SelectUserRequest(Integer
requestID)
<boundary>
PostRequestUI
public void PostRequest(Integer usernameID,
ArrayList<Task> requestInfo, Time deliveryTime,
String deliveryLocation)
<control>
PostRequestController
public void PostUserRequest(Integer usernameID,
ArrayList<Task> requestInfo, Time deliveryTime,
String deliveryLocation)
public void ValidateRequest(Integer usernameID,
ArrayList<Task> requestInfo, Time deliveryTime,
String deliveryLocation)
<entity>
Request
private int requestID
public User requester
private User fulfiller
private ArrayList<Task> requestInfo
private Time deliveryTime
private String deliveryLocation
private int requestStatus
<boundary>
DeleteRequestUI
public void DeleteRequest(Integer requestID)
<control>
DeleteRequestController
public void DeleteUserRequest(
Integer requestID)
<boundary>
RateFulfillerUI
public void RateFulfiller(Integer usernameID,
float rating)
<control>
RateFulfillerController
public void RateUser(Integer usernameID,
float rating)
<entity>
Task
private String requestInfo
private String sourceLocation
private int taskStatus
<control>
ManageUserRecordController
public void ViewUserAccount(Integer usernameID)
<boundary>
ManageRecordsUI
public void Login(String username, String password)
public void ViewDashboard(Integer usernameID,
String username, String upmail, String college,
String course, String location, Integer rating,
Boolean disableToggle, Boolean verifyToggle)
<control>
VerifyAccountController
public void VerifyAccount(Integer usernameID,
Boolean veriyToggle)
<boundary>
DisableAccountUI
public void SettingsView ()
public void DisableOwnAccount(Boolean disableToggle)
<control>
DisableAccountsController
public void DisableAccount(User a,
Boolean disableToggle)
<boundary>
UpdateAccountsUI
public void ViewViolators(Integer usernameID,
String username, String date)
public void DisableUserAccount (Integer usernameID,
Boolean disableToggle)
public void SettingsView ()
public void UpdateOwnAccount(Integer usernameID,
String username, String upmail, String college,
String course, String location, Integer rating,
Boolean disableToggle, Boolean verifyToggle)
<control>
UpdateAccountsController
public void DisableAccount(User a,
Boolean disableToggle)
public void UpdateInformation(Integer usernameID,
String username, String upmail, String college,
String course, String location, Integer rating,
Boolean disableToggle, Boolean verifyToggle)
<control>
ViewAccountsController
public void ViewUserAccount(Integer usernameID,
String username, String upmail, String college,
String course, String location, Integer rating,
Boolean disableToggle, Boolean verifyToggle)
<boundary>
ViewAccountsUI
public void ViewDashboard(Integer usernameID,
String username, String upmail, String college,
String course, String location, Integer rating,
Boolean disableToggle, Boolean verifyToggle)
public void SelectUserAccount(Integer usernameID,
String username)

Boundary Classes:

Class Name	**Description**
ManageRecordsUI	This is the interface of the administrator to the system whenever he or she needs to manage user recorda. *Responsibilities*: public void Login(String username, String password) public void ViewDashboard(Integer usernameID, String username, String upmail, String college, String course, String location, Integer rating, Boolean disableToggle, Boolean verifyToggle) public void ViewViolators(Integer usernameID, String username, String date)
DisableAccountsUI	This is the interface of the user whenever he or she wants to disable his/her own account. *Responsibilities*: public void SettingsView () public void DisableUserAccount (Boolean disableToggle)
UpdateAccountsUI	This is the interface of the administrator whenever he or she selects an account to be updated. The admin can only update the status (active/disable) of the user. This is the interface of the user whenever he or she wants to update his/her own account information. *Responsibilities*: public void ViewViolators(Integer usernameID, String username, String date) public void DisableUserAccount (Integer usernameID, Boolean disableToggle) public void SettingsView () public void UpdateOwnAccount(Integer usernameID, String username, String upmail, String college, String course, String location, Integer rating, Boolean disableToggle, Boolean verifyToggle)
ViewAccountsUI	This is the interface of the administrator or user whenever he/she wants to view another user account. *Responsibilities*: public void ViewDashboard(Integer usernameID, String username, String upmail, String college, String course, String location, Integer rating, Boolean disableToggle, Boolean verifyToggle) public void SelectUserAccount(Integer usernameID, String username)
RateFulfillerUI	This is the interface for the requester to rate their fulfiller *Responsibilities*: public void RateFulfiller(Integer usernameID, float rating)
ComplainUserUI	This is the interface for the user to complain another user *Responsibilities*: public void selectUser(String username) public void enterComplainBody(String complainBody) public void submitComplaint(String complainantUsername, complainedUsername)
ViewRequestsUI	This is the interface for viewing and selecting requests. *Responsibilities*: public void ViewRequests(Integer usernameID) public void SelectRequest(Integer requestID)

PostRequestUI	This is the interface for posting new requests. *Responsibilities*: public void PostRequest(Integer usernameID, ArrayList<Task> requestInfo, Time deliveryTime, String deliveryLocation)
DeleteRequestUI	This is the interface for deleting a posted request. *Responsibilities*: public void DeleteRequest(Integer requestID)
ManageAcceptedRequestsUI	This is the interface for the fulfiller to manage their own accepted requests (ie. update, delete, add a request) *Responsibilities*: public void viewAcceptedRequests(username, usernameID)
RateRequesterUI	This is the interface for the fulfiller to rate their requester *Responsibilities*: public void enterRating(Interger rating) public void enterAdditionalComments(String addtlCommentsBody) public void submitRating(String username, Integer usernameID, Integer rating, Integer requestID)

Control Classes:

Class Name	**Description**
ManageUserRecordController	This is the control that allows user (can be admin) to view a user account to the system. *Responsibilities*: public void ViewUserAccount(Integer usernameID)
VerifyAccountController	This is the control that allows the admin to verify user accounts by checking if the verification email is received from the UP Mail. This extends ManageUserRecordController *Responsibilities*: public void VerifyAccount(Integer usernameID, Boolean veriyToggle)
DisableAccountsController	This is the control that allows the admin to disable user accounts if determined necessary, and also allows user to disable his/her own account if he/she opts out of the requester-fulfiller pool. *Responsibilities*: public void DisableAccount(User a, Boolean disableToggle)
UpdateAccountsController	This is the control that allows the admin to update status (active/disabled) of user accounts, and also allows user to updates his/her own account information. *Responsibilities*: public void DisableAccount(User a, Boolean disableToggle) public void UpdateInformation(Integer usernameID, String username, String upmail, String college, String course, String location, Integer rating, Boolean disableToggle, Boolean verifyToggle)
ViewAccountsController	This is the control that allows the admin and general user to view account information. *Responsibilities*: public void ViewUserAccount(Integer usernameID)
RateFulfillerController	This is the control that allows a requester to rate their fulfiller *Responsibilities*: public void RateUser(Integer usernameID, float rating)
ComplainUserController	This is the control that allows a user to submit a complaint on another user *Responsibilities*: public void saveComplaint(String complainantUsername, Integer complainantID, String complaintBody, complainedUsername, Integer complainedID)
ViewRequestsController	This is the control that allows a user to view requests. *Responsibilities*: public void ViewUserRequests(Integer usernameID) public void SelectUserRequest(Integer requestID)
PostRequestController	This is the control that allows a user to view requests. *Responsibilities*: public void PostRequest(Integer usernameID, ArrayList<Task> requestInfo, Time

	deliveryTime, String deliveryLocation) public void ValidateRequest(Integer usernameID, ArrayList<Task> requestInfo, Time deliveryTime, String deliveryLocation)
DeleteRequestController	This is the control that allows a user to delete a request they posted. *Responsibilities*: public void DeleteUserRequest(Integer requestID)
ManageAcceptedRequestsController	This is the control that allows a user to manage their accepted requests (ie. update, delete, add) *Responsibilities*: public void viewAcceptedRequest(String username, Integer usernameID, Integer requestID)
RateRequesterControl	This is the control that allows a fulfiller to rate their requester *Responsibilities*: public void saveRating(String username, Integer usernameID, Integer rating, Integer requestID)

Entity Classes:

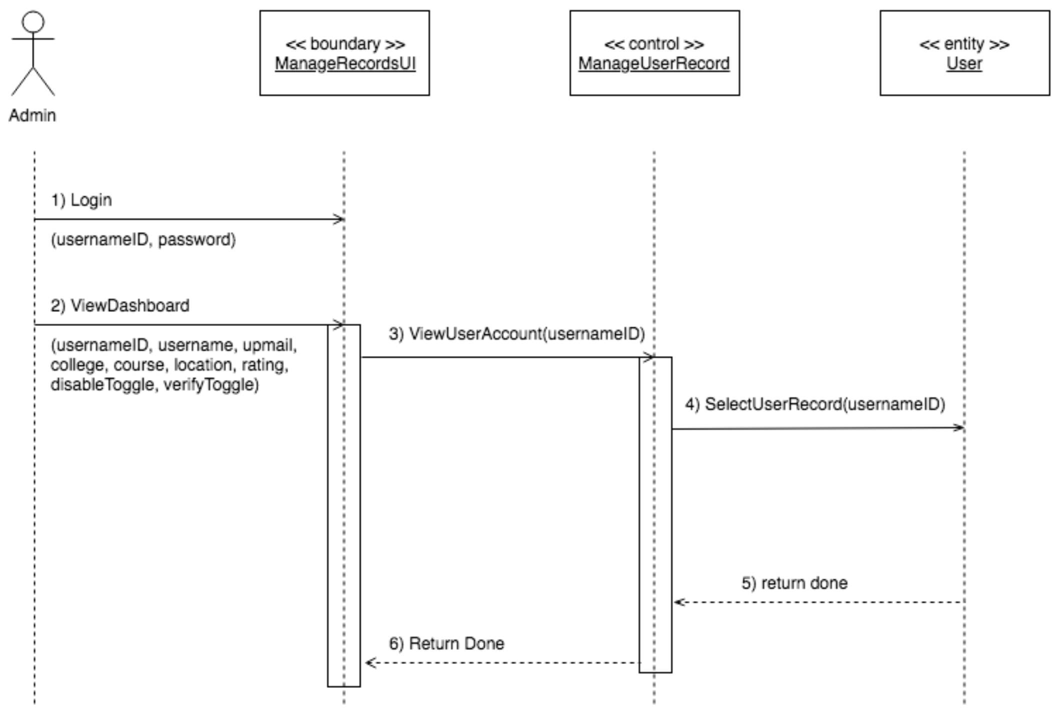
Class Name	**Description**
User	This is the entity class User, which contains the data about the athlete. Admin is just a special instance of User. *Attributes*: private int usernameID public String username private String upmail private String college private String course private String location private float rating *Methods*: public void ViewUserAccountDetails(Integer usernameID, String username, String upmail, String college, String course, String location, Integer rating, Boolean disableToggle, Boolean verifyToggle)
Request	This is the entity class Request, which contains the information about the request of the user. *Attributes*: public User requester private User fulfiller private ArrayList<Task> requestInfo private Time deliveryTime private String deliveryLocation private int requestStatus
Task	This is the entity class Task, which contains the data about the tasks. This contains the actual information of the request of user. *Attributes*: private String requestInfo private String sourceLocation private int taskStatus

Behavioral Model:

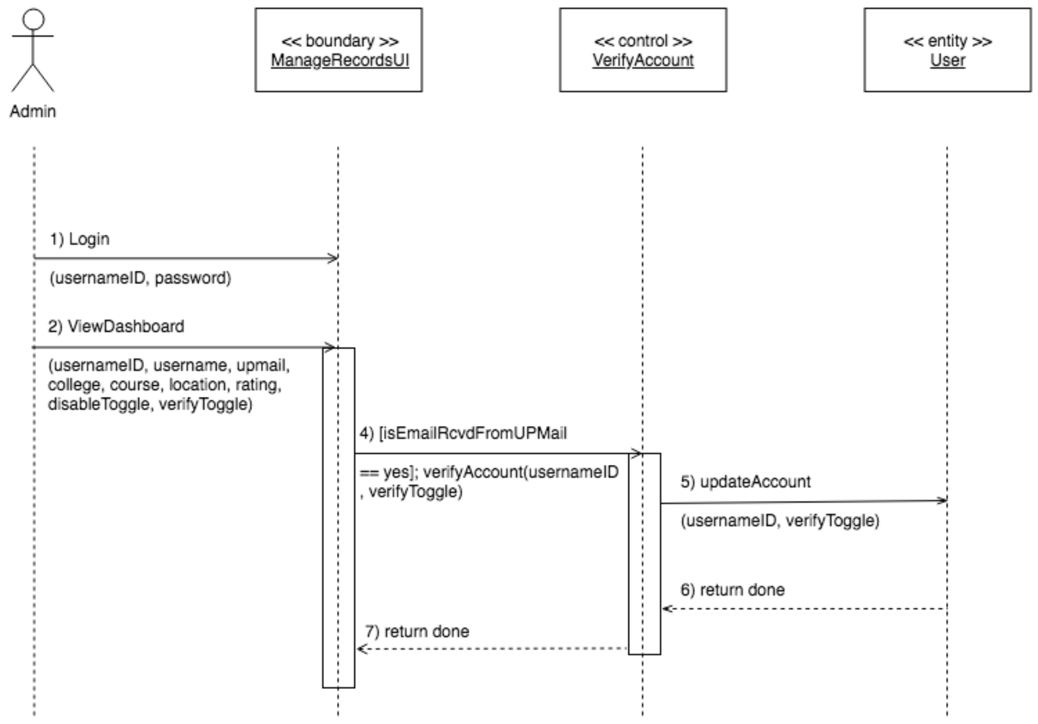
***Use-Case Name*:** Use-Case 1.0 Manage User Record

Description: Administrator can manage the records of all user accounts. This use case is extended to two more use-cases: verifying and disabling user accounts.

Scenario 1: Administrator manages all user accounts record (Basic Flow)


Admin
<< boundary >> ManageRecordsUI
<< control >> ManageUserRecord
<< entity >> User
1) Login
(usernameID, password)
2) ViewDashboard
(usernameID, username, upmail, college, course, location, rating, disableToggle, verifyToggle)
3) ViewUserAccount(usernameID)
4) SelectUserRecord(usernameID)
5) return done
6) Return Done

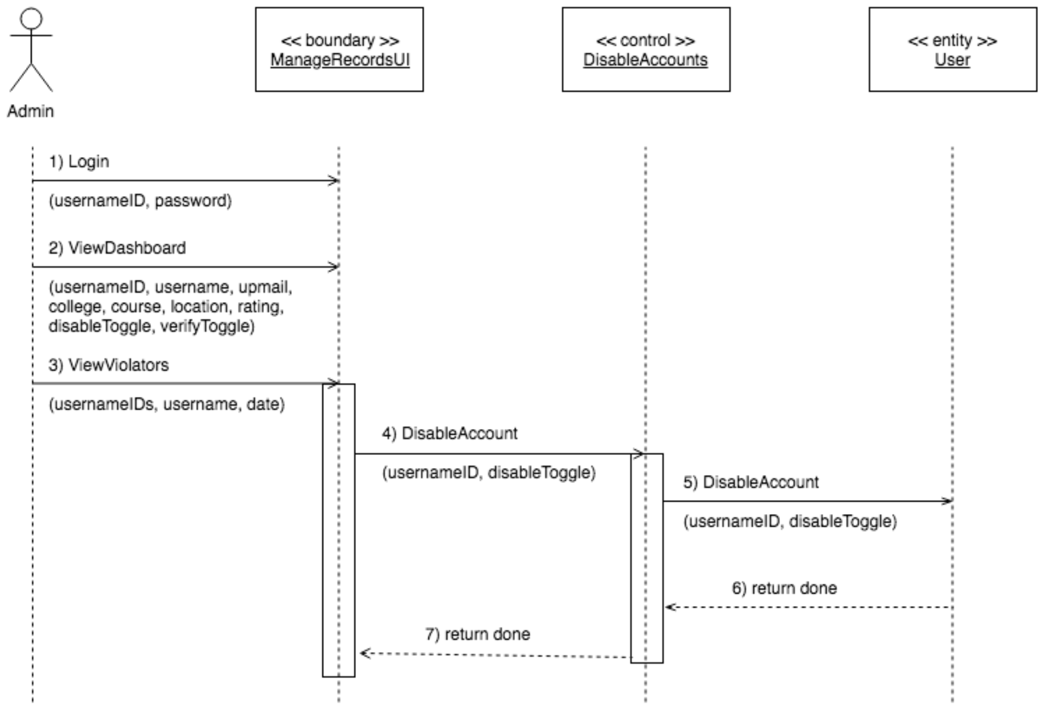
Scenario 2: Administrator verifies user account (Use Case 1.0 Verify User Account)


Admin
<< boundary >> ManageRecordsUI
<< control >> VerifyAccount
<< entity >> User
1) Login
(usernameID, password)
2) ViewDashboard
(usernameID, username, upmail, college, course, location, rating, disableToggle, verifyToggle)
4) [isEmailRcvdFromUPMail == yes]; verifyAccount(usernameID , verifyToggle)
5) updateAccount
(usernameID, verifyToggle)
6) return done
7) return done

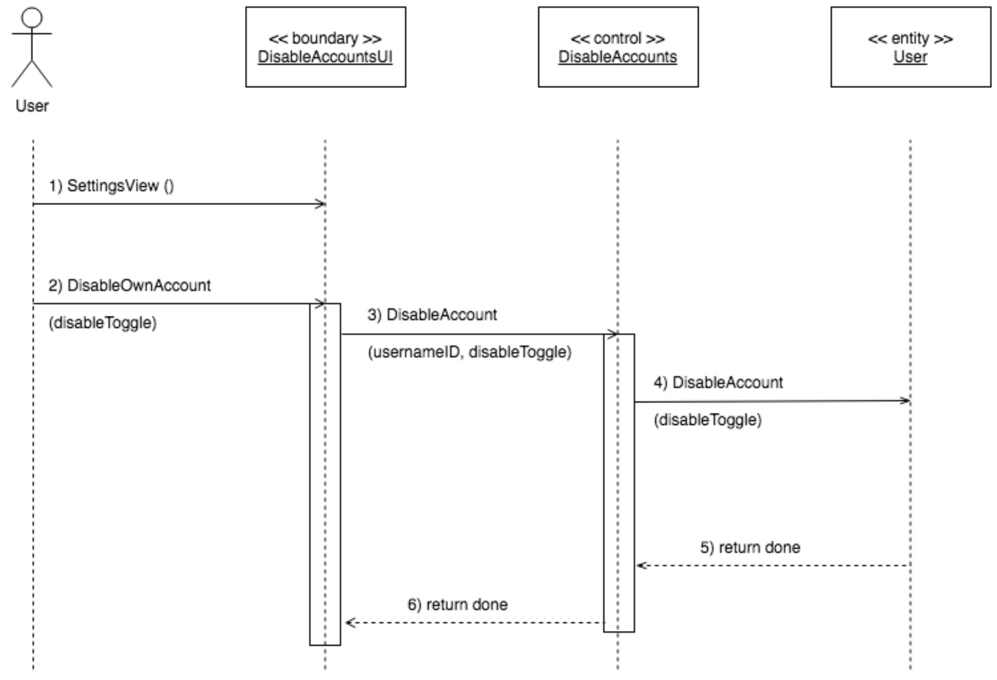
***Use-Case Name*:** Use-Case 2.0 Disable User Account

Description: Administrator can disable (with deliberation) user accounts that are complained by other users, or disable user accounts that violate the Iskolivery's terms and conditions. General user can also opt to disable their own user account.

Scenario 1: Administrator disables user account (Basic Flow)


Admin
<< boundary >> ManageRecordsUI
<< control >> DisableAccounts
<< entity >> User
1) Login
(usernameID, password)
2) ViewDashboard
(usernameID, username, upmail, college, course, location, rating, disableToggle, verifyToggle)
3) ViewViolators
(usernameIDs, username, date)
4) DisableAccount
(usernameID, disableToggle)
5) DisableAccount
(usernameID, disableToggle)
6) return done
7) return done

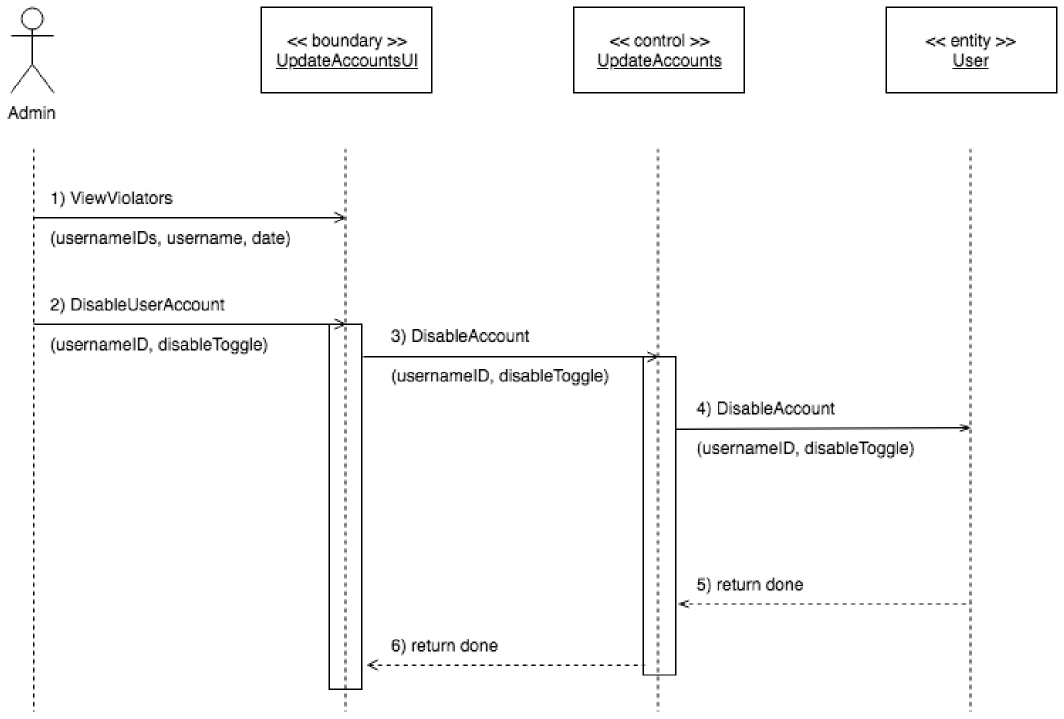
Scenario 2: User disables his/her own user account



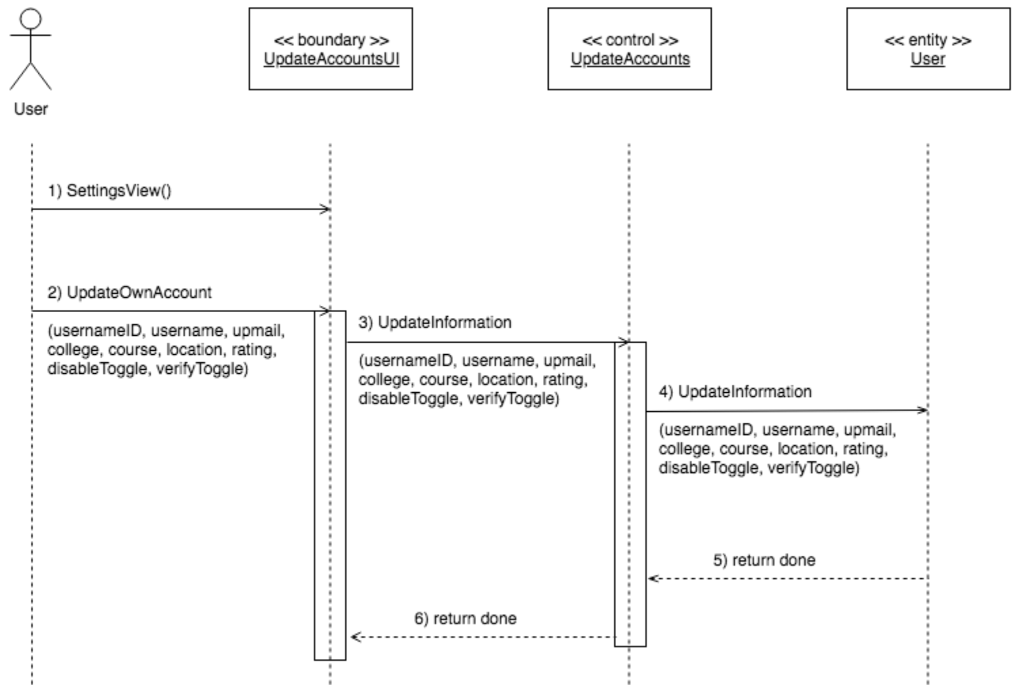
***Use-Case Name*:** Use-Case 3.0 Update User Account Info

Description: General user can update their user account information. Updated information is useful for the "matchmaking" algorithm of the application and contacting the user. The only attribute that the admin can update is whether the account is still active or disabled.

Scenario 1: Administrator updates user account (Basic Flow)



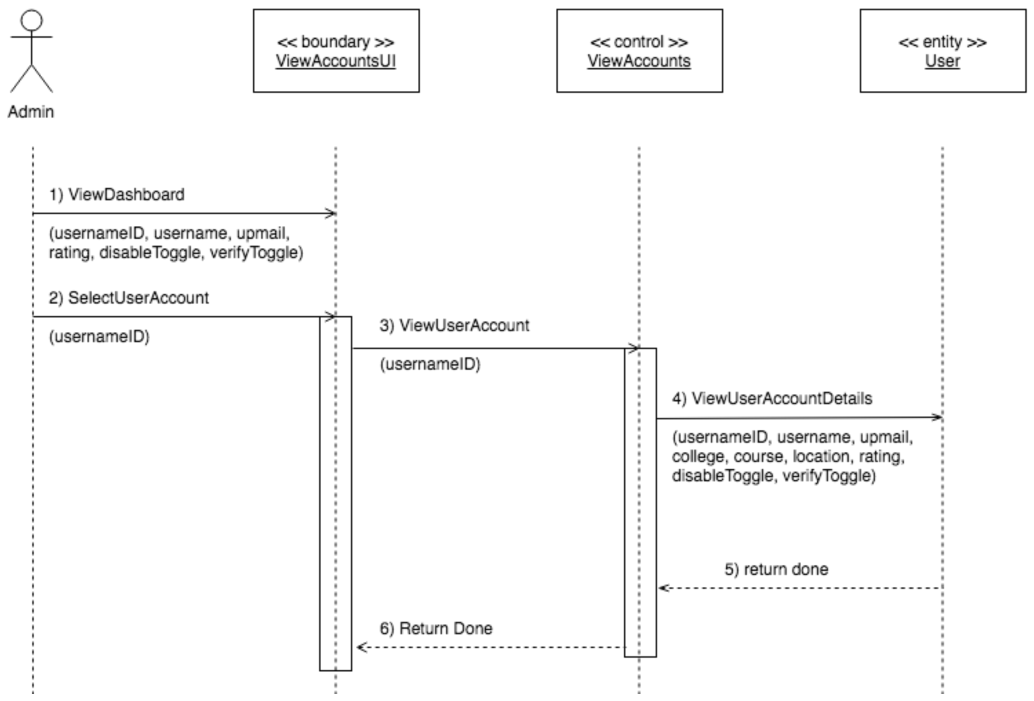
Scenario 2: User updates his/her own user account


User
<< boundary >>
UpdateAccountsUI
<< control >>
UpdateAccounts
<< entity >>
User
1) SettingsView()
2) UpdateOwnAccount
(usernameID, username, upmail, college, course, location, rating, disableToggle, verifyToggle)
3) UpdateInformation
(usernameID, username, upmail, college, course, location, rating, disableToggle, verifyToggle)
4) UpdateInformation
(usernameID, username, upmail, college, course, location, rating, disableToggle, verifyToggle)
5) return done
6) return done

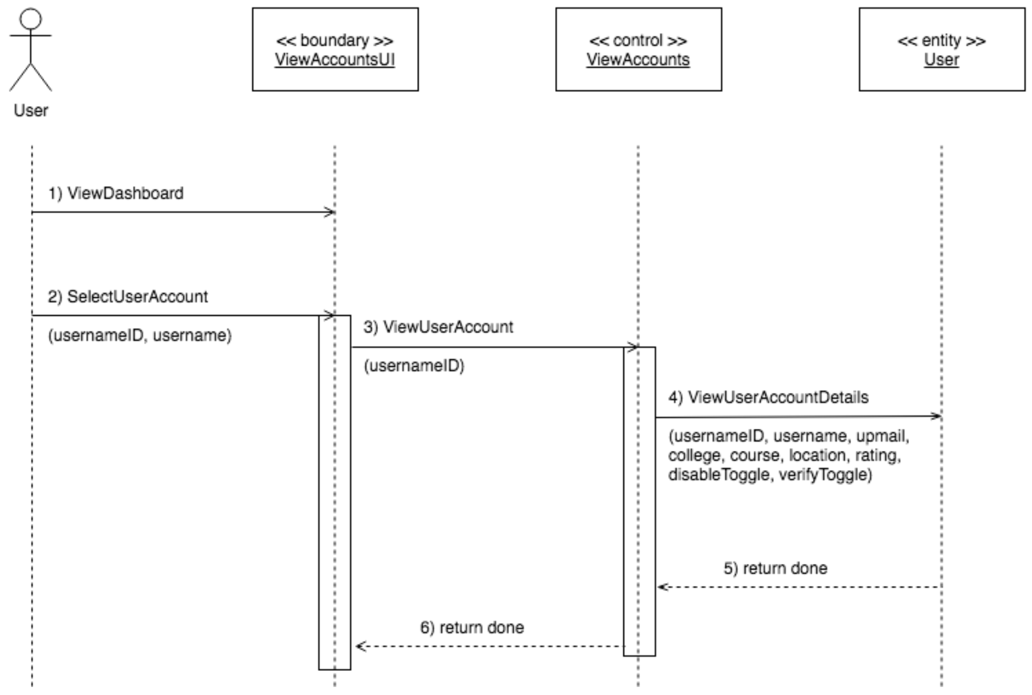
Use-Case Name: Use-Case 4.0 View User Account Info

Description: General user can view the information of other users. This is helpful for requesters to know the rating of their fulfillers, and vice versa. For the admin side, this is useful for deliberating complained users.

Scenario 1: Administrator views user account (Basic Flow)


Admin
<< boundary >> ViewAccountsUI
<< control >> ViewAccounts
<< entity >> User
1) ViewDashboard
(usernameID, username, upmail, rating, disableToggle, verifyToggle)
2) SelectUserAccount
(usernameID)
3) ViewUserAccount
(usernameID)
4) ViewUserAccountDetails
(usernameID, username, upmail, college, course, location, rating, disableToggle, verifyToggle)
5) return done
6) Return Done

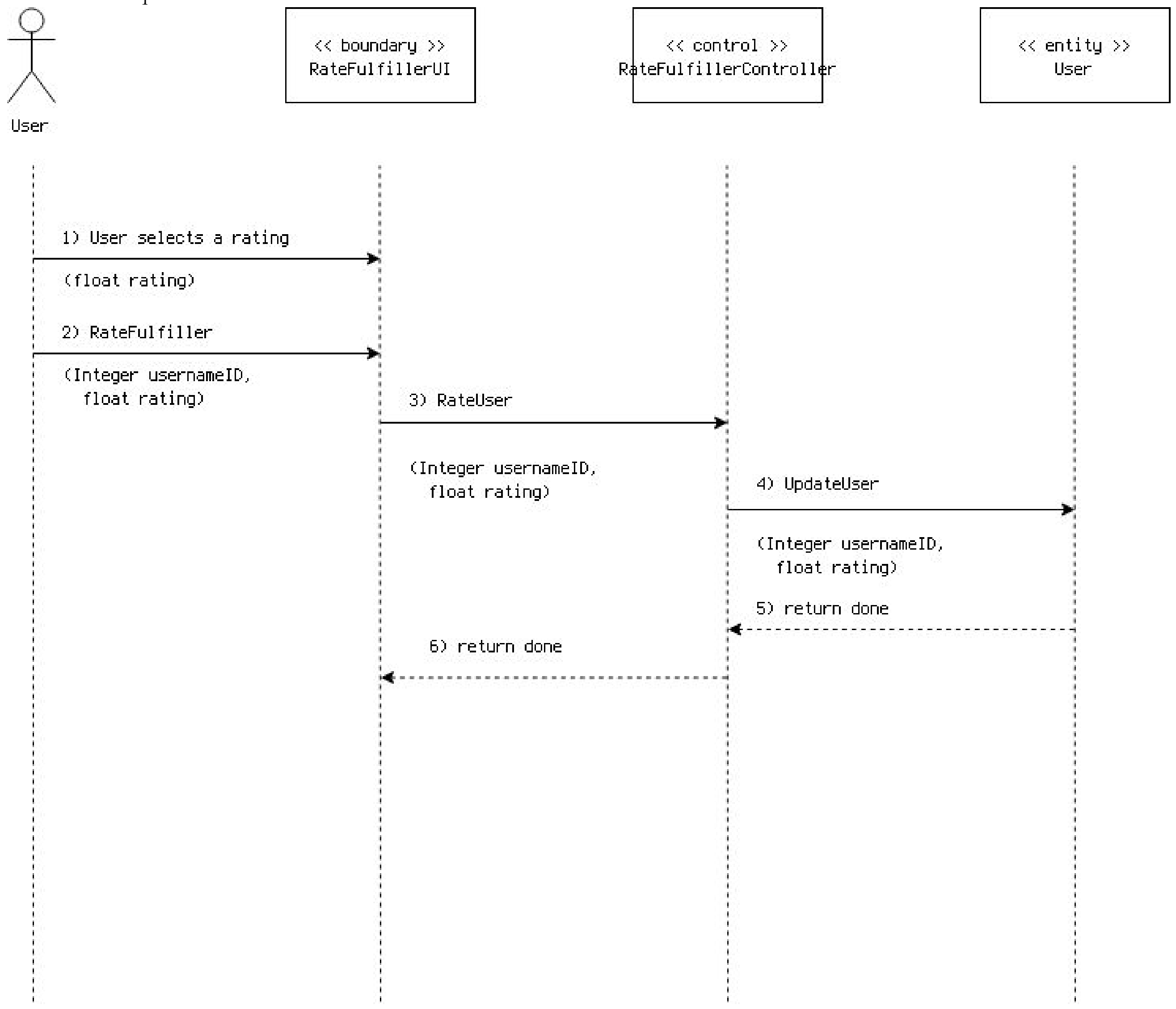
Scenario 2: User views other user account


User
<< boundary >> ViewAccountsUI
<< control >> ViewAccounts
<< entity >> User
1) ViewDashboard
2) SelectUserAccount
(usernameID, username)
3) ViewUserAccount
(usernameID)
4) ViewUserAccountDetails
(usernameID, username, upmail, college, course, location, rating, disableToggle, verifyToggle)
5) return done
6) return done

***Use-Case Name*:** Use-Case 5.0 Rate Fulfiller

Description: After a request is fulfilled by a fulfiller, the requester must give a rating of how pleasant their experience was with the fulfiller. This will be a number from 1 - 5, with 1 being the least pleasant experience and 5 being the most. A user's total rating is the average of all the ratings he or she has received.

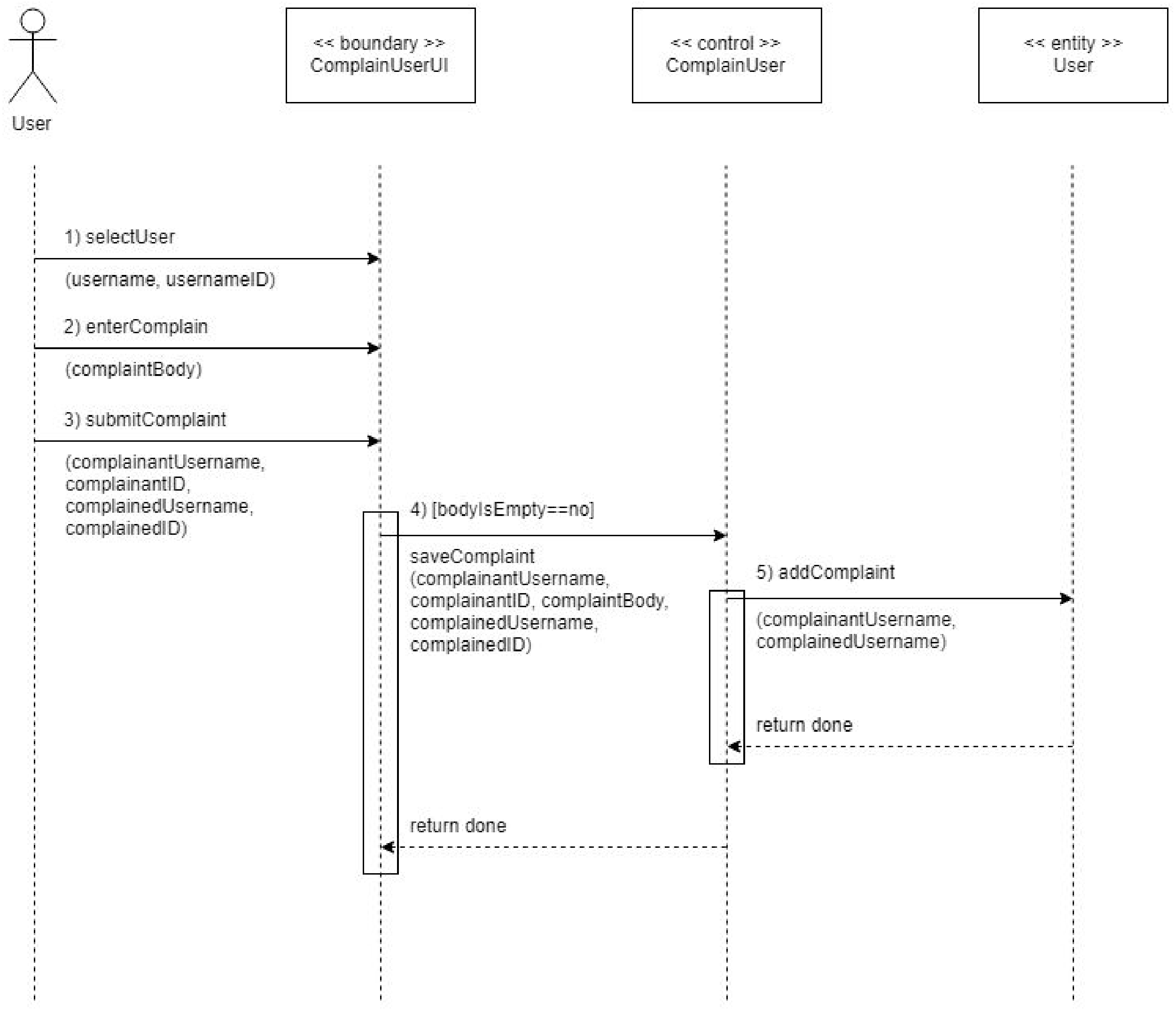
Scenario 1: Requester rates a fulfiller.



***Use-Case Name*:** Use-Case 6.0 Complain User

Description: A user can make a complaint on another user (ideally, if the user being complained about is practicing disruptive behavior). This shall be reviewed by the administrator.

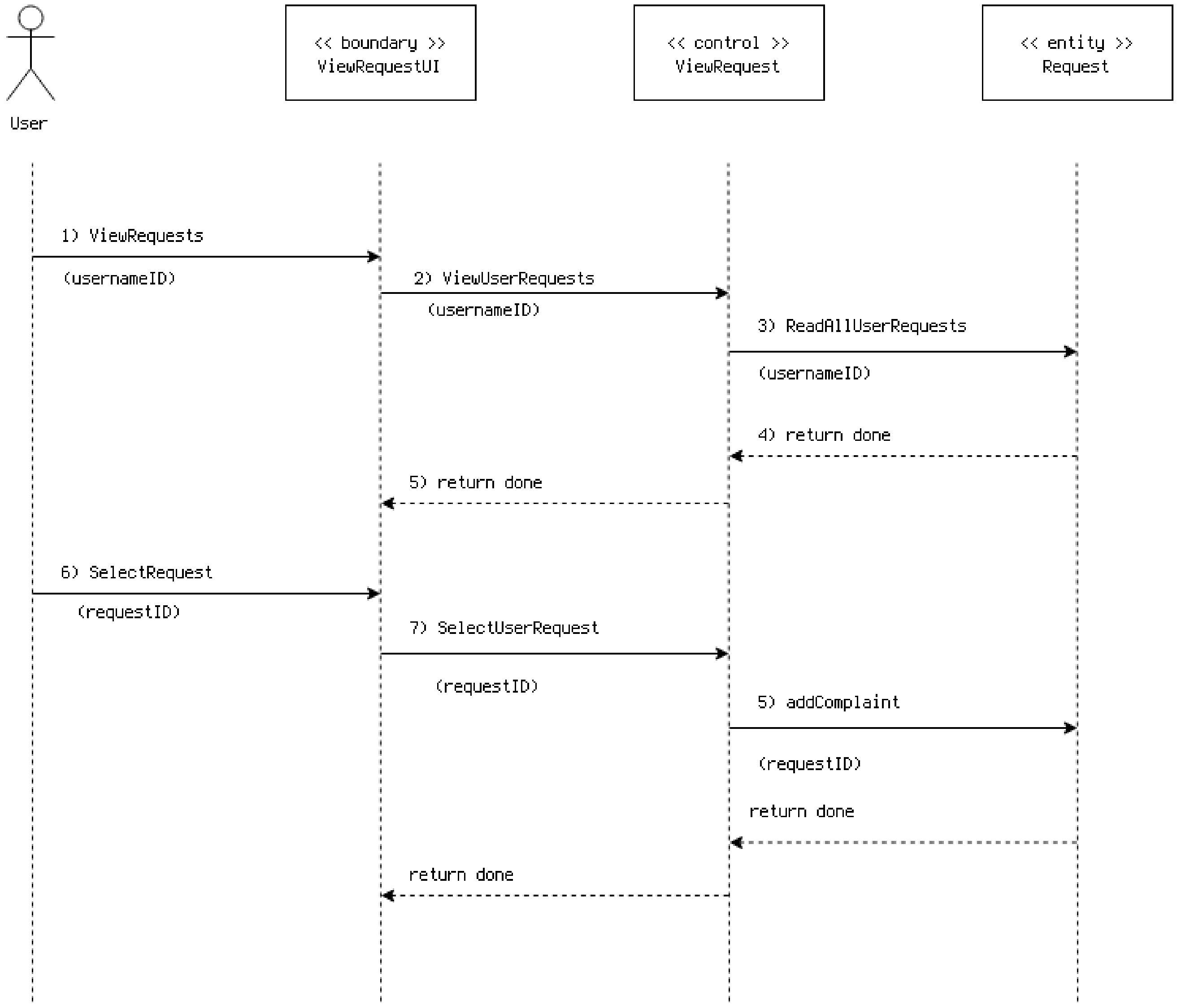
Scenario 1: User submits a complaint on another user (Basic Flow)


User
<< boundary >> ComplainUserUI
<< control >> ComplainUser
<< entity >> User
1) selectUser
(username, usernameID)
2) enterComplain
(complaintBody)
3) submitComplaint
(complainantUsername, complainantID, complainedUsername, complainedID)
4) [bodyIsEmpty==no]
saveComplaint (complainantUsername, complainantID, complaintBody, complainedUsername, complainedID)
5) addComplaint
(complainantUsername, complainedUsername)
return done
return done

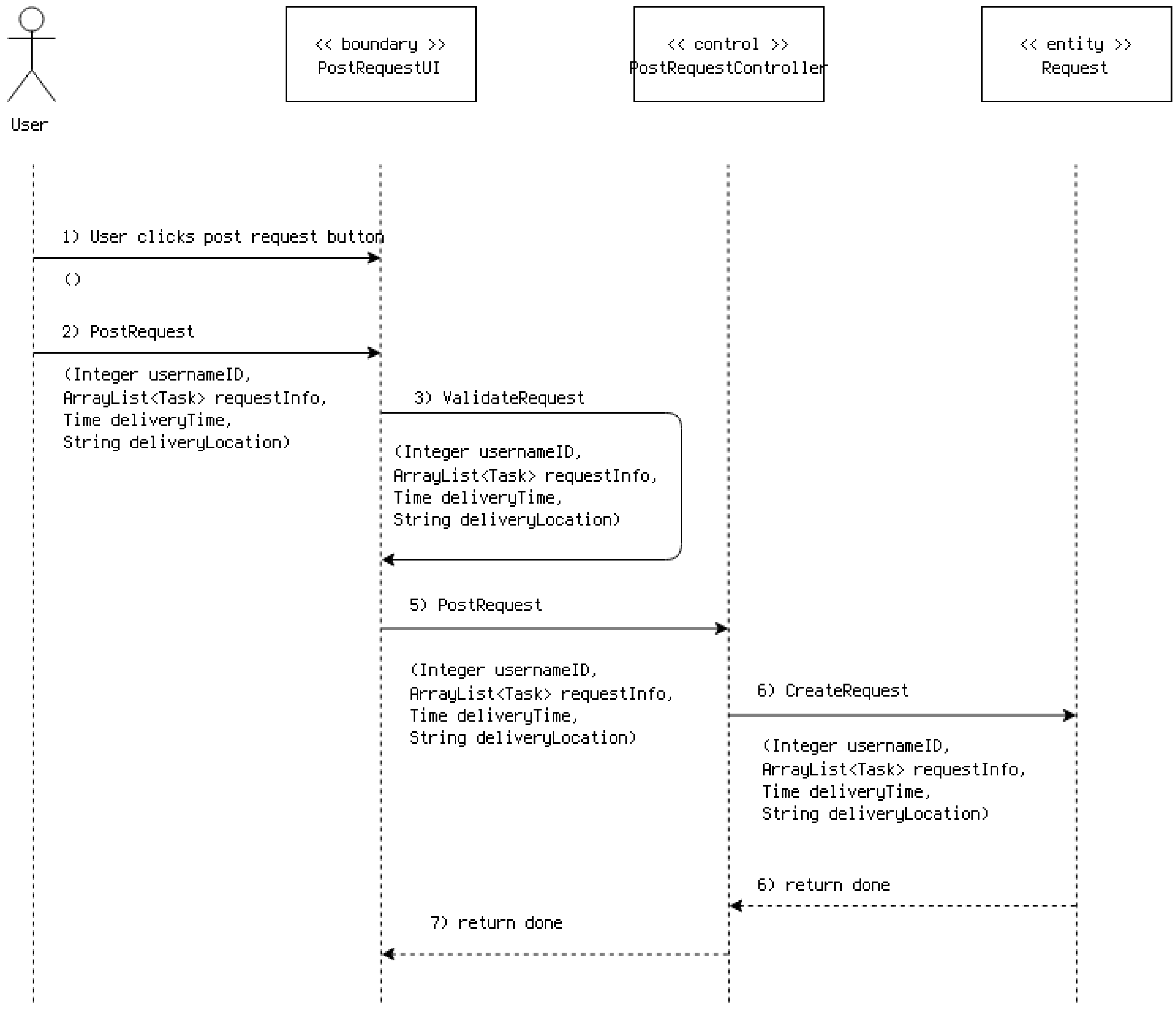
***Use-Case Name*:** Use-Case 7.0 Manage Requests

Description: Every user can post requests for other users to fulfill. After they are posted, users can manage their requests. The user that has posted a certain request is referred to as its requester in the context of that request.

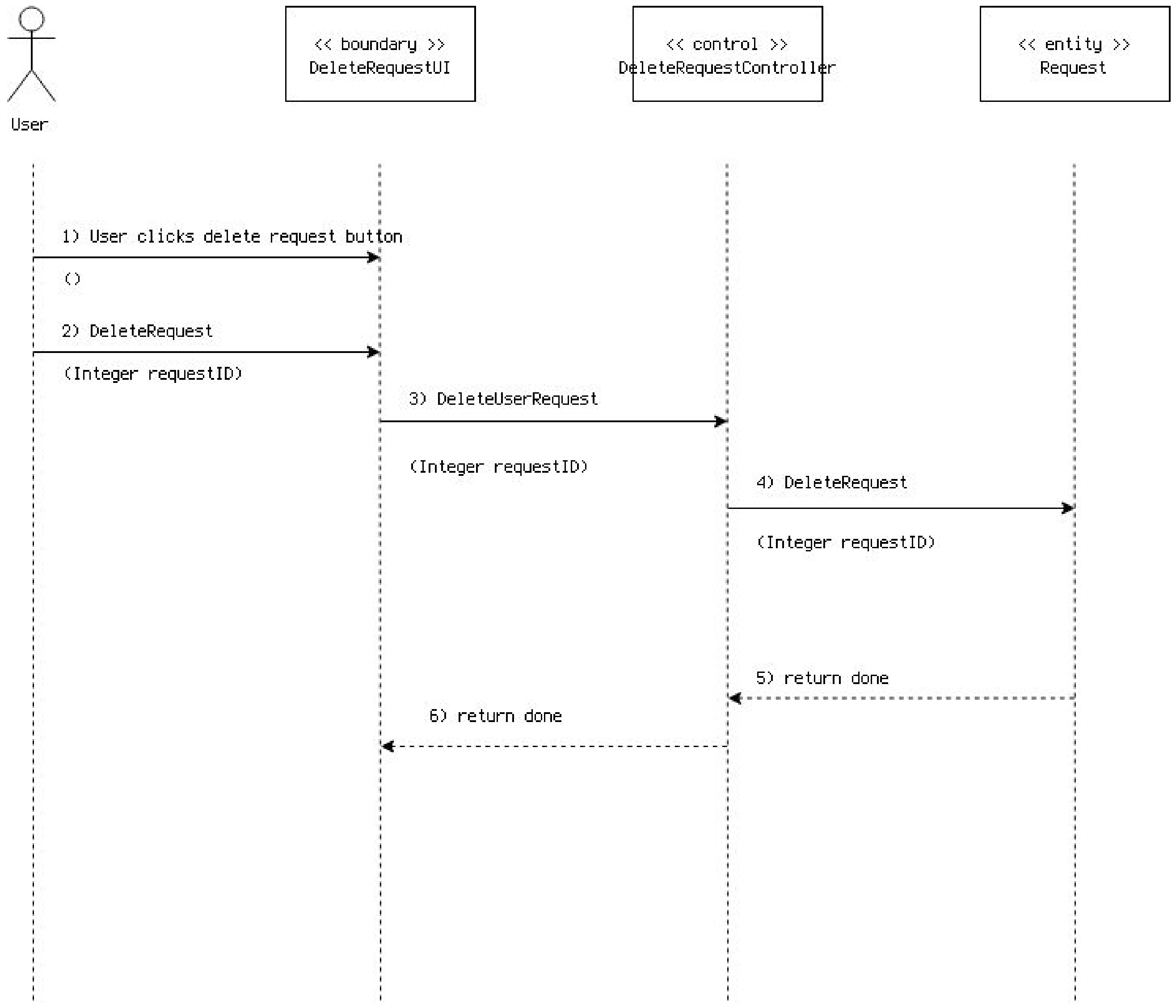
Scenario 1 & 2: Requester views and selects posted requests (Basic Flow)



Scenario 3: Requester posts a request (Basic Flow)



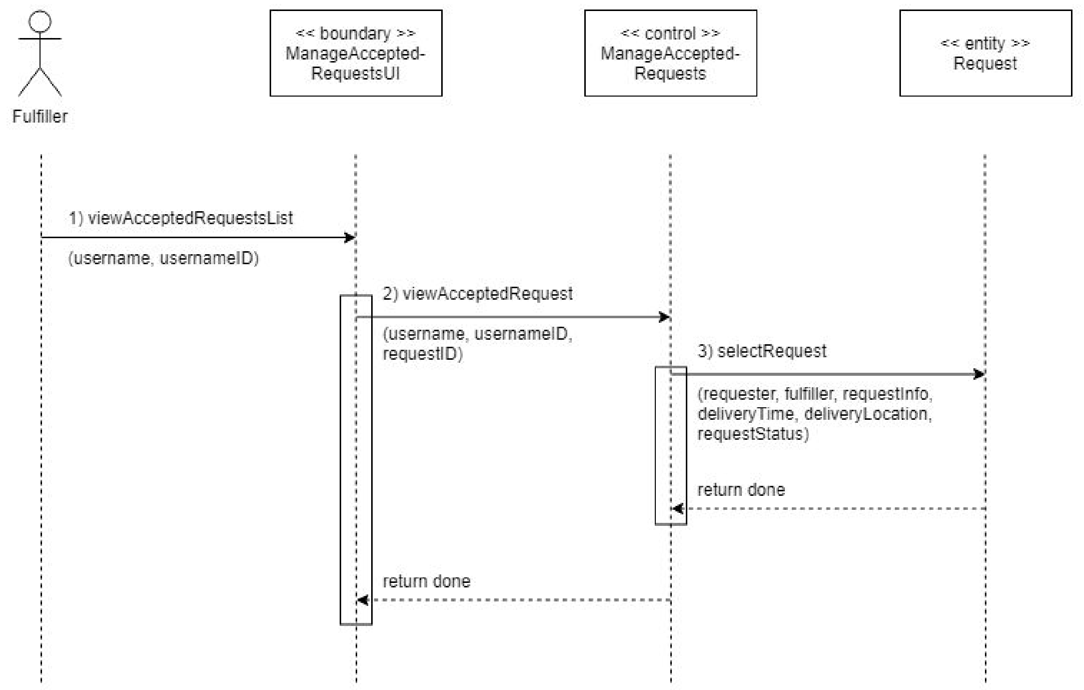
Scenario 4: Requester deletes a request (Basic Flow)



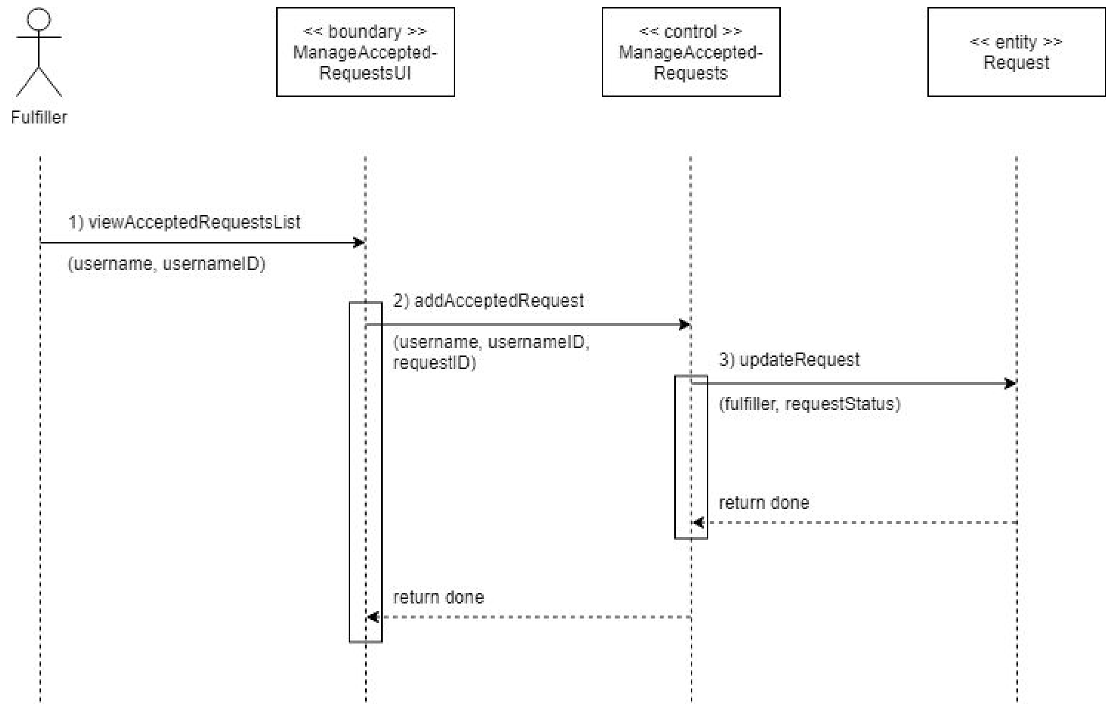
***Use-Case Name*:** Use-Case 8.0 Manage Accepted Requests

Description: A fulfiller can manage their accepted requests. They may either accept a request or cancel an accepted request.

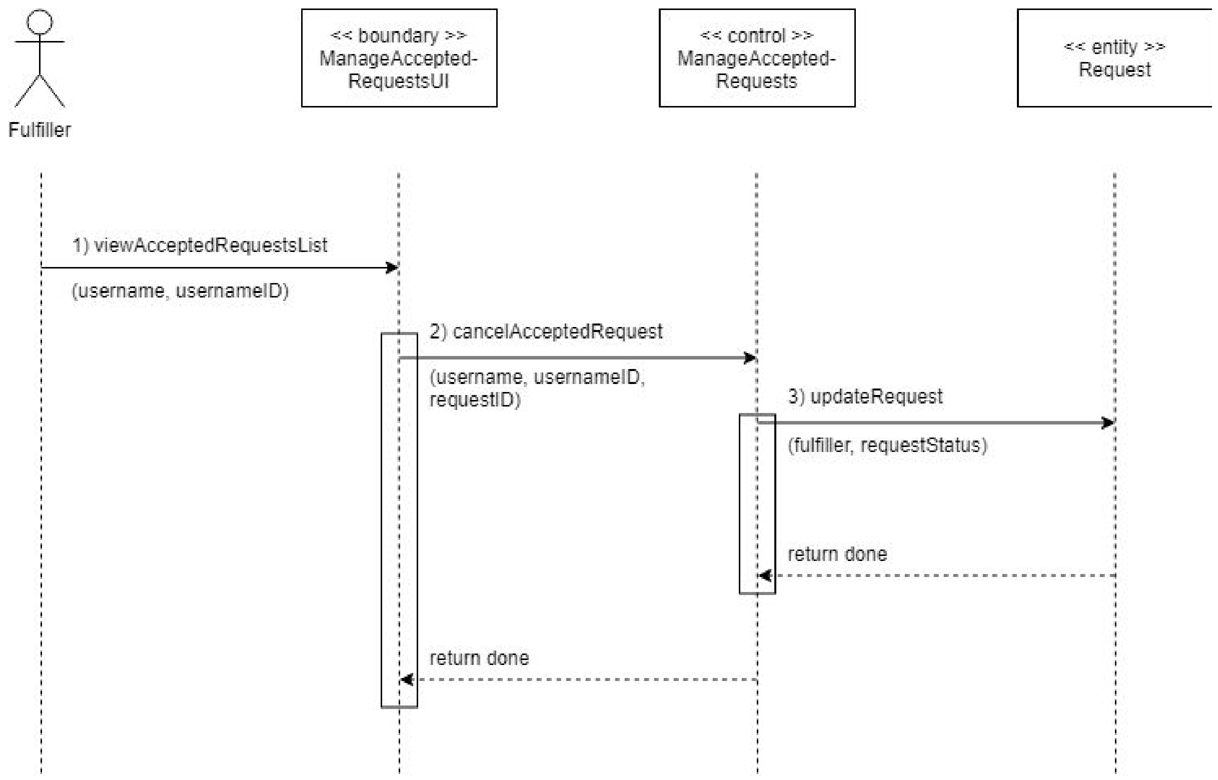
Scenario 1: Fulfiller manages their accepted requests (Basic Flow)


Fulfiller
<< boundary >> ManageAccepted-RequestsUI
<< control >> ManageAccepted-Requests
<< entity >> Request
1) viewAcceptedRequestsList
(username, usernameID)
2) viewAcceptedRequest
(username, usernameID, requestID)
3) selectRequest
(requester, fulfiller, requestInfo, deliveryTime, deliveryLocation, requestStatus)
return done
return done

Scenario 2: Fulfiller accepts a request (Use Case 8.1 Accept Request)


Fulfiller
<< boundary >> ManageAccepted-RequestsUI
<< control >> ManageAccepted-Requests
<< entity >> Request
1) viewAcceptedRequestsList
(username, usernameID)
2) addAcceptedRequest
(username, usernameID, requestID)
3) updateRequest
(fulfiller, requestStatus)
return done
return done

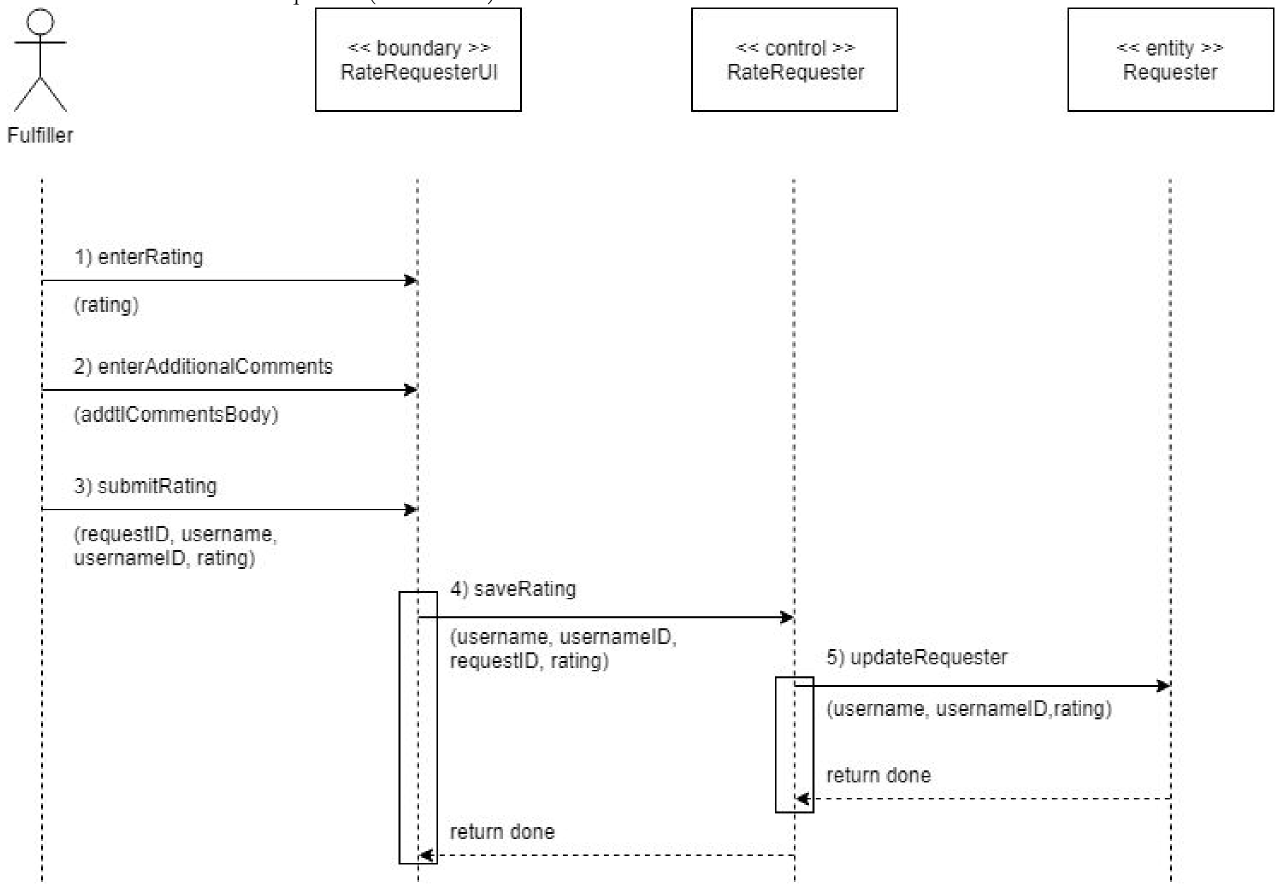
Scenario 3: Fulfiller cancels an accepted request (Use Case 8.2 Cancel Accepted Request)


Fulfiller
<< boundary >> ManageAccepted-RequestsUI
<< control >> ManageAccepted-Requests
<< entity >> Request
1) viewAcceptedRequestsList
(username, usernameID)
2) cancelAcceptedRequest
(username, usernameID, requestID)
3) updateRequest
(fulfiller, requestStatus)
return done
return done

***Use-Case Name*:** Use-Case 9.0 Rate Requester

Description: A fulfiller can rate a requester after fulfilling a request based on their ease of communication and transaction with the requester

Scenario 1: Fulfiller rates requester (Basic Flow)


Fulfiller
<< boundary >> RateRequesterUI
<< control >> RateRequester
<< entity >> Requester
1) enterRating
(rating)
2) enterAdditionalComments
(addtlCommentsBody)
3) submitRating
(requestID, username, usernameID, rating)
4) saveRating
(username, usernameID, requestID, rating)
5) updateRequester
(username, usernameID,rating)
return done
return done