BingeBuddy
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:
Jose, Giovan Samuel R.
Pineda, Lance Dominic M.
Sison Abelardo III P.

In partial fulfillment of Academic Requirements
for the course
CS 191 Software Engineering I
of the
1st Semester, AY <2019-2020>

Unique Reference:

The documents are stored in the https://github.com/Riler4899/Cs-191-Show-Tracker referenced with Grp. 8-BingeBuddy-Analysis Model.pdf.

Purpose:

To further analyze the use cases for the app and their relations to actors, as well as the entities needed for processes

Audience:

The audience would be the people who wish to view and/or comment on any problems with the implementation of the system

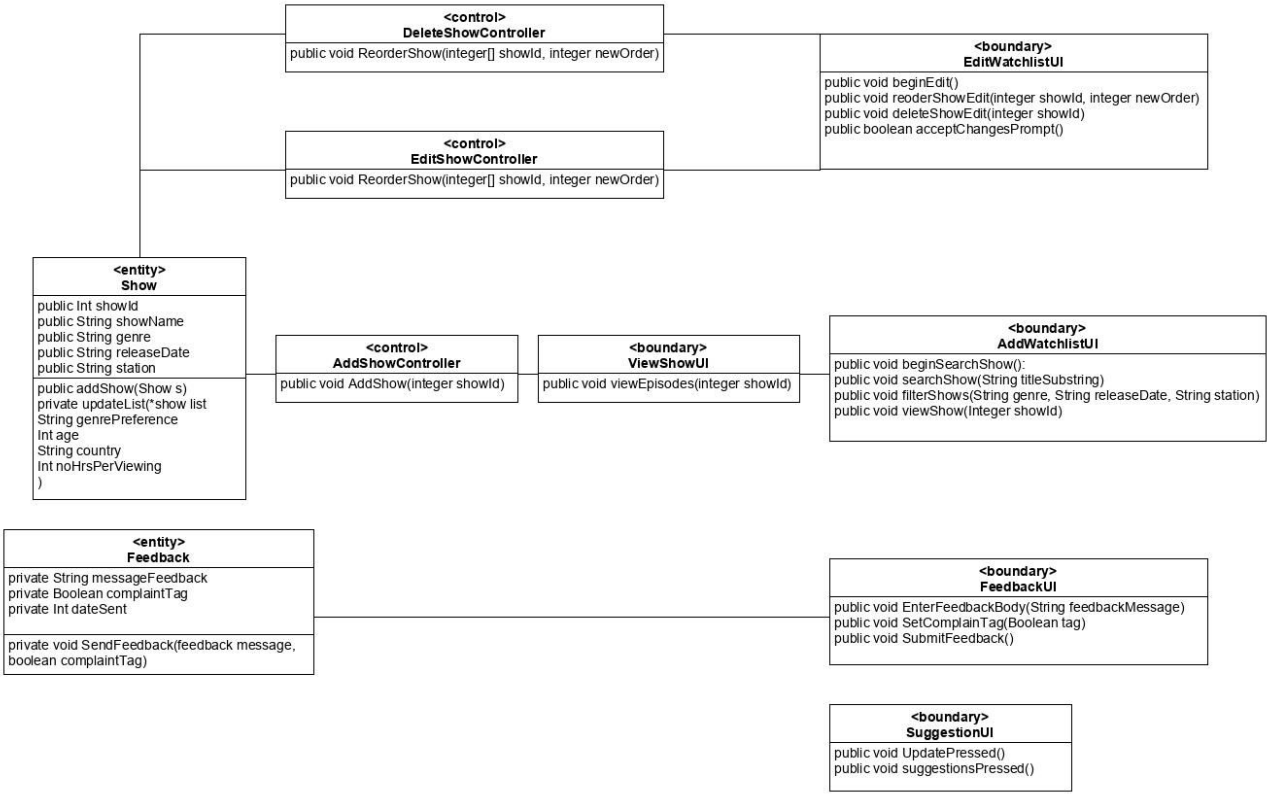
Revision Control:

Revision Date	*Person Responsible*	*Version Number*	*Modification*
10/02/19	Pineda, Lance Dominic	1.0	Initial Document
10/03/19	Pineda, Lance Dominic	2.0	Added boundaries, controls and entities
10/03/19	Sison, Abelardo	2.1	Added more classes and updated Use Case 5.0
10/03/19	Sison, Abelardo	2.2	Edited footer
10/03/19	Jose, Giovan Samuel R	3.0	Revised Control and Boundary Classes. Added analysis model. Added behavioral model for Use Case 1.0

***System Name*:** BingeBuddy Show Tracker System

Description: The system shows how users may create watchlists, maintain and delete show entries, view other show suggestions, and also give feedback about the app.

***Analysis Model*:**



Boundary Classes:

Class Name	Description
AddWatchlistUI	This is the interface of the user whenever he or she decided to add a show to the watchlist. *Responsibilities* public void beginSearchShow(): public void searchShow(String titleSubstring) public void filterShows(String genre, String releaseDate, String station) public void viewShow(Integer showId)
EditWatchlistUI	This is the interface of the user whenever he or she decided to edit the watchlist. (After select and holding a show in the watchlist) *Responsibilites*: public void beginEdit() public void reoderShowEdit(integer showId, integer newOrder) public void deleteShowEdit(integer showId) public boolean acceptChangesPrompt()
ViewWatchlistUI	This is the default interface where the watchlist can be viewed. *Responsibilites*: public void searchShowWatchlist(String titleSubstring) public void filterShowWatchlist(String genre, String releaseDate, String station) public void viewShow(Integer showId)
ViewShowUI	This is the interface whenever the user clicks on a show *Responsibilites*: public void viewEpisodes(integer showId)
FeedbackUI	This is the interface for making feedback or complaint *Responsibilites*: public void EnterFeedbackBody(String feedbackMessage) public void SetComplainTag(Boolean tag) public void SubmitFeedback()
SuggestionUI	Interface for suggestions

	Responsibilites: public void UpdatePressed() public void suggestionsPressed()

Control Classes:

Class Name	**Description**
MaintainWatchlistController	This is the control that maintains the list of shows that the user has accumulated. This is considered an abstract class.
AddShowController	This is the control that adds shows to a watchlist of a user. It extends MaintainWatchlistController *Responsibilities*: public void AddShow(integer showId)
EditShowController	This is the control that performs various modifications to a show in the watchlist It extends MaintainWatchlistController *Responsibilities*: public void ReorderShow(integer[] showId, integer newOrder)
DeleteShowController	This is the control that deletes shows from a user's watchlist.. It extends MaintainWatchlistController *Responsibilities*: public void DeleteShow(integer[] showId)
SendFeedbackController	This is the control that sends feedback to the server. It can also cancel should the user decide to withdraw his or her comment. *Responsibilities*: public void SendFeedback(feedback message, boolean complaintTag, boolean cancel) public String CancelFeedback()
SuggestionController	This is the control that handles the suggestions *Responsibilities*: public *show showSuggestions(UserName) public void updateSuggestions(Username) public String getGenrePreference() public Int getAge() public String getCountry() public Int getNoHrsPerViewing()

Entity Classes:

Class Name	Description
Show	This is the entity class Show, which contains data about a show *Attributes:* public Int showId public String showName public String genre public String releaseDate public String station *Methods:* public addShow(Show s) private updateList(*show list String genrePreference Int age String country Int noHrsPerViewing)
User	This is the entity class User, which contains information about the user himself or herself *Attributes:* private String userName private String genrePreference private Int age private String country private Int noHrsPerViewing private *Show suggestionList *Methods* private updateList(*show list String genrePreference Int age String country Int noHrsPerViewing) public String getUserName() public String getGenrePreference() public Int getAge() public String getCountry()

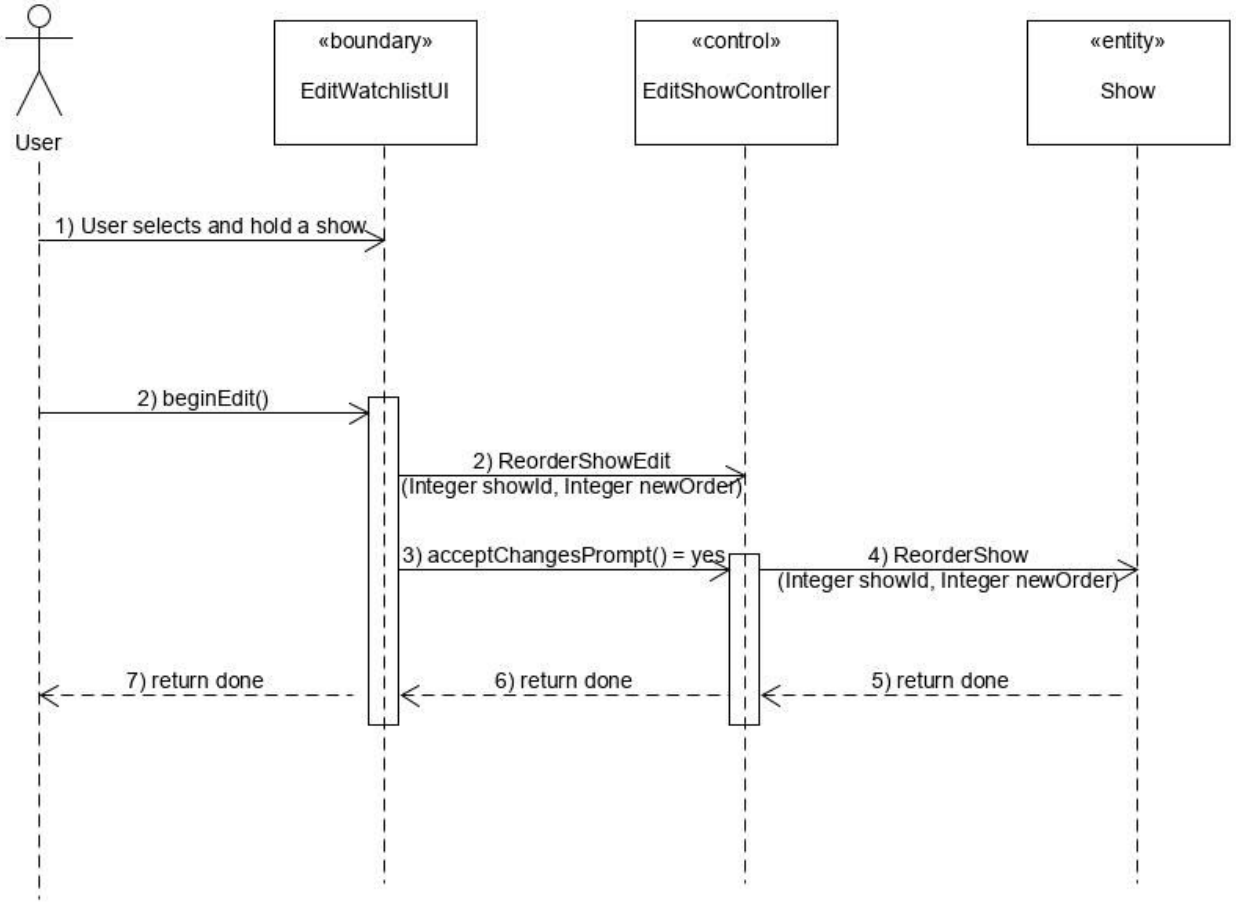
	public Int getNoHrsPerViewing() public *Show GetSuggestionList()
Feedback	This is the entity class of a feedback message *Attributes:* private String messageFeedback private Boolean complaintTag private Int dateSent *Methods:* private void SendFeedback(feedback message, boolean complaintTag)

Behavioral Model:

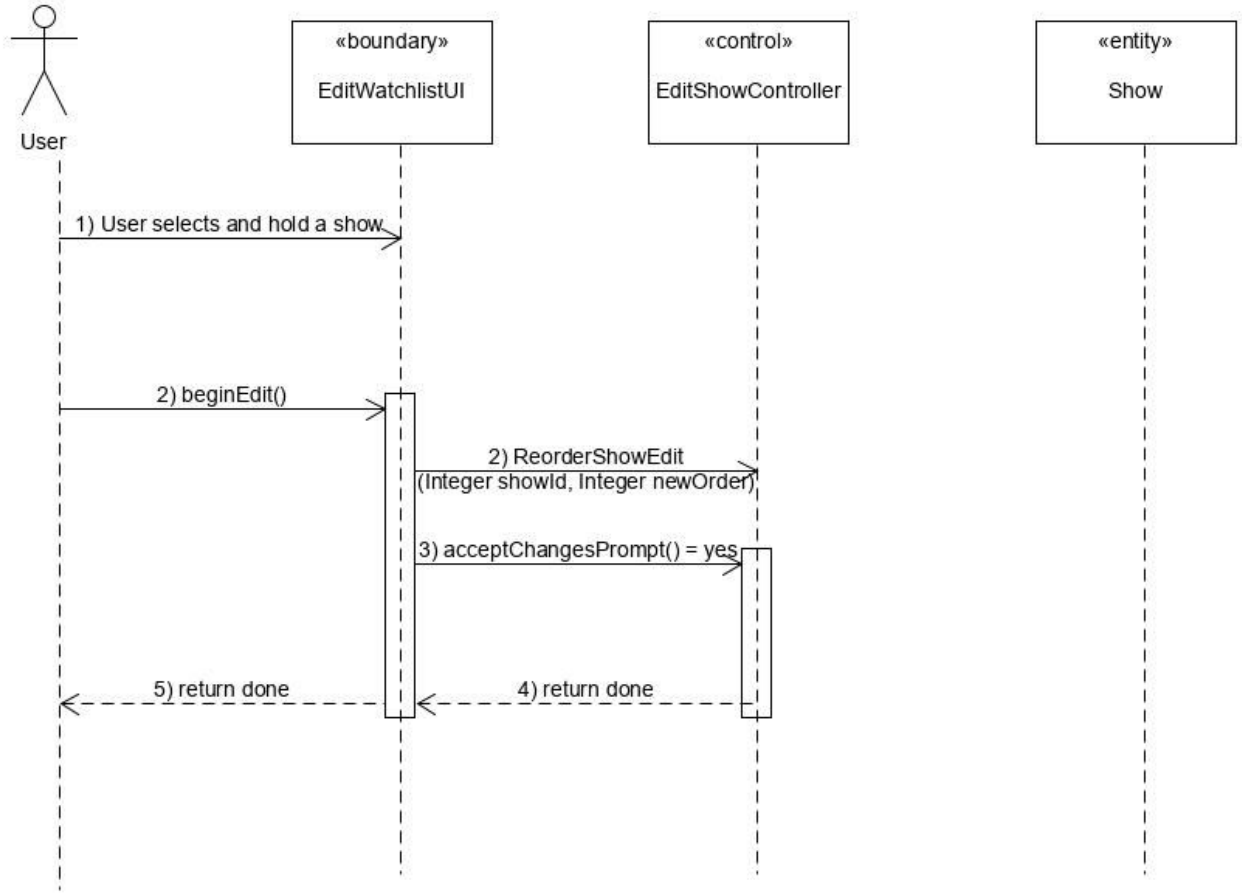
***Use-Case Name*:** 1.0 Maintain Watchlist

Description: This use case contains the general actions needed to maintain a watchlist. These include adding, editing, deleting and viewing watchlists. This also includes updating watchlists, though this action is just a combination of the other actions.

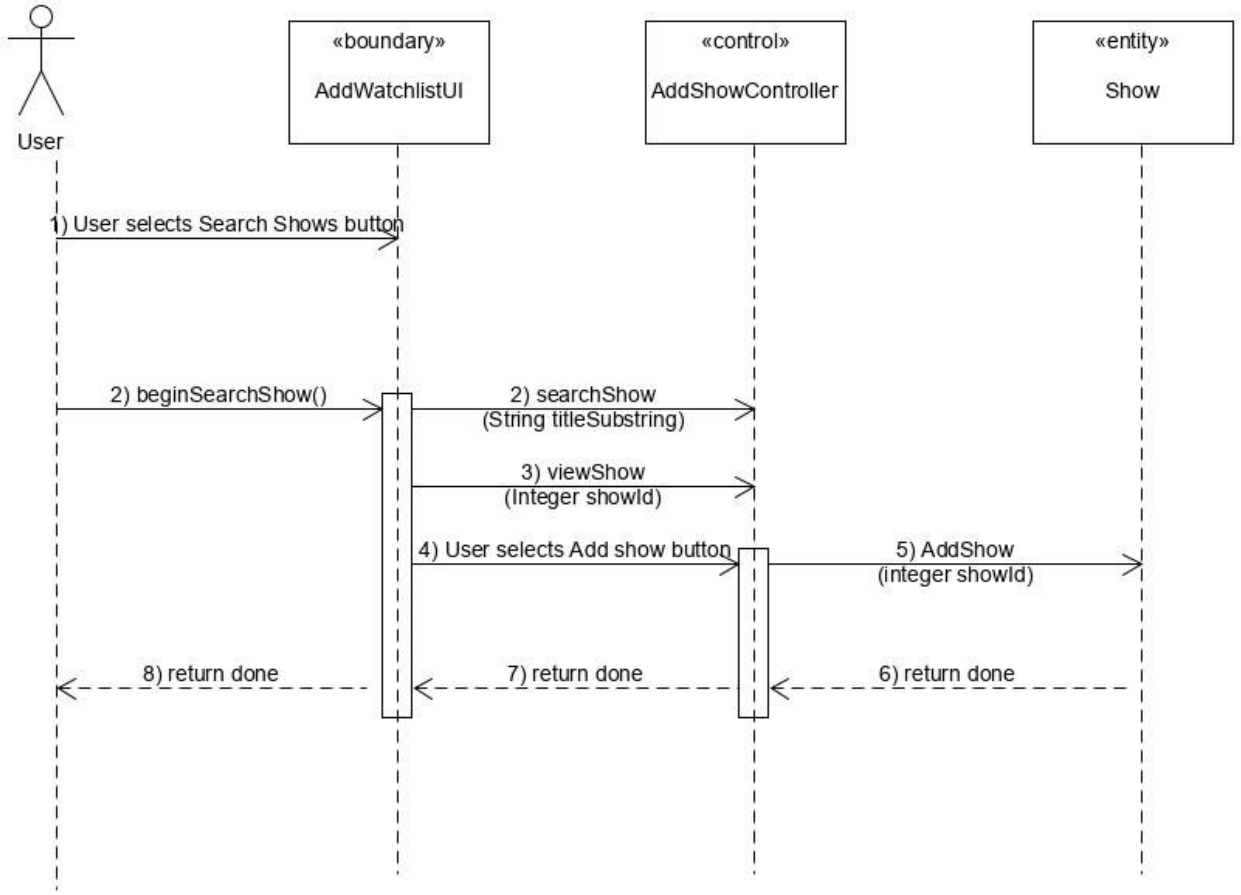
Scenario 1a: User edits a show and accepts changes (Basic Flow)



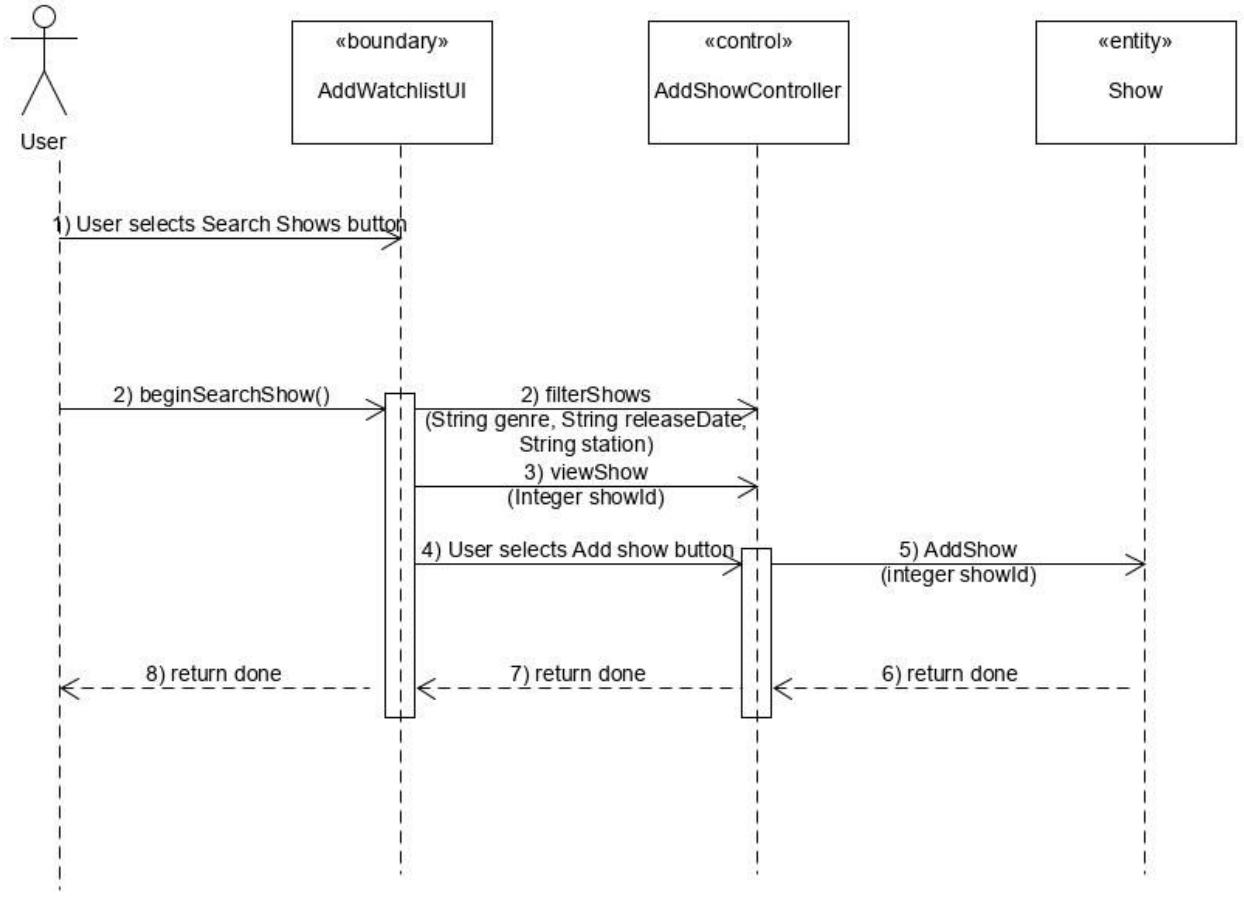
Scenario 1b: User edits a show and cancels changes.



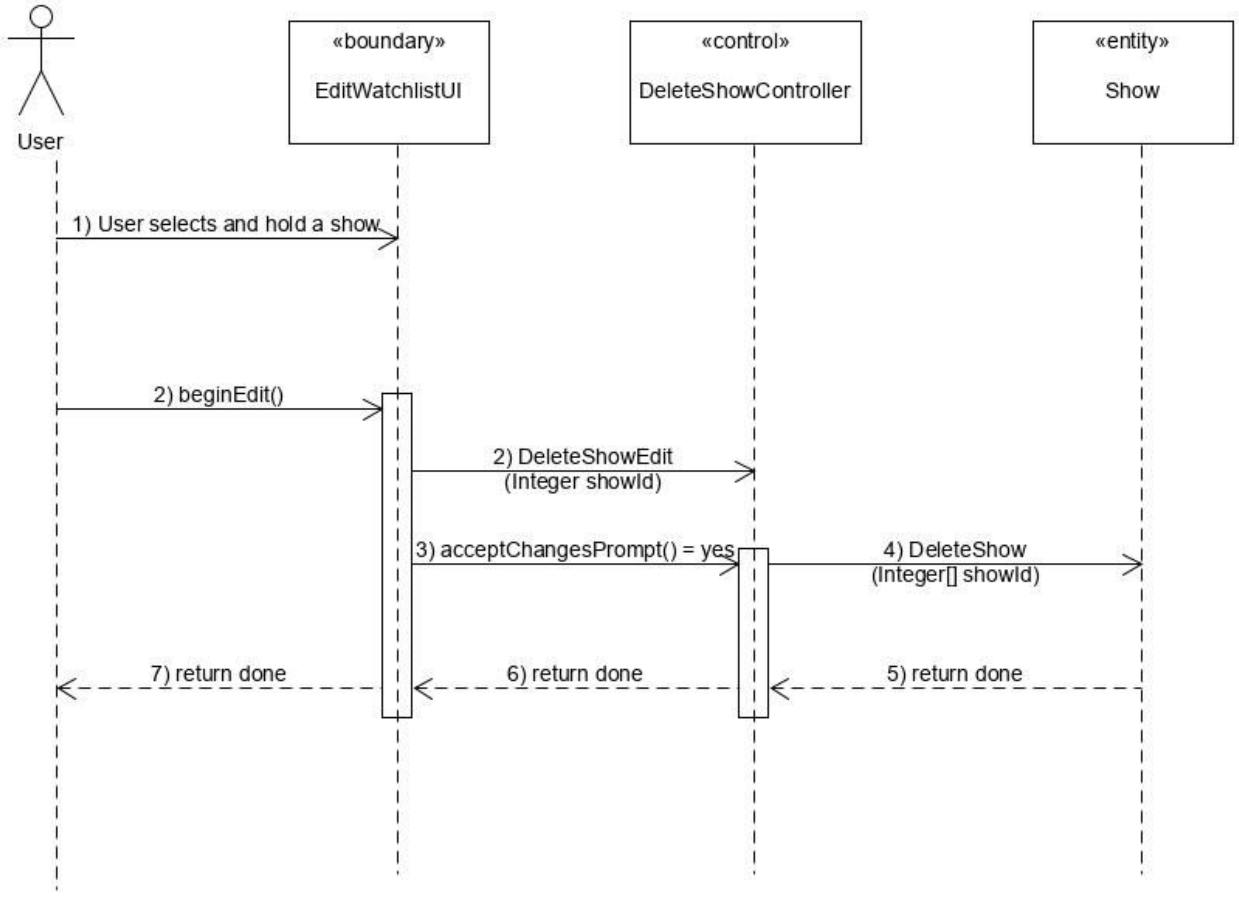
Scenario 2a: User adds a show and searches using search bar



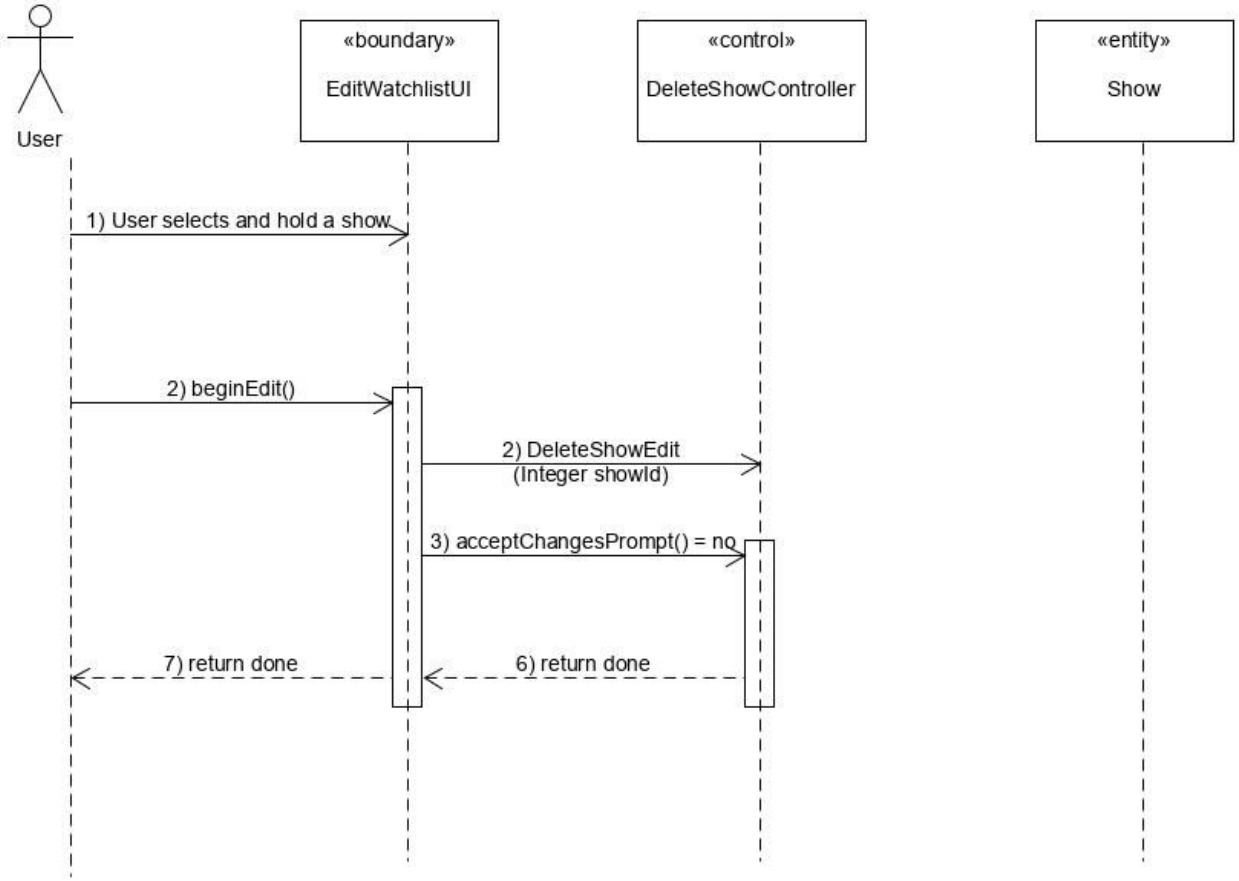
Scenario 2b: User adds a show and uses filters.



Scenario 3a: User deletes a show from the watchlist and accepts changes



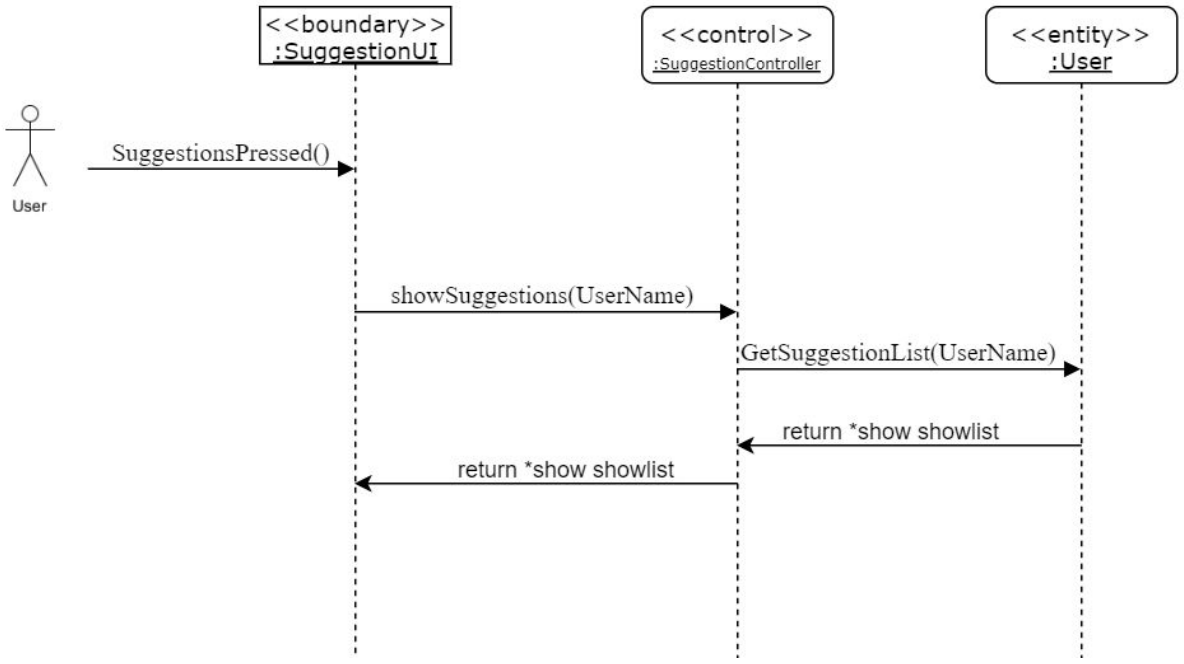
Scenario 3b: User deletes a show from the watchlist and cancels changes



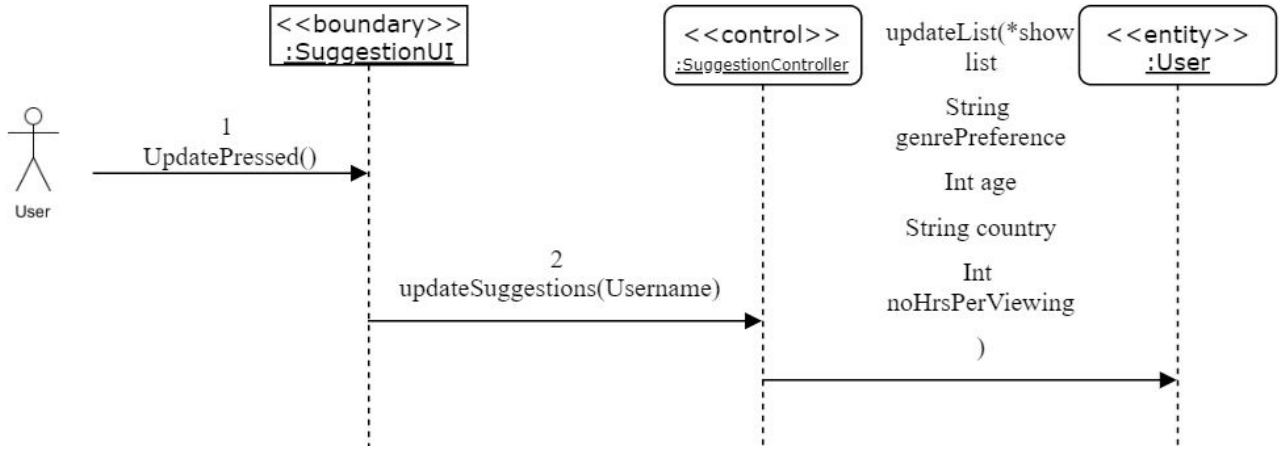
***Use-Case Name*:** 3.0 View Suggestions

Description: This use case is for the user to view other shows the user has a potential of liking. The attributes needed to determine this come from the demographics of the user. This use case is for making the app another avenue for the user to explore other shows and for building their watchlists.

Scenario 1: Basic Flow



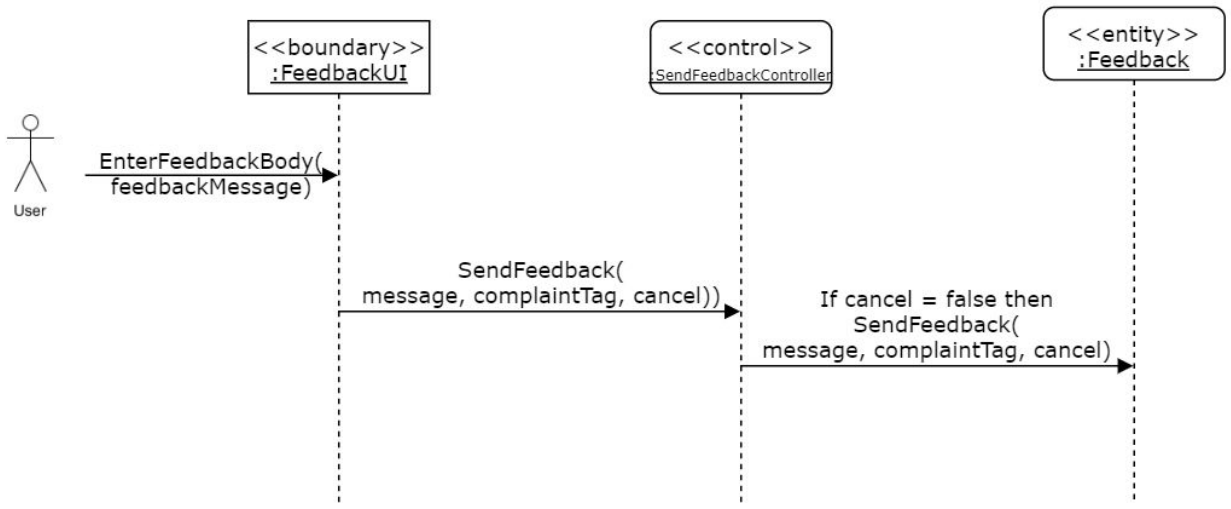
Scenario 2: Update Suggestion List



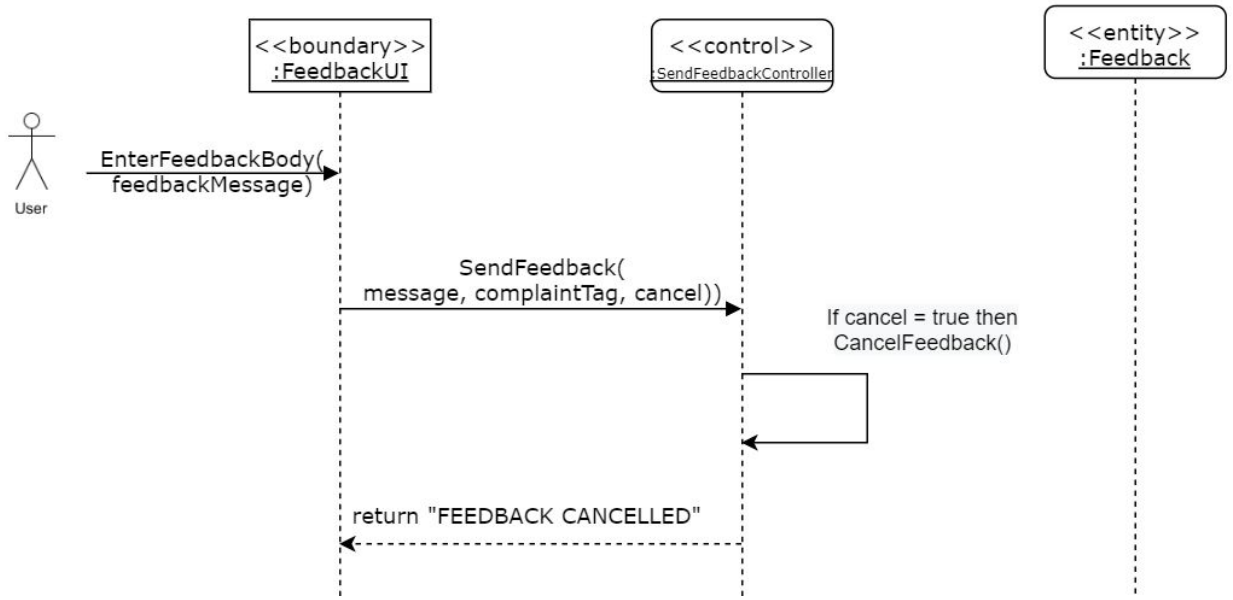
***Use-Case Name*:** 5.0 Give Feedback

Description: This use case is for the user to give suggestions on how to make the application better. The actions required involve sending messages to the developers so that any criticisms may be noted. Users may also send in complaints is ever there is any problems they encounter with the app.

Scenario 1: User has comments and suggestions or complaints (Basic Flow)



Scenario 2: Cancel feedback


<<boundary>>
:FeedbackUI
<<control>>
:SendFeedbackController
<<entity>>
:Feedback
User
EnterFeedbackBody(
feedbackMessage)
SendFeedback(
message, complaintTag, cancel))
If cancel = true then
CancelFeedback()
return "FEEDBACK CANCELLED"