Project Portfolio Page: EzWatchlist

This document serves to show the various contributions Jared has made to the development of **EzWatchlist** in an understandable manner.

More About The Project

EzWatchlist is a desktop application that allows users to keep track of movies and tv series that they want to watch or have watched. It was developed by a group of 5 students from the National University of Singapore taking the Software Engineering module including myself under the premise that we had to either optimize or morph an existing AddressBook application. We chose the latter. Another constraint was ensuring that the application had a command-line interface.

Main Features

- Adding, editing and deleting Shows
- Keeping track of shows that are watched
- Online search functionality
- Statistics about user’s watchlist

Summary of Contributions

This section provides a summary of my coding, documentation and other helpful contributions to EzWatchlist.

Major Contributions

- Implemented of the **watch** feature for users to keep track of episodes of their tv series and movies watched
    - The **watch** feature allows users to update the watch list if the show has been watched
    - The watch feature enables users to keep track of shows that they have and have not watched that they can plan out what they want to watch in the future
    - The watch feature also enables users to keep track of the episodes in a TV series that have been watched

Minor Contributions

- Updating to the user and developer guides to make them more readable, including the use of diagrams and screenshots of the application
- Updating to EzWatchlist **Contact Us** page
- Writing of tests for EzWatchlist to reduce the possibility of bugs when using EzWatchlist

- Integrating of the features of EzWatchlist to improve the user-experience of the application

View code contributions here.

Contributions to User Guide

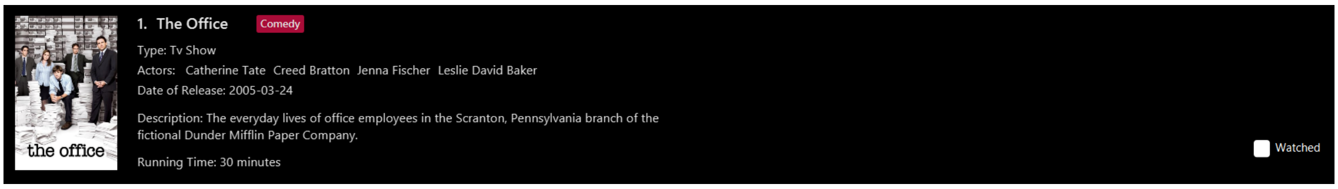
This section serves to showcase my contributions to the User-Guide, as well as my ability to write documentation for users in a concise and understandable manner.

Mark/Unmark as watched : `watch`

To mark an unwatched show in the watchlist as watched, use the command format listed below.
Format: `watch INDEX [e/EPISODE_NUMBER] [s/SEASON_NUMBER]`

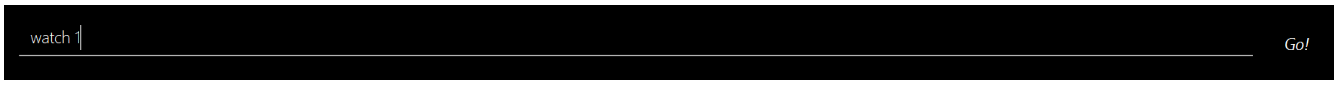
Example Usage:

1. User wants to mark "The Office" as watched.



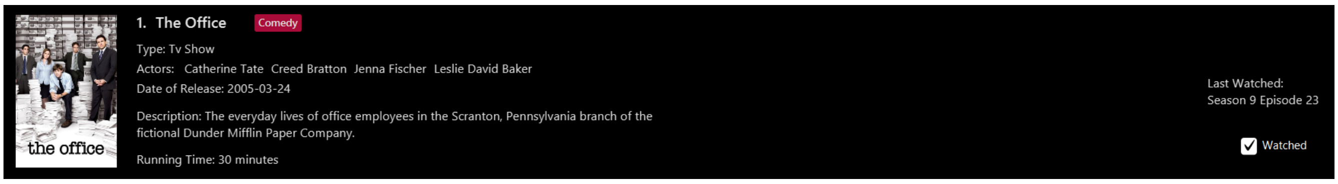
1. The Office Comedy
Type: Tv Show
Actors: Catherine Tate Creed Bratton Jenna Fischer Leslie David Baker
Date of Release: 2005-03-24
Description: The everyday lives of office employees in the Scranton, Pennsylvania branch of the fictional Dunder Mifflin Paper Company.
Running Time: 30 minutes
Watched

2. User enters `watch 1` into the command box in the watchlist tab.



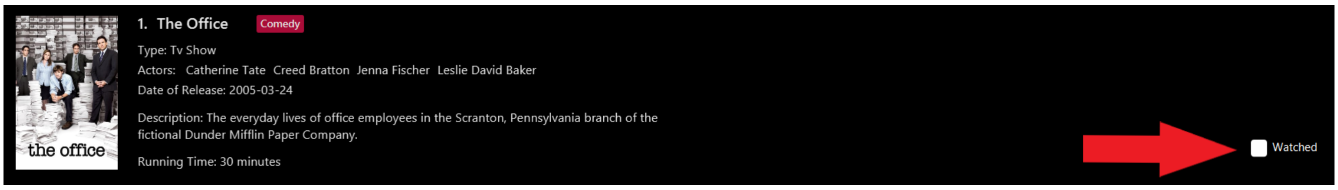
watch 1

3. The show can now be viewed under the watched tab by clicking the watched tab or hitting the keyboard shortcut `2`.



1. The Office Comedy
Type: Tv Show
Actors: Catherine Tate Creed Bratton Jenna Fischer Leslie David Baker
Date of Release: 2005-03-24
Description: The everyday lives of office employees in the Scranton, Pennsylvania branch of the fictional Dunder Mifflin Paper Company.
Running Time: 30 minutes
Last Watched:
Season 9 Episode 23
✓ Watched

Alternatively, users can click on the watched checkbox to toggle between whether a show is watched as indicated by the red arrow in the image below.



1. The Office Comedy
Type: Tv Show
Actors: Catherine Tate Creed Bratton Jenna Fischer Leslie David Baker
Date of Release: 2005-03-24
Description: The everyday lives of office employees in the Scranton, Pennsylvania branch of the fictional Dunder Mifflin Paper Company.
Running Time: 30 minutes
Watched

- The `index` refers to the index number shown in the displayed watchlist. The index **must be a positive integer** 1, 2, 3, …
- Any number of the optional fields may be provided.
- Having only the index of the show will mark/unmark the show as watched.
- Having the index and the episode number of the show will update the cumulative number of episodes of the show that are watched.
- Having the index and the season number of the show will update the cumulative number of seasons of the show that are watched.
- Having the index, season number and the episode number of the show will update the last watched episode to be the indicated episode of the indicated season of the show.

TIP Using the `watch` command on an already watched show will un-mark the show as watched.

Examples:

- `watch 1`
  Marks/un-marks the first show of the list as watched.
- `watch 2 e/20`
  Marks the first 20 episodes of the second show of the list as watched.
- `watch 2 s/5`
  Marks all episodes of the first 5 seasons of the second show as watched.
- `watch 3 s/5 e/2`
  Marks all episodes up to and including the second episode of the fifth season of the third show in the list as watched.

Contributions to Developer Guide

This sections showcases my contribution to the EzWatchlist Developer guide for the `watch` function.

[Feature] Mark/unmark as watched feature

The watch feature allows users to mark or unmark shows as watched. It also allows users to keep track of the latest episode of a TV series that they have watched.

Implementation

The mark/unmark as watched mechanism is facilitated by `WatchCommand` which can be found under the commands package. It extends `Command` and uses the `WatchCommandParser` to process the command entered by the user.

Given below is an example usage scenario and how the mark/unmark as watched mechanism works at each step.

Step 1. The user launches the application, and executes `watch 1 s/2 e/3` command to update the latest watched episode of the first show in the list.

Step 2. Entering the command calls `WatchListParser#parseCommand()`, which in turn returns a new `WatchCommandParser` and the `WatchCommandParser#parse()` command is called.

Step 3. A new `WatchCommand` is created, with the index of the show being parsed as a field of the `WatchCommand`. A new `WatchShowDescriptor` is also created to relay the episode number and season number to the `WatchCommand` object.

Step 4. The `WatchCommand#execute()` method is called, referencing the current `model`, and the show that is in the current `FilteredShowList` is referenced based off the current `model`.

NOTE If the `index` is out of bounds, a new `CommandException` is thrown.

Step 5. A copy of the show is created through the use of `WatchCommand#createEditedShow()`, with the new total number of seasons and episodes updated if there are any changes. A new isWatched value of the show is also determined based on the number of episodes that are watched.

The following activity diagram below explains the summarizes the calculation of the number of episodes watched.

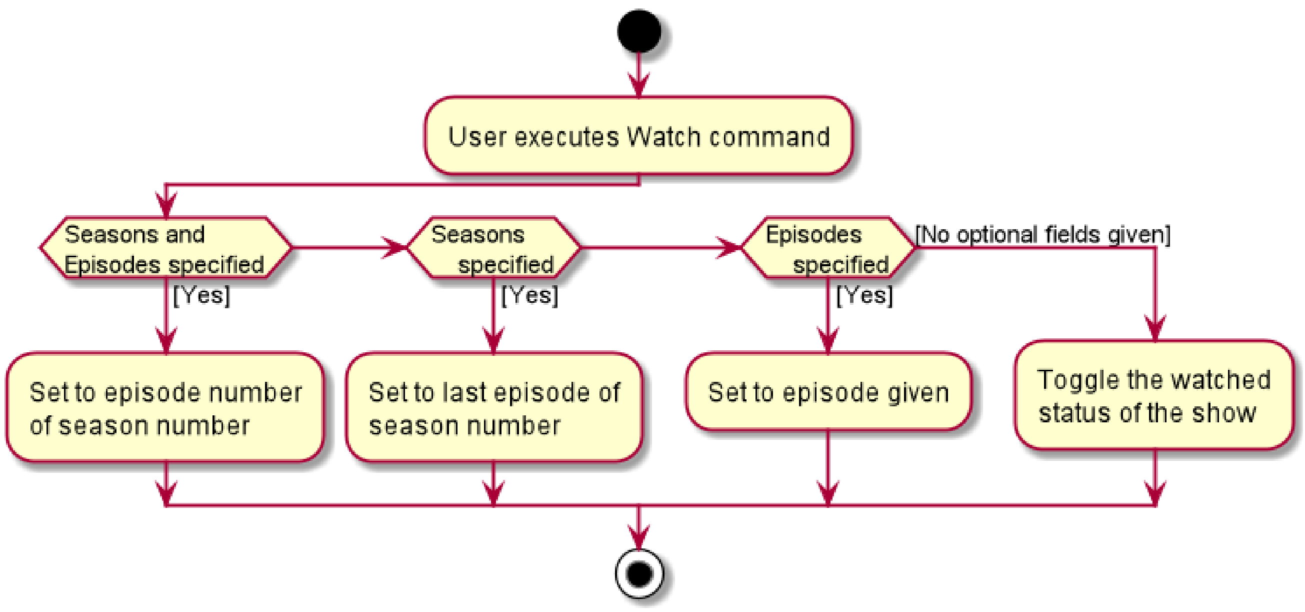

User executes Watch command
Seasons and Episodes specified
Seasons specified
Episodes specified
[No optional fields given]
[Yes]
[Yes]
[Yes]
Set to episode number of season number
Set to last episode of season number
Set to episode given
Toggle the watched status of the show

Figure 1. Calculating the Number of Episodes Watched

Step 6. The show in the current show list is updated to the newly created copy with the updated watched status and latest episode watched, and a `CommandResult` with the new watched status of the show is created.

The following sequence diagram shows how the watch operation works:

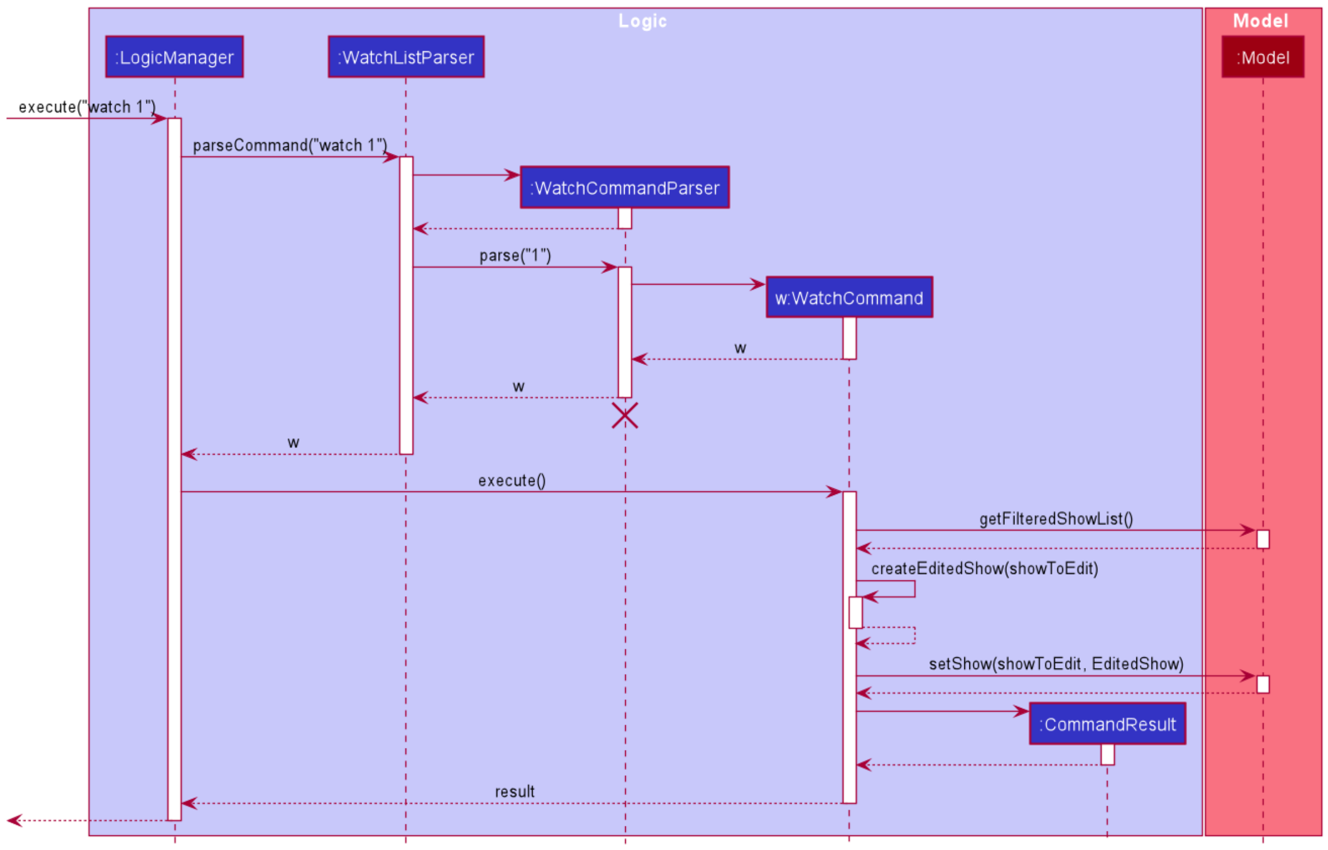


Figure 2. WatchSequenceDiagram Showing Events in Sequence

Design Considerations

Aspect: Creating a new WatchCommand instead of an altered EditCommand

- **Alternative 1 (current choice):** Creating a new WatchCommand class for changing the 'watch' status of a show.
    - Pros: Enables for greater cohesion since there is a specific command for editing the 'watch' status of a show.
    - Cons: Requires longer code, and the code is also repetitive since its implementation is similar to that of the edit command
- **Alternative 2:** Use the WatchCommandParser to create a new EditCommand object that edits the watch status of the show.
    - Pros: Less code repetition and shorter code in general.
    - Cons: This will mean that there is less cohesion of the code and perhaps greater dependencies since more classes depend on the EditCommand class.